Formal Modelling and Analysis of Safety-Critical Interactive Systems using Coloured Petri Nets

A thesis

submitted in fulfilment

of the requirements for the degree

of

Doctor of Philosophy in Computer Science

at

The University of Waikato

by

SAPNA JAIDKA

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2020

Abstract

To gain confidence in safety-critical interactive systems, formal modelling and analysis plays a vital role. Generally, existing techniques focus either on modelling the user interface or on modelling the functionality of a system. Although there are many benefits to using the individual models for different purposes, it requires a lot of work to do the coupling of functional behaviour with interactive elements for analysis. Therefore, further investigation into the modelling and analysis techniques was required that models all the parts (user interface, interaction and functional) into a single model.

This research aims to apply formal methods for modelling and specifying the user interface, interaction and functional aspects of a safety-critical system in a single model using Coloured Petri Nets (CPN), then investigating the model to ensure that the system behaves as expected. The approach developed in this thesis has its starting point in several existing, accepted formal specification techniques. From this existing basis, we create a Coloured Petri Net model of a system which has the required features of existing formalisms, taking into account all three aspects (user interface, interaction and functional), hence our investigation of the combination of formalisms to achieve their combined strength.

There are several reasons for using Coloured Petri Nets. Coloured Petri Nets provide a graphical representation and hierarchical structuring mechanism, and a state space verification technique, which allows querying the state space to investigate behaviours of a system. There are several tools that support Coloured Petri Nets including the CPN Tool which helps in building CPN models and allows simulation and analysis using state spaces.

In this thesis, the findings of our investigation into modelling and analysis of safety-critical interactive systems are presented. We describe the technique developed to model and analyze an interactive system using Coloured Petri Nets. The technique is illustrated using a simplified infusion pump example.

Then we present a case study of the Niki T34 Infusion Pump to show that we have retained all the expressiveness that we need of existing formalisms. Lastly, we present a small example of a nuclear reactor control system to show that now we can use the Coloured Petri Nets alone to model and analyze the user interface, interaction and functionality of safety-critical interactive systems and also to show that the scope of this technique is not limited to just the medical domain.

Acknowledgements

Hare Krishna - first and foremost - My Lord who has been very kind to me for bestowing upon me the courage, strength, and ability for completing this research journey successfully.

I would like to thank my supervisors, Steve Reeves and Judy Bowen, without whom this PhD would not have been possible. I have been very fortunate to have you as my supervisors. Thank you, Steve and Judy, for all of your guidance, support, and encouragement over the years. You have taught me how to focus on the important aspects, and how to look at the bigger picture. Your advice and insight over the past few years have been invaluable. I will forever appreciate the time you have each put into my personal and professional development.

I am also grateful to the University of Waikato for funding this research through the generous Doctoral Scholarship and for funding travel to the international conference. Without this, I would not have been able to undertake or continue the doctoral study.

Many thanks to my friends and colleagues for their invaluable help. I appreciate each of your unique contributions at different stages of my PhD journey.

Finally, I would like to express my deepest love and gratitude to my family for all their encouragement and support throughout this journey. My husband, Karan, and son, Tejas, for giving their unconditional love and support throughout. You both have shown immeasurable patience and understanding. My elders, Ashwani Kumar Verma, Sudesh Verma, Manju Jaidka and Raksha Rani, for their endless prayers and support. I am indebted to all of you for your unconditional love and emotional encouragement in my life.

Publications

Some of the research presented in this thesis is published in the following:

- S. Jaidka, S. Reeves, and J. Bowen, "Modelling safety-critical devices: Coloured Petri Nets and Z," in *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pp. 51–56, ACM, 2017 [1].

- S. Jaidka, S. Reeves, and J. Bowen, "A coloured Petri Net approach to model and analyze safety-critical interactive systems," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 347–354, IEEE, 2019 [2] .

- S. Jaidka, S. Reeves, and J. Bowen, "Formal modelling of safety-critical interactive devices using Coloured Petri Nets," 2019. To appear in *The 8th Formal Methods for Interactive Systems workshop, participants proceedings FMIS, 2019.*

As primary author for these publications the technical contributions come from my research, the other authors on these papers are my supervisors.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Interactive systems are systems that take into consideration the information or instructions provided by a user, and which produce an output based on the underlying system functionality. These systems require a significant amount of interaction with humans. Simple calculators with buttons, web browsers, mobile phones, editors, remote controls, medical infusion pumps are examples of systems that require human computer interaction and can therefore be considered as interactive systems. Interactive systems must help and allow the user to accomplish their desired goals. This is the main purpose of an interactive system [8]. Interactive systems are used to make a human's life more easy, safe and productive.

Interactive systems are usually composed of two parts: the *user interface* and the *functional* core. The user interface is made up of widgets which could be buttons, sensors, touch-screen or speech. Each widget has a certain set of behaviours associated with it which control and change the interactive and functional behaviour of the system. The functional core of the system deals with the operations, calculations and information it takes and returns to the user interface. A user interface must be simple and easy to understand so that users can complete their intended task efficiently and with minimal difficulty.

There are various interactive systems which, if they fail, can cause significant damage to property, the environment or even human life [9]. Such interactive systems are referred to as *safety-critical interactive systems*. There are

many systems that are considered safety-critical interactive systems where the interaction occurs via a user. These systems are considered as safety-critical because of the unbearable damage they can cause. For example, failure of the control system of a nuclear reactor could endanger the lives of millions of people and could impact the environment, failure of an avionics system could result in the death of hundreds of people and failure of a medical infusion pump could risk the life of a patient.

Due to advances in science and technology, there has been an increase in complexity of safety-critical interactive systems, and this complexity can be a source of errors. There are various reasons for the failure of safety-critical interactive systems, like faulty design and software errors [10]. There could be mistakes in the specification of the system, mistakes in the software or there could be mistakes in the hardware design. Poor user interfaces in such systems would leave the user feeling insecure, annoyed and frustrated and errors in the functionality of safety-critical systems can have severe consequences like deaths [11] [12] [13].

Another issue is that the nature of these systems is not thoroughly understood when paired with human use. It is important to understand that users are central to the system development process. In so many cases, whenever an error occurs, the blame is on the user. For example, if a nurse inputs the dose or delivery values for an infusion incorrectly he or she could endanger the patient connected to the pump. But it is not always the uses error that leads to accidents, most of the times an accident is a consequence of bad system design. So considering techniques to model and analyze user interfaces and interaction is also important in addition to the functionality.

Researchers have been working for many years to solve problems or issues in safety-critical interactive systems due to poor user interfaces and functionality, for example, in avionics [14] or in medical infusion pumps [15] [16] to ensure that safety-critical interactive systems are safe before entering the market and function correctly in all situations. The use of formal methods for modelling and analysis is often recommended as a way of raising confidence in

such systems. Therefore, it is necessary to model and analyse safety-critical interactive systems using formal methods for both interaction design and functional specifications to ensure that the device meets all requirements.

Formal methods is the field of software engineering that is aimed at developing techniques to plan, develop and analyze a system in order to improve reliability. Formal methods greatly increase our understanding of a system by revealing inconsistencies, ambiguities, and incompleteness that might otherwise go unnoticed [17]. Formal methods and languages are used to describe a system before actually building it to help ensure that no errors are introduced into the development phase. Formal methods are used in human computer interaction in many different ways, one example being modelling and analyzing hardware and software systems.

Modelling is a universal technique that can be used across many activities in system development. A model is an abstract representation of relevant characteristics of a system. Models are helpful in investigating the behaviour of the system and its properties.

However, safety-critical interactive systems are often modelled and verified in two separate parts; the user interface and interaction are modelled and verified separately to the functionality of the system. This becomes evident when we consider typical techniques such as those discussed in chapter two. This results in two different sets of models, one for user interface/interactivity and one for functionality. The model of the user interface and interaction can be verified for usability or task satisfaction. The model of the functional part can be verified for code and logical correctness.

Although the two parts of the safety-critical interactive systems can be modelled and verified separately, errors can still arise. There are certain properties of safety-critical interactive systems which can only be verified when these two parts, functional and user interface/interaction, are combined together. For example, medical infusion pumps must satisfy a property that "Changing settings, such as the patient's weight or infusion duration, while the pump is infusing, should either not be allowed, or at least require confir-

mation"[1]. We have to combine the two separate models, functional and user interface/interaction, to verify this property. This property is hard to verify by using the two individual models. The functional specification does not highlight what users are allowed to do while the pump is in the infusing state. In order to check if the user can actually change the settings while infusing, we have to combine the user interface/interaction model with the functional model.

The drawback of combining the models is that a lot of work is required to do the coupling of functional behaviour with interactive elements to ensure consistency [18][19]. This means that despite extensive research into modelling and analysis of safety-critical interactive systems, no strategy can guarantee that a complete system (user interface, interaction and functionality) is modelled and analyzed perfectly. Hence our investigation into modelling and analysis of safety-critical interactive systems and combination of models in order to overcome the difficulty of coupling of functional behaviour with interactive elements.

Models are often expressed using some modelling language which could be formal, informal or semi-formal [20]. In this thesis we will be using a formal language to express the model. Formal models are expressed in a language that has a well defined syntax, semantics and a logic. There is a wide variety of formal languages and notations. There are a number of languages that come under the frame of formal methods, such as Z, Spin, Promela, Uppaal, Petri Nets, CSP, CCS etc. The focus of this research work is on CPN [21].

Initially the main focus of formal methods was on modelling, specifying and analyzing the functional part of a system. User-interfaces and interaction were not considered as important as functional aspects because systems used to be very simple in terms of interactivity. But now as interfaces have become more complex, so their design and analysis is very obviously important.

There were some formal methods which were used to model and verify both user-interface and interaction, for example, Jacob in [22] has used techniques

[1] `https://rtg.cis.upenn.edu/gip/HazardAnalysis`

based on state transition diagrams and BNF to specify the user-interface of the Military Message System (MMS) project and Dix and Runciman in [23] focused on creating abstract models for user interfaces and interaction. However, formal methods for system development and formal methods for modelling user-interfaces and interaction were considered separate.

In this thesis we aim to apply formal methods for modelling and specifying the user interface, interaction and functional aspects of the system in a single model. The proposed technique has its starting point in several formal specification techniques: Z, Presentation Interaction Models (PIMs) and Presentation Models (PMs) (as for instance in [24]). From this (existing) basis we create a Coloured Petri Net (CPN) model of a system which will have the required aspects of Z, PMs and PIMs expressed within it. In summary, our plan here is to show how an existing, accepted way of formally modelling systems via PM/PIM/Z can be re-cast in the single formalism of CPN, and then in future, having shown the CPN models are as expressive as the PM/PIM/Z models, we can move straight from the system or requirements to a CPN model of it.

We will illustrate this work with a case study of a medical system, particularly an infusion pump. These computer-based safety-critical interactive devices are widely used in hospitals to dispense medicine to patients. There is a rapid change in medical systems because of the advancements in computer technology. These devices are now quite small in size and are highly portable. But even as the devices are becoming smaller, their complexity is increasing significantly. It is challenging for a systems engineer to develop error-free software-based safety-critical devices. There have been a large number of accidents and serious harm caused because of the inadequacy of these sorts of devices [25].

There is a lot of ongoing work in the field of safety-critical medical devices. The U.S. Food and Drug Administration (FDA) has a significant role regulating medical devices and is responsible for ensuring the safety and efficiency of medical devices. The FDA has an ongoing research project related to model-

based software development named the Generic Infusion Pump Project[2]. The goal of the project is to develop a set of safety models and reference specifications that can be used by manufacturers to verify safety properties of infusion pumps. One model has been made by the researchers using UPPAAL [26]. But still there is no such method that has been accepted by FDA for validation of the infusion pumps.

We also present a case study on the control system of a nuclear power plant to show that the technique we present is not restricted to only the medical domain.

1.1 Problem statement

Formal verification techniques are now widely applied for ensuring correctness and safety of safety-critical interactive systems. Moreover, most approaches built so far either focus on the user interface/interaction or on the functional part of the device. There is very little work done on modelling and analyzing a system which have all the three parts in one model. This study aims to apply formal methods for modelling, specifying and analyzing the user interface, interaction and functional aspects of a system in a single model.

1.2 Research Questions

The main research questions investigated in this thesis are:

1. How can we create formal models of safety-critical interactive systems that have all the parts (user interface, interaction and functional) in a single model, in a way which allows us to investigate the behaviour of a system?

2. How can we combine existing accepted formalisms to synthesize another formal method which builds a model as in 1?

3. What are the benefits of the new method?

[2]See `https://rtg.cis.upenn.edu/gip/`

1.3 Approach

The approach to address the research is divided into the following steps:

1. **A technique to express existing, accepted light-weight models for user interface and interaction in Coloured Petri Nets.** This approach addresses research questions one and two. We will present rules and demonstrate how to model a user interface and interactive behaviour using Coloured Petri Nets based on an existing method.

2. **A technique to express an existing formal specification technique for specifying functionality of safety-critical interactive systems in Coloured Petri Nets.** This approach addresses research questions one and two. We will demonstrate how we can model the functionality of a system using Coloured Petri Nets by expressing the required aspects of an existing formalism in Coloured Petri Nets.

3. **Investigation of a model developed by the above stated technique.** This approach addresses research questions one, two and three. We will investigate the behaviour of the user interface/interaction and functional part of the system that are combined into a single model.

1.4 Thesis Structure

This thesis is structured in eight chapters and is organized as follows:

In chapter two we discuss background material that relates to our work, with a particular focus on relevant literature which seeks to address similar research questions to our own. We will summarize existing approaches to explain how our work relates to these and differs from these.

This will be followed by chapter three which will present an overview of formal methods and tools used in this research work. This will include an introduction to four formalisms: *presentation model, presentation interaction model, Z and Coloured Petri Nets.*

In chapter four we will demonstrate how to express *presentation models* in Coloured Petri Nets. Then we will model *presentation models (PM) and presentation interaction model (PIM)* in Coloured Petri Nets. This chapter will also present the formal definition of designing CPNs for modelling PIMs. Then we will investigate a model to check if the user interface and interaction of a safety-critical interactive system is behaving as expected.

In chapter five we will demonstrate how to express Z specifications (as used to specify functional aspects of the safety-critical interactive systems) in Coloured Petri Nets. Then we will explain the process of extending the CPN model of PM/PIM by adding functionality (Z) to it. This chapter will also present the formal definition of designing CPNs for modelling PM/PIM/Z models. Then we will give methods for investigating the combined model.

In chapter six we will present a case study for a medical device, the Niki T34 Infusion Pump. In this chapter we will create and analyze a CPN model of the Niki T34 infusion pump using techniques presented in chapter four and five.

In chapter seven we present another case study on the reactor control system of a nuclear power plant to demonstrate how we can move straight to Coloured Petri Nets alone.

In chapter eight we finish with concluding remarks and a discussion for future work.

1.5 Summary

In this chapter we introduced safety-critical interactive systems and their two main components: user interface/interaction and functional. We discussed how interactive systems are modelled and verified in two separate parts, user interface/interaction and functional. We further discussed the importance of formal methods in modelling and analysis of safety-critical interactive systems. Lastly, we presented our research questions and approaches to answering these questions.

Chapter 2

Literature Review

2.1 Introduction

In this chapter we introduce the relevant literature related to this research work. We start with a discussion on modelling and verification. Then we briefly introduce the formal methods used in this research work. This is followed by an introduction to the research area. Next, we discuss how formal methods have been used for modelling and analyzing safety-critical systems. We discuss the importance of formal methods in the field of human computer interaction. We also discuss existing techniques which are relevant and form the basis of this research.

2.2 Modelling and Analysis

Modelling is a well-accepted engineering technique. A model is a representation of selected aspects of a system and hence is an abstraction: parts of the system are hidden in order to control complexity, or to highlight parts of a system. Models are helpful for variety of purposes. They help in better understanding of a system and also help users to visualize the working of a system under development. They also help in understanding and identification of relationships between various aspects of a system, for example, the relationship between the user interfaces and interaction with the underlying system

functionality.

Models provide significant support for the development of safety-critical interactive systems. Models of safety-critical interactive systems describe the behaviour of a system in regards to its functionality and user interfaces and interactions between a user and a system. Models are often expressed using some modelling language which could be formal, informal or semi-formal [20]. Formal models are described using a formal language that has a well-defined syntax and semantics. Once we have a formal model of a system ready, various techniques are then applied to these models to assess the quality of safety-critical interactive systems. We can then investigate and analyze the behaviour of the model as per the requirements written as properties. If there are any flaws and the model is not behaving as expected, then the flaws are removed and the model is updated and analyzed again. Model checking is one of several ways to analyze a model.

Model checking allows us to investigate whether a set of requirements which are specified as properties are satisfied or not. A property is a general statement describing an expected behaviour of the system. For model checking, firstly, a system under analysis is formally modelled using some formal language or notation. Then, ideally, the entire state space is analyzed to determine whether the set of requirements/ properties are true or not. One of the main drawback of state space method is the state explosion problem. In this work we reduce the maximum value of some variables to keep the state space small for analysis.

Model checking has been applied for verification of safety-critical interactive systems in domains such as healthcare [15] and avionics [27].

In this research we create a formal model of a safety-critical interactive systems specifying the underlying functionality and user interfaces and interaction using a formal language and then apply a model checking technique to analyze a system.

The focus of this research is on safety-critical interactive systems, so we take into consideration both *safety requirements* and *usability requirements*.

Safety requirements help in ensuring that the system follows the guidelines and usability requirements help in preventing users from making mistakes. There exist approaches that analyze usability properties or functional properties or both. These are discussed in detail later in this chapter.

The following section gives an overview of the formal methods and tools used in this thesis.

2.3 Formal Methods and Tools

In this research we use Coloured Petri Nets [28] to model and analyze safety-critical interactive systems, and we use these as the basis for our modelling technique which we will discuss in detail later in chapters three, four and five. The focus of this research is to create a model of the implemented system and investigate the behaviour of a modelled system using Coloured Petri Nets. This research is not about using the model for the subsequent transformation to an implemented system.

Coloured Petri Nets (CPN) is a language for the modelling and validation of complex systems [29]. Coloured Petri Nets is a discrete-event modelling language combining Petri nets with the functional programming language Standard ML. Petri nets [30] [31] provide the foundation of the graphical notation and the basic primitives for modelling concurrency, communication, and synchronization. Standard ML provides the primitives for the definition of data types, describing data manipulation, and for creating compact models. A CPN model of a system is an executable model representing the states of the system and the events (transitions) that can cause the system to change state. High level Petri Nets have been used for modelling and analysis of safety-critical systems which can be seen in [32, 33, 34, 35].

We use the CPN Tool [36] for editing, simulating and analysis of Coloured Petri Net models [37]. It is a well-established tool with a good on-going support. It also provides many analysis tools such as automatic state-space generation, state-space analysis, symbolic execution, which allows us to perform

validation and check safety properties etc.

We use Coloured Petri Nets to combine the user-interface and interaction and functional aspects in a single model by taking advantage of existing techniques (PM/PIM/Z). Formal methods based on a single formal foundation often have their own strengths and weaknesses and are thus suitable to specify some aspects (control, data, structure, behaviour etc.) and types (sequential, concurrent, distributed, real-time etc.) of software systems [38]. It is not only theoretically interesting but also practically useful to investigate the relationships between the different formal methods and to integrate them to achieve their combined strength. Formal method integration is one of the major future research directions of the formal methods community [17] [7].

Many researchers have attempted to combine Z with other concurrent models including CSP [39], promela [40], temporal logic [41], User Centered Design Techniques(PM & PIM) [42]. Chubb in [43] has presented an approach using Z and state transition diagrams to model windowing and event handling.

There also exist a few studies which have combined Z and Petri Nets [38]. The strength of the two languages can be preserved, while alleviating their individual drawbacks [44]. Z is helpful in specifying the state space and sets of operations but we cannot express the sequence of occurrence of the operations as well as their combinations. It is important to talk about the dynamic behaviour of a interactive system, particularly in safety-critical domains.

Bolt's case study [45] shows that Petri Nets are a well established language for describing the behavioural aspects of a system. The case study describes modelling and analysis of multimodal interactive systems. The strengths of Petri Nets are also shown in [46] where Weyers has shown the verification and validation of interactive system using high level Petri nets. ICOs [35] and the APEX framework [47] are some more existing formalisms which are discussed in detail later in this chapter which follow an object-oriented approach for the functional aspect of the system and use Petri Nets for showing dynamic behaviour of the system. We will discuss the relevant approaches in detail and explain how our work is different from these approaches later in this chapter.

The following sections shed light on this research area of this work.

2.4 Research Area

There are two strands of our research: formal methods and human computer interaction. We will investigate how both of these strands are used together, with a specific focus on modelling and analyzing safety-critical interactive systems. Our motive is to contribute to both formal methods and human computer interaction by presenting a technique to model and analyze safety-critical interactive systems. By this we expose the power of formal methods to HCI and conversely, it exposes to formal methods, the applications and potential in HCI.

Formal methods are mathematically-based languages, techniques, and tools for specifying, validating and verifying hardware and software systems. Formal methods help in increasing our knowledge of a system by revealing inconsistencies, ambiguities, and incompleteness that might otherwise go unnoticed [17]. These languages and notations are used to model and verify a system to help ensure that no errors are introduced into the development phase. There are many languages and notations which come under the umbrella of formal methods; Petri nets [48], UPPAAL Model Checker [49], Z [50] and B [51] are some of the examples. Similar to software development techniques, formal methods continue to evolve. Whichever of the existing formal methods we choose, our motive is to correctly model and verify a system before the actual implementation in order to ensure that the system adheres to the set of requirements. The formal methods used in this research work are Coloured Petri Nets [21], presentation models [24], presentation interaction models [24] and Z [50]. These formal methods are discussed in detail in chapter three.

In the early years when user interfaces were very simple, formal methods were used to model and verify the functionality of a system. How users interact with a system was not given much prominence because interactions were quite simple. But now, as interfaces have become more complex, so their design

and analysis is very obviously important. Problems could be largely solved if designers had methods and tools to provide indications about the most effective interaction and presentation techniques to support the possible user activities [52]. This is the reason that human computer interaction has become an important area of research.

Human Computer Interaction (HCI) is a multidisciplinary field that studies humans and their cognitive abilities as well as designing computing systems and user interfaces and interaction at different levels of abstractions. Formal methods are used in different ways covering many aspects of HCI. Formal methods in HCI is of great significance, especially for systems that are used in safety-critical domains like power generation, health care etc. Formal methods have great benefits in the design process because of the increasing level of complexity in the user interfaces and interactions in the safety-critical interactive systems. HCI and formal methods aim to provide reasoning about a system before its implementation to ensure that it satisfies a set of requirements [53]. An interactive system or/and its user interface is specified using formal methods and then usability evaluation techniques are applied for assessing the usability of a system, for example, as mentioned by Thimbleby in [54].

Researchers have been working on using formal methods for modelling user interfaces and interactions for more than two decades [55]. However, systems are often modelled and investigated in two separate parts; functionality is modelled and analyzed separately to the the user interface and interaction of the system. As the focus of this research is on safety-critical systems, so we will see how formal methods are used for modelling and analysis of safety-critical systems.

2.5 Formal methods for modelling and analysis of safety-critical systems

Safety-critical systems are likely to be more robust if the developers evaluate and examine the software using formal methods early in the development

phase. Various techniques have been developed by the software engineering and human computer interaction communities for modelling software and its interactions. We can examine the safety properties of the model and analyze the model and then the developer will derive the software which implements that model. The resulting software will be far more robust compared to that developed using traditional methods like code walkthroughs, testing and manual inspection. The Laboratory of Software Engineering at the Food and Drug Administration (FDA) has been working with academic collaborators under the National Science Foundation/FDA Scholar-in-Residence programme to develop and refine model-based engineering methods and associated verification techniques [16].

Formal methods have been used in the safety-critical domain for a long time [56]. There have been several studies involving the use of formal method-based analysis of safety-critical systems [17]. The use of formal methods in the production of systems should be viewed as a means of delivering enhanced quality rather than establishing correctness [57].

Formal methods have also been used in the field of avionics. Palanque and others did a case study on air traffic control [58]. They used Petri Nets to describe the tasks that have to be managed by the controller and operational formalism that allows one to express the data flows and synchronizations. The results shown in the paper prove that formal methods can provide fruitful insights at various stages of the development life cycle of safety-critical interactive applications. Because of the major changes in the air traffic management with time, researchers are engaged in applying different formal methods and techniques for the verification of air traffic control which is evident in [59].

Lahtinen and others discussed the role of formal methods in software development in the area of nuclear engineering [60]. They put forward model checking, a computer-aided formal method for verifying the correctness of a system design model, as a promising approach to system verification. They developed a systematic methodology for modelling safety-critical systems in the nuclear domain. Two case studies are reviewed and they were able to find

errors using formal methods.

Formal methods have been used in the railway domain for more than twenty five years [61]. Researchers have used different formal methods and techniques to model and investigate various areas of railway systems. Lecomte et al. in [62] presented work in which they used B to specify the functionality of platform screen door controllers used by RATP (the organization that operates bus and metro public transport in Paris). The models they built were proved, guaranteeing a correct behaviour of the system and the process that was used during system specification and design were also qualified by French authorities. Gaied et al. used stochastic P-time Petri nets for modelling the railway transport networks in Sahel, Tunisia [63]. That model was then used to analyze the network traffic and evaluate the performance of the system. Although a lot of research has been carried out, as the technology is growing, so there is a need to come up with new methods to analyze these systems.

Healthcare is another safety-critical domain where the use of formal methods is playing a significant role. Petri nets can play the role of a generic framework for architectural decisions for control systems, allowing verification/simulation, an important bridge in the requested traceability by regulatory bodies [64]. Figueiredo represented the main advantages of Petri nets that it is able to be used in the construction of reference models for the medical devices domain with controller characteristics and conducted a case study on specification of a medical device.

Arney [16] showed that formal methods have long been suggested as a means to design and develop medical device software. However, most manufacturers avoid using these formal tools and techniques as to them these techniques are very complex and time consuming. Because of this, errors are discovered when a device is developed and is in the market. The reference model for a generic patient controlled analgesic infusion pump was developed by Arney to illustrate the approach to software conformance checking.

Majma and Babamir [65] used Fuzzy Petri Nets for the specification and verification of the behaviour of a medical monitoring system called INS. They

proposed a visual and a mathematical model through which one can investigate the device behaviours against a user's requirements by determining the device's unsafe and hazard statuses. Harrison et al. [66] showed the use of formal techniques to contribute to the risk analysis of a new neonatal dialysis machine.

In our research work we are using a case study of a medical infusion pump which is a contribution to the healthcare domain.

So far we have discussed how formal methods have been applied to modelling and analysis of safety-critical systems in various domains, like, healthcare, avionics, railways and nuclear domain. Another equally important aspect is to analyze how humans interact with these safety-critical systems. We will see in the coming section the use of formal methods in the field of human computer interaction.

2.5.1 Formal methods in HCI

Formal methods are used in human computer interaction in many different ways, one example being modelling and verifying hardware and software systems. As hardware and software systems are growing in functionality, there is a certain increase in complexity and it is more likely there is greater risk of error in the system. These errors may cause significant damage to the environment, property or even human life. Safety for us is a property of a system that says it will not endanger human life or the environment. So, safety should be a central consideration in the development of safety-critical interactive systems. There are many devices that are considered as safety-critical interactive systems where the interaction occurs via a user or, perhaps, via an automatic manufacturing system where sensors are interacting among themselves. A primary goal of a system engineer is to develop reliable systems and the use of formal methods can help in achieving this goal.

Formal methods can be used to specify several different aspects of safety-critical interactive systems. There exist various techniques that use formal methods for modelling interactive systems as objects, interactors, components or agents, for example, [67, 68, 69]. There also exist several approaches that

do not use composition of smaller parts and model interactive systems as a whole, for example, [70, 71, 72]. We will discuss these approaches in detail later in this chapter.

Formal methods have been used for HCI for many years now. In 1990s when research in this field started, existing formal methods and tools were used in HCI for the design process. This is evident by looking at Jacky's work in [73] where Z [50] has been used for modelling the interface of a radiation machine. As stated earlier, techniques were developed that use existing formal methods for modelling and verifying interactive elements of a system. For example, Paterno et al. in [74] used LOTOS [75] for modelling and evaluating the usability of user interfaces. Doherty and Harrison in [76] used VDM-based notation [77] for the description of interactors and formal reasoning about the functionality of a user interface.

There also exist several other techniques that used existing formal methods for describing user interfaces and interactions of interactive systems. Based on the existing methods, several new formalisms were then developed which have given a new direction to this research field.

ICO [35] is an example of a formalism that was developed based on the existing formalism of Petri Nets. Barboni et al. in [78] used ICO on a cockpit display system for describing the interactive widgets, user applications and user interface servers. ICOs are discussed in detail in the next section.

Thimbleby in [79] has presented work to detect errors in safety-critical interactive systems related to number entry. The formal notation used in Thimbleby's work is based on, and is equivalent, to Hoare triples [80]. The main reason to choose a lightweight notation is that it takes little effort to learn and can be used immediately.

Silva et al. [81] reverse engineer an abstract model of a graphical user interface and ensure it satisfies the set of requirements. They have also described the IVY project[1] which is used for analysis of interactive system design. Campos and Harrison in [82] have used the IVY tool with Modal Action Logic

[1] `http://www.di.uminho.pt/research/ivy?set_language=en&cl=en`

(MAL) for modelling and analysis of interactive systems. The case study on the BBraun Infusomat pump has been presented which raised issues around the relation of the user interface to the underlying system behaviour. Researchers have also worked on creating models that can be used in the design process of interactive systems in a more general manner to ensure the requirements are met, for example, [83, 84]. In our research we are taking the models described in [83] and re-expressing them using Coloured Petri Nets. This is discussed in detail in chapter 4.

These general modelling approaches have to tackle the problem of separation of concerns between interface and functional elements, while at the same time managing the relationship between the two [7]. One of the focuses of our research is to address this issue and bridge the gap between the user interface and underlying system functionality.

Researchers have been using these formal methods and tools for modelling and verifying interactive systems. We will discuss some of these existing techniques in detail.

2.6 Existing Techniques

There has been a lot of work done on modelling and analysis of safety-critical interactive systems, but still accidents do happen. So there is a need to come up with new techniques to ensure the safe use of such systems. We will discuss some techniques from the last two sections. This is not intended as an exhaustive list of techniques, as we only discuss those relevant to this work.

Early approaches, as mentioned earlier, used existing formal languages and notations to model and verify interactive systems which can be seen in work done by Abowd in [67]. A framework for the formal description of users, systems and user interfaces is presented in [67] in which the modelling of interactive systems is done as a collection of agents which is specified using Z [50]. Another technique was presented by Abowd et. al. in [85] which described interactive systems via a tabular interface with the help of Action Simulator (a

tool that allows the representation of propositional production system (PPS) [86] in tabular format). The tabular specification then can be translated into SMV [87] and properties are then formalized using temporal logic. The work in [85] shows verification of various usability properties (reversibility, deadlock freedom and undo) and functional properties (state inevitability, strong task completeness, event constraint, state avoid-ability, rule set connectedness). Details of this work can be found in [85, 87]. The tabular format represents information about the states and actions that leads to state changes. But this technique is not suitable for large and complex systems and this technique has not been applied to any safety-critical systems. Moreover, there is no obvious way to differentiate between the user interface and interactions and the underlying system functionality.

Another example is the PIE model which was initially described by Dix et al. [88]. The main aim of the PIE model (Figure 2.1) is to describe the user interfaces in terms of the possible inputs (P) and their effects (E) and an interpretation function between these two (I) [3]. This permits certain usability properties such as predictability (predict the effect of commands), undo-ability (user can reverse the effect of any command), total reachability (user can get anywhere from anywhere) etc to be formalized and analyzed. The PIE model creates an abstract model of user interfaces and omits other details such as internal system functionality, hardware characteristics etc. Although the PIE model allows formal modelling and analysis of user interfaces, it does not provide ways to link the model to other parts of a system such as underlying system functionality.

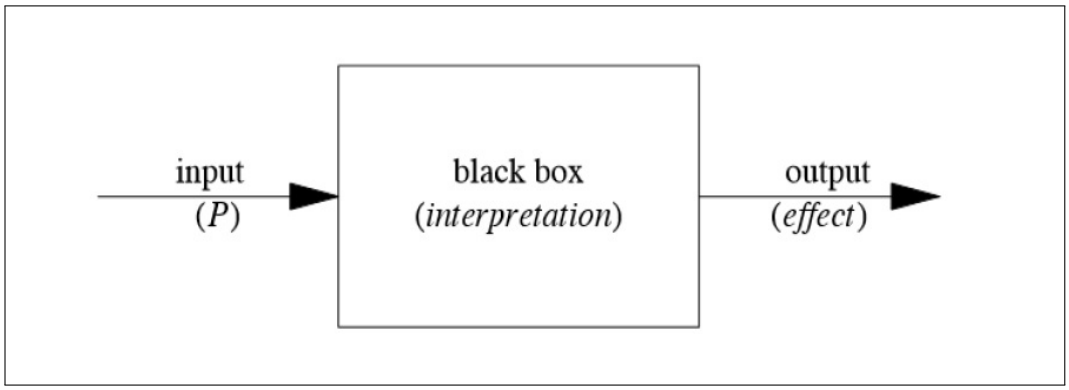

Figure 2.1: The PIE Model [3]

Interactors is one of the most influential concepts. Interactors were originally developed at University of York by Duke and Harrison [89] and since then they have formed the basis for much research and are still used in many different forms. The York interactor [4] as shown in Figure 2.2 has an internal state that represents functionality which is reflected through a rendering relation (rho) onto a perceivable state (P). The interface between an interactor and its environment consists of a set of events.

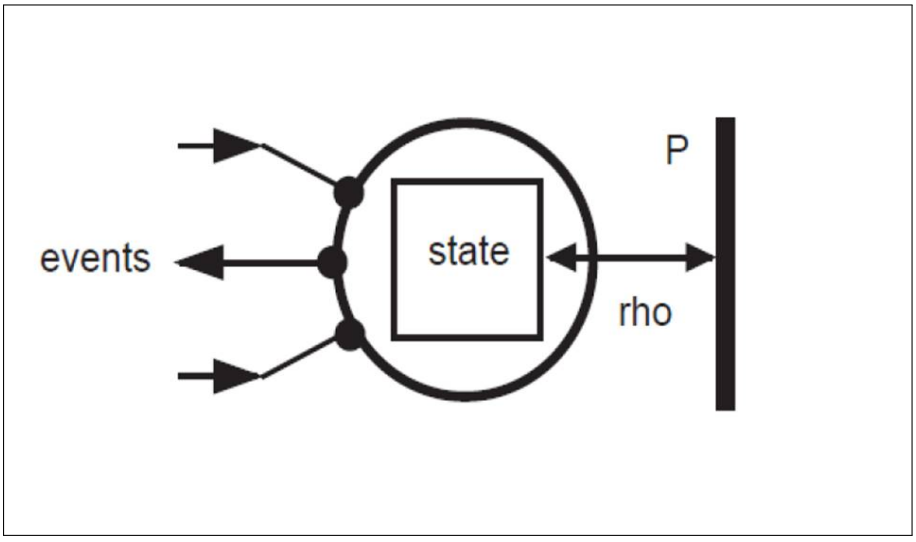

Figure 2.2: The York Interactor [4]

The York interactors are described using Z. They have also explored the use of other formalisms such as VDM and VVSL [90]. Interactors allows underlying system functionality, interaction and presentation to be captured in a single

component. The York interactor model also allows formal verification via theorem proving. Various usability properties can also be verified, such as total reachability. This approach has been applied to case studies in the safety-critical domain such as an aircraft's fuel system [91]. Although interactors have been influential, concurrent behaviour cannot be expressed [89].

The York interator forms the basis of many other research works. One such example is the work proposed by Campos et al. in [92] where they used Modal Action Logic (MAL) for implementation of the York interactor and used model checking for the verification. Campos and Harrison presented the MAL interactor language to describe interactors based on MAL [93]. A tool called i2smv is also proposed that helps in translating the MAL specifications into the SMV input language of the SMV model checker. To contribute to usability engineering, the IVY toolbox as shown in Figure 2.3 was developed by Campos and Harrison [5].

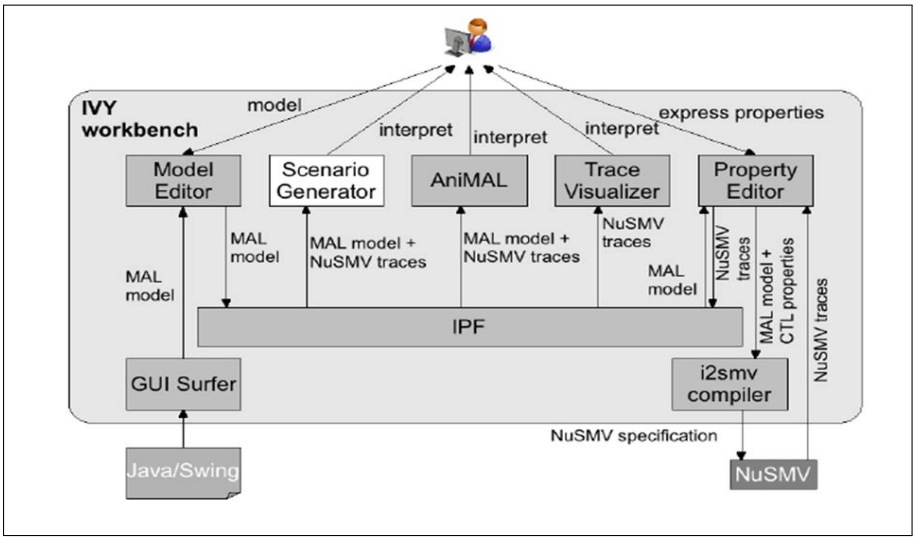

Figure 2.3: The IVY Tool Architecture [5]

The main components of the IVY tool (Figure 2.3) are: a *model editor* that helps in creating a model in the MAL interactor language, a *reverse engineering component* that takes the user interfaces written in Java/Swing and generates specifications in the MAL interactor language, a *model animator* that takes a model and creates a prototype, a *property editor* that helps a designer to select

a patterns of properties and generate the logical formulae, a *compiler i2smv* translates the MAL specifications into the SMV input language. In the IVY tool, the properties are specified using Computation Tree Logic [94] allowing verification of usability (feedback, behavioural consistency, reversibility, completeness) and functional properties as explained in [95]. This approach is used to model the core functionality and user interfaces of interactive systems. It also covers some assumptions about user behaviours [96].

The approach presented by Campos and Harrison has been applied to various case studies in the safety-critical domain. The approach has been used to model and verify infusion pumps [82]. Another area where this approach has been applied is aerospace [97].

Campos et al. also presented the approach for modelling and verification of interactive systems using Prototype Verification System (PVS) [98] and PVD theorem prover. Harrison et al. [99] explained the use of a theorem prover for the verification of properties of interactive systems which is an alternative to model checking. To do so, MAL specifications were translated into PVS. In [100] the authors presented templates for model-based analysis of usability and safety aspects of user interface software design in which they have used PVS for modelling and analysis and presented a case study on an infusion pump. Harrison et al. are also working closely with the US Food and Drug Administration (FDA) and applied the same approach to ensure that user related FDA requirements are true of an infusion pump [101].

Palanque et al. [102] presented a technique for modelling and verifying interactive system using Petri Nets. Petri Nets have graphical representations which makes it easier to understand as compared to textual descriptions. Initially, it was referred to as Petri Nets with Objects (an extension of Petri Nets). The work has been extended over the years and has been given the name Interactive Cooperative Object (ICO) formalism [35]. ICOs allows modeling and testing of interactive systems which can also be fully implemented by integrating Java code in the models. ICOs uses concepts borrowed from object-oriented approach such as polymorphism, encapsulation and inheritance

to describe the attributes and the functions/operations.The major difference between our research and ICOs is around levels of abstraction, because they take an object-oriented view whereas we are committed to more abstract models.

Model checking techniques can be applied to the models created using the ICO formalism for the verification of usability and functional properties. Modelling and verification can be done using Petshop [103]. Palanque et al. in [104] explained how this work permits the integration of task models and system models. The task models were first described using Petri Nets. More recently the HAMSTERS [105] notation is used to model user tasks. HAMSTERS task models focus on the set of steps a user will take to complete a certain task. With HAMSTERS, they start with task models, whereas in our research we allow a model to start with functional aspect, user interface or interaction. Various usability properties, such as predictability, deadlock freedom, availability can be verified. ICOs has been applied to various case studies, particularly in avionics which can be seen in [106, 58].

With ICOs, analysis is performed on the underlying Petri Net which is a simplified version of the original Petri Net. A drawback is that properties verified on the underlying Petri Nets are not necessarily true in the original Petri Net [35]. Thus, they used the result of classical property analysis as an indicator that highlights potential problems in the Petri Net. Verifying the properties directly on the ICOs model is still a research challenge for the Petri nets community. In our research we are using Coloured Petri Nets as the main formal method and not any simplified version of it which will overcome the problem as mentioned in the existing work.

Bowen and Reeves in [107, 24] have presented an approach in which user interfaces are formally expressed using presentation models and presentation interaction models. A presentation model (PM) [24] describes the existence, category and behaviour of the widgets (interactive elements) in various states of a user interface. A presentation and interaction model (PIM) describes the transitions between states [24]. A PIM gives us a view of the dynamic changes

of the user-interface. A PIM is the combination of a presentation model and a finite state machine (FSM). The main objective of the approach is the use of lightweight models of user interfaces and interactions in combination with a formal model of underlying system functionality [107]. The underlying system functionality is specified using Z [50]. The focus of the work is to use the models during the design process but it is also possible to reverse engineer existing systems and create models of existing systems [70].

This approach has been applied to several case studies. Presentation models and presentation interaction models are used in the design process of user interfaces for the PIMed tool [42]. However, the models are verified manually in this work which is prone to errors and consumes a lot of time. In [83], PMs and PIMs have been used for another case study related to a medical infusion pump. In this work verification is done with the help of the ProZ tool to check usability properties expressed in linear temporal logic (LTL), like total reachability (a user can go to any state from any other state) and deadlock detection (to verify that a user can't enter a state which leads to no action). Another case study to which these formal methods have been applied is the nuclear power plant control system [7]. This approach (PM/PIM/Z) has been applied on various case studies which results in three separate models.

In this research we take the existing approach (PM/PIM/Z) of Bowen and Reeves and re-express it in a new way using Coloured Petri Nets. In the coming chapters we will show that each formalism in the original existing method (PM/PIM/Z) can be expressed in Coloured Petri Nets, thus showing that there is no loss of expressiveness in moving from the established PM/PIM/Z method to Coloured Petri Nets. Having done that we would then in future use Coloured Petri Nets alone for our modelling, knowing that we have retained all the expressiveness that we need.

The motive for doing this move is two-fold: The existing method results in three separate models, and the drawback with this is that a lot of work is required to do the coupling of functional behaviour with interactive elements to ensure consistency [18][19]. Moreover, these models need to be combined in

order to verify safety properties about a safety-critical interactive system which might relate to functional constraints, interface constraints or both. The new technique results in a single model capturing both functional and interaction aspects of safety-critical interactive systems and all the connections between them. So, the benefit of having all aspects in a single model prevents us from extra work required to combine them later for analysis.

2.7 Summary

In this chapter we discussed uses of modelling and verification in general. We have also discussed formal methods and human computer interaction and described how our work relates to this field. We have also explained how formal methods have been used in the human computer interaction field. This is followed by discussion of different types of existing techniques for modelling and verifying safety-critical interactive systems, i.e., relevant research work required to understand our technique.

We have discussed how presentation models, presentation interaction models and Z are used to model user interfaces, interactions and functionality of interactive systems. In our research work we use this existing technique and re-express it in Coloured Petri Nets. To do this, we need to understand its syntax and semantics so that there is no loss of expressiveness in moving from the established PM/PIM/Z method to Coloured Petri Nets. All these formalisms are explained in detail in the next chapter.

Chapter 3

Formal Methods and Tools

3.1 Introduction

This chapter provides an introduction to four formalisms: *presentation models, presentation interaction models, Z and Coloured Petri Nets.*

3.2 Presentation Model

The *Presentation Model* [42] describes the behaviour of widgets (interactive elements) of a user interface. A presentation model can be developed for interfaces at various levels, for example, prototypes, design artefacts, implemented systems etc. The presentation model represents the behaviour of the user-interface and how a user accesses that behaviour. It is important to note that widgets are not just buttons or drop-down menu or check boxes. Widgets can be anything through which interaction can take place, for example, speech, touch screen etc. Widgets are categorized using the widget categorization hierarchy given in [108]. Widgets can fall in one of the three categories depending upon their behaviour: *Event Generator* (causes operation to occur when a user interacts with them), *Event Responder* (provide information back to the user) or *Container* (group of other widgets, like drop-down menu). A presentation model typically consists of several component presentation models which could be understood as the states of the user interface.

Presentation models consist of two parts: declaration and definition [24].

$$\langle PresentationModel\rangle ::= \langle declaration\rangle \; \langle definition\rangle$$

The declarations introduce the three sets of identifiers which can be used within the definitions.

$$\begin{aligned}\langle declaration\rangle \; ::= \; & WidgetName\{\langle ident\rangle\}^{+} \\ & Category\{\langle ident\rangle\}^{+} \\ & Behaviour\{\langle ident\rangle\}^{*}\end{aligned}$$

WidgetName is a list of names of widgets. Category refers to the description of widget categories. Behaviour shows what behaviours a widget has associated with it.
A definition consists of one or more identifiers for presentation models and expressions which give the values for those identifiers.

$$\begin{aligned}\langle definition\rangle &::= \{\langle pname\rangle is \langle pexpr\rangle\}^{+} \\ \langle pname\rangle &::= \langle ident\rangle \\ \langle pexpr\rangle &::= \{\langle widgetdescr\rangle\}^{+} \mid \langle pname\rangle : \langle pexpr\rangle \mid \langle pname\rangle\end{aligned}$$

Each expression is either a widget description, a presentation model identifier concatenated with another expression or a presentation model identifier. Each state of the system is described in a separate component presentation model by the means of widget descriptions. The collection of component presentation models then forms the full interface presentation model.
A widget description consists of a triple: the widget name, the category and the set of behaviours associated with the widget. The syntax of a widget

description is as follows:

$$\langle widgetdescr \rangle ::= (\langle widgetname \rangle, \langle category \rangle, (\{\langle behaviour \rangle\}*))$$
$$\langle widgetname \rangle ::= \langle identifier \rangle$$
$$\langle category \rangle ::= \langle identifier \rangle$$
$$\langle behaviour \rangle ::= \langle identifier \rangle$$

The names are assigned to widgets according to what is visually represented on the system or device. For example, there could exist a label "ON/OFF" on the button which clearly gives indication of its underlying behaviour then the name given to that widget in the presentation model is "OnOffButton" or something similar to that. There must be an easily identifiable mapping between the widget descriptions of the presentation model and the widgets of the prototype making it easier for the system developers to relate the model to the prototype.

Behaviours are divided into two categories. The first is called a system behaviour (S-behaviour) which refers to the underlying non-interactive system functionality where the user actions directly affect the state of the system. These behaviours allow users to interact with the system by changing or retrieving information. The second category is called an interactive behaviour (I-behaviour) that represents user interface functionality, which changes things about the user interface itself, like changing screens.

3.3 Simple Infusion Pump Example

We consider a simplified infusion pump to help explain the process. The prototype of a simple infusion pump which is used as an example is as shown in Figure 3.1. There are a total of seven widgets in this device: a display, on-off button, info button, up and down button, start and stop button.

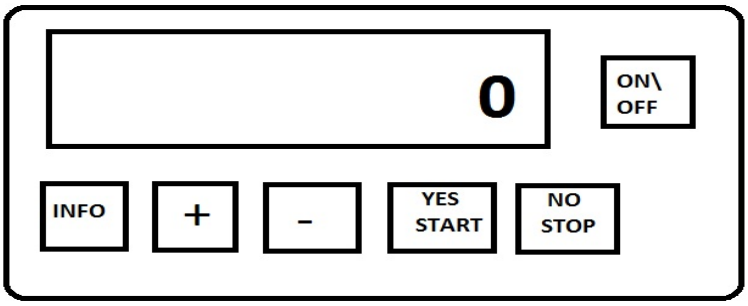

Figure 3.1: Prototype of Simple Infusion Pump

The simple infusion pump prototype works as follows:

1. Switch ON the pump
2. Press INFO button and select Battery Life from the menu and press YES button to confirm the availability of sufficient battery charge and NO if the battery charge is insufficient to complete the entire task
3. Set the time of infusion and volume to be infused using the up and down keys (+/- keys)
4. Press Start to start infusing and information will be displayed on the display

There are certain requirements for the infusion pump that are listed below:

1. Volume to be infused shall cover the range from 1 to 3 ml.
2. Pump can not operate for more than 3 hours.
3. A user can never get into a state where no action can be taken.
4. A user can never get into a cycle without the possibility of making effective progress.
5. A user is not allowed to make any changes when the pump is infusing.
6. The infusion rate for the pump is automatically calculated.

The device has a particular set of operations like *set duration, set volume and start infusing*.

Table 3.1 shows the presentation model of the simple infusion pump prototype.

WidgetName	Display NoButton YesButton InfoButton
	MinusButton PlusButton OnOffButton
Category	ActCtrl Responder
Behaviour	S_displaystartmessage S_displaybatterylife
	S_decreasevolume S_increasevolume
	S_setrate S_decreasetime S_increasetime
	S_infusing S_settime I_confirmrate
	I_settime I_init I_info
	I_infuse I_setvolume S_setvolume
	S_displayinfusingmsg
init is	(Display, Responder, (S_displaystartmessage))
	(NoButton, ActCtrl, ())
	(YesButton, ActCtrl, ())
	(InfoButton, ActCtrl, (I_info))
	(MinusButton, ActCtrl, ())
	(PlusButton, ActCtrl, ())
	(OnOffButton, ActCtrl, ())
info is	(Display, Responder, (S_displaybatterylife))
	(NoButton, ActCtrl, (I_init))
	(YesButton, ActCtrl, (I_setvolume))
	(InfoButton, ActCtrl, ())
	(MinusButton, ActCtrl, ())
	(PlusButton, ActCtrl, ())
	(OnOffButton, ActCtrl, ())
setvolume is	(Display, Responder, (S_decreasevolume,
	S_increasevolume))
	(NoButton, ActCtrl, (I_info))

	(YesButton, ActCtrl, (I_settime, S_setvolume))
	(InfoButton, ActCtrl, ())
	(MinusButton, ActCtrl, (S_decreasevolume))
	(PlusButton, ActCtrl, (S_increasevolume))
	(OnOffButton, ActCtrl, ())
settime is	(Display, Responder, (S_decreasetime, S_increasetime))
	(NoButton, ActCtrl, (I_setvolume))
	(YesButton, ActCtrl, (S_settime, I_confirmrate))
	(InfoButton, ActCtrl, ())
	(MinusButton, ActCtrl, (S_decreasetime))
	(PlusButton, ActCtrl, (S_increasetime))
	(OnOffButton, ActCtrl, ())
confirmrate is	(Display, Responder, (S_setrate))
	(NoButton, ActCtrl, (I_settime))
	(YesButton, ActCtrl, (S_setrate_I_infuse))
	(InfoButton, ActCtrl, ())
	(MinusButton, ActCtrl, ())
	(PlusButton, ActCtrl, ())
	(OnOffButton, ActCtrl, ())
infuse is	(Display, Responder, (S_displayinfusingmsg))
	(NoButton, ActCtrl, ())
	(YesButton, ActCtrl, ())
	(InfoButton, ActCtrl, ())
	(MinusButton, ActCtrl, ())
	(PlusButton, ActCtrl, ())
	(OnOffButton, ActCtrl, ())

Table 3.1: Presentation Model of Simple Infusion Pump Prototype

The presentation model describes widgets using a triple consisting of a widget name, its category and behaviours. The model has seven widgets and

each widget falls under one of the two categories: *ActCtrl or Responder*. The simple infusion pump's presentation model has eleven S-behaviours, which represent functionality and six I-behaviours which affect the user interface itself. In table 3.1, *init, info, setvolume, settime, confirmrate and infuse* are six component presentation models. Each component presentation model consists of a set of widget triples. For example, the *init* component presentation model comprises of seven sets of the widget triple. The first set of triples means that the widget *Display* is of category *Responder* has the *S_displaystartmessage* behaviour associated with it.

There are several benefits of presentation models for user-interface design. We can ensure that terminology used throughout the user-interface design is consistent. For example there are multiple ways to perform a particular action using different widgets. So there will be repetition of a particular behaviour within the widget descriptions in the model. We can identify common behaviours within the presentation model and examine the widgets which have this behaviour associated with them and use this to consider issues of consistency. Another important user-interface property is reactivity which can be easily determined by analyzing the widgets whose category is in the *Event Generator* hierarchy as they allow the user to interact and generate behaviour. The presentation model provides a different view of a system that makes things clear which are not necessarily obvious in the prototype alone.

The limitation of a presentation model is that it does not provide information about the dynamic behaviour of the user-interface. Although I-behaviours provide some information about a user interaction, that is not enough to provide certainty about which behaviour is available to a user. It does not show how various states of the user-interface are connected and how a user can navigate to different states and access the behaviours. This ability to navigate to various states of user-interface is missing from the presentation model.

In order to provide the information about dynamic behaviour of a user-interface, the presentation and interaction model (PIM) is used. The presentation and interaction model uses the presentation model as its basis and is

explained in the next section.

3.4 Presentation and Interaction Models

The presentation and interaction model (PIM) models the transitions between the states. The PIM gives us a view of the dynamic changes of the user-interface. A PIM is the combination of a presentation model and a finite state machine (FSM). PIMs have been described as FSMs using the μCharts language [109]. This is just one way of expressing a PIM. There is an alternative way of expressing a PIM instead of using μ-Charts. One other such method is Coloured Petri Nets. We will see in the chapter four how the presentation models are expressed in Coloured Petri Nets and how we can construct the PIM using Coloured Petri Nets.

Explaining FSM and μCharts is beyond the scope of this research, but we will describe them briefly here. Finite state machines consist of states and transitions. A *state* represents a behavioural node of the system in which it is waiting for an event to occur and the system is in one of these states at any given time. The state can change to another state when an event occurs, that is called a *transition*. The μCharts language has a visual representation called μcharts (the language is μCharts with a upper case C, and the visual representations are μcharts with a lower case c) which includes FSMs. A description of how a PIM can be visualized in μCharts can be found in [109].

A mathematical visualization of μchart of a PIM is a six-tuple (Q, $\sum$, δ, q_0, F, R) where Q is a finite set of states, $\sum$ a finite set of action labels which are taken from the I-behaviour sets of the presentation models, δ is a transition function which takes a state and an input label and returns a state, q_0 is a start state which describes the initial status of the system, F is an accept state (referred to as a final state), and a relation, R, which relates to presentation models. The relation between presentation models and states of the μcharts is used to link the current active state in presentation models and a specific state which the μchart is in. Once there exists a connection, then it means that this

part of the user-interface described in the presentation model is visible to the user and available for interaction. A condition of well-formedness of a PIM is given in [24] as follows: "A PIM of a presentation model is well-formed iff the labels on transitions out of any state are the names of I-behaviours which exist in the behaviour set of the presentation model which is associated with that state." This means that we can only make a transition between states if an appropriate I-behaviour exists in the component presentation model related to the starting state of the transition.

The PIM is derived by creating a single state for each of the component presentation models and creating the transitions between states based on the relevant I-behaviours. The PIM for the simple infusion pump can be represented visually as in Figure 3.2.

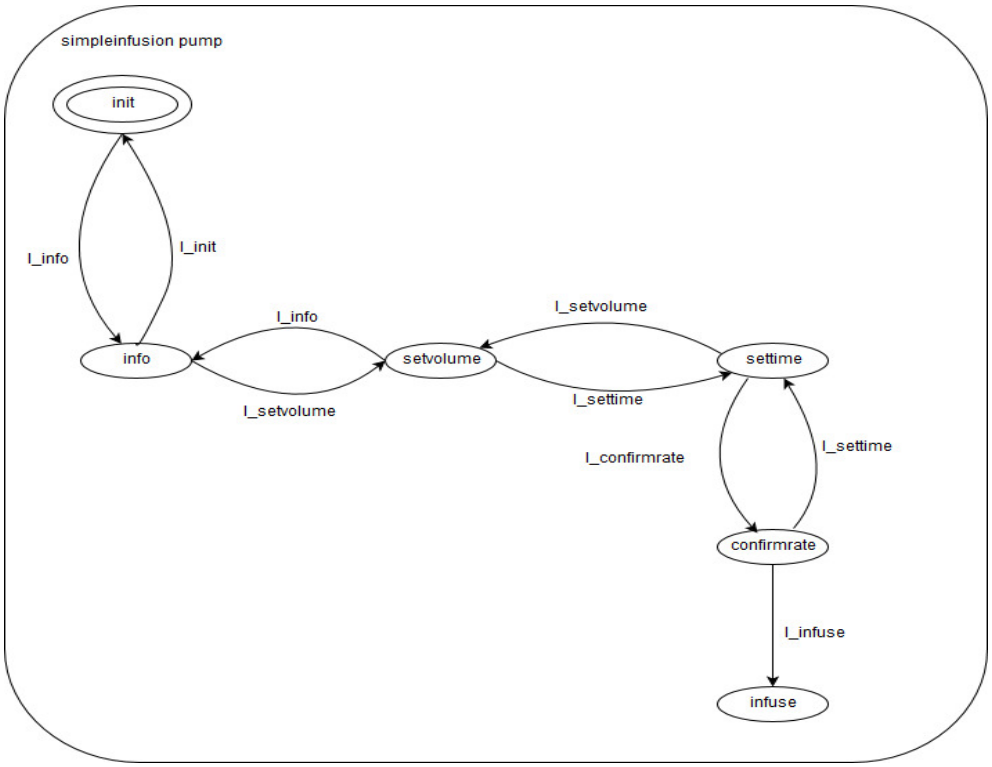

Figure 3.2: PIM for simple infusion pump prototype

In figure 3.2, six ovals represents six component presentation models and labels on the arcs represents the relevant I-behaviours associated with each component presentation models. The double oval represents the start state. Overall the PIM gives a formal meaning to the I-behaviours and provides an

abstract transition model of the system's navigational possibilities.

The PIM can be used to check various properties of the user-interface such as reachability, deadlock and the complexity of the navigational sequences. The main benefit of visualizing PIMs using μ-Charts is the structuring they allow, which prevents state space explosion. PIMs have various benefits but there are some things that PIMs don't consider. An important feature to be considered while modelling the user-interface is simulation. Simulation is available via the Z semantics of μ-Charts, for example using ProB. It is important to investigate the behaviour of the modelled system using simulation which helps in error-detection [110].

A PIM models the interactivity of a system i.e., it gives meaning to the I-behaviours of the presentation model. There must also be some way to specify the underlying functionality of a system, which gives formal meaning to the S-behaviours of the presentation model. Z specification is used to describe this underlying functionality.

3.5 An Introduction to Z

Z is a formal specification language which is used to specify and model systems. Z was developed by Oxford University's Programming Research Group. Z is based on mathematical logic such as propositional logic, predicate logic and typed set theory. Z specifications can be recognized by the use of the schema. In this section we are going to introduce Z briefly with small examples. More detailed information can be found in [50] [111] [112].

The main feature of Z is the schema notation which makes it different from other existing formal notations. It provides a very graceful way of presenting the state of a system and the ways in which that state changes. The Z notation also defines a schema calculus to combine schemas.

We begin by providing some background on logic, set theory, types, relations and functions. Note that we provide only the brief introduction to the terminology used in this work.

3.5.1 Types, Logic, Relations, Sets and Functions

• Types

Z uses typed set theory. A *type* is basically a set of common elements. A type is assigned to every expression that appears in a Z specification. When x is declared as $x : S$ then the type of x is S. For example, $\mathbb{Z}$ is a type of all integers. When n is declared as $n : \mathbb{Z}$, it defines n as integer. Types are an important part of Z as they help in finding specification mistakes.

Z provides a single built-in type, namely the type of integers $\mathbb{Z}$. The mathematical toolkit [113] is another important part of Z. It includes additional types and operations. For example, it contains types such as $\mathbb{N}$ (natural numbers).

Although Z provides only one built-in type, a specifier has a number of ways to define new types. One of the ways to do so is to declare a new *given type*. We can declare given types by simply writing the name of the type in square brackets, for example

$$[NAME]$$

defines a new type $NAME$. We can declare a variable $x : NAME$ which can be read as x is a $NAME$. There is no further information about how the name can be written. This is an example of abstraction, i.e, we can use x without specifying any information. Any further information can be added later when required.

Another significant type constructor is the *free type* constructor which is similar to given types but with additional constraints. For example,

$$INFUSING ::= Yes \mid No$$

defines a type $INFUSING$ containing exactly two different constants Yes and No.

There are also a number of ways to construct new types from existing

types. The *power set* constructor $\mathbb{P}$ is the basic type constructor which is used often. For example, the set $\{tim, bob\}$ is of type $\mathbb{P}\, NAME$, given that each of the names is of type $NAME$. The *Cartesian product* is another frequently used type constructor. For example, NAME $\times \mathbb{Z}$ is a type consisting of ordered pairs, e.g. $(tim, 3)$ is of type $NAME \times \mathbb{Z}$.

• Logic

Propositional and predicate logic is used by the Z notation to state the relationship between the components of a system [112]. The propositional logic used by Z contains the common connectives with their usual meaning and order of precedence: negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$), implication ($\Rightarrow$), equivalence ($\Leftrightarrow$).

Predicate logic introduces quantifiers into the language with the notion of free and bound variables. The two types of quantifiers are:
Universal quantification (for all) which has the form

$$\forall\, x : T \mid p \bullet q$$

and is interpreted as for all x of type T satisfying the predicate p, q holds.
Existential quantification (there exists) has the form:

$$\exists\, x : T \mid p \bullet q$$

and is interpreted as there exists an x of type T satisfying the predicate p such that q holds.

• Sets

Set theory is the other keystone of the Z specification. Membership ($\in$), empty set ($\emptyset$), subset ($\subseteq$) and equality ($=$) are all defined as usual. Sets can be built

by listing the elements, like

$$\{yes, no\}$$

Sets can also be given by set comprehension, $\{n : T \mid P(n)\}$, which is the set of all elements n in T satisfying the predicate $P(n)$, for example,

$$\{n : \mathbb{Z} \mid n \geq 0\}$$

which is interpreted as the set of all natural numbers.

There is one more type of comprehension, $\{n : T \mid P(n) \bullet Q(n)\}$ which is the set of all expressions $Q(n)$ where n satisfies P and n is in T. For example,

$$\{n : \mathbb{Z} \mid n > 0 \bullet n * 2\}$$

is the set of all positive even numbers. This is because the expression $n * 2$ gives a positive even number for all n in $\mathbb{Z}$ that satisfy the predicate $n > 0$.

Furthermore, union ($\cup$), intersection ($\cap$) and difference ($\setminus$) have their usual meaning.

• Abbreviations

Abbreviations are vital to the readability of a specification. In Z they are written as:

$$name == expression$$

For example, we could define $Teachers == \{bob, charlie, alan\}$ which introduces a set consisting of the three elements listed. An abbreviation does not introduce new values. So, the elements *bob*, *charlie* and *alan* have to have been previously defined.

Another way would be to use an axiomatic definition (discussed later) or a free type. For example, $Teachers ::= bob \mid charlie \mid alan$ would introduce *Teachers* as a type consisting exactly of these three different elements, which

had not been previously defined.

• Relations

Relations are another important part of Z. Z makes extensive use of relations. A relation is a set of ordered pairs. If X and Y are sets, then $X \leftrightarrow Y$ represents the set of all relations between the sets X and Y, i.e., the set of all sets of ordered pairs whose first elements are members of X and whose second elements are members of Y. $X \leftrightarrow Y$ is defined as $\mathbb{P}(X \times Y)$. Usually, the maplet notation $x \mapsto y$ is used as an alternative to (x, y) for defining relations.

For example, the relation *teaches* is used to record which subject is taught by which teacher. If the group of teachers is defined by

$$Teachers == \{bob, charlie, alan\}$$

and the subjects is defined by

$$Subjects == \{dbms, hci, oop\}$$

and if *teaches* is an element of $Teachers \leftrightarrow Subjects$, then the statement *'Bob teaches dbms'* could be formalized as $Bob \mapsto dbms \in teaches$.

To extract information, *domain* and *range* functions are included in the toolkit. The domain of a relation R of type $X \leftrightarrow Y$ is the set of elements in X related to anything in Y i.e. $\text{dom } R = \{x : X;\ y : Y \mid x \mapsto y \in R \bullet x\}$. The range of the relation R is the set of elements in Y to which any element of X is related, i.e. $\text{ran } R = \{x : X;\ y : Y \mid x \mapsto y \in R \bullet y\}$. For example, given the relation *teaches* we might have $\text{dom } teaches = \{bob, charlie, alan\}$ and $\text{ran } teaches = \{dbms, hci, oop\}$.

• Functions

The set of all *partial functions* $X \nrightarrow Y$ from X to Y is the set of all relations between X and Y such that $x \in X$ is related to at most one $y \in Y$. A function f from X to Y is said to be *total*, denoted *f:* $X \rightarrow Y$, if $\text{dom } f = X$, i.e.

if it relates each member of X to exactly one member of Y. For example, we can write *count:* $NAME \rightarrow \mathbb{N}$ for a function *count* such that *count(n)* is the number of letters in a given name n, or *names* : $\mathbb{N} \nrightarrow \mathbb{P}\ NAME$ for a function that gives all the names of a given length. Every name has a number of letters it consists of, hence *count* is total but there is atleast one natural number such that there cannot be a name of that length, hence *names* is partial.

Functions can be *injective, surjective or bijective*. A function from X to Y is *injective*, if each $y \in Y$ is related to no more than one $x \in X$. A function from X to Y is *surjective*, if its range is equal to Y. A function is *bijective*, if it is both injective and surjective.

So far we have discussed the basic language constructs used in Z specifications. Next we introduce the main feature of Z, i.e schemas.

3.5.2 Schemas in Z

Z Schemas are used to specify the state space and operations of a system. Schemas are the most identifiable feature of Z.

3.5.2.1 Schema syntax

A schema consists of three parts: the declarations, the predicates and an optional name which can be written as:

Name

Declarations

Predicates

The schema can also be written horizontally like:

$$Name \mathrel{\widehat=} [Declarations \mid Predicates]$$

The declaration part contains a list of declarations of observations and the predicate part puts constraints on these observations. For example: the *Num*

schema has two observations n and x which are of type integer. The predicate section adds constraints to the observations, i.e. n should be less than 10 and the value of x should be greater than 15.

Num

$$n : \mathbb{Z}$$
$$x : \mathbb{Z}$$

$$n < 10$$
$$x > 15$$

3.5.2.2 Axiomatic definition

Observations are local to the schema they are declared in. Global constants can also be introduced by means of *axiomatic definitions* so that all schemas have access to those constants. An *axiomatic definition* is written as:

Declarations

Predicate

The predicate part is optional and is used to constrain the values introduced in the declaration part. For example

$$hours : \mathbb{P}\,\mathbb{N}$$
$$minutes : \mathbb{P}\,\mathbb{N}$$
$$seconds : \mathbb{P}\,\mathbb{N}$$

$$hours = 0 \mathinner{..} 24$$
$$minutes = 0 \mathinner{..} 59$$
$$seconds = 0 \mathinner{..} 59$$

introduces global constants *hours*, *minutes*, and *seconds* as powersets of nat-

ural numbers and the predicate part specifies a constraint on their values.

3.5.2.3 Schema Inclusion

A schema can be included in another schema which is referred to as schema inclusion. A schema could be included by just listing its name in the declaration part of a schema. For example, we have the following two schemas:

$S1$

$p, q : \mathbb{Z}$

$p \neq q$

$S2$

$r, s : \mathbb{N}$

$r \geq s$

We can form a new schema by including two existing ones:

$S1S2$

$S1$

$S2$

If expanded, schema S1S2 will be:

$S1S2$

$p, q : \mathbb{Z}$

$r, s : \mathbb{N}$

$p \neq q \land r \geq s$

3.5.2.4 Decorations and Conventions

There are some conventions/decorations that are often used when writing Z specifications.

Inputs and Outputs: Suffix ? is used to indicate an input observation and ! is used to indicate an output observation. For example, in the *Add* schema $n?$ is an input observation and $x!$ is an output observation.

Add

$$n? : \mathbb{Z}$$
$$x! : \mathbb{Z}$$
$$x! = n? + 1$$

States Schemas and Operation Schemas: *State schemas* describe the state space of the system under consideration. For example, schema *Counter* is the state schema of a simple counter with a declaration part consisting of two observations: a current *value* and a *limit* and a predicate part that puts constraint on these observations, i.e. *value* should be less than or equal to *limit*:

Counter

$$value : \mathbb{N}$$
$$limit : \mathbb{N}$$
$$value \leq limit$$

For Z specifications, there must be atleast one possible initial state, which is specified by another schema with the same signature as the state schema. An initial schema for the *Counter* is written below:

InitCounter

$Counter$

$value = 0$

$limit = 100$

An operation is specified in Z with a predicate relating the state before and after the operation. For example, an increment operation for the counter is specified as:

Increment

$\Delta Counter$

$value' = value + 1$

The declaration $\Delta Counter$ means that the state *Counter* is changed by the invocation of the operation. In the predicate section, the new value of the counter is primed ($value'$) while the old value is unprimed. The predicate states that the new value of the counter is the old value plus one.

Let us consider another operation schema *Display* as written below:

Display

$\Xi Counter$

$v! : \mathbb{N}$

$v! = value$

Here the declaration $\Xi Counter$ means that the operation cannot change the state of *Counter*, i.e., $value' = value$

3.5.2.5 Schema Calculus

Schemas can be combined using schema calculus. We will briefly describe some of the schema calculus used in this work.

Schema Conjunction: Schema conjunction is very much related to schema inclusion. The schema resulting from the conjunction of the schemas $S1$ and $S2$ contains both $S1$ and $S2$. We write $S1 \land S2$ to denote the conjunction of these two schemas. For example, consider two schemas $S1$ and $S2$:

$S1$

$x : \mathbb{N}$

$y : \mathbb{N}$

$x > 10$

$S2$

$z : \mathbb{N}$

$z < 10$

A new schema formed by conjoining $S1 \land S2$ is:

S

$x : \mathbb{N}$

$y : \mathbb{N}$

$z : \mathbb{N}$

$x > 10$

$z < 10$

This can also be written as: $S \mathrel{\widehat=} S1 \land S2$.

Schema conjunction allows one to specify different aspects of a system separately. It can be usefully applied both on operation and state schemas to combine those aspects to form a complete description.

Schema Disjunction: Schema disjunction is often applied on operation schemas to handle separate cases. For example, consider schemas $S1$ and $S2$:

$S1$

$$x : \mathbb{Z}$$

$$x \in \mathbb{N}$$
$$even(x)$$

$S2$

$$x : \mathbb{Z}$$

$$x \in \{-1, 1\}$$

A new schema formed by disjunction $S1 \lor S2$ is:

$$x : \mathbb{Z}$$

$$(x \in \mathbb{N} \land even(x)) \lor x \in \{-1, 1\}$$

which allows exactly all values of x allowed by either $S1$ or $S2$.

Schema Quantification: The schema quantification of a schema U results in a new schema V containing a subset of the components of U in its declaration, with a predicate that is obtained from U by quantifying over the removed components. Quantification is used to express universal or existential

properties of the given schema.

- **Existential Quantification:** Given a schema $U == [x : X;\ Decl_U \mid pred_U]$ where $Decl_U$ consists of declarations but for $x : X$, then the existential quantification over x in U is:
  $\exists\, x : X \bullet U == [Decl_U \mid \exists\, x : X \bullet pred_U]$
  Thus, $\exists\, x : X \bullet U$ is a schema on all components of U except x.

  For example, consider the schema *State* with two observations x and y of type $\mathbb{N}$ and predicate $x<10$.

  State

  $x : \mathbb{N}$

  $y : \mathbb{N}$

  $x < 10$

  the *Operation* schema below includes the observations in *State*.

  Operation

  $\Delta State$

  $x' = x + 1$

  $y' = x$

  The following schema shows $\exists$ *State* $\bullet$ *Operation*:

  $x', y' : \mathbb{N}$

  $\exists\, x, y : \mathbb{N} \bullet x < 10 \land x' = x + 1 \land y' = x$

- **Universal Quantification:** It is also possible to universally quantify over schemas. Given a schema $U == [x : X; Decl_U \mid pred_U]$ *where* $Decl_U$

consists of declarations but for $x : X$, then the universal quantification over x in U is

$\forall\ x : X \bullet U == [Decl_U \mid \forall\ x : X \bullet pred_U]$

Thus, $\forall\, x : X \bullet U$ contains all observations of U but x.

Precondition Calculation: The precondition of an operation characterizes all the states for which the outcome of the operation is properly defined. In order to make a precondition of a given operation explicit one needs to calculate it.

The precondition, pre *Op*, of an operation *Op==[Δ State; ins?; outs! | pred]* on a state *State* with inputs *ins*? and outputs *outs*! is defined by:

$preOp = \exists\, State';\ outs! \bullet Op$

This hides the afterstate and outputs, resulting in a schema that only includes beforestate and input observations. Thus, pre *Op* is another schema on State and *ins*?, indicating on which before states and inputs the operation is applicable. The precondition is, based on this definition, a rather abstract predicate. This predicate is usually simplified applying, for example, the one-point rule and other equivalences [50].

We will now illustrate more about Z schemas with a simplified infusion pump example.

3.5.3 Simplified Infusion Pump Z Example

We consider the same example of a simple infusion pump as presented in section 4.3.1. The presentation model of a simple infusion pump has S-behaviours associated with the widgets. Meaning to those behaviours are given by Z. So for each of the S-behaviours, there exists a Z schema.

The associated definitions for the simple infusion pump are as given below:

$INFUSING ::= Yes \mid No$

$BATTERY ::= sufficient \mid insufficient$

$CHAR ::= initialisingpump$

These three definitions introduce three types; $INFUSING$, $BATTERY$, and $CHAR$. $INFUSING$ and $BATTERY$ each have two values in them. $CHAR$ has one value.

The *SimpleInfusionPump* schema defines the state space of the model. It says that in each state in the state space there are seven observations: *battery* of type $BATTERY$, *infusing* of type $INFUSING$, while the remaining observations *timer*, *volume*, *volumeleft*, *infusionrate* and *timeleft* all have the type natural numbers.

SimpleInfusionPump

$$
\begin{aligned}
&battery : BATTERY \\
&timer : \mathbb{N} \\
&volume : \mathbb{N} \\
&volumeleft : \mathbb{N} \\
&infusionrate : \mathbb{N} \\
&timeleft : \mathbb{N} \\
&infusing : INFUSING
\end{aligned}
$$

In the initial state of the specification, the initial values of the simple infusion pump observations are as below:

Init

$$
\begin{aligned}
&SimpleInfusionPump \\
&\hline
&battery = sufficient \\
&timer = 0 \\
&volume = 0 \\
&infusionrate = 0 \\
&timeleft = 0 \\
&volumeleft = 0 \\
&infusing = No
\end{aligned}
$$

The *DisplayStartMessage* schema displays the output *initialisingpump* on the screen of the device. None of the other values of the state space schema change as represented by the declaration $\Xi SimpleInfusionPump$.

DisplayStartMessage

$$\Xi SimpleInfusionPump$$
$$display! : CHAR$$

$$display! = initializingpump$$

The *DisplayBatteryLife* schema displays the current status of the battery which could be *sufficient* or *insufficient*.

DisplayBatteryLife

$$\Xi SimpleInfusionPump$$
$$display! : BATTERY$$

$$display! = battery$$

The operations *IncreaseVolume* and *DecreaseVolume* changes the value of a dose to be infused using the up and down keys.

IncreaseVolume

$$\Delta SimpleInfusionPump$$

$$battery' = battery$$
$$timer' = timer$$
$$volume' = volume + 1$$
$$volumeleft' = volumeleft$$
$$infusionrate' = infusionrate$$
$$timeleft' = timeleft$$
$$infusing' = infusing$$

DecreaseVolume

$\Delta SimpleInfusionPump$

$battery' = battery$

$timer' = timer$

$volume' = volume - 1$

$volumeleft' = volumeleft$

$infusionrate' = infusionrate$

$timeleft' = timeleft$

$infusing' = infusing$

The *SetVolume* schema changes the value of the *volumeleft* to the new value of the *volume*.

SetVolume

$\Delta SimpleInfusionPump$

$battery' = battery$

$timer' = timer$

$volume' = volume$

$volumeleft' = volume'$

$infusionrate' = infusionrate$

$timeleft' = timeleft$

$infusing' = infusing$

The operations *IncreaseTime* and *DecreaseTime* changes the value of the *timer* using the up and down keys.

IncreaseTime

$\Delta SimpleInfusionPump$

$battery' = battery$

$timer' = timer + 1$

$volume' = volume$

$volumeleft' = volumeleft$

$infusionrate' = infusionrate$

$timeleft' = timeleft$

$infusing' = infusing$

DecreaseTime

$\Delta SimpleInfusionPump$

$battery' = battery$

$timer' = timer - 1$

$volume' = volume$

$volumeleft' = volumeleft$

$infusionrate' = infusionrate$

$timeleft' = timeleft$

$infusing' = infusing$

The *SetTime* schema changes the value of *timeleft* to the new value of the *timer*.

SetTime

$\Delta SimpleInfusionPump$

$battery' = battery$

$timer' = timer$

$volume' = volume$

$volumeleft' = volume'$

$infusionrate' = infusionrate$

$timeleft' = timer'$

$infusing' = infusing$

The *SetRate* schema sets the *infusionrate* by dividing the $volume'$ by $timer'$.

SetRate

$\Delta SimpleInfusionPump$

$battery' = battery$

$timer' = timer$

$volume' = volume$

$volumeleft' = volumeleft$

$infusionrate' = volume' \mathsf{div} timer'$

$timeleft' = timeleft$

$infusing' = Yes$

So far we have seen that presentation models describe the behaviours (I-behaviours and S-behaviours) of the widgets of the entire user interface. A PIM gives meaning to the I-behaviours and shows the navigational possibilities. Z gives meaning to the S-behaviours. This results in three different models. To have a look at the widgets and behaviours we have to refer to the PM, to see the navigational possibilities we refer to the PIM and to see the underlying system functionality we refer to the Z specification. We will show in this thesis

that we can actually combine all these three models into one using another formal method, Coloured Petri Nets.

3.6 Coloured Petri Nets

3.6.1 Background

Petri net theory [30] [31] was introduced in 1962 by Carl Adam Petri in his PhD thesis [114]. In system engineering, computer science, and many other areas, Petri nets are a powerful modelling formalism. Petri nets combine a well-defined mathematical theory with a graphical representation of the dynamic behaviour of systems. The theoretical aspects of Petri nets allow precise modelling and analysis of system behaviour, while the graphical representation of Petri nets enable visualization of the modelled system state changes. This combination is the main reason for the great success of Petri nets [115].

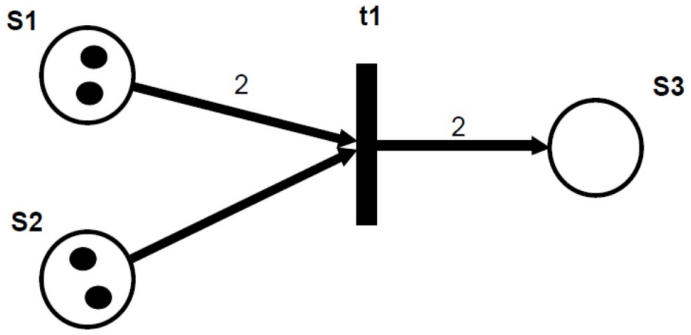

Figure 3.3: An example of Petri nets

A Petri net consists of a set of *places* drawn as circles, a set of *transitions* drawn as rectangles and a set of *arcs* which connects places to transitions and transitions to places with the associated weight as shown in Figure 3.3. Typically, places represent states whereas transitions represent an action to be taken or an event that can occur.

In general, a Petri net is a bipartite graph as shown in Figure 3.3 and the bi-partition is between places and transitions, which means that places cannot be connected to places and transitions cannot be connected to transitions. Each place may contain different number of small black dots which are known as *tokens*. Tokens are distributed on the places which is known as *marking*. An initial state is called the *initial marking*. Transitions are enabled only when sufficient tokens are available and only those transitions can occur which are enabled. The occurrence of a transition changes the state of the system.

However, there are systems that contain complex data which influence the behaviour of the system. This means that there was a need for a modelling language which makes it possible to represent data in an adequate way. Petri net models become very large, complicated and unreadable for complex systems. So another kind of Petri net, called High Level Petri Nets [116] were introduced to overcome this problem. The first successful type of High Level Petri Net was called Predicate/Transition Nets (Pr/T-nets). This net class was developed by Hartmann Genrich and Kurt Lautenbach from Petri's group at Schloss Birlinghoven [117].

An ordinary Petri net has no types and no modules and there was only one kind of token. So to represent two processes of similar kind, there are two separate subnets. This repetition of nets make the model complex. To overcome this, the Predicate/Transition net (Pr/T-net) was introduced. The main idea behind the Predicate/Transition net is that the tokens can be distinguished from each other so that instead of having different nets, a single net is used with different kinds of tokens. Tokens are said to be coloured in contrast to ordinary Petri net which are indistinguishable and drawn as black dots. Although the invention of token colours was a big step forward, it still had some limitations because there was only one class of token colours. So to overcome this limitation, the next step forward was achieved by the development of Coloured Petri Nets at Aarhus University, Denmark in 1979. Coloured Petri Nets allowed the use of a number of different sets of colours with which it became possible to distinguish between token colours. It was convenient to use

data types to define colour sets. By doing so, token colours became structured and type checking became possible.

3.6.2 Introduction to Coloured Petri Nets

Coloured Petri Nets (CPN) is a language for the modelling and validation of systems [29]. It is a modelling language combining Petri nets with the functional programming language Standard ML [118]. Petri nets [30] [31] provide the foundation of the graphical notation and the basic primitives for modelling concurrency, communication, and synchronization. Standard ML provides the primitives for the definition of data types, describing data manipulation, and for creating compact models. A CPN model of a system is an executable model representing the states of the system and the events (transitions) that can cause the system to change state.

3.6.2.1 CPN Tool

There are lot of tools that can be used for CPN [36][119]. CPN Tool v.4.0.1 from [37] is used in this thesis. CPN Tool was initially developed by the CPN group at Aarhus University in Denmark. It was maintained and developed further by AIS group at Eindhoven University of Technology in the Netherlands since 2010. CPN Tool is a tool for editing, simulating and analysis of coloured Petri Nets [37]. Using CPN Tool, it is possible to investigate the behaviour of the modelled system using simulation, to verify properties by means of state space methods and model checking, and to conduct simulation-based performance analysis. User interaction with CPN Tool is based on direct manipulation of the graphical representation of the CPN model using interaction techniques, such as tool palettes and marking menus.

3.6.2.2 Coloured Petri Nets Components

This section introduces the basic terminology and the structure of a CPN model using a very simple example shown in Figure 3.4.

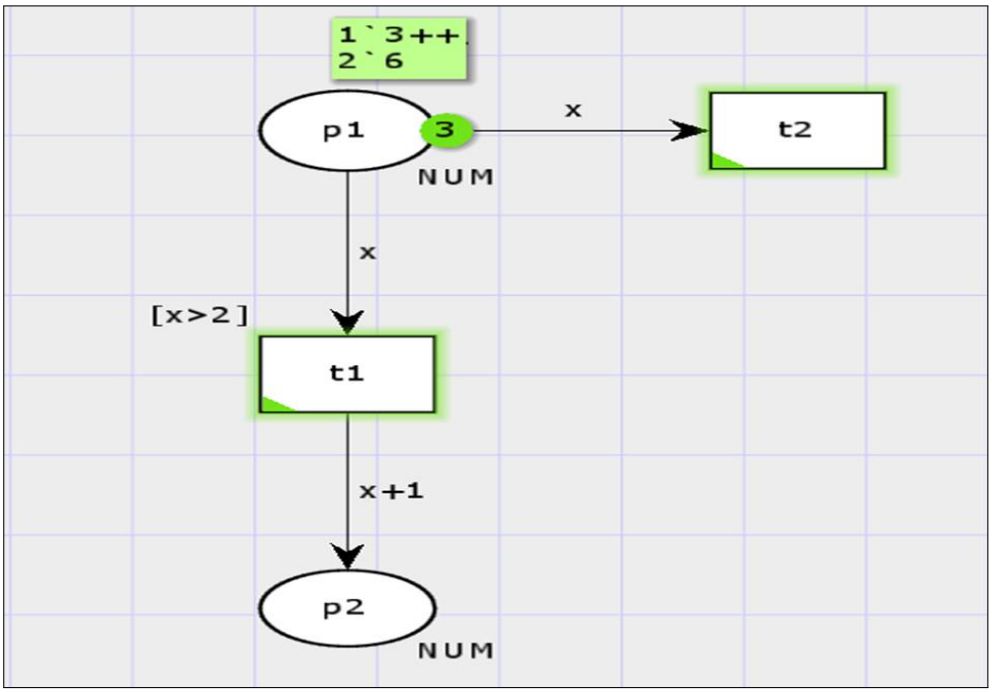

Figure 3.4: Model to illustrate Basic Components of Coloured Petri Nets

Places represent the states of the model and are drawn as ovals. The names of the places are written inside the oval and usually they don't have any formal meaning but are important for the readability of the model. In this thesis, however, the names of the places will have a formal meaning as the names of the places would be the same as the names of the component presentation model (discussed in the next chapter).

Transitions are drawn as rectangles that represent the actions of a CPN model. The name of the transition is written inside the rectangle. Transitions can have guards associated with them which are located at the top or bottom of the transitions by default. Guards are like preconditions which should be true to make the transition enabled.

Arcs are the arrows that show the flow of the model. Arcs connect places and transitions together. Each arc has an arc inscription that tells the quantity and the type of token colours that are transferred from a place to a transition or from a transition to a place.

The set of places, transitions and arcs are denoted by P, T and A respectively. In figure 3.4, there are two places, two transitions and three arcs. The

set of places, transitions and arcs are:

$$P = \{p1,\ p2\}$$
$$T = \{t1,\ t2\}$$
$$A = \{(p1,t1),\ (p1,t2),\ (t1,p2)\}$$

Each place is associated with one ***colour set*** that determines the type of data the place contains. The colour set is written as an inscription at the bottom left or right of the place. There are different categories of colour sets which can be found in [28]. In Figure 3.4, each place has *NUM* colour set which is an integer colour set defined as:

$$colset\ NUM = int;$$

Similar to ordinary Petri nets, CPN also has tokens distributed on the places which is known as ***marking***. Each token has an attached data value which is referred to as a ***token colour*** which belongs to a specific type which is decided by the colour set attached to a place. A marking represents the state of the CPN model which is determined by the number of tokens and the token colours on individual places. We use the multi-set m_{p1} to denote the multi-set over the colour set *NUM* corresponding to the marking of the place $p1$ in Figure 3.4:

$$m_{p1} = 1`3 ++ 2`6$$

This indicates that the place $p1$ has the marking which contains 1 token of data value 3 and 2 tokens of data value 6.

A ***multi-set*** is just like a set except that an element may occur several times, and we keep track of that number for each distinct element. So a "normal" set is just a multi-set where every element appears just once [28]. The multi-set m_{p1} can be defined as:

$$m_{p1}(s) = \begin{cases} 1 & if\ s=3 \\ 2 & if\ s=6 \\ 0 & otherwise. \end{cases}$$

3.6.2.3 Formal Definition of CPN

In this section we present CPN definition by Jensen [28] as they will be used later in chapter 4 and 5 as the theoretical basis for our models.

Definition 3.6.1. The following are assumed to be defined:

(i) *EXPR* denotes the set of expressions provided by the inscription language, i.e., CPN ML [28].

(ii) Given an expression $e \in EXPR$, the ***type*** of e is represented by $Type[e]$.

(iii) The set of ***variables*** in an expression e is denoted by $Var[e]$.

(iv) V denotes the set of (all) variables.

(v) By S_{MS}, we denote the set of all *multi-sets* over the set S.

Let us now formally define elements that constitute a net inscription, i.e., colour sets, arc expressions and guards. We will explain these elements and other graphical conventions using the example shown in figure 3.4.

Arc expressions are the expressions written on the arcs. In figure 3.4, for the arc expressions we have the following variables:

$$Var[e] = \begin{cases} \{\text{x}\} & \textit{if } e = x \\ \{\text{x}\} & \textit{if } e = x + 1 \end{cases}$$

$\sum$ denotes the ***colour sets*** (that determines the type of data) for the CPN model. Given a set of variables V, $\forall\, v \in V : Type[v] \in \sum$, i.e., a variable must have a type that is already given in $\sum$. A finite set of non-empty colour sets $\sum$ in figure 3.4 is:

$$\sum = \{NUM\}$$

A finite set of variables V is:

$$V = \{x\}$$

The ***node function*** maps each arc into a pair where the first element is the source node and the second is the destination node. The two nodes must be of different kinds, i.e., one must be a place while the other is a transition.

The ***colour set function*** is a function, $C : P \rightarrow \sum$ that maps each place to its corresponding colour set. The colour set function for the CPN model in figure 3.4 is:

$$C(p) = \begin{cases} NUM & \textit{if } p \in \textit{\{p1, p2\}} \end{cases}$$

A ***guard*** is a Boolean expression attached to a transition. For a transition to be enabled it is a necessary condition that the guard of the transition should evaluate to *True*. A function $G : T \rightarrow EXPR$ is called a ***guard function*** and assigns every transition $t \in T$ to a boolean expression, i.e., $Type[G(t)] = Bool$. All variables in $G(t)$ must have types that belong to $\sum$. A missing guard is considered as a shorthand for the guard expression which always evaluates to *True*. The CPN model, in figure 3.4, has a guard function defined as:

$$G(t) = \begin{cases} x > 2 & \textit{if t= t1} \\ True & otherwise \end{cases}$$

A function $E : A \rightarrow EXPR$ is called an ***arc expression function*** which assigns to every $a \in A$ an expression $E(a)$. Similar to the guards of the transitions, all variables in $E(a)$ must have types that belong to $\sum$. For an arc $(p, t) \in A$, connecting a place $p \in P$ and a transition $t \in T$, it is required that the type of the arc expression is the multi-set over the colour set $C(p)$ of the place p, i.e., $Type[E(p, t)] = C(p)_{MS}$. This is for the directed arc from a place to a transition. Similarly it can be applied to a directed arc from a transition to a place. For an arc $(t, p) \in A$ it is required that $Type[E(t, p)] = C(p)_{MS}$. For the model in figure 3.4, the arc expression function is defined as:

$$E(a) = \begin{cases} x & \textit{if a} \in \textit{\{(p1,t2),(p1,t1)\}} \\ x + 1 & \textit{if a = (t1,p2)\}} \end{cases}$$

An initialization function maps an initial marking to each place. The ***initialization function*** $I : P \rightarrow EXPR$ assigns an initialization expression

$I(p)$ to each place p. An initialization expression $I(p)$ is required to evaluate to a multi-set over the colour set of the place p, i.e., $Type[I(p)] = C(p)_{MS}$. The initialization expression must be a closed expression, i.e., it cannot have any variables. For the model in figure 3.4, the initialization function is given by:

$$I(p) = \begin{cases} 1`3 + + \ 2`6 & \textit{if } p = \{p1\} \\ \emptyset_{MS} & \textit{otherwise} \end{cases}$$

$\emptyset_{MS}$ is used to denote the empty multi-set.

As per the explanation of the above functions, a non-hierarchical Coloured Petri net is defined as:

Definition 3.6.2. A Coloured Petri Net is a tuple CPN = $(\sum, P, T, A, N, C, G, E, I)$ satisfying the following requirements [21]:

(i) $\sum$ is a finite set of non-empty types, called colour sets.

(ii) P is a finite set of places.

(iii) T is a finite set of transitions.

(iv) A is a finite set of arcs such that:
- $P \cap T = P \cap A = T \cap A = \emptyset$.

(v) N is a node function. It is defined from A into $P \times T \cup T \times P$.

(vi) C is a colour function. It is defined from P into $\sum$.

(vii) G is a guard function. It is defined from T into expressions such that:
- $\forall\ t \in T : [Type(G(t)) = Bool \wedge Type(Var(G(t))) \subseteq \sum]$.

(viii) E is an arc expression function. It is defined from A into expressions such that:
- $\forall\ a \in A : [Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \sum]$

where $p(a)$ is the place of $N(a)$.

(ix) I is an initialization function. It is defined from P into closed expressions such that:

- $\forall\ p \in P : [Type(I(p)) = C(p)_{MS}]$.

3.6.2.4 Dynamic behaviour of a CPN Model

The dynamic behaviour of the CPN model is described as the change of marking with the occurrence of transitions. In this section we will talk about markings, binding elements and enabling and occurrence of a step (finite multi-set of binding elements)

A distribution of tokens on the places is called a *marking*. A marking M is a function that maps each place p into a multi-set of values $M(p)$ representing the marking of p. The individual elements in the multi-set $M(p)$ are called ***tokens***. The multi-set of tokens present on a place p in a marking M is required to match the type of the place, i.e., $M(p) \in C(p)_{MS}$.

A ***binding*** b of a transition t is a function that maps each variable v of the transition t to a value $b(v)$ belonging to the type of the variable v, i.e., $b(v) \in Type[v]$. Bindings are written as $\langle var_1 = val_1, var_2 = val_2, ..., var_n = val_n \rangle$, where $var_1, var_2, ..., var_n$ are the variables in $Var(t)$ and val_i is the value bound to the variable var_i. A ***binding element*** is a pair (t, b) consisting of a transition t and a binding b of t.

Enabling of a Step

Definition 3.6.3. For a binding element to be ***enabled*** in a marking M there are two conditions to satisfy:

- In order for a binding to be enabled, the corresponding guard expression must evaluate to *True*.
- For each place p, an arc expression $E(p, t)$ has to be evaluated against the binding b and $E(p, t)\langle b \rangle \leq M(P)$ must hold. This means that for each place p there should be enough tokens there of the right form so

that transition t can remove the required number of tokens of that form when it occurs.

Definition 3.6.4. A ***step*** Y is a non-empty, finite multi-set of binding elements.

A step Y is ***enabled*** in a marking M iff the following property is satisfied [21]:

$$\forall\, p \in P : \Sigma_{(t,b)\in Y}\ E(p,t)\langle b\rangle\ \leq M(p)$$

which says that all the binding elements that are going to take part in the step are enabled.

To explain the concept of enabling of a step, let us consider a very simple example as shown in Figure 3.5. The CPN model shown in Figure 3.5 has two places $p1$ and $p2$ drawn as ellipses and one transition $t1$ drawn as a rectangle. The colour set declaration is shown in the box at the upper left corner of Figure 3.5. The declaration of the colour set *NUM* tells us that each token on $p1$ and $p2$ is an integer.

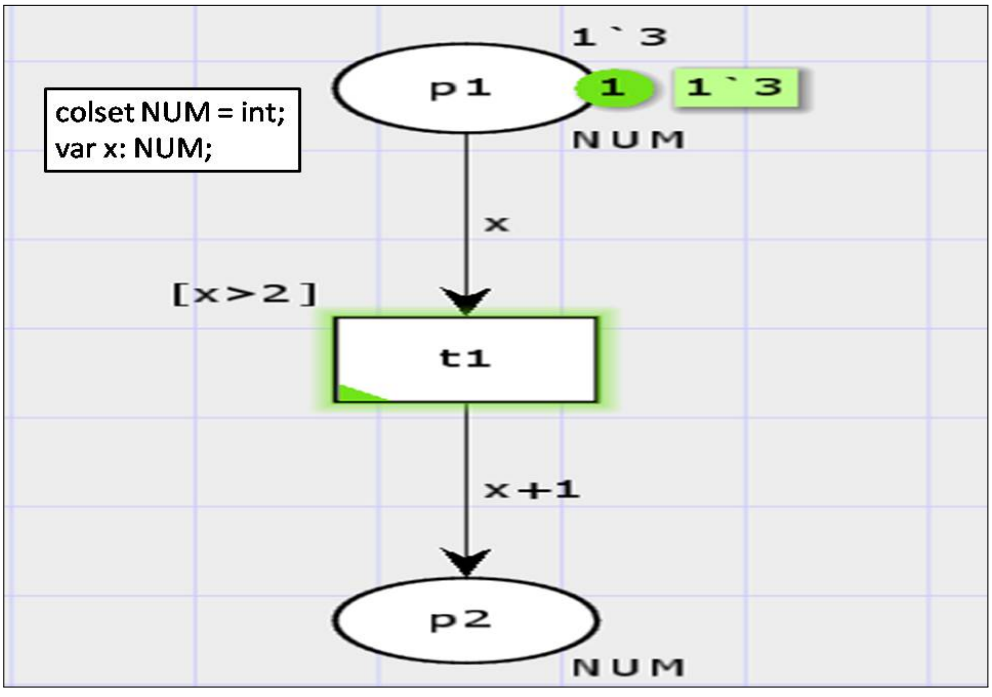

Figure 3.5: Example to Illustrate Enabling of Transition

In a given marking, the enabling rule specifies when a step (consisting of

a multi-set of binding elements) becomes enabled.

The ***marking*** of the places for the CPN model in Figure 3.5 is given by:

$$M(p) = \begin{cases} 1`3 & \textit{if } p{=}p1 \\ \emptyset & \textit{otherwise} \end{cases}$$

The variables for the transitions for the model in the Figure 3.5 are:

$$Var(t) = \begin{cases} x & \textit{if } t{=}t1 \end{cases}$$

The ***initial marking*** (1`3 shown in black text in Figure A)on place p1), denoted by M_0 , is obtained by evaluating the initialization expression. The initialization expression does not contain any variables and its evaluation is with the empty binding (denoted by $\langle\rangle$), i.e., $M_0(p) = I(p)\langle\rangle$ for each $p \in P$. The initial marking for the example in Figure 3.5 is given by:

$$M_0(p) = \begin{cases} 1`3 & \textit{if } p{=}p1 \\ \emptyset & \textit{otherwise} \end{cases}$$

As stated earlier, the transitions have guards attached to them and the arcs carry expressions with them (arc expressions). These two determine the enabling and occurrence of a step as per Definition 3.6.4. For a binding element (t, b) where t is a transition and b is a binding, the guard expression $G(t)$ of the transition is evaluated for the binding b (i.e. the variables in the guard expression are replaced by the expressions associated with each variable by the binding) and the result is written as $G(t)\langle b\rangle$.

Similarly, the arc expression $E(a)$ (for any arc a) is also evaluated for the binding b and the result is written as $E(a)\langle b\rangle$. For an arc $a = (p, t)$, which connects a place p and a transition t , the arc expression $E(p, t)$ denotes the arc expression on the input arc from p to t. When no such arc exists, we define $E(p, t) = \emptyset_{MS}$. Analogously, $E(t, p)$ denotes the arc expression on the output arc from t to p. When no such arc exists, we define $E(t, p) = \emptyset_{MS}$.

Let us see how the transition *t1* is enabled in the initial marking. For a CPN model given in Figure 3.5 consider a binding element $(t1, b_{t1})$, where:

$$b_{t1} = \langle\ x{=}3\ \rangle$$

Marking of the place *p1* is:

$$M(p1) = 1`3$$

The arc expression *E(p1,t1)*, from place *p1* to the transition *t1* is x. When the binding b_{t1} is applied to the arc expression *E(p1,t1)* we get:

$$E(p1,t1)\ \langle\ b_{t1}\ \rangle = 1`3$$

As per Definition 3.6.4, the place must have enough tokens of the right form in order for a step to happen (recall that an expression like $1'3$ means we have one token of the form 3, and that the occurrence of just 3 on the arc is shorthand for a single token, i.e. $1'3$). We have:

$$E(p1,t1)\ \langle\ b_{t1}\ \rangle = 1`3 \leq 1`3 = M(p1)$$

So the condition, $E(p,t)\langle b\rangle \leq M(p)$ for this case is satisfied.

Also there is a guard $[x > 2]$ on the transition $t1$ which must evaluate to *True* to make transition $t1$ enabled. We have:

$$G(t1)\langle\ b_{t1}\ \rangle = 3>2 = True$$

As the guard evaluates to *True* and there are enough tokens on the input place, so a transition $t1$ is enabled.

For the same CPN model given in Figure 3.5 now consider the binding element $(t1, b'_{t1})$, where we assume:

$$b'_{t1} = \langle\ x=8\ \rangle$$

The arc expression *E(p1,t1)*, from place *p1* to the transition *t1* is x. When the binding b_{t1} is applied to the arc expression *E(p1,t1)* we get:

$$E(p1,t1)\ \langle\ b'_{t1}\ \rangle = 1`8$$

As per the Definition 3.6.4, the place must have enough tokens of the right form in order for a step to happen. We have:

$$E(p1,t1)\ \langle\ b'_{t1}\ \rangle = 1`8 \not\leq 1`3$$

So the condition, $E(p,t)\langle b\rangle \leq M(p)$ for this case is not satisfied.

Also there is a guard $[x > 2]$ on the transition $t1$ which must evaluate to *True* to make a transition $t1$ enabled. We have:

$$G(t1)\langle\ b'_{t1}\ \rangle\ =\ 8{>}2{=}\ True$$

In this case, the guard evaluates to *True*, but there are no tokens of data value *8*. So a transition *t1* is not enabled for the binding $\langle x = 8\rangle$.

Occurrence of a Step

Definition 3.6.5. When an enabled binding element (t, b) occurs, a multi-set of tokens is removed from the input place and a multi-set of tokens is added to the output place of the occurring transition. The multi-set of tokens removed from the input place p, when t occurs in b is given by $E(p,t)\langle b\rangle$, and the multi-set of tokens added to an output place p is given by $E(t,p)\langle b\rangle$, which means that the new marking M' is reached when an enabled binding element (t, b) occurs in a marking M which is given by:

$$\forall\ p \in P\ :\ M'(p)\ =\ (M(p)\ -\ E(p,t)\ \langle b\rangle)\ +\ E(t,p)\ \langle b\rangle$$

Definition 3.6.6. When a step Y is enabled in a marking M_1 it may occur, changing the marking M_1 to another marking M_2, defined by:

$$\forall p \in P{:}\ M_2(p){=}(M_1(p)\ -\ \textstyle\sum_{(t,b)\in Y}\ E(p,t) < b >){+}\sum_{(t,b)\in Y}\ E(t,p) < b >$$

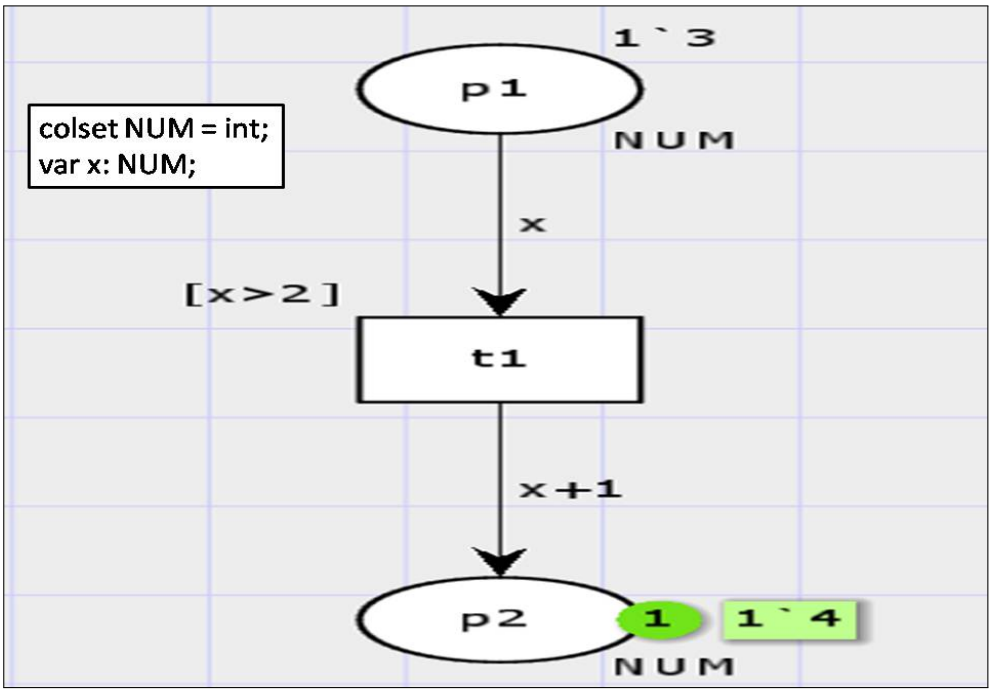

Figure 3.6: Example to Illustrate Occurrence of Transition

As the transition $t1$ is enabled, it can occur and change the marking as per Definition 3.6.6. A new marking $M'(p)$ for place p is given by:

$$M'(p) = (M(p) - E(p,t)\langle b \rangle) + E(t,p)\langle b \rangle$$

We have

$$M(p1) = 1`3$$

$$E(p1,t1)\langle b_{t1} \rangle = 1'3$$

$$E(t1,p2) = x + 1$$

$$E(t1,p2)\langle b_{t1} \rangle = 1'4$$

In Figure 3.5, transition $t1$ is enabled. The occurrence of the transition $t1$ removes tokens from place $p1$ and adds them to the output place $p2$ determined by the arc expression $x + 1$ as shown in Figure 3.6. So, x will be bound to 3 and with the arc expression $(x + 1)$ evaluation we will get one token of 4 at the output place $p2$.

To summarise, for a Coloured Petri net CPN=$(\sum, P, T, A, N, C, G, E, I)$:

1. A ***marking*** is a function M that maps each place $p \in P$ into a multi-set of token $M(p) \in C(p)_{MS}$

2. The ***initial marking*** M_0 is defined by $M_0(p) = I(p)$ for all $p \in P$.

3. The ***variables of a transition*** t are denoted $Var(t) \in V$ and consist of the variables appearing in the guard of t and in the arc expressions of arcs connected to t .

4. A ***binding*** of a transition t is a function b that maps each variable $v \in Var(t)$ into a value $b(v) \in Type[v]$. The set of all bindings for a transition t is denoted $B(t)$.

5. A ***binding element*** is a pair (t, b) such that $t \in T$ and $b \in B(t)$. The set of all binding elements $BE(t)$ for a transition t is defined by $BE(t) = \{(t, b) \mid b \in B(t)\}$. The set of all binding elements in a CPN model is denoted BE.

6. A ***step*** $Y \in BE_{MS}$ is a non-empty, finite multi-set of binding elements.

3.6.2.5 Example CPN Model

Lets consider one more example as shown in Figure 3.7. This is a CPN model showing a part of the medical infusion pump which allows the user to set the volume of the drug to be infused, set the duration of infusion and then confirm settings. The CPN model shown in Figure 3.7 has three places *setvolume, settime* and *confirmsettings* drawn as ellipses and three transitions *I_setvolume, I_settime* and *I_confirmsettings* drawn as rectangles. From the declarations (in the box at the upper right corner of Figure 3.7), it can be seen what the colour sets are. The three colour sets *timeset*, *volumeset* and *rate* are of type integer and colour set *state* is the record of these three colour sets. The declaration of the colour set *state* tells us that each token on places *setvolume*, *settime* and *confirmsettings* have a token colour of type record.

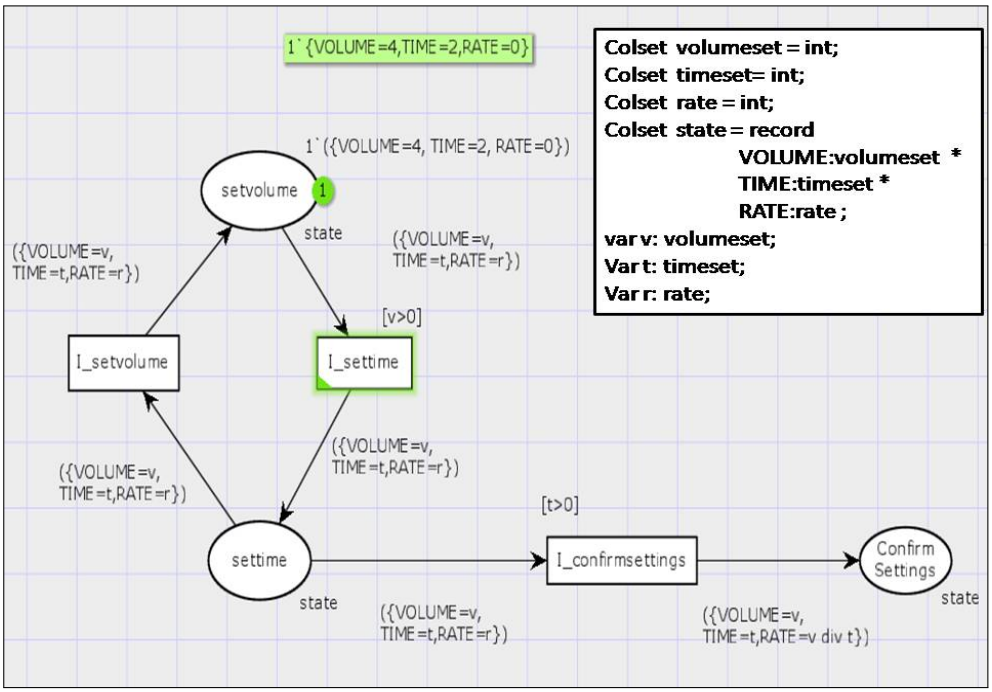

Figure 3.7: Another Example to Illustrate Enabling of Transition

In the Figure 3.7, there is one token of ($\{VOLUME = 4, TIME = 2, RATE = 0\}$) as an initial marking on place *setvolume*. It is clear from the Figure 3.7 that the initial marking M_0 is:

$$M_0 = \begin{cases} 1`(\{VOLUME = 4, TIME = 2, RATE = 0\}) & \text{if p=setvolume} \\ \emptyset & \text{if p=settime} \\ \emptyset & \text{if p=confirmsettings} \end{cases}$$

For a CPN model given in Figure 3.7 consider a binding element:

$$b_{I\_settime} = \langle\ v{=}4,\ t{=}2,\ r{=}0\ \rangle$$

The arc expression *E(setvolume,I_settime)*, from place *setvolume* to the transition *I_settime* is *{VOLUME=v, TIME=t, RATE=r}*. When the binding $b_{I\_settime}$ is applied to the arc expression *E(setvolume,I_settime)* we get:

$$E(setvolume,I\_settime)\ \langle\ b_{I\_settime}\ \rangle = 1`(\{VOLUME{=}4,\ TIME{=}2,\ RATE{=}0\})$$

As per Definition 3.6.4, the place must have enough tokens of the right form in order for a step to happen. We have:

$$E(setvolume,I\_settime)\ \langle\ b_{I\_settime}\ \rangle\ =\ 1`(\{VOLUME=4,\ TIME=2,\ RATE=0\}) \leq 1`(\{VOLUME=4,\ TIME=2,\ RATE=0\})\ =\ M(setvolume)$$

So the condition, $E(p,t)\langle b\rangle \leq M(p)$ for this case is satisfied.

Also there is a guard $[v > 0]$ on the transition *I_settime* which must evaluate to *True* to make transition *I_settime* enabled. We have:

$$G(I\_settime)\langle\ b_{I\_settime}\ \rangle\ =\ 4>0\ =\ True$$

As the guard evaluates to *True* and there are enough tokens on the input place, so a transition *I_settime* is enabled.

Thus we conclude that transition *I_settime* is enabled in the initial marking because the input place contains at least the token to which the corresponding arc expression evaluates and also the guard $(v > 0)$ on the transition is *True*.

When the transition *I_settime* occurs, it changes the marking M_0 to another marking M_1 and the token is removed from the *setvolume* place and added to the *settime* place as shown in Figure 3.8.

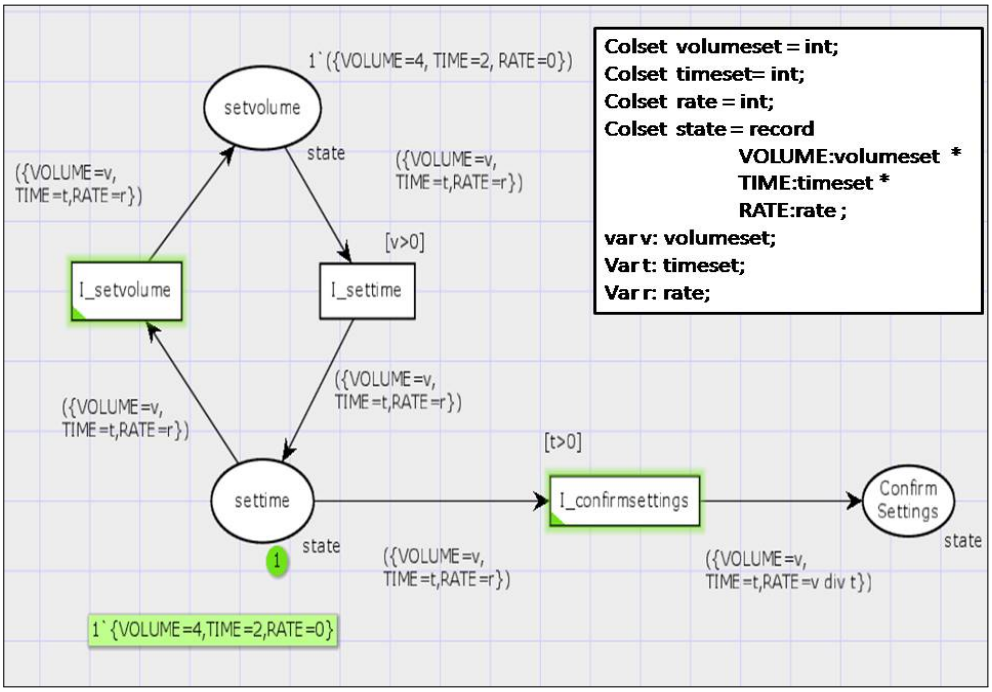

Figure 3.8: Another Example to Illustrate Occurrence of Transition

A new marking $M'(p)$ for place p is given by:

$$M'(p) = (M(p) - E(p,t)\langle b \rangle) + E(t,p)\langle b \rangle$$

We have

$$M(setvolume) = 1`(\{VOLUME = 4, TIME = 2, RATE = 0\})$$

$$E(setvolume, I\_settime)\langle b_{I\_settime} \rangle = 1`(\{VOLUME = 4, TIME = 2, RATE = 0\})$$

$$E(I\_settime, settime) = \{VOLUME = v, TIME = t, RATE = r\}$$

$$E(I\_settime, settime)\langle b_{I\_settime} \rangle = 1`(\{VOLUME = 4, TIME = 2, RATE = 0\})$$

So the new marking M_1 is:

$$M_1 = \begin{cases} \emptyset & \text{if p=setvolume} \\ 1`(\{VOLUME = 4, TIME = 2, RATE = 0\}) & \text{if p=settime} \\ \emptyset & \text{if p=confirmsettings} \end{cases}$$

3.6.2.6 Heirarchical Coloured Petri Nets

Hierarchical Coloured Petri Nets allow models to be divided into modules. They allow the model to be split into manageable parts with well-defined interfaces. It is really useful when it is impractical to draw a CPN model of a large system as a single net. Moreover, there is always a need of abstractions that make it possible to concentrate on only a few details at a time by hiding some information. A Coloured Petri net model can be organized into several pages. There are two ways to interconnect these several pages: *substitution transitions* and *fusion places* [120].

Substitution Transitions

Substitution transitions allow models to be built in a top-down manner. If a transition is a substitution transition, it has a sub-page related to it, which

includes a more detailed description of the model. For example, in figure 3.9, the transitions *setvolumeBehaviours* and *settimeBehaviours* are substitution transitions. It is represented by a double rectangle.

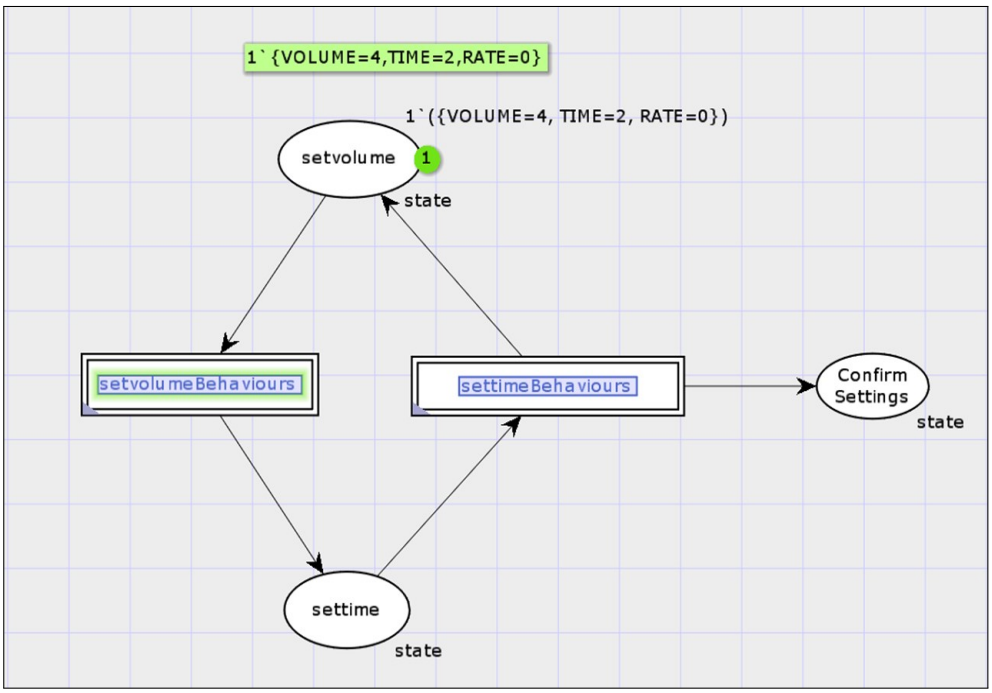

Figure 3.9: Substitution Transitions

The subpage of the transition *setvolumeBehaviours* is shown in Figure 3.10.

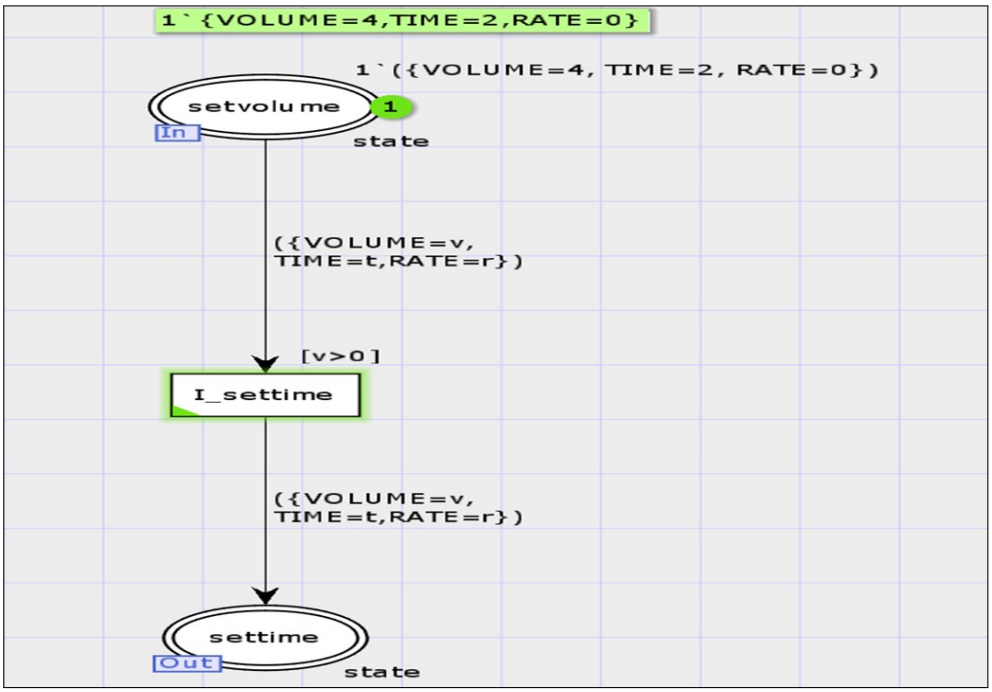

Figure 3.10: Subpage of transition *setvolumeBehaviours* of Figure 3.9

The CPN sub-pages interface through port and socket places. Sub-pages have port places, which allow them to receive, deliver or receive and deliver tokens from the higher level pages. For example, in Figure 3.10, the place *setvolume* is an input port and it is marked with an *In-tag* and the place *settime* is an output port and is marked with an *Out-tag*.

The places connected to the substitution transition are called *sockets*. In Figure 3.9, the places *setvolume*, *settime* and *confirmsettings* connected to the substitution transitions *setvolumeBehaviours* and *settimeBehaviours* are called sockets. Sockets are related to port places on the corresponding sub-pages by providing *port assignments*.

Fusion Places

Fusion places include a set of places, which are functionally identical, so they have the same marking. A set of fusion places is a fusion set. Members of a fusion set can belong to a single page or be distributed across different pages. Also, there are several types of fusion sets [120]; however in this thesis only the global fusion sets are considered. They can have members from different

pages. A global fusion place is indicated by a FusionSet-tag (shown in blue rectangles) next to the fusion place. For example, in Figures 3.11 and 3.12, the places with FusionSet-Tags *setvolume*, *settime* and *confirmsettings* are the fusion places. A place with FusionSet-tag *sevolume* in Figures 3.11 and 3.12 belongs to the same group. That is why they have the same marking. When a token is added/removed at one of these places, an identical token will be added/removed at all the other places that belong to *setvolume* FusionSet-Tag.

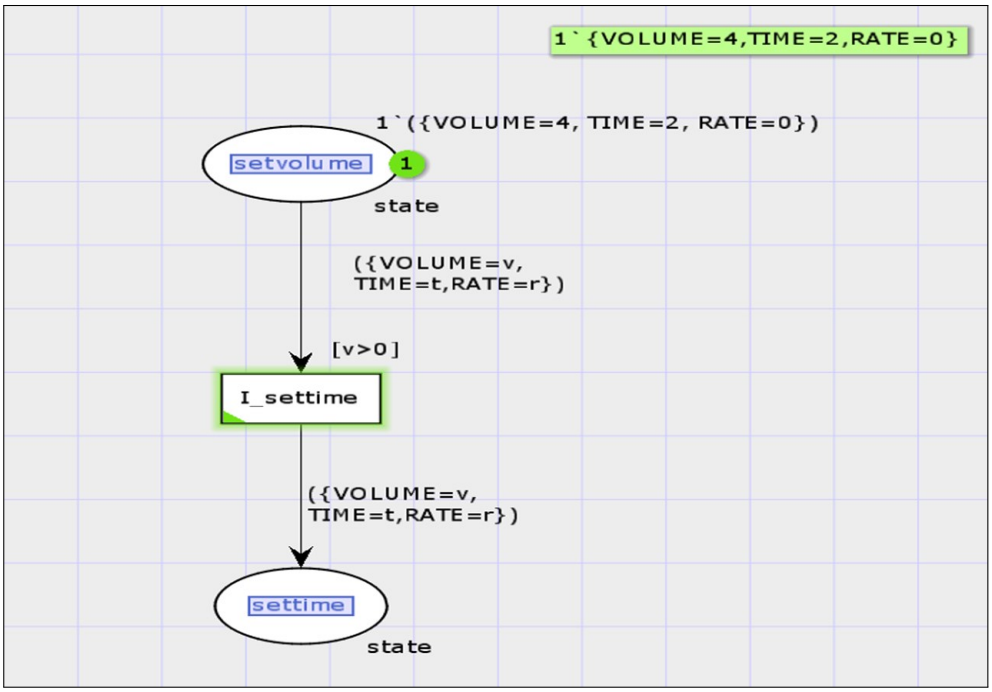

Figure 3.11: Fusion Places page 1

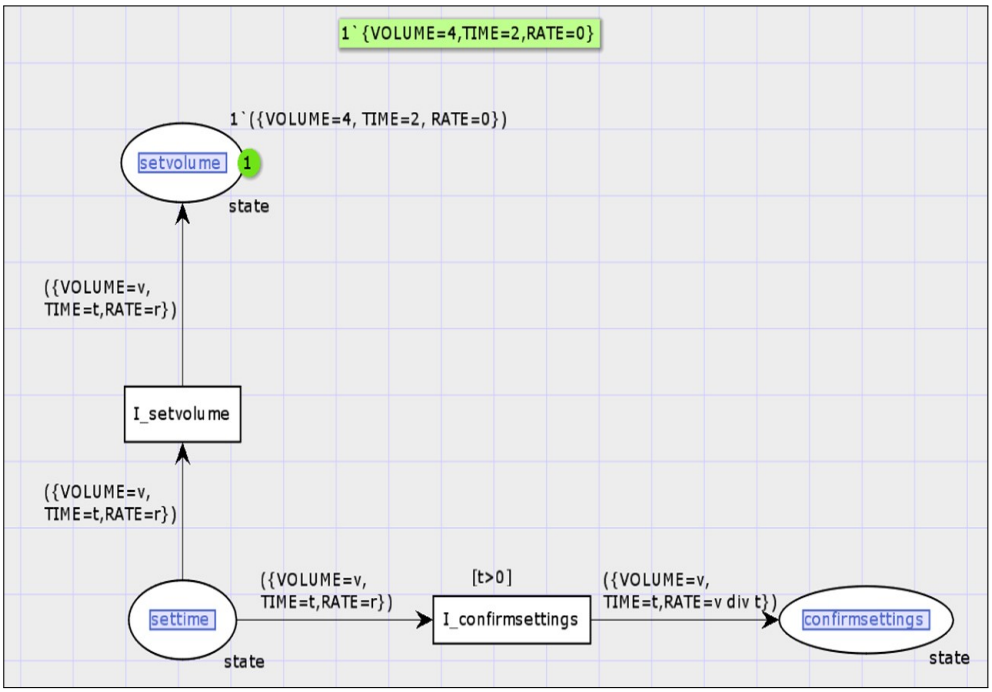

Figure 3.12: Fusion Places page 2

3.6.3 Analysis of Coloured Petri Nets

There are four ways that a Coloured Petri Net model is analyzed:

- The first method is *interactive simulation*. It allows the user to understand and to debug the model. A user can execute a CPN model and investigate the behaviour of the model in detail. This method is used in this thesis.

- The second method is *automatic simulation*. This is quite similar to executing a program. It executes hundreds of transitions at a very fast speed. As in this work we are concentrating on safety-critical interactive devices, we are not using this method because it is not suitable for this context.

- The third method uses a *state space graph* which is also known as a reachability graph or an occurrence graph. The state space includes all possible markings that can be reached from the initial marking. It is represented by a directed graph where the nodes represent the markings

and the arcs the occurring binding elements. We are using this method to analyze our models.

- The fourth method is *place invariants*. This method is very similar to the use of invariants in ordinary program verification. The user constructs a set of equations which are proved to be satisfied for all reachable system states. We are not using this method in our work.

In this thesis, we will be using interactive simulation to investigate the behaviour of the model and the state space graph method to investigate the properties.

3.6.3.1 State Space Graph

The state space graph is a set of nodes that are connected by arcs. Compared with other analysis techniques such as invariant analysis, state space methods have several advantages [121]. Firstly, state spaces can be automatically constructed, which provides computer-aided analysis and verification of the behaviour of the modelled system. Furthermore, the state space tool is fully integrated in CPN Tool, which is the software used in this thesis for constructing the CPN models. Secondly, the state space includes a lot of information about the behaviour of the system, which can answer a large set of analysis and verification questions. Thirdly, the state space methods can be used for debugging and testing the system.

The major disadvantage of state space methods is the state explosion problem. The problem with very large state spaces is that they cannot be generated with limited computational resources (e.g. memory) and if they can, they may be difficult to analyze. Researchers have been working hard to try to find ways of solving or alleviating the problem. Thus, several state space reduction techniques have been developed [28]. But in this work there was no need to use any such technique as we reduce the maximum value of some variables to keep the state space small for analysis but does not mean we miss checking things like upper bounds and lower bounds of duration.

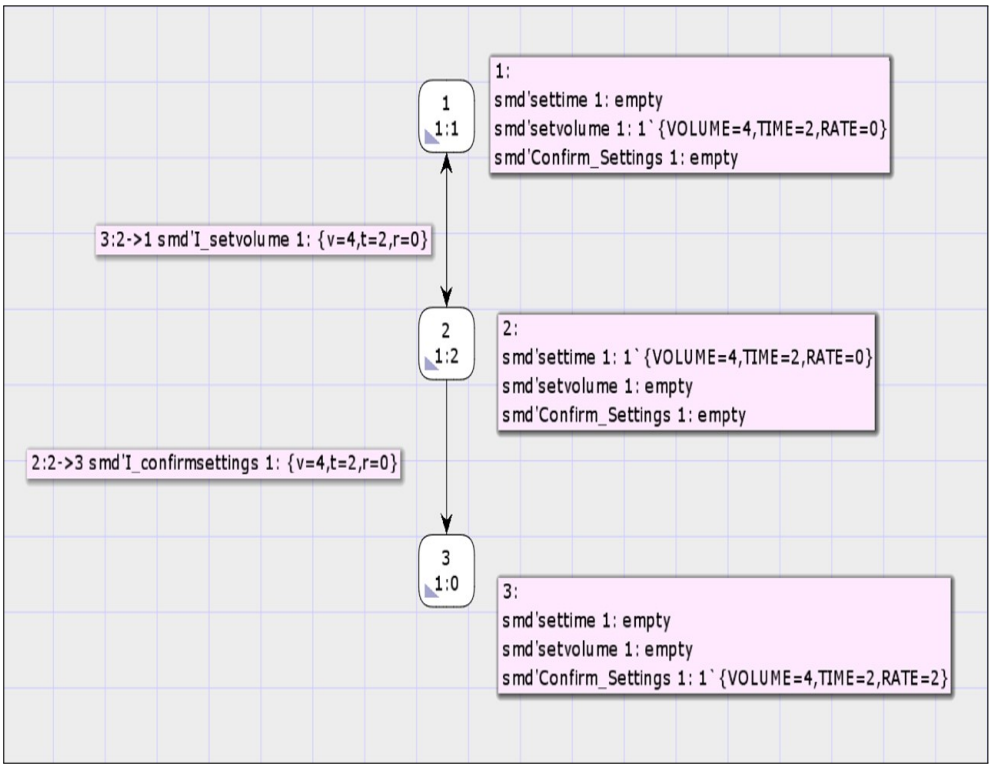

Figure 3.13: State Space Graph of the CPN model given in Figure 3.7

The great advantages of the state space methods listed previously make it a good choice to analyze models of safety-critical interactive systems. We will consider the same example model as shown in Figure 3.7 to explain the use of a state space graph. The state space graph of the model is given in Figure 3.13.

There are three reachable markings or nodes in Figure 3.13, represented by rounded rectangular boxes. Each marking has an identification number located at the top. Also, in each marking box, there are two numbers separated by a colon (":"), which represent the number of input and output arcs, respectively. The node at the top is the initial marking and has the identification number 1.

An arc box is shown in the solid box next to the arc in Figure 3.13 which includes the identification number of the arc and some marking information (located next to the number). The marking information, represented by m1->m2, indicates that the marking m2 can be reached from marking m1. For example, marking 1 can be reached from marking 2 and it includes the tran-

sition *I_setvolume*.

The marking of each place can also be shown in the boxes next to the nodes. For example, in the initial marking (i.e. node 1), the place *setvolume* has one token with the value : $1`\{VOLUME = 4, TIME = 2, RATE = 0\}$. The rest of the places have no tokens at that point.

3.6.3.2 Strongly Connected Component Graph

A strongly connected component (SCC) of the state space is a maximal subgraph, whose nodes are mutually reachable from each other [28]. A SCC graph has a node for each SCC and arcs that connect each SCC node with other SCCs. A SCC without incoming arcs is called the initial SCC, and a SCC without outgoing arcs is called a terminal SCC. Each node in the state space belongs only to one SCC, so the SCC graph will be smaller than or equal to the corresponding state space [122].

Figure 3.14 shows the SCC graph for the example shown in Figure 3.7. The full state space graph as shown in Figure 3.13 has three nodes and three arcs, while the SCC graph has two nodes and one arc. Node number ~1 is the initial marking and ~2 is the terminal SCC node. Each SCC node has information about the number of nodes and arcs, which have been grouped in that SCC node. The numbers of arcs, which connect each SCC node to others, are indicated next to the arcs. For example, node ~1 (i.e. initial SCC) has two nodes and two arcs and the node has no input arcs and one output arc going to the SCC node ~2.

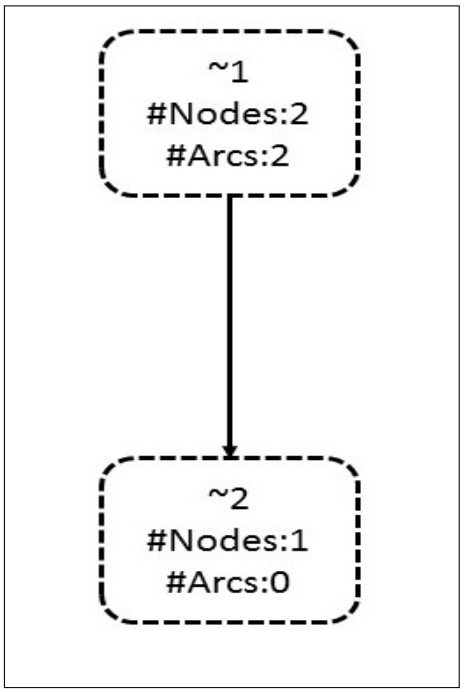

Figure 3.14: SCC Graph

Each node of an SCC graph is a strongly connected component while each arc of an SCC graph is an ordinary state space arc [119]. The SCC reduces the state space nodes which are mutually reachable from each other into a single node. If no nodes can be reduced, the SCC should have the same number of nodes as the state space has. We have an SCC arc for each state space arc that starts in one SCC and ends in another. As per [123], a state space graph is isomorphic to its SCC graph if there are the same number of nodes and arcs in both the graphs. If there are no cycles in the state space graph and all the SCCs are trivial, i.e. all SCCs contain single node and no arcs, then the state space and its SCC graph will have same number of nodes and arcs, hence isomorphic.

3.6.3.3 State Space Report

The state space report can also be generated using the CPN Tool [37]. The state space report for the example given in Figure 3.7 is given in APPENDIX A.

The state space report includes statistical information about the size of the

state space and SCC graph. It also has information about integer upper and lower bounds and multi-set bounds on places, i.e boundedness properties. In addition, the report provides information about home and liveness properties such as home markings, dead markings, dead transitions and live transitions.

3.6.4 Behavioural Properties Of Coloured Petri Nets

There are certain properties that describe the expected behaviour of the CPN model. We will describe these dynamic properties of Coloured Petri Nets in an informal way. More details and formal definitions of these properties can be found in [28].

3.6.4.1 Reachability

The notation M_n represents the marking of node number n. M_n is reachable from M_1 if there exists an occurrence sequence [28] from marking M_1 to M_n. For example, in Figure 3.13, M_2 is reachable from M_1.

3.6.4.2 Home Markings

A marking that can always be reached from all other reachable markings is referred to as a home marking. For example, in Figure 3.13, M_3 is a home marking.

3.6.4.3 Dead Markings

A dead marking is a marking with no enabled binding elements. For example, in Figure 3.13, M_3 is a dead marking as there is no enabled transition for that marking.

3.6.4.4 Dead Transitions

A transition that is not enabled at all in any reachable marking is called as a dead transition. There is no dead transition for the CPN model given in Figure 3.7.

3.6.4.5 Live Transitions

A transition that can always occur again is referred to as a live transition. There is no live transition for the CPN model given in Figure 3.7.

3.7 Summary

In this chapter we have introduced four formalisms: *presentation model, presentation interaction model, Z and Coloured Petri Nets.* We have discussed and illustrated the definitions and basic concepts of all the four formalisms by means of simple examples. Presentation models give the description of the behaviour of the interactive elements of the user interface, presentation interaction models give the navigational possibilities and Z specifies the underlying functionality of a system. We have also given a brief introduction to Coloured Petri Nets. In our work we are going to express presentation model, presentation interaction model and Z in Coloured Petri Nets so that we can have one single model that have all the parts of a system (user interface, interaction and functional).

Chapter 4

Modelling user interface and interaction using Coloured Petri Nets

4.1 Introduction

As we have seen in Chapter 3, presentation models (PM) [42] give a model of the user interface of a system. Each state of the user interfaces is described separately in a component presentation model as a set of triples consisting of a widget, its category and behaviours associated with that widget. The presentation interaction model (PIM) [42] is like a state transition diagram, where component presentation models are abstracted into states and transitions are labelled with I-behaviours from those component presentation models.

In this chapter we show how to express presentation models in Coloured Petri Nets [29]. Then we present a technique to express presentation interaction models using Coloured Petri Nets too. The presentation interaction models created using Coloured Petri Nets shows the component presentation models as a current marking on the current state. This way we have both presentation models and presentation interaction models in a single model which is useful for investigating the behaviour of a model as we don't have to look at two separate models. We can see by simulating through the CPN model

of the PIM if the behaviours specified in the presentation model are actually available to users or not. We are going to use an example of a simplified infusion pump to explain the process of expressing presentation models and presentation interaction models in Coloured Petri Nets.

The achieved model is then analyzed to see if it is behaving as expected. Certain general properties that can be applied to any interactive system are investigated using the state space method of Coloured Petri Nets.

4.2 Expressing Presentation Model in Coloured Petri Nets

In this section we will see how we can express presentation models in Coloured Petri Nets. Presentation models consist of declarations and definitions [24]. We will first look into the declaration part of the presentation model which introduces the three sets of identifiers which can be used within the definitions. The three set of identifiers are called : *WidgetName, Category and Behaviour*. These three sets of identifiers can be written in CPN-ML as:

- **WidgetName:** A set of identifiers that are the names of the widgets on a device. It is written in CPN as:

$$colset\ WidgetName = with\ wid_1 \mid wid_2 \mid ... \mid wid_{n_w};$$

- **Category:** A set of identifiers that are the names of the category of widgets. It is written in CPN as:

$$colset\ Category = with\ cid_1 \mid cid_2 \mid ... \mid cid_{n_c};$$

- **Behaviour:** A set of identifiers that shows what behaviour a widget has associated with it. It is written in CPN as:

$$colset\ Behaviour = with\ bid_1 \mid bid_2 \mid ... \mid bid_{n_b};$$

As we need a list of behaviours for widget description (explained below), a new colset *Behaviours* is declared as:

$$colset\ Behaviours = list\ Behaviour;$$

which is a list of above declared colset *Behaviour*.

We will consider the same example of a simple infusion pump from the last chapter. Table 4.1 shows how we can express the declaration part of the presentation model of the simple infusion pump in Coloured Petri Nets. The first part of the table shows the original way of writing the declaration part of the presentation model and the second part shows its translation in Coloured Petri Nets.

Presentation Model	
WidgetName	Display NoButton YesButton InfoButton MinusButton PlusButton OnOffButton
Category	ActCtrl Responder
Behaviour	S_displaystartmessage S_displaybatterylife S_decreasevolume S_increasevolume S_setvolume S_setrate S_decreasetime S_increasetime S_infusing S_settime I_confirmrate I_settime I_init I_info I_infuse I_setvolume S_displayinfusingmessage
CPN-ML	
colset WidgetName =	with Display | NoButton | YesButton | InfoButton | MinusButton | PlusButton | OnOffButton;
colset Category =	with ActCtrl | Responder;

```
colset  Behaviour  =   with   S_displaystartmessage |
                       S_displaybatterylife  |
                       S_decreasevolume |
                       S_increasevolume  |  S_Setvolume  |
                       S_setrate  |  S_decreasetime |
                       S_increasetime  | S_infusing  |
                       S_settime  |  I_confirmrate  |
                       I_settime  |  I_init  |
                       I_info  |  I_infuse  |  I_setvolume  |
                       S_displayinfusingmessage;
colset  Behaviours  =  list   Behaviour;
```

Table 4.1: Declaration part of the presentation model of the simple infusion pump in CPN

WidgetName is an enumeration type that represents the names of the seven widgets *(Display, NoButton, YesButton, InfoButton, MinusButton, PlusButton* and *OnOffButton*) of a simple infusion pump. Each of these widgets are categorized using the widget categorization hierarchy given in [108]. In this case the widget *Display* is of category *Responder* and all the six buttons are under *ActionControl* category. These two categories are represented by an enumerated colour set *Category* with two identifiers: *Responder* and *ActCtrl* (short form for ActionControl). There is a total of seventeen behaviours associated with these widgets, represented by an enumerated colour set *Behaviour*. As one widget can have more than one behaviour, we need a list colour set *Behaviours*.

Now we look at the definition part of the presentation model [24]. Our main concern is the widget description. A widget description is a tuple consisting of the widget name, the category and the list of behaviours associated with the widget. A widget description can be written in CPN as:

$$colset\ widgetdescr = product\ WidgetName\ *\ Category\ *\ Behaviours;$$

Each state of the system is described in a separate component presentation model (pmodel) by the means of widget descriptions. This can be written in CPN as:

$$colset\ pmodel = list\ widgetdescr;$$

To define the component presentation model we will use a value declaration feature of CPN-ML. A value declaration binds a value to an identifier. The syntax for doing this is:

$$val\ name = expression;$$

where *name* is the name of the component presentation model. Consider the *init* component presentation model from the same example:

```
init is  (Display, Responder, (S_displaystartmessage))
         (NoButton, ActCtrl, ())
         (YesButton, ActCtrl, ())
         (InfoButton, ActCtrl, (I_info))
         (MinusButton, ActCtrl, ())
         (PlusButton, ActCtrl, ())
         (OnOffButton, ActCtrl, ())
```

Table 4.2: init component presentation model

This can written in CPN-ML as:

```
val   init =  [(Display, Responder, [S_displaystartmessage]),
               (NoButton, ActCtrl, [ ]),
               (YesButton, ActCtrl, [ ]),
               (InfoButton, ActCtrl, [I_info ]),
               (MinusButton, ActCtrl, [ ]),
               (PlusButton, ActCtrl, [ ]),
               (OnOffButton, ActCtrl, [  ])];
```

Table 4.3: init component presentation model in CPN

This is how the component presentation models are expressed in Coloured Petri Nets. The collection of component presentation models forms the full interface presentation model. With a presentation model, it is easy to understand what the states of the device are and what widgets are available to the user in every state and what the behaviour of those widgets are. But to understand the navigational possibilities, it is always better to have some graphical representation. Another model, i.e., a presentation and interaction model (PIM) is used for this purpose. In this thesis we will represent PIMs using Coloured Petri Nets as explained in the next section.

4.3 Expressing Presentation Interaction Models in Coloured Petri Nets

In this section we will see how we can represent a PIM in Coloured Petri Nets using the simplified infusion pump example. It is quite straightforward to express a PIM in CPN for devices with a small number of states. For example, the PIM for the simple infusion pump given in Figure 3.2 has just six component presentation models/states. These states are represented by places in CPN and the transitions are represented by the CPN transitions. The start state is represented by the initial marking on the places. The original PIM [24] does not show the component presentation models. But now we are able to have a representation of the component presentation model as a part of the representation of the PIM in CPN in the form of tokens on the places, as we shall see.

The non-hierarchical CPN model of the PIM is fine if the states are few. But for complex devices, the number of states increases and the model becomes large and complex. So we will use hierarchical CPNs to model PIMs, where each page is interconnected by fusion places.

4.3.1 Example Net

Figure 4.1 shows an overview of a CPN model of the simple infusion pump.

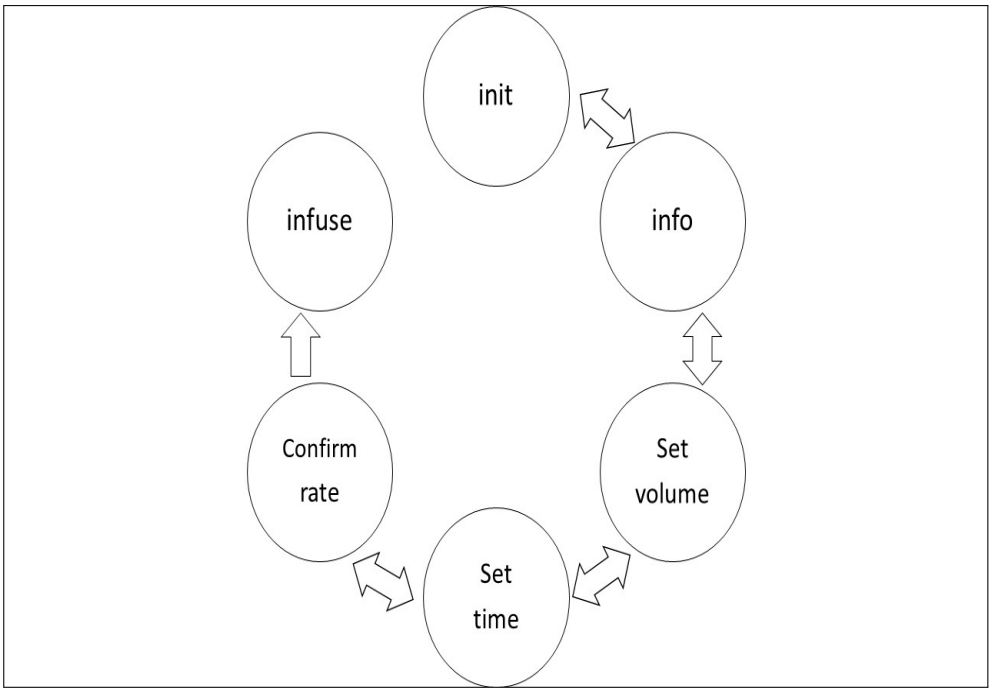

Figure 4.1: An overview of a CPN model of the simple infusion pump

There are six component presentation models for the simple infusion pump as shown in Table 4.1, therefore, the CPN model of a PIM of the simple medical device will have six pages which are interconnected by fusion places. Each page represents the individual component presentation model showing the navigational possibilities. Figure 4.2 - 4.7 shows the six pages.

The presentation model declaration for the simple infusion pump is as shown in the Table 4.4.

```
(* ============ Presentation Model Declaration ============ *)
colset  WidgetName  =  with   Display |  NoButton |  YesButton |
                       InfoButton |  MinusButton |  PlusButton |
                       OnOffButton;
colset  Category  =    with   ActCtrl  |  Responder;
colset  Behaviour  =   with   S_displaystartmessage |
                       S_displaybatterylife  |  S_decreasevolume |
```

```
                        S_increasevolume  |  S_setvolume  |
                        S_setrate  |  S_decreasetime |
                        S_increasetime  |  S_infusing  |
                        S_settime  |  I_confirmrate  |
                        I_settime  |  I_init  |
                        I_info  |  I_infuse  |  I_setvolume  |
                        S_displayinfusingmessage;
colset  Behaviours  =   list   Behaviour;
```

Table 4.4: Presentation Model Declaration for Simple Infusion Pump in CPN

The presentation model definition of the simple infusion pump is given in Table 4.5. The constants *init*, *info*, *setvolume*, *settime*, *confirmrate* and *infusing* represents the component presentation models of the simple infusion pump prototype.

```
(* ============= Presentation Model Definition ============= *)


colset  widgetdescr =  product WidgetName*Category*Behaviours;
colset  pmodel =       list widgetdescr;
val init =             [(Display, Responder,
                       [S_displaystartmessage]),
                       (NoButton, ActCtrl, [ ]),
                       (YesButton, ActCtrl, [ ]),
                       (InfoButton, ActCtrl, [I_info]),
                       (MinusButton, ActCtrl, [ ]),
                       (PlusButton, ActCtrl, [ ]),
                       (OnOffButton, ActCtrl, [ ])];

val info =             [(Display, Responder,
                       [S_displaybatterylife]),
```

```
                        (NoButton, ActCtrl, [I_init]),
                        (YesButton, ActCtrl, [I_setvolume]),
                        (InfoButton, ActCtrl, [ ]),
                        (MinusButton, ActCtrl, [ ]),
                        (PlusButton, ActCtrl, [ ]),
                        (OnOffButton, ActCtrl, [ ])];

val setvolume =         [(Display, Responder, [S_decreasevolume,
                        S_increasevolume]),
                        (NoButton, ActCtrl, [I_info]),
                        (YesButton, ActCtrl, [I_settime,
                        S_setvolume]),
                        (InfoButton, ActCtrl, [ ]),
                        (MinusButton, ActCtrl, [S_decreasevolume]),
                        (PlusButton, ActCtrl, [S_increasevolume]),
                        (OnOffButton, ActCtrl, [ ])];

val settime =           [(Display, Responder, [S_decreasetime,
                        S_increasetime]),
                        (NoButton, ActCtrl, [I_setvolume]),
                        (YesButton, ActCtrl, [S_settime,
                        I_confirmrate]),
                        (InfoButton, ActCtrl, [ ]),
                        (MinusButton, ActCtrl, [S_decreasetime]),
                        (PlusButton, ActCtrl, [S_increasetime]),
                        (OnOffButton, ActCtrl, [ ])];

val confirmrate =       [(Display, Responder, [S_setrate]),
                        (NoButton, ActCtrl, [I_settime]),
                        (YesButton, ActCtrl, [I_infuse]),
                        (InfoButton, ActCtrl, [ ]),
```

```
                    (MinusButton, ActCtrl, [ ]),
                    (PlusButton, ActCtrl, [ ]),
                    (OnOffButton, ActCtrl, [ ])];

val infuse =        [(Display, Responder,
                    [S_displayinfusingmsg]),
                    (NoButton, ActCtrl, [ ]),
                    (YesButton, ActCtrl, [ ]),
                    (InfoButton, ActCtrl, [ ]),
                    (MinusButton, ActCtrl, [ ]),
                    (PlusButton, ActCtrl, [ ]),
                    (OnOffButton, ActCtrl, [ ])];
```

Table 4.5: Presentation Model Definitions for Simple Infusion Pump in CPN

Let us take a closer look at all the pages. The first page is the *init* page as shown in Figure 4.2. The two places: *init* and *info* represent the states of the system. Note that all the places in the model are fusion places and belong to the corresponding fusion sets as shown by a blue rectangle at the lower left corner of each place. Each place has an associated *colour set* which determines the type of data the place may contain. It is written at the bottom right of each place. The colour sets are shown in Tables 4.4 and 4.5. In this case each place belongs to just one colour set, *pmodel*, which is a list of colour set *widgetdescr* (a product colour set comprised of *WidgetName*, *Category* and *Behaviours*).

In Figure 4.2, the place *init* has $1'init$ as its initial marking which means that the place has one token with the value *init*. Initially, the remaining places do not contain any tokens. In a model, a marking of places is shown in a green box. For example, in Figure 4.2, the current marking on the place *init* (green box) shows the definition of the *init* component presentation model as given in Table 4.5 which gives information about the available widgets in that state

and what behaviours are attached to those widgets in that state. The marking shows that there is only one I-behaviour: *I_info*, so there is just one transition namely *I_info* on this page. This means that from the *init* state, the user can go to the *info* state by firing the transition *I_info*. As we are modelling PIM in CPN which gives meaning to the I-behaviours, so the CPN model will contain the transitions which represent I-behaviours.

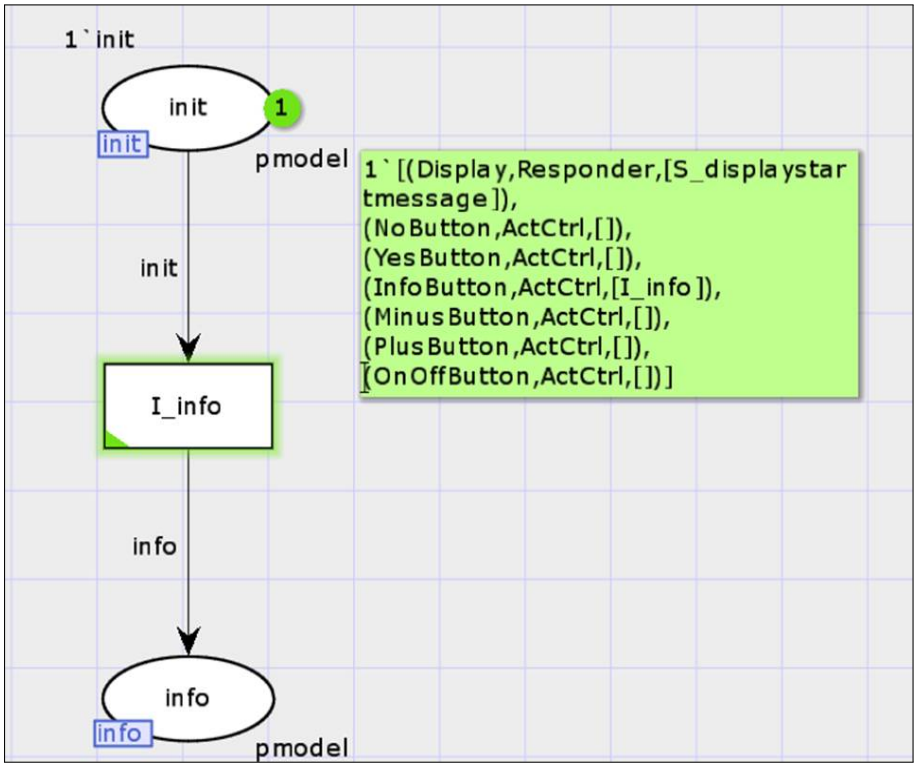

Figure 4.2: init page of simple infusion pump

All the arc expressions carry constants in this model, there are no variables. Therefore, a binding will always be empty. So, all arc expressions and guards are evaluated against the empty binding. This makes the models very simple to understand, and is a consequence of the fact that tokens (at the moment) contain no variables either. We shall see that when we come to model Z operation schemas in our CPN model that variables come into play in an important way.

Let us see how the transition *I_info* is enabled in the initial marking in Figure 4.2. The marking of the place *init*, $M(init)$, is $1'init$. The arc

expression, $E(init, I\_info)$, from place *init* to the transition *I_info* is *init*. As there are no variables, there are only empty bindings. As per Definition 3.6.3, the place must have enough tokens of the right form in order for a step to happen (recall that an expression like $1'init$ means we have one token of the form *init*, and that the occurrence of just *init* on the arc is shorthand for a single token, i.e. $1'init$). We have:

$$E(init, I\_Info)\langle\rangle = 1'init\langle\rangle = 1'init \leq 1'init = M(init)$$

So the condition, $E(p,t)\langle b\rangle \leq M(p)$ for this case is satisfied. Also, a missing guard is considered as a shorthand for the guard expression which always evaluates to True. Both these conditions mean the transition *I_info* is enabled in the initial marking.

As the transition *I_info* is enabled, it can occur and change the marking as per Definition 3.6.4. A new marking $M'(p)$ for place p is given by:

$$M'(p) = (M(p) - E(p,t)\langle b\rangle) + E(t,p)\langle b\rangle$$

We have

$$M(init) = 1`init$$

$$E(init, I\_info)\langle\rangle = 1`init$$

$$E(I\_info, info)\langle\rangle = 1`info$$

So the token *1'init* is removed from the input place *init* and a token *1'info* is added to the output place *info*. Figure 4.3 shows the change of marking when the transition *I_info* has occurred and we have moved from the *init* page to the *info* page.

Note that we can see the component presentation model details (the green box shows the value of the token in a place) which makes following the simulation of a system very straightforward, and is a feature of the CPN version of the PM/PIM model: all aspects of what was before spread across two models are now in one place.

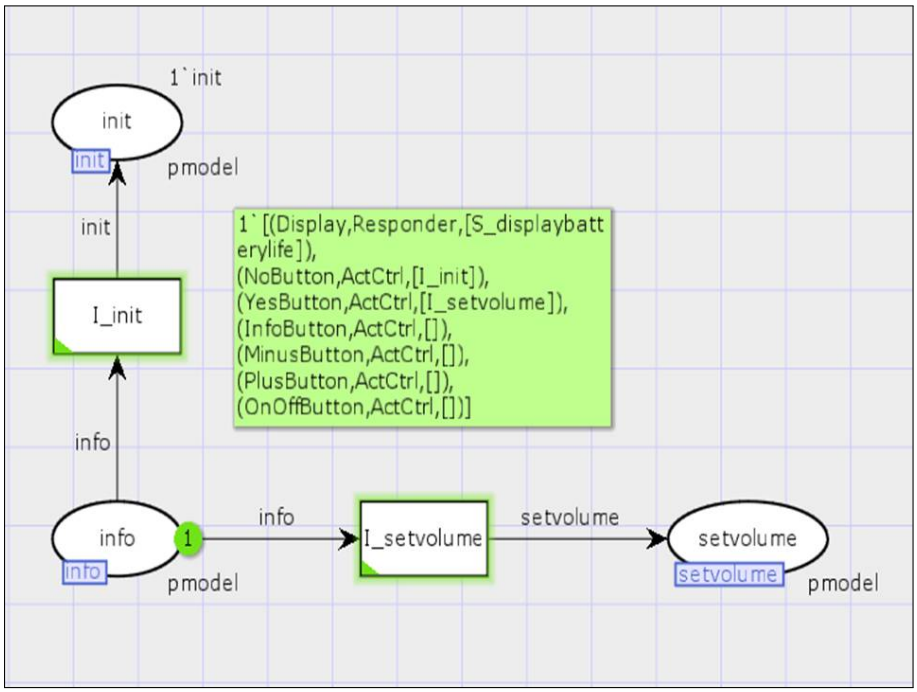

Figure 4.3: info page of simple infusion pump

Now we look at the *info* page as shown in Figure 4.3. This page shows the navigational possibilities from the *info* state. The *info* component presentation model has two I-behaviours: *I_init* and *I_setvolume*, i.e from the *info* state we can go to either the *init* state or the *setvolume* state. Therefore, the model in Figure 4.3 has three places: *init*, *info* and *setvolume* and two transitions *I_init* and *I_setvolume*. All the places will have type *pmodel* and the corresponding constants are attached to the arc expressions.

Figure 4.4 shows the structure of the *setvolume* page. This state allows the user to set the volume to be infused. There are three places in the model: *setvolume*, *info* and *settime*. A user can go to the next state i.e., *settime* by firing the transition *I_settime* or can go to the previous state *info* by firing the *I_info* transition. If a user is in the *setvolume* state, then the marking on the place *setvolume* will have a token that will give information about the available widgets and its associated behaviour. Occurrences of transitions will update the corresponding markings on the places.

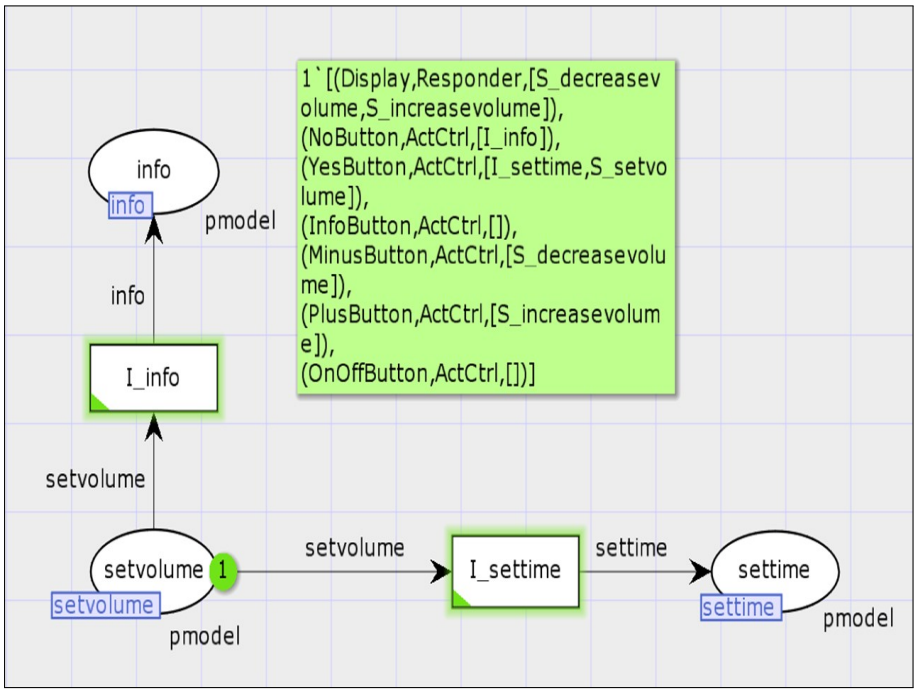

Figure 4.4: setvolume page of simple infusion pump

Figures 4.5-4.7 shows the *settime* page, the *confirmrate* page and the *infuse* page which will have the places and transitions corresponding to their component presentation models given in table 4.5.

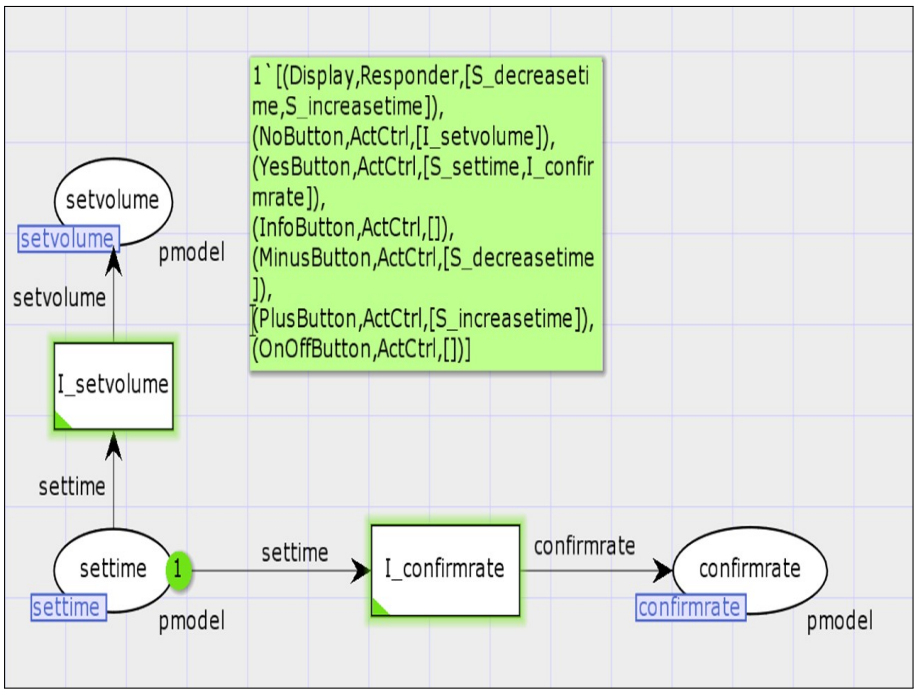

Figure 4.5: settime page of simple infusion pump

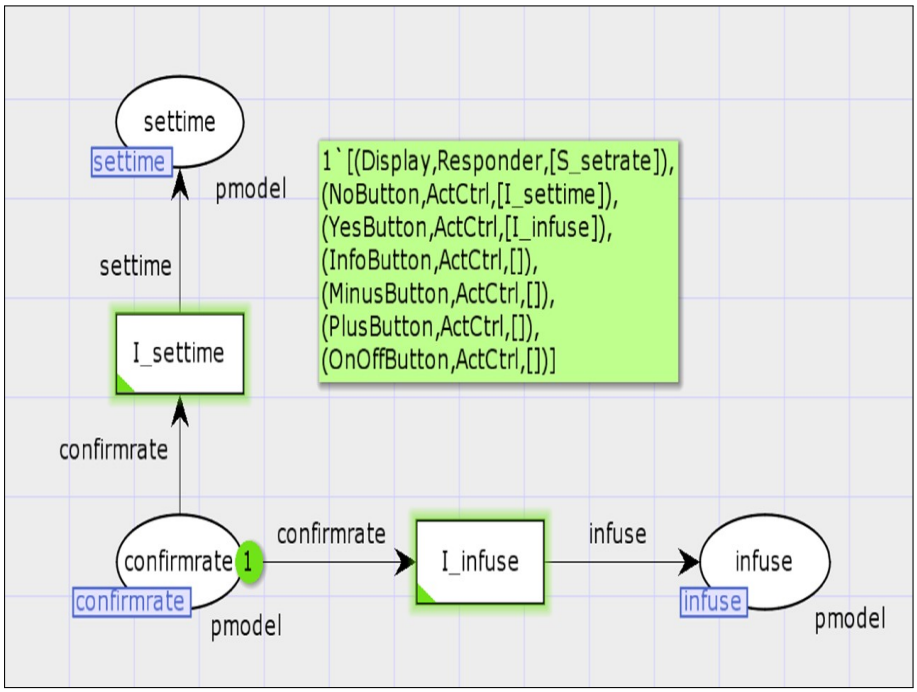

Figure 4.6: confirmrate page of simple infusion pump

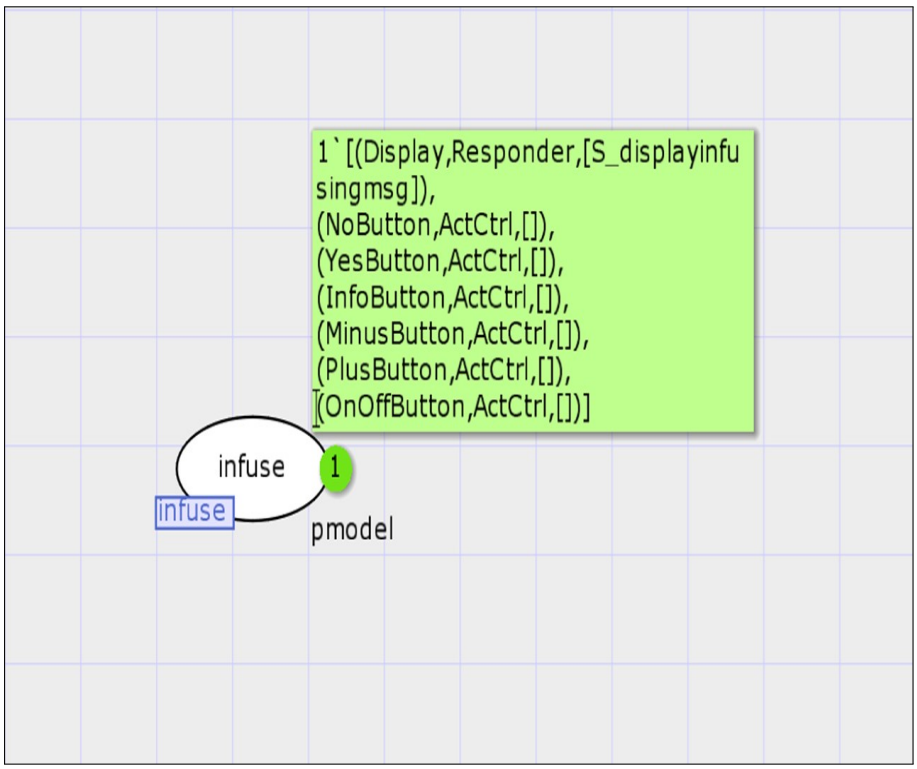

Figure 4.7: infuse page of simple infusion pump

Now we will summarize the rules for creating a PIM using hierarchical

Coloured Petri Nets.

1. The number of pages in a CPN model of a PIM is the same as the number of component presentation models in the PIM. For example, a PIM of the simple infusion pump shown in Figure 4.1 has six component presentation models, so there are six pages in the CPN model of this device. These pages represent an individual component presentation model.

2. The names of the places are exactly the names of the component presentation models. For the simple infusion pump, there are six places: *init, info, setvolume, settime, confirmrate and infuse*. These places have fusion tags (the blue boxes) named the same as the name of the corresponding component presentation models.

   As every page represents an individual component presentation model, so every page will have at least one place which represents that component presentation model. The page will contain additional places depending upon the I-behaviours present in that component presentation model. For example, the component presentation model *init* has one I-behaviour, which means that from the place *init* a user can go to one other state *info*. The page *info* will have two places: *init* and *info* where *init* is the current state and *info* is the state a user can go to from *init*.

3. Every place in the model will be of one type, i.e. *pmodel*.

4. The transitions give formal meaning to the I-behaviours in the presentation model. The names of the transitions that represent I-behaviours will be the same as the name of the respective I-behaviours in the presentation model. The number of transitions on each page depends upon the number of I-behaviours in that corresponding component presentation model. For example, the component presentation model *init* has one I-behaviour, so the page *init* contains one transition with the same name as the I-behaviour.

5. If a component presentation model (which has the same name as a place

in the Coloured Petri Net model) contains an I-behaviour, then that means we are moving from one place to another via that I-behaviour. So there exist two arcs: the first from that component presentation model (place) to the I-behaviour (transition) and the second from the same I-behaviour (transition) to another component presentation model (place). For example, there is an arc from place *init* to a transition *I_info* and another arc from transition *I_info* to the place *info*.

6. Each token on a place will have a token colour that belongs to type *pmodel*.

7. An arc expression must be of type *pmodel*, which is a set of constants whose values (as already defined) represent component presentation models. So an arc expression is one of the constants defined as representing (in fact, naming) a component presentation model from the original PIM we are modelling. An arc from a place will be labelled with the constant expression which is the same as the name of that place, and similarly an arc to a place will be labelled with an expression which is the name of that place.

4.3.2 Formal Definition of CPN for modelling User Interface and Interaction

In this section we formalize the definition of how we use CPN for representing the combination of presentation models and presentation interaction models (user interface and interaction).

In CPN, the finite set of component presentation models is represented by a finite set of places P, where the names of the places are exactly the names of the component presentation models. The finite set of I-behaviours is represented by a finite set of transitions T, where the names of the transitions are exactly the names of the I-behaviours. A start state is represented by the initial marking.

Definition 4.3.1. A non-hierarchical Coloured Petri Net for modelling a user interface and interaction is a tuple $(PM, K, P, T, I, \Sigma, A, C, G, E)$ such that:

(i) PM is a finite set of colour sets for representing presentation model declarations from PM, such that:
$PM = \{WidgetName,\ Category,\ Behaviour,\ Behaviours,\ widgetdescr,\ pmodel\}$, where

- $colset\ WidgetName = with\ wid_1 \mid wid_2 \mid ... \mid wid_{n_w}$;
  • where wid_s are the names of the widgets in the various component presentation models in the PM.
- $colset\ Category = with\ cid_1 \mid cid_2 \mid ... \mid cid_{n_c}$;
  • where cid_s are the names of the category of the widgets in the various component presentation models in the PM. These names are taken from the widget categorization hierarchy given in [108].
- $colset\ Behaviour = with\ bid_1 \mid bid_2 \mid ... \mid bid_{n_b}$;
  • where bid_s are the names of the S-behaviours and I-behaviours associated with the widgets in the various component presentation models in the PM.
- $colset\ Behaviours = list\ Behaviour$;
- $colset\ widgetdescr = product\ WidgetName * Category * Behaviours$;
- $colset\ pmodel = list\ widgetdescr$;

(ii) K is a finite set of constants that represents the component presentation models by their names and is such that $Type[K] = pmodel$.

(iii) P is a finite set of places, the same size as K, representing the component presentation models where the names of the places and names of the constants representing component presentation models are same. A constant k in the set K is associated with a place p in the set P with the same name.

(iv) T is a finite set of transitions representing the I-behaviours of the PIM.

(v) I is an initialization function that assigns an initial marking to each place. The initialization function $I : P \rightarrow EXPR$ assigns an initialization expression $I(p)$ to each place p such that: $I(p) = k \in K$, i.e., $I(p)$ can be a constant, k, representing a component presentation model. The initial marking, denoted by M_0 , is obtained by evaluating the initialization expression. $M_0 \neq \emptyset_{MS}$ represents the start state of the PIM.

(vi) Σ is a finite set of non-empty types, called colour sets with $PM \subseteq \Sigma$.

(vii) A is a finite set of arcs as given in definition 3.6.2.

(viii) C is a colour function. It is defined from P into Σ as given in Definition 3.6.2.

(ix) G is a guard function as given in definition 3.6.2.

(x) E is an arc expression function such that:

• For an arc $(p, t) \in A$, connecting a place $p \in P$ and a transition $t \in T$, it is required that the arc expression $E(p, t)$ is the constant $k \in K$ which represents the component presentation model of the place p. This is for the directed arc from a place to a transition. Similarly it can be applied to a directed arc from a transition to a place. For an arc $(t, p) \in A$ it is required that $E(t, p)$ is the constant $k \in K$ which represents the component presentation model of the place p.

Definition 4.3.2. A Hierarchical Coloured Petri Net for modelling a user interface and interaction is a tuple (S, FS, FT) such that:

(i) S is a finite set of pages such that each page $s \in S$ is a non-hierarchical CPN model as above

$$(PM_s, K_s, P_s, T_s, I_s, \Sigma_s, A_s, C_s, G_s, E_s)$$

(ii) $F \subseteq P_s$ is a finite set of fusion sets.

(iii) FT is a fusion type function.

So far we have discussed how to model the user interface and interaction of an interactive system using Coloured Petri Nets. This is achieved by expressing a presentation model and a presentation interaction model in Coloured Petri Nets as seen in Sections 4.2 and 4.3. In the next section we will analyze the CPN model of a user interface and interaction of safety-critical interactive systems.

4.4 Analysis

As discussed in Section 3.6.3, in this work we use a state space method to analyze the behaviour of a system. There are some properties that need to be investigated for any safety-critical interactive system. These properties include:

1. deadlock freedom: make sure that a user can never get into a state where no action can be taken

2. livelock freedom: make sure that a user can never get into a cycle (when sequences are executed indefinitely) without the possibility of making effective progress

3. reachability properties: reachability properties deal with what can be done through the user interface, i.e., checking if all the behaviours are available to a user at some point during interaction.

We use the simplified infusion pump example to show how the state space method can be used to investigate the properties of a system. All the properties are analyzed by examining the nodes of the state space graph or the SCC graph. The SCC graph is used for checking reachability among multiple nodes as it is smaller and does not have cycles. The queries are implemented in CPN-ML as it is supported by the CPN Tool. Normally model checking uses a form of temporal logic to express properties. In our case, we have found the use of CPN-ML queries to be of immediate benefit with the CPN Tool.

Firstly, we generate the state space graph for the CPN model of a user interface and interaction of the simple infusion pump. Figure 4.8 shows the state space graph for the simple infusion pump model. There are six reachable markings or nodes (1 to 6) in Figure 4.8, represented by rounded boxes. Each node has an identification number located at the top.

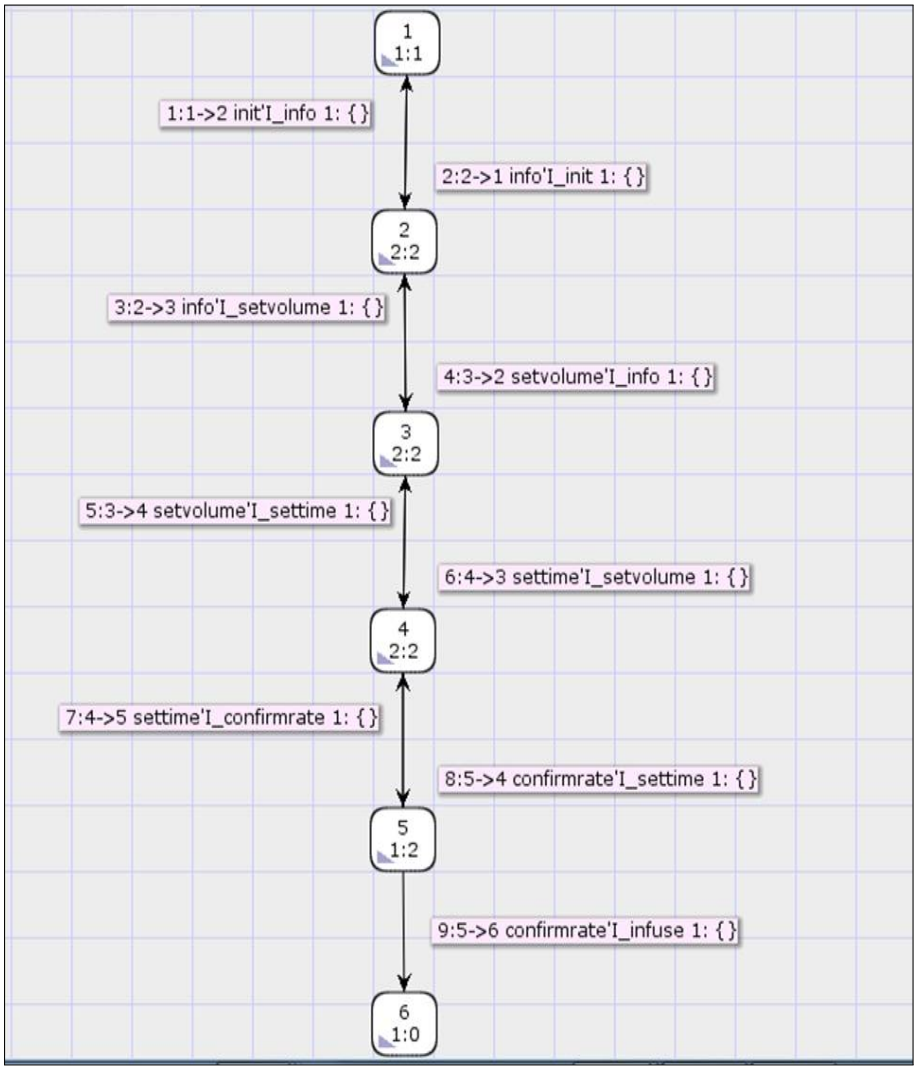

Figure 4.8: State space graph of simple infusion pump

An arc box is shown in the solid box next to the arc in Figure 4.8 which includes the identification number of the arc and some marking information (located next to the number). The node/marking information, represented by m1->m2, indicates that the node m2 can be reached from node m1. A user can go from *init* state to *info* state (arc 1) and from *info* state to *init* state

(arc 2). From *info* state a user can go to the *setvolume* state (arc 3) and from *setvolume* a user can also go back to *info* state (arc 4). After setting the volume, a user can go to *settime* state (arc 5). If a user wants to make some changes in the volume to be infused, then a user can go back to the *setvolume* state (arc 6). After setting the time, the user needs to confirm the rate and goes to the *confirmrate* state (arc 7). If a user wants to make some changes, then a user is allowed to go back to the *settime* state (arc 8). After confirming the rate, a user enters the *infuse* state (arc 9).

As shown in figure 4.8, there is one dead marking/node (i.e. 6). This is obviously expected as we have intentionally made our model like this. There is no state a user can go to after entering the *infuse* state.

The marking of each place can also be shown in the boxes next to the nodes. The marking for node 1 is given in Table 4.6.

```
1:
infuse'infuse 1 :              empty
info'init 1 :                  1‘ [(Display,Responder,
                               [S_displaystartmessage]),
                               (NoButton,ActCtrl,[]),
                               (YesButton,ActCtrl,[]),
                               (InfoButton,ActCtrl,[I_info]),
                               (MinusButton,ActCtrl,[]),
                               (PlusButton,ActCtrl,[]),
                               (OnOffButton,ActCtrl,[])]
info'info 1 :                  empty
info'setvolume 1 :             empty
confirmrate'settime 1 :        empty
confirmrate'confirmrate 1 :    empty
confirmrate'infuse 1 :         empty
setvolume'info 1 :             empty
setvolume'settime 1 :          empty
setvolume'setvolume 1 :        empty
init'init 1 :                  1‘ [(Display,Responder,
                               [S_displaystartmessage]),
                               (NoButton,ActCtrl,[]),
                               (YesButton,ActCtrl,[]),
                               (InfoButton,ActCtrl,[I_info]),
                               (MinusButton,ActCtrl,[]),
                               (PlusButton,ActCtrl,[]),
                               (OnOffButton,ActCtrl,[])]
init'info 1 :                  empty
settime'settime 1 :            empty
settime'setvolume 1 :          empty
settime'confirmrate 1 :        empty
```

Table 4.6: marking of node 1 for simple infusion pump model

In Table 4.6, the place *init* that appears on the *init* page and *info* page belongs to the same fusion set and has one token with the value : *[(Display,Responder,[S_displaystartmessage]), (NoButton,ActCtrl,[]), (YesButton, ActCtrl,[]), (InfoButton,ActCtrl,[I_info]), (MinusButton,ActCtrl,[]), (PlusButton,ActCtrl,[]), (OnOffButton,ActCtrl,[])]*. The rest of the places have no tokens.

Figure 4.9 shows the SCC graph for the simple infusion pump prototype. The full state space graph as shown in Figure 4.8 has six nodes and nine arcs, while the SCC graph has two nodes and one arc. Node ˜1 is the initial SCC

node and ~2 is the terminal SCC node (represented by solid rectangle). Each SCC node has information about the number of nodes and arc, which have been grouped in that SCC node. The number of arcs, which connect each SCC node to others, are indicated next to the arcs. For example, node ~1 has five nodes and eight arcs and the node has no input arcs and one output arc going to the SCC node ~2.

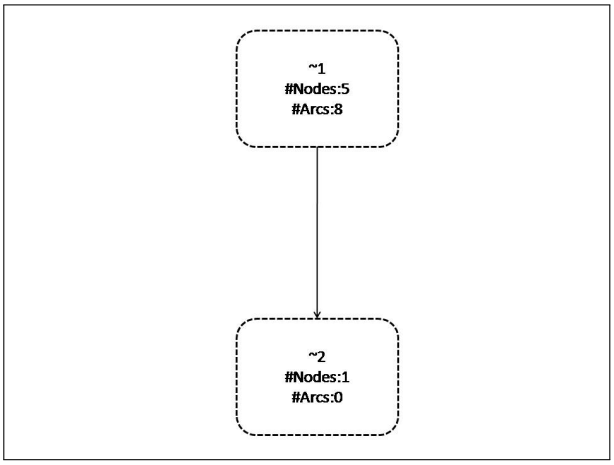

Figure 4.9: SCC Graph of simple infusion pump

The detailed state space report of CPN model of user interface and interaction of simple infusion pump is given in Appendix A and is summarized in Table 4.7.

State Space Graph	
Number of Nodes	6
Number of Arcs	9
SCC Graph	
Number of Nodes	2
Number of Arcs	1
Number of Dead Markings	1 (Node [6])
Dead Transition Instances	None
Live Transition Instances	None

Table 4.7: statistics of the CPN model of user interface and interaction of simple infusion pump

4.4.1 Notations

Following are the notations that are used for investigating the general properties of a CPN model:

- $[M\rangle$ denotes the set of markings that are reachable from marking M.
- $[M_0\rangle$ denoting the set of markings reachable from the initial marking M_0.
- $M[t\rangle$ denotes that transition $t \in T$ is enabled at marking M.
- B(t) denotes set of all bindings for transition t.

4.4.2 Investigation of General Properties

4.4.2.1 Deadlock freedom

Deadlock means that a user enters into a state in which no action can be taken, which might (clearly) be a problem for a safety-critical system. In CPN, a system is said to be deadlock-free if no dead marking (a marking with no enabled transition leaving it) can be reached from the initial marking. This means that from an initial state, the user will never get into a state in which no actions can be taken. In Figure 4.8, M_6 is a dead marking. A CPN model is deadlock-free if for each marking, M, reachable from the initial marking, M_0, there is an enabled transition, t, i.e $M[t\rangle$.

Definition: The CPN model is deadlock-free iff $\forall M \in [M_0\rangle, \exists t \in T : M[t\rangle$.

The CPN tool can detect deadlock-freeness. The state space report generated by the CPN tool gives details about the dead markings in the model. We can check the dead markings directly using the in-built functions for finding the dead markings. The function that lists the dead markings in the CPN model is *ListdeadMarkings*(). The output of this function is the list of nodes for which there are no enabled transitions.

For example, we can detect deadlock-freeness for the CPN model of the simple infusion pump. In our simple infusion pump model *infusing* is the final state and none of the other states is reachable from there. If there exists only one dead marking, M_6, this means that the model is working as expected. Table 4.8 shows the dead marking in the model.

`ListDeadMarkings()`
`output:` `val it =[6]:  Node list`

Table 4.8: Showing dead marking for simple infusion pump model

This is one way of detecting whether the model is deadlock-free or not.

Of course, in the simple infusion pump we are supposed to get a dead marking as from *infusing* no other state is reachable. That is how we have intentionally made our model. *Infusing* is the expected terminal marking in our case. There can be cases in real systems where we have expected terminal markings. So there should be a way of detecting deadlock-freeness in such systems.

If we want to check for deadlocks in such systems, we first have to find the terminal markings (i.e. dead markings that are a correct part of the design, like a legitimate final state) and then see in the list of all dead markings if there exists a marking (or markings) which does not exist in the list of terminal markings. If there exist such markings then it means that the system has improper termination i.e. deadlock, otherwise it is free from deadlocks.

Table 4.9 show how we detect the dead markings that are not intended (i.e. not terminal markings) for our model.

```
fun ValidTerminalMarking n = (Mark.infuse'infuse 1 n =
1`[(Display, Responder, [S_displayinfusingmsg]),
(NoButton, ActCtrl, []),
(YesButton, ActCtrl, []),
(InfoButton, ActCtrl, []),
(MinusButton, ActCtrl, []),
(PlusButton, ActCtrl, []),
(OnOffButton, ActCtrl, [])] )
fun InValidTerminal()=PredNodes(ListDeadMarkings(),
fn n => not (ValidTerminalMarking n),
NoLimit);
```

```
output:
val it =[]:  Node list
```

Table 4.9: Showing invalid terminal nodes for simple infusion pump model

The definition of *ValidTerminalMarking* states that the marking on place *infuse* on page *infuse* is a terminal (i.e. correct) marking. The function *InValidTerminal* searches through all the dead markings. If the dead marking matches the markings given by *ValidTerminalMarking*, then it results in an empty list, otherwise we get a list of all the dead markings which are other than the terminal markings. In this case we have an empty list as value, i.e. all the dead markings are terminal markings. Hence there are no deadlocks.

4.4.2.2 Livelock

Livelock, i.e. a cycle in which no progress is being made, occurs when sequences are executed indefinitely without possibility of making effective progress through the states of the system. A livelock is detected when the state space contains a cycle that leads to no markings outside the cycle. In this case, once the cycle is entered it will be repeated forever. A CPN model which behaves properly should usually be free from livelocks. Therefore, it is important to check that the CPN model does not contain livelocks, unless intended. There might be a case for which a model *should* have a livelock state, but this is very uncommon. So searching for livelock states is undoubtedly useful and should be a part of any analysis.

It is quite easy to detect livelocks in a small and simple model by simply

looking at it. But for real devices, this is not possible. So, there should be some uniform approach that finds such livelock states. A convenient way to check the absence of livelocks in a CPN model is to study the automatically generated graph of strongly connected components (SCC graph).

Depending on the structure of the generated state space graph, model checking the absence of livelocks takes one of the following two forms [122]:

1. ***If the state space graph contains no self-loops and the state space graph is isomorphic to the SCC graph, then the state space has no livelocks.***

   To use the above definition, we first need to check if there exists self-loops in state space graph. The query shown in table 4.10 verifies the absence of self-loop terminal nodes.

```
fun SelfLoopNodes n = (OutNodes (n) = [n])
fun SelfLoopTest()=PredNodes(EntireGraph,
fn n => (SelfLoopNodes n),
NoLimit);
```

Table 4.10: Query to verify the absence of Self Loop Terminal Nodes in CPN

   The CPN-ML code given in Table 4.10 finds the self loop terminal nodes. The function *SelfLoopNodes* uses the in-built function *OutNodes* to get a list of successsor nodes. The argument *PredNodes* specifies a function which maps each node into a boolean value. The nodes which evaluate to false are ignored; the others take part in further analysis. The val *EntireGraph* denotes the set of all nodes in the state space. The val *NoLimit* specifies the limit, i.e., that the search continues until the entire search area has been traversed. If a code returns an empty list, then there are no self loop terminal nodes which means that the CPN model is livelock free.

   If we apply this function to the CPN model of the simple infusion pump then we an empty output list as shown in Table 4.11. This means that

there are no self loop terminal nodes in the state space graph of the simple infusion pump.

```
fun SelfLoopNodes n = (OutNodes (n) = [n])
fun SelfLoopTest()=PredNodes(EntireGraph,
fn n => (SelfLoopNodes n),
NoLimit);
```

```
Output:
val it =[] :  Node list
```

Table 4.11: Detecting the absence of Self Loop Terminal Nodes in Simple Infusion Pump model

Secondly, we need to check if the state space graph is isomorphic to its SCC graph. In our example, the number of nodes and arcs in the state space graph of the simple infusion are not equal to the number of nodes and arcs in the SCC graph as shown in the Table 4.7 and also the strongly connected components consists of more than one node. Hence, we cannot say that the state space graph is isomorphic to its SCC graph. Therefore we can not prove the absence of livelock using this definition.

2. ***If the state space contains self-loops or if there is at least one strongly connected component that consists of more than one node, then we need to examine if all terminal SCCs are trivial that is, they consist of a single node and no arcs. A non-trivial terminal SCC represents a livelock in the model [122].***

   We have seen in the point 1 above that the state space graph and its SCC graph have a different number of nodes and arcs, therefore the strongly connected components consist of more than one node. So there is a need to check the if all terminal SCCs are trivial. Table 4.12 shows CPN-ML code to find all non-trivial terminal SCC. The function *ListTerminalSCCs* uses the function *SccTerminal* to get a list of SCC terminal nodes (i.e., nodes having no outgoing arcs). The function *Livelock* finds the non-trivial terminal SCC nodes. If we get an empty list, i.e. if all terminal SCCs are trivial, then the CPN model is free from

livelock using point 2. The search area in this function is the list of all SCC terminal nodes. The val *NoLimit* specifies an infinite limit, i.e., that the search continues until the entire search area has been traversed.

```
fun ListTerminalSCCs()=PredAllSccs(SccTerminal);
fun Livelock()=PredSccs(ListTerminalSCCs(),
fn n => not (SccTrivial n),
NoLimit);
```

Table 4.12: Query to find a non-trivial terminal SCC node

Now we apply the function given in Table 4.12 to the CPN model of a simple infusion pump which results in an empty list as shown in Table 4.13. Hence, the model is free of livelock.

```
fun ListTerminalSCCs()=PredAllSccs(SccTerminal);
fun Livelock()=PredSccs(ListTerminalSCCs(),
fn n => not (SccTrivial n),
NoLimit);
```

```
Output:
val it=[ ] :  Scc list
```

Table 4.13: Query to find a non-trivial terminal SCC node in Simple Infusion Pump

4.4.2.3 Reachability Properties

Reachability is another important property that needs to be considered for safety-critical interactive systems. Reachability concerns checking whether all behaviours described in the presentation models can be obtained by a user by applying some sequence of commands at some point in their interaction. As we have just modelled I-behaviours, we will check if all the I-behaviours described in the presentation model are available to a user at some point in their interaction.

Reachability functions in CPN Tool [119] allow checking reachability of one node from another. Suppose we wish to check the CPN model of the user interface and interaction of the simple infusion pump for the following:

1. Is it possible to reach the *setvolume* state from the *info* state and vice versa?

2. Is it possible to reach the *init* state from the *info* state and vice versa?

3. Is it possible to reach the *setvolume* state from the *settime* state and vice versa?

4. Is it possible to reach the *confirmrate* state from the *settime* state and vice versa?

5. Is it possible to reach the *infusing* state from the *confirmrate* state and vice versa?

To check all of these reachabilities, we first need to find the nodes that represent the *init*, *info*, *sevolume*, *settime*, *confirmrate* and *infusing* states from the state space graph. The CPN-ML codes given in Tables 4.14 - 4.19 gives the list of *init*, *info*, *sevolume*, *settime*, *confirmrate* and *infusing* nodes respectively.

```
fun initnodes ()=SearchAllNodes( fn n=>
let
fun initarcs(nil)=false |
initarcs(a ::  al :  Arc list) =
if ArcToTI a = TI.info'I_init 1
then
true
else
initarcs(al);
in
initarcs(InArcs(n))
end, fn n => n, [], op::);
```

```
output:
val it=[1]:  Node list
```

Table 4.14: Returns all *init* nodes in the state space graph

```
fun infonodes ()=SearchAllNodes( fn n=>
let
fun infoarcs(nil)=false |
infoarcs(a ::  al :  Arc list) =
if ArcToTI a = TI.init'I_info 1 orelse
ArcToTI a = TI.setvolume'I_info 1
then
true
else
infoarcs(al);
in
infoarcs(InArcs(n))
end, fn n => n, [], op::);
```

```
output:
val it=[2]:  Node list
```

Table 4.15: Returns all *info* nodes in the state space graph

```
fun setvolumenodes ()=SearchAllNodes( fn n=>
let
fun setvolumearcs(nil)=false |
setvolumearcs(a ::  al :  Arc list) =
if ArcToTI a = TI.info'I_setvolume 1 orelse
ArcToTI a = TI.settime'I_setvolume 1 then
true
else
setvolumearcs(al);
in
setvolumearcs(InArcs(n))
end, fn n => n, [], op::);
```

```
Output:
val it=[3]:  Node list
```

Table 4.16: Returns all *setvolume* nodes in the state space graph

```
fun settimenodes ()=SearchAllNodes( fn n=>
let
fun settimearcs(nil)=false |
settimearcs(a ::  al :  Arc list) =
if ArcToTI a = TI.setvolume'I_settime 1 orelse
ArcToTI a = TI.confirmrate'I_settime 1
then
true
else
settimearcs(al);
in
settimearcs(InArcs(n))
end, fn n => n, [], op::);
```

```
output:
val it=[4]:  Node list
```

Table 4.17: Returns all *settime* nodes in the state space graph

```
fun confirmratenodes ()=SearchAllNodes( fn n=>
let
fun confirmratearcs(nil)=false |
confirmratearcs(a ::  al :  Arc list) =
if ArcToTI a = TI.settime'I_confirmrate 1
then
true
else
confirmratearcs(al);
in
confirmratearcs(InArcs(n))
end, fn n => n, [], op::);
```

```
output:
val it=[5]:  Node list
```

Table 4.18: Returns all *confirmrate* nodes in the state space graph

```
fun infusingnodes ()=SearchAllNodes( fn n=>
let
fun infusingarcs(nil)=false |
infusingarcs(a ::  al :  Arc list) =
if ArcToTI a = TI.confirmrate'I_infuse 1
then
true
else
infusingarcs(al);
in
infusingarcs(InArcs(n))
end, fn n => n, [], op::);
```

```
output:
val it=[6]:  Node list
```

Table 4.19: Returns all *infusing* nodes in the state space graph

1. **Is it possible to reach the *setvolume* state from the *info* state and vice versa?**

   Now to check whether the *setvolume* node is reachable from *info* node, we apply the function given in Table 4.20 which returns *true* which means that the *setvolume* node is reachable from the *info* node.

```
Reachable(2,3)
```

```
output:
val it=true:  bool
```

Table 4.20: Reachability function to test reachability from *info* to *setvolume* node

   If we want to know the path to reach the *setvolume* node from the *info* node, the function given in Table 4.21 is evaluated which tells the path from the *info* node to the *setvolume* node.

```
Reachable'(2,3)
```

```
output:
A path from node 2 to node 3 is:[2,3]
val it=true:  bool
```

Table 4.21: Reachability function to find the path from *info* to *setvolume*

Similarly, we can check if the *info* node is reachable from the *setvolume* node by applying the function given in Table 4.22 which shows that user can reach the *info* state from the *setvolume* state.

Reachable(3,2)
output: val it=true: bool

Table 4.22: Reachability function to test reachability from *setvolume* to *info* node

2. **Is it possible to reach the *init* state from the *info* state and vice versa?** The reachability test written in Tables 4.23 and 4.24 shows that it is possible to reach the *init* state from the *info* state and vice versa as expected.

Reachable(2,1)
output: val it=true: bool

Table 4.23: Reachability function to test reachability from *info* to *init* node

Reachable(1,2)
output: val it=true: bool

Table 4.24: Reachability function to test reachability from *init* to *info* node

3. **Is it possible to reach the *setvolume* state from the *settime* state and vice versa?**

   The reachability test written in Tables 4.25 and 4.26 shows that it is possible to reach the *setvolume* state from the *settime* state and vice-verse as expected.

Reachable(3,4)
output: val it=true: bool

Table 4.25: Reachability function to test reachability from *setvolume* to *settime* node

Reachable(4,3)
output: val it=true: bool

Table 4.26: Reachability function to test reachability from *settime* to *setvolume* node

4. **Is it possible to reach the *confirmrate* state from the *settime* state and vice versa?**

   The reachability test written in Tables 4.27 and 4.28 shows that it is possible to reach the *settime* state from the *confirmrate* state and vice versa as expected.

Reachable(4,5)
output: val it=true: bool

Table 4.27: Reachability function to test reachability from *settime* to *confirmrate* node

Reachable(5,4)
output: val it=true: bool

Table 4.28: Reachability function to test reachability from *confirmrate* to *settime* node

5. **Is it possible to the reach *infusing* state from the *confirmrate* state and vice versa?**

   The reachability test written in Table 4.29 shows that it is possible to

reach the *infusing* state from the *confirmrate* state but the reachability test written in Table 4.30 returns false which means that a user can not return to the *confirmrate* state when the pump is in the *infusing* state. This is one of the requirement of a simple infusion pump as mentioned in Section 4.3.1.

`Reachable(5,6)`
`output:` `val it=true:  bool`

Table 4.29: Reachability function to test reachability from *confirmrate* to *infusing* node

`Reachable(6,5)`
`output:` `val it=false:  bool`

Table 4.30: Reachability function to test reachability from *infusing* to *confirmrate* node

We can repeat this process for each of the other pairs and investigate the reachability.

Reachability functions provided by the CPN tool also allows checking total reachability which means the ability to get anywhere from anywhere. The *AllReachable* function in the CPN tool determines whether all the reachable markings are reachable from each other. This is the case iff there is exactly one strongly connected component[119]. This helps in verifying that all of the behaviours can be accessed by a user and also ensures that a user can return back to the state they came from. Allowing users to return back to the previous states is important as it allows correcting errors. The *AllReachable* function will return true or false as a result. For the simple infusion pump model, this function evaluates to false as shown in Table 4.31.

AllReachable()
output: val it=false: bool

Table 4.31: Total reachability function

Of course, the CPN model of the user interface and interaction of the simple infusion pump does not have total reachability as we cannot return to any other state when the user is in infusing state. It is also quite clear by looking into the graphs in Figures 4.8 and 4.9. The strongly connected component graph has two nodes: node ~1 has five nodes which are all reachable from each other and node ~2 has only one node. As there are more than one node in strongly connected component graph which means that total reachability is not achieved.

4.4.3 Summary

In this chapter we have discussed how presentation models and presentation interaction models can be expressed in Coloured Petri Nets. On the basis of the presentation model, another model, i.e. the presentation and interaction model, is constructed using Coloured Petri Nets which give meaning to the *I-behaviours* given in the presentation model. In this chapter we have summarized the rules to create presentation interaction model using Coloured Petri Nets. The CPN model thus obtained is a combination of the presentation model and the presentation and interaction model. We can see the navigational possibilities and the available widgets and its associated behaviours in the same model. The investigation of the behavioural properties show that there is no significant problem and the model is behaving as expected as per the assumptions made.

Chapter 5

Modelling Functionality using Coloured Petri Nets

5.1 Introduction

Having modelled a user interface and interaction in Coloured Petri Nets, we now move to the final part, modelling the functionality. In the original (PM/PIM/Z) model, Z is used to specify the functionality of a system. In our work we want to have a single model, so we will express a Z specification in Coloured Petri Nets and then extend the CPN model of a user interface and interaction (from the previous chapter) by adding functional aspects to it.

In the case of a CPN model, we add the meaning of S-behaviours as transitions, reflecting the predicate part of the relevant Z operation schema. We will also have one fusion place named Z added to every page of the model that contains all the observations of the state schema. In the end we are aiming to be able to see what the values of the observations are in each state during simulation.

Since the resulting CPN model will contain both I-behaviours and S-behaviours, we can have user-interface and interaction and functional parts of a safety-critical system in a single model.

To add functionality to the model, we must know how to express Z in Coloured Petri Nets. From the previous chapter, we have a CPN model which

has:

1. Pages, one for each component presentation model, interconnected by fusion places;

2. Fusion places that represent the states and have the same name as the component presentation model;

3. Transitions that represent I-behaviours of the presentation model.

5.2 Expressing Z in Coloured Petri Nets

In this section we will explain how the kinds of Z specification used in the existing PM/PIM/Z models can be expressed in Coloured Petri Nets. This is done using colour sets, an initial expression and arc inscriptions. We will start with expressing Z types [124][112] in CPN.

Built-in type: Z provides a single built-in type, namely the type of integers $\mathbb{Z}$. For example in Z, n can be declared as n: $\mathbb{Z}$ such that the value of n is an integer. We can write this in Coloured Petri Nets using the integer colour set. So the syntax:

$$colset\ INT = int;$$

will create a colour set INT which defines INT as integers, and

$$var\ n : INT;$$

will declare a variable n such that the value of n is an integer.

In Z, we can just declare the type and can add the information about that type later. For example, in Z we can declare a type $[NAME]$ without giving any information about it. But in CPN, it is not possible to add a colour set without giving any further information. If the Z type $[NAME]$ is expressed in CPN, we have to provide more information about it.

Cartesian Product: The Cartesian product is another frequently used type constructor. It is a type consisting of ordered pairs. We can use the product colour set of CPN to represent such Z types.
The Syntax for writing Z Cartesian product types in Coloured Petri Nets is:

$$colset\ \langle z\_type\_name\rangle = product\langle colset\_name_1\rangle * \langle colset\_name_2\rangle * ...* \langle colset\_name_n\rangle;$$

where $colset\_name_1 ... colset\_name_n$ are already defined colour sets which represents types in Z that need to form an ordered pair.
For example, in Z, $NAME \times NUM$ is a type consisting of ordered pairs. This can be written in CPN as:

$$colset\ NAME = string;$$
$$colset\ NUM = int;$$
$$colset\ NAMEXNUM = product\ NAME * NUM;$$

$NAME$ is a colour set of strings and NUM is a colour set of int. Colour set $NAMEXNUM$ represents an ordered pair of $NAME$ and NUM.

Free Type: Another significant type constructor is the *free type* which can have additional constraints. Simple free types in Z are declared in a BNF-like notation, as,

$$Z\_type\_name ::= constant_1 \mid constant_2 \mid ... \mid constant_n$$

To declare free types in CPN, we can use the enumerated colour sets where enumerated values are explicitly named as identifiers in the declaration. The syntax for writing free types in CPN is:

$$colset\ \langle Z\_type\_name\rangle = with\ \langle constant_1\rangle \mid \langle constant_2\rangle \mid ... \mid \langle constant_n\rangle;$$

For example, we might have a free type $SHAPES$ containing exactly two dif-

ferent constants *circle* and *rectangle* which is written in Z as:

$$SHAPES ::= circle \mid rectangle$$

The same can be expresses in CPN as:

$$colset\ SHAPES = with\ circle \mid rectangle;$$

There are other types that may occur on the right-hand side of a free type definition in Z. An expression like $anint \ll \mathbb{Z} \gg$ denotes an integer value labelled with the constructor function *anint*. Thus in Z a type of values which are either Booleans or integers is defined by:

$$INTORBOOL ::= anint \ll \mathbb{Z} \gg \mid abool \ll \mathbb{B} \gg$$

To represent such types in CPN, we use the union colour set. A union colour set is a disjoint union of previously declared colour sets. The syntax to represent free types with constructor functions is:

$$colset\ \langle Z\_type\_name \rangle = union\ id_1 : colset\_name_1 + id_2 : colset\_name_2 + \ldots + id_n : colset\_name_n;$$

where $id_1 ... id_n$ are the identifiers and $colset\_name_1 .. colset\_name_n$ are the previously declared colour sets.
So *INTORBOOL* type can be written in CPN-ML as:

$$colset\ INT = int;$$
$$colset\ BOOL = bool;$$
$$colset\ INTORBOOL = union\ anint : INT\ +\ abool : BOOL;$$

Powersets: The power set operator $\mathbb{P}$ (giving "the set of all subsets" of a set) is an elementary type constructor often used in Z. For example, we might have a observation *NUM* of type $\mathbb{P}\,\mathbb{Z}$, meaning that *NUM* is a set of integers. If we

want to write such a type in CPN, then in this work we have decided that the list colour set is used[1]. The syntax for writing power sets of Z based on this decision in CPN is:

$$colset\ \langle z\_type\_name \rangle = list\ \langle colset\_name \rangle;$$

We can write $\mathbb{P}\,\mathbb{Z}$ in CPN as:

$$colset\ INT = int;$$
$$colset\ NUM = list\ INT;$$

Axiomatic Definition: If we have an axiomatic definition:

$$\begin{array}{|l} hours : \mathbb{P}\,\mathbb{N} \\ minutes : \mathbb{P}\,\mathbb{N} \\ \hline hours = 0\,..\,24 \\ minutes = 0\,..\,59 \end{array}$$

This can be written in CPN as:

$$colset\ hours = int\ with\ 0..24;$$
$$colset\ minutes = int\ with\ 0..59;$$

Z schema: Z schemas are used to specify the state space and operations of the system. To write the declaration part of a Z schema in CPN, we use the record colour set. The syntax for writing this in CPN is:

$$colset\langle Z \rangle = record\ id_1 : type_1 * id_2 : type_2 * \ldots * id_n : type_n;$$

where $id_1..id_n$ are Z observations and $type_1..type_n$ are their corresponding types (which are already declared colour sets using the rules as explained above) as

[1]We model only systems with finite components, so modelling the power set with lists is no restriction on our expressiveness.

they appear in the declarations of the schema we are modelling. For example, the *SimpleInfusionPump* schema has two observations *timer* and *volume* and both are of type natural number:

SimpleInfusionPump

$$timer : \mathbb{N}$$
$$volume : \mathbb{N}$$

$$timer \geq 1 \land timer \leq 10$$
$$volume \geq 1 \land volume \leq 10$$

This can be modelled in CPN as follows:

$$colset\ nat = int\ with\ 1..10;$$
$$colset\ Z = record\ timer : nat * volume : nat;$$
$$var\ timer : nat;$$
$$var\ volume : nat;$$

Assume we have a state schema *SimplePump* given below which has three observations *duration*, *volume* and *rate* (which can be any natural number).

SimplePump

$$duration : \mathbb{N}$$
$$volume : \mathbb{N}$$
$$rate : \mathbb{N}$$

This can be expressed in CPN as:

$$colset\ nat = int;$$

$$colset\ Z = record\ duration : nat * volume : nat * rate : nat;$$

$$var\ duration : nat;$$

$$var\ volume : nat;$$

$$var\ rate : nat;$$

The predicate part of a schema will give us expressions on arcs of certain transitions, as we will see later.
In its initial state, all the initial values are set to zero:

Init

SimplePump

$duration = 0$

$volume = 0$

$rate = 0$

The *Init* schema of Z is represented as an initial marking in Coloured Petri nets.

$$I(Init) = 1`\{duration = 0, volume = 0, rate = 0\}$$

where *Init* is a place whose initial marking is $1`\{duration = 0, volume = 0, rate = 0\}$. This is illustrated in the figure 5.1.

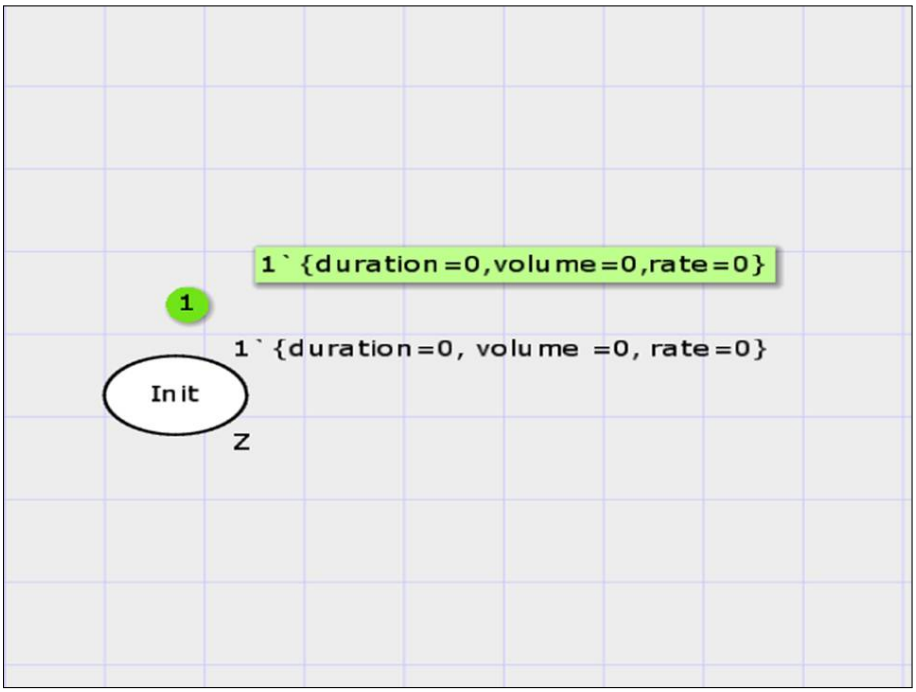

Figure 5.1: Init schema in CPN

Now suppose we have one operation schema *SetVolume* as written below:

SetVolume

$\Delta SimplePump$

$volume' = 4$

$time' = time$

$rate' = rate$

In Figure 5.2, on the arc going into place/state *setvolume*, we have names in the set in the tuple on the arc that correspond to the observations in the *SetVolume* Z schema. In any state change, a primed observation in the Z schema indicates the value *after* the operation has successfully taken place and the unprimed observations represent the observations *before* the operation happens. So, for example, we can see that the Z schema *SetVolume* requires that $volume'$ (i.e. the value of the observation *volume* after the operation has happened) is four. This is expressed in the CPN version by the arc going into

the state *SetVolume* where we have *volume* = 4, i.e. the value of *volume* after the *I_setvolume* transition has happened is 4 as shown in Figure 5.3.

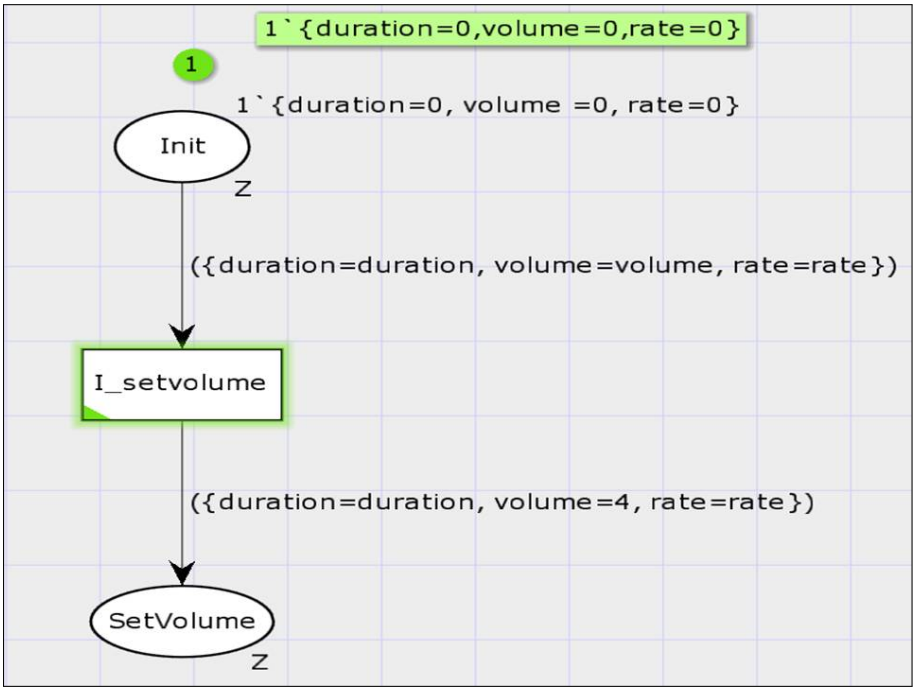

Figure 5.2: setvolume schema in CPN before operation

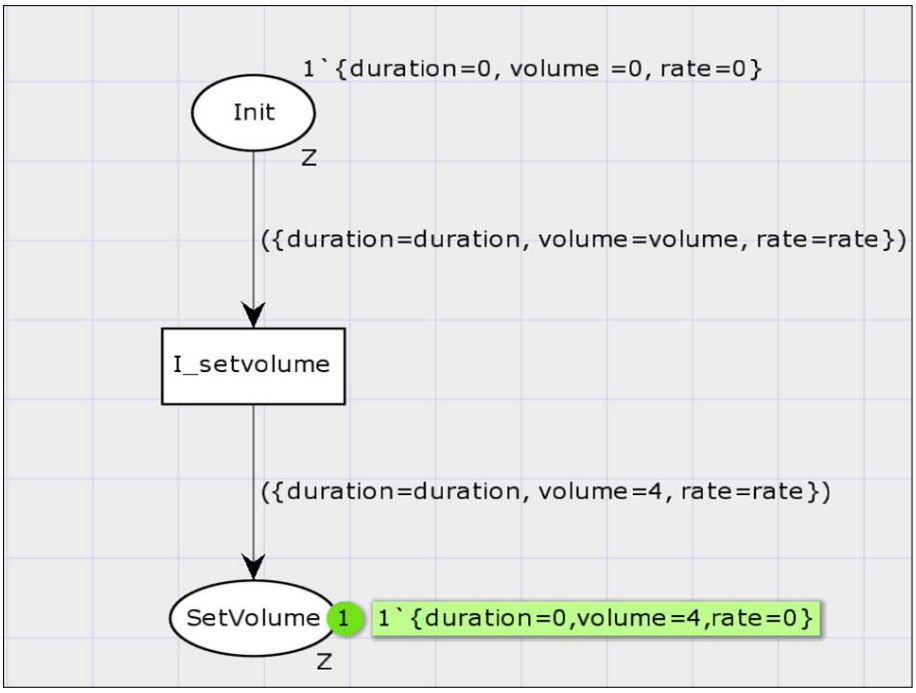

Figure 5.3: setvolume schema in CPN after operation

This is how the state schema and operation schema are expressed in CPN. We have expressed the features of Z as needed in this thesis; we do not need to express the entire Z language in Coloured Petri Nets.

5.2.1 Integration Rule

Now we will summarize and outline the process for adding S-behaviours written in Z to the CPN model of a user interface and interaction from previous chapter:

1. For every Z type used to specify the underlying functionality of a system, create a corresponding colour set.

2. For modelling the Z state schema, we use a record colour set. Variables are declared in CPN for every observation in the state schema. The types of the variables are the same as the observations declared in the Z state schema. We call these variables *observation-name variables*.

3. A fusion place named *Z* (note the italic font) is added to every page of the CPN model. A Z colour set declared in step two above that represents a state schema is assigned to this fusion place.

4. An initial marking of a fusion place *Z* represents the Z initialisation schema.

5. Guards in CPN represent a precondition in a Z operation. The guard must be *True* for a transition to be enabled, thus modelling the case of preconditions in Z.

6. If a component presentation model has S-behaviours, then there will be input and output arcs going to and from *Z* to the corresponding S-behaviour transition.

7. Transitions in CPN that represent the S-behaviours of the presentation model will be added so that they have same name as the names of the

corresponding Z operation schemas. If S-behaviours and I-behaviours occur simultaneously, then they will be merged into a single transition.

8. An output arc from fusion place Z to an S-behaviour transition has an expression which will assign each observation-name variable, declared in (2) above and appearing in the Z operation schema associated with this S-behaviour.

9. An input arc to the place Z from the same S-behaviour transition as in (8) has an expression which assigns each observation-name variable to an expression which gives it its new value reflecting the right-hand side of the relevant equation in the operation schema being modelled.

   For example, if the operation schema has the equation $value' = value + 1$ where *value* is an observation in the state space, then the expression due to (8) will contain the assignment $value = value$ and the expression due to (9) will contain the assignment $value = value + 1$. Taken together these two assignments model the relevant equation in the relevant operation schema. This is done for every equation in the predicate part of the relevant operation schema.

5.3 Formal Definition of CPN for modelling User Interface, Interaction and Functionality

Definition 5.3.1. A non-hierarchical Coloured Petri Net for modelling a user interface, interaction and functionality is a tuple *(Z, PM, K, P, T, I,* Σ*, A, C, G, E)* such that:

(i) Z is a finite set of colour sets representing the Z state schema of the original model with declarations $id_1 : type_1 ... id_n : type_n$, such that:
$Z = \{type_1, type_2, ..type_n, Z\}$, where

- colset $type_1$;
- .....

- colset $type_n$;
- colset $Z = record\ id_1 : type1 * ... * id_n : type_n$;

(ii) PM is a finite set of colour sets representing presentation model declarations as given in definition 4.3.1.

(iii) K is a finite set of constants that represents (by their names) component presentation models as given in definition 4.3.1.

(iv) P is a finite set of places such that $\mid P \mid = \mid K \mid +1$. The constant k in the set K can be mapped to the place p in the set of P with the same name. There is an additional place named Z that represents the state schema.

(v) T is a finite set of transitions representing both I-behaviours and S-behaviours.

(vi) I is an initialization function that assigns an initial marking to each place.

(vii) Σ is a finite set of non-empty types, called colour sets, with $PM \subseteq \Sigma$ and $Z \subseteq \Sigma$.

(viii) A is a finite set of arcs as given in definition 4.3.1.

(ix) C is a colour function as given in definition 4.3.1.

(x) G is a guard function as given in definition 4.3.1.

(xi) E is an arc expression function such that:

- For an arc $(p, t) \in A$, connecting a place $p \in P$ and a transition $t \in T$ where t is an I-behaviour transition, it is required that the arc expression $E(p, t)$ is the name $k \in K$ which represents the component presentation model of the place p. Similarly for a directed arc from a transition representing an I-behaviour to a place.

- For an arc $(p, t) \in A$ connecting a place $p \in P$, where p represents a state in the Z state space, and a transition $t \in T$, representing an S-behaviour which is itself given a meaning by the Z operation schema S, it is required that the arc expression $E(p, t)$ assigns each observation name appearing in S to a new, unique variable (say s).
- For an arc $(t, p) \in A$ connecting a transition $t \in T$, representing an S-behaviour which is itself given a meaning by the Z operation schema S, and a place $p \in P$, where p represents a state in the Z state space, it is required that the arc expression $E(t, p)$ assigns each observation name in S to an expression (in terms of the new, unique variables introduced in the previous paragraph) which defines that observation's new value. This expression is, therefore, essentially the right-hand side of the equation from the predicate part of S which relates the primed version of the observation to its new value.

Definition 5.3.2. A hierarchical Coloured Petri Net for modelling a user interface, interaction and functionality is a tuple HCPN $= (S, FS, FT)$ such that:

(i) S is a finite set of pages such that:
Each page $s \in S$ is a non-hierarchical CPN: *(Z_s, PM_s, K_s, P_s, T_s, I_s, Σ_s, A_s, C_s, G_s, E_s).*

(ii) $FS \subseteq P_s$ is a finite set of fusion sets.

(iii) FT is a fusion type function.

5.4 Example Net

We will take the example of the simple infusion pump as given in Section 4.3.1. We have the CPN model with six pages (Figures 4.2 - 4.7) showing the user interface and interaction (PM/PIM) of the simple infusion pump. The CPN model shows all the states of the device and also shows all the I-behaviours. In

this section we will extend the same model and add S-behaviours to it, whose meaning is originally given in Z.

The Z for the simple infusion pump example is given in Section 3.5.3. The fee type definition and state schema for the pump are given below:

$$INFUSING ::= Yes \mid No$$
$$BATTERY ::= sufficient \mid insufficient$$

SimpleInfusionPump

$battery : BATTERY$
$timer : \mathbb{N}$
$volume : \mathbb{N}$
$volumeleft : \mathbb{N}$
$infusionrate : \mathbb{N}$
$timeleft : \mathbb{N}$
$infusing : INFUSING$

There are two free types: *BATTERY* and *INFUSING* and seven observations: *battery*, *timer*, *volume*, *volumeleft*, *timeleft*, *infusionrate* and *infusing*. The observations *battery* and *infusing* are of type *BATTERY* and *INFUSING* respectively. All the other five observations are of type $\mathbb{N}$. So there are a total of three types in the Z specification of the simple infusion pump: *BATTERY*, *INFUSING* and $\mathbb{N}$ which are expressed in CPN as shown in Table 5.1 as per rule 1 and 2 in Section 5.2.1.

- Colour set *INFUSING* is declared as the enumerated colour set that can have exactly two values *Yes* or *No*.
- Colour set *BATTERY* is also declared as enumerated colour set that can have two values *sufficient* or *insufficient*.
- Colour set *NAT* is declared as integer colour set.

- Colour set Z is a record colour set with a record of all the seven observations of the state schema with its corresponding types. This colour set represents the state schema of the simple infusion pump.

- As the Z operation schemas would be expressed as arc inscriptions so we need to declare variables which could be bound to different values of their respective colour sets during simulation. There are seven variables *battery*, *timer*, *volume*, *volumeleft*, *timeleft*, *infusionrate* and *infusing*.

```
colset INFUSING =   with Yes | No;
colset BATTERY =    with sufficient | insufficient;
colset NAT =        int;
colset Z =          record battery:BATTERY *
                    timer:NAT *
                    volume:NAT *
                    volumeleft:NAT *
                    infusionrate:NAT *
                    timeleft:NAT *
                    infusing:INFUSING;
var battery :       BATTERY;
var timer :         NAT;
var volume :        NAT;
var volumeleft :    NAT;
var infusionrate :  NAT;
var timeleft :      NAT;
var infusing :      INFUSING;
```

Table 5.1: Z Types in CPN

Now we will extend the six pages (Figures 4.2 - 4.7) by adding S-behaviour transitions to them.

Init Page

In its initial state, we assume that the battery charge is sufficient and the pump is not infusing. All the other observations have their initial values set to zero. *Init* schema for the pump is written below:

Init

SimpleInfusionPump

$battery = sufficient$

$timer = 0$

$volume = 0$

$infusionrate = 0$

$timeleft = 0$

$volumeleft = 0$

$infusing = No$

Figure 5.4 shows the structure of the *Init* page of simple infusion pump.

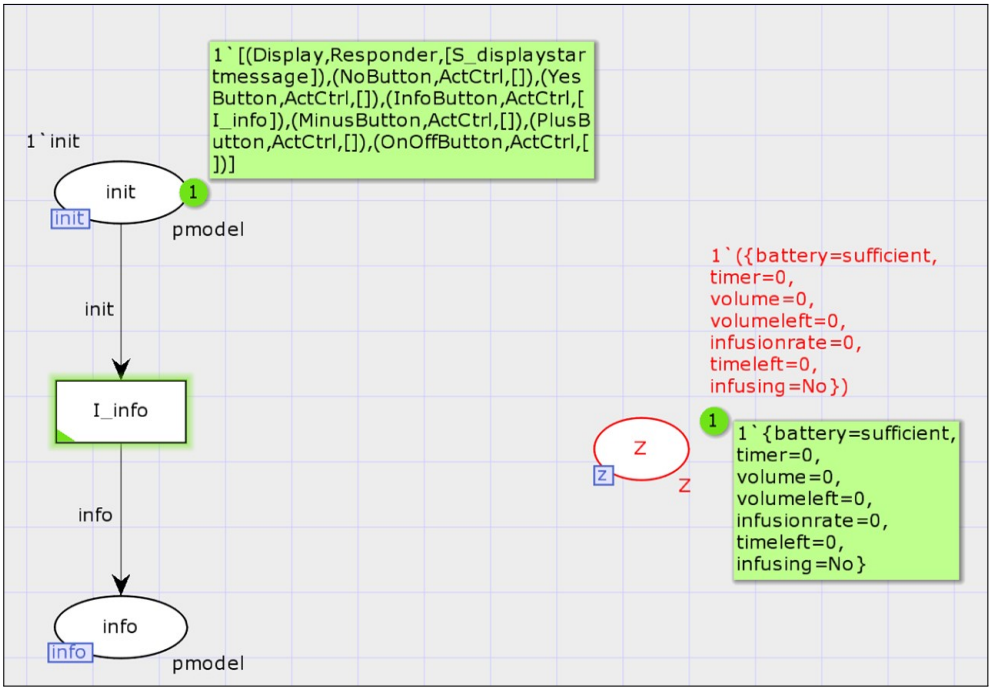

Figure 5.4: Init page

We have one fusion place named Z added to the model which is of type colour set Z as per rule 3 in section 5.2.1. The *Init* schema is expressed in the CPN model as the initial marking as per the rule 4 in section 5.2.1. In figure 5.4, there are two initial markings. A marking on the place *init* represents the presentation model and a marking on the place Z represents the *Init* schema.

There is one S-behaviour, *S_displaystartmessage* associated with the *init* presentation model as shown by the marking of the place *init* in Figure 5.4. The Z schema for this S-behaviour is:

DisplayStartMessage

$\Xi SimpleMedicalDevice$

$display! : CHAR$

$display! = initializingpump$

There is one output observation *display* which displays a message *"initializingpump"* on the screen of the device. In this work we are abstracting what is being displayed on the screen of the device. Therefore we hide these details in order to make the model simpler and reduce the state space. The display part is very important as there are certain properties of the device that are related to the display, for example, the number entry error detection. But in this work we are currently focused on interactivity and so not considering this aspect, so we have abstracted this from the model.

If we *do* want to include the output observation in the CPN model, the model of the *init* page will look as in Figure 5.5. The model is the same as shown in Figure 5.4 with one more transition *S_displaystartmessage* and one extra place *display*. The Z *DisplayStartMessage* schema is expressed in CPN in Table 5.2. Colour set *CHAR* is the enumeration colour set with the value *initializingpump* and the colour set *display* is of type *CHAR* colour set. As we need variables to form the arc inscriptions, so a variable *display* is declared. This model also have a reset arc and an inhibitor arc. Reset arcs consume all tokens from a place and are useful for resetting parts of a model when a

choice is made. This is used so that the transition *S_displayStartMessage* is enabled for one time. Inhibitor arcs test that no tokens are present on a given place. When the transition *I_Info* fires, any token present on the place *display* is removed because of the presence of an inhibitor arc. This is done to avoid multiple token collection on the place *display*.

If other pages have S-behaviours which just display messages on a screen of the device, then it could be modelled in a similar manner. But in order to reduce the state space and reduce the complexity of the model, we are hiding these details.

```
colset CHAR =     with initializingpump;
colset display =  CHAR;
var display :     CHAR;
```

Table 5.2: Z output observation CPN

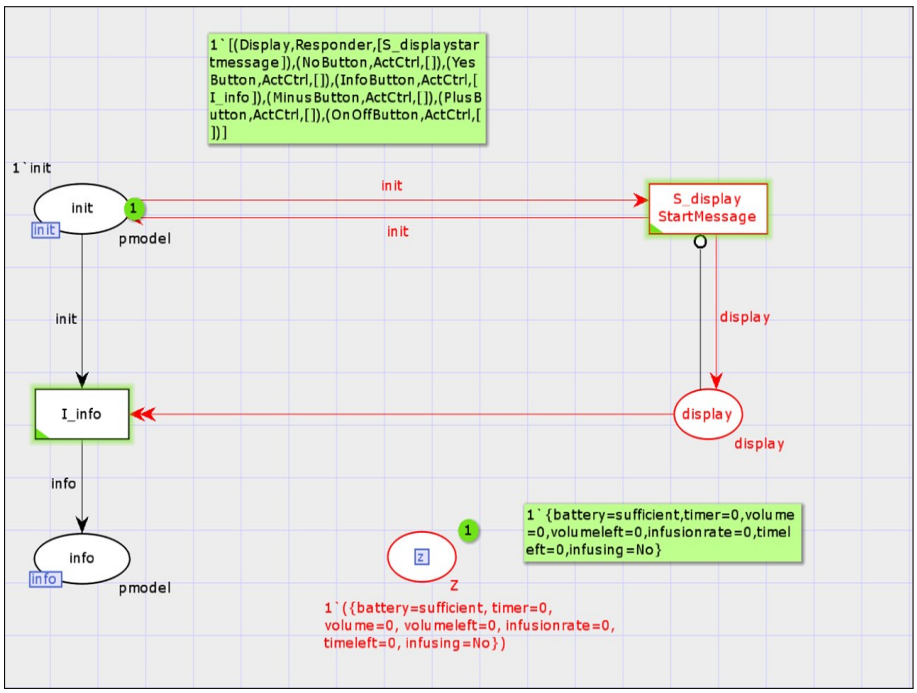

Figure 5.5: Init page with Display

Info Page

The structure of the *Info* page is shown in Figure 5.6. This page also has one additional place *Z* which gives information about the values of the observations

in the *info* state. In the marking of the place *info*, we can see that there is just one S-behaviour: *S_displaybatterylife*. Again we are abstracting the display messages, so its S-behaviour is not modelled. Hence, there is no transition representing S-behaviour in this page.

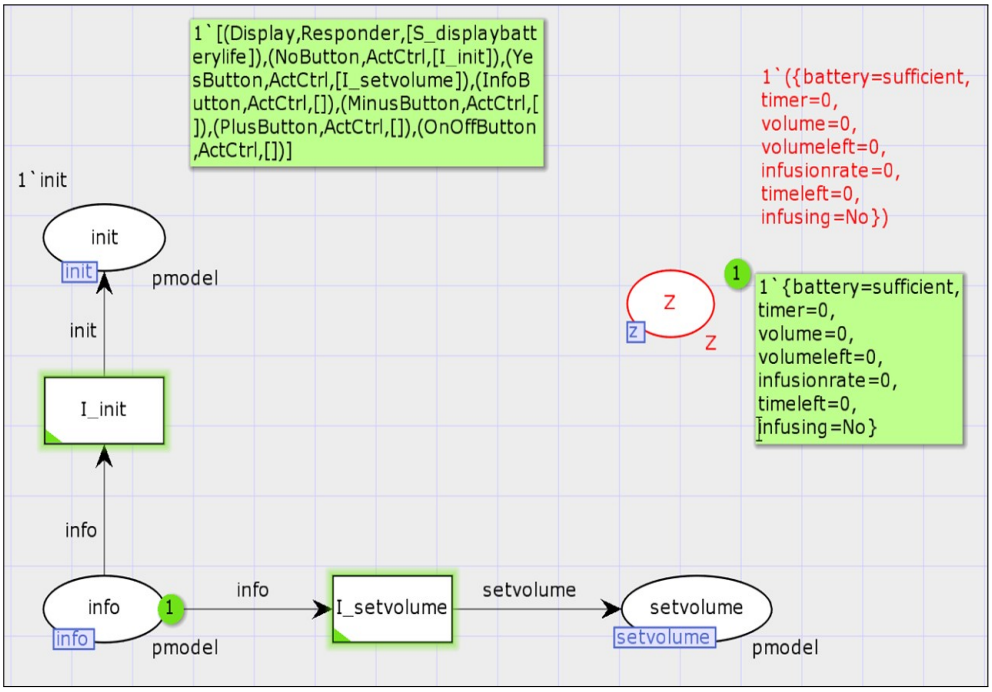

Figure 5.6: Info page

SetVolume Page

Figure 5.7 shows the structure of the *setvolume* page. We are extending the *SetVolume* page of Figure 4.4 which has three places (*info, setvolume and settime*) and two transitions (*I_settime and I_info*) by adding S-behaviours to the model. We have one fusion place named Z added to the model which is of type colour set Z as per rule 3 in section 5.2.1. The token on the *setvolume* place shows the definition of the *setvolume* component presentation model which has three S-behaviours: *S_DecreaseVolume*, *S_IncreaseVolume* and *S_setvolume*. According to the rule 7 in section 5.2.1, *S_DecreaseVolume* and *S_IncreaseVolume* are added as transitions with the same name. As the transitions *S_setvolume* and *I_settime* occur simultaneously, we have single transition for both. Occurrence of these transitions changes the value of observations shown by the marking of the place Z.

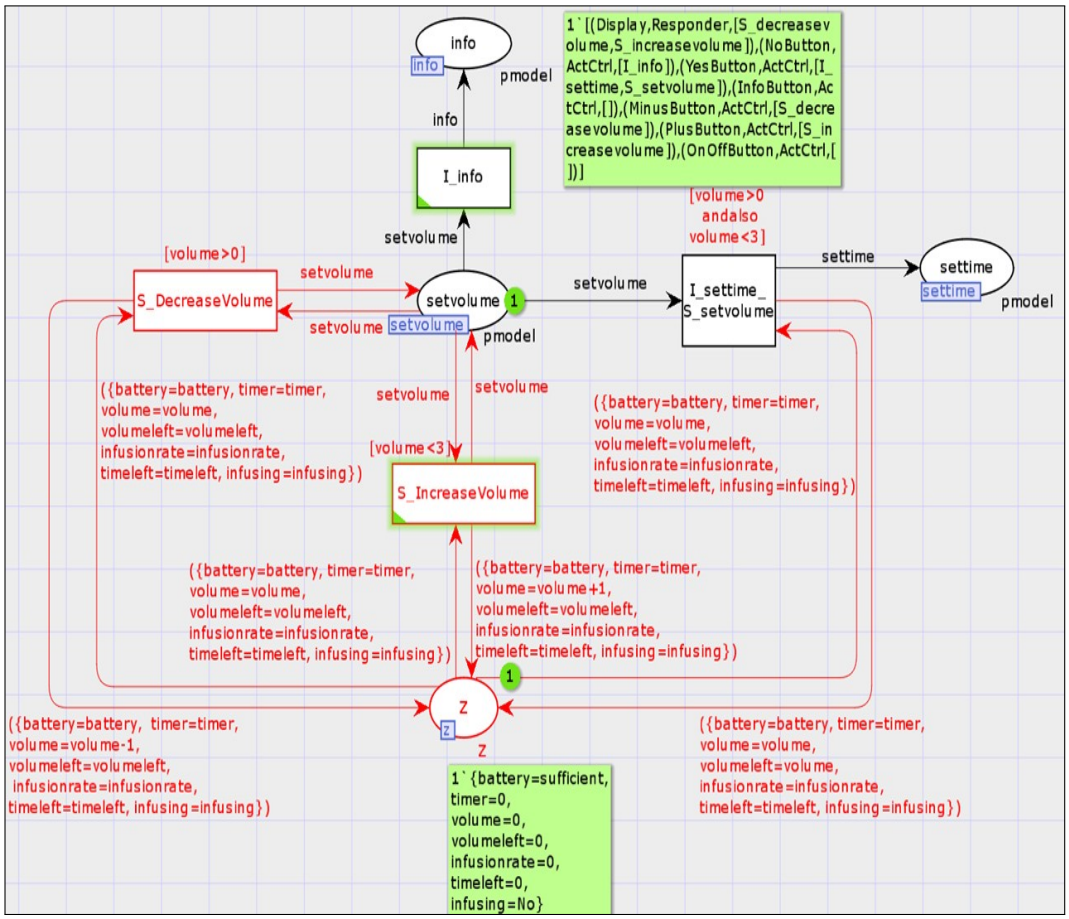

Figure 5.7: SetVolume page

The operations *IncreaseVolume* and *DecreaseVolume* allow the user to set a dose to be infused using the up and down keys. The Z operation schema *IncreaseVolume* is:

IncreaseVolume

$\Delta SimpleInfusionPump$

$battery' = battery$

$timer' = timer$

$volume' = volume + 1$

$volumeleft' = volumeleft$

$infusionrate' = infusionrate$

$timeleft' = timeleft$

$infusing' = infusing$

According to the rule 6, input and output arcs are added to and from the fusion place Z to the *S_IncreaseVolume* transition.

As per rule 8, the expression from the fusion place Z to the transition *S_IncreaseVolume* simply contains assignments which set each variable to its current value (where the variables are the ones that model the observations from the Z operation schema *S_IncreaseVolume* where they will appear on the left of each equation in primed form). This set of assignments "picks up" the current values of the variables ready to be used by the second arc, i.e. an input arc to the place Z from the same *S_IncreaseVolume* transition.

As per rule 9, this second arc, the one to the place Z from the same *S_IncreaseVolume* transition, assigns each variable to its new value, as given by the right-hand side of each equation in the *IncreaseVolume* operation schema. Taken together these two arcs express the intent of the equations in the operation schema *IncreaseVolume*.

Table 5.3 shows the relevant arc expression for the second arc, and that should be compared with the predicate part of the *IncreaseVolume* schema. This arc expression follows from the rules for forming arc expressions as given in Definition 5.3.1.

Arc	Arc Expression
`S_IncreaseVolume to Z`	({battery=battery, timer=timer, volume=volume+1, volumeleft=volumeleft, infusionrate=infusionrate, timeleft=timeleft, infusing=infusing})

Table 5.3: Arc Expression S_IncreaseVolume to Z

When the transition *S_IncreaseVolume* is enabled and fired (as per Definitions 3.6.4 and 3.6.6), the new value of the observation *volume* will be old value of the observation *volume* plus one.

We can also add guards to the transitions. If we want that the transition *S_IncreaseVolume* should not be enabled if the value of the observation *volume* is greater than or equal to three, then we can add the guard $[volume < 3]$ to the transition *S_IncreaseVolume* as shown in Figure 5.7.

Similarly, we add *S_DecreaseVolume* transition to the model. Z operation schema *DecreaseVolume* is written below:

DecreaseVolume

$\Delta SimpleInfusionPump$

$battery' = battery$

$timer' = timer$

$volume' = volume - 1$

$volumeleft' = volumeleft$

$infusionrate' = infusionrate$

$timeleft' = timeleft$

$infusing' = infusing$

In Figure 5.7, there is a transition with the name *S_DecreaseVolume* which represents the S-behaviour. The outgoing arc expression from the transition *S_DecreaseVolume* to the place *Z* is the same as the *DecreaseVolume* Z operation schema. The value of the observation *volume* cannot be negative, so the guard $[volume > 0]$ is added to the transition. When this transition fires,

the new value of the observation *volume* will be old value of the observation *volume* minus one.

The third Z operation schema *S_SetVolume* is:

SetVolume

$\Delta SimpleInfusionPump$

$battery' = battery$

$timer' = timer$

$volume' = volume$

$volumeleft' = volume'$

$infusionrate' = infusionrate$

$timeleft' = timeleft$

$infusing' = infusing$

As the behaviours *S_setvolume* and *I_settime* occur simulatneously, so there is one transition with the name *I_settime_S_setvolume*. The guard *[volume>0 andalso volume<=3]* on this transition means that the transition will only be enabled if the value of the observation *volume* is greater than zero and less than or equal to three. The occurrence of this transition makes two changes. First, it will change the state from *setvolume* to *settime*. Second, the new value of the observation *volumeleft* will be the new value of the observation *volume*.

Now let's have a look at enabling and occurrence of a step as per Definitions 3.6.4 and 3.6.6 in the *setvolume* page. In the Figure 5.7, two transitions: *I_info* and *S_IncreaseVolume* are concurrently enabled as there exists bindings for the variables (in the guard and the surrounding arc expressions). We can see in Figure 5.7 that the the transition *S_IncreaseVolume* uses a same *setvolume* token as the *I_info* transition. One of these transitions can occur in the current state as shown in Figure 5.7. Also the transition *S_IncreaseVolume* is

concurrently enabled with itself. The enabled step looks as follow:

$$S = 1`(I\_info, \langle\rangle) \ + \ 1`(S\_IncreaseVolume, \langle battery = sufficient, timer = 0,$$
$$volume = 0, infusionrate = 0, timeleft = 0, volumeleft = 0, infusing = No\rangle)$$
$$+ \ 1`(S\_IncreaseVolume, \langle\rangle)$$

The effect of the step is the sum of the effects of the individual binding elements. This means that when the *S_IncreaseVolume* fires, the occurrence of the step S moves a *({battery=sufficient, timer=0, volume=1, infusionrate=0, timeleft=0, volumeleft=0,infusing = No})* token to the place Z and moves a *setvolume* token to the place *setvolume*. The change of marking with the occurrence of step S is shown in Figure 5.8.

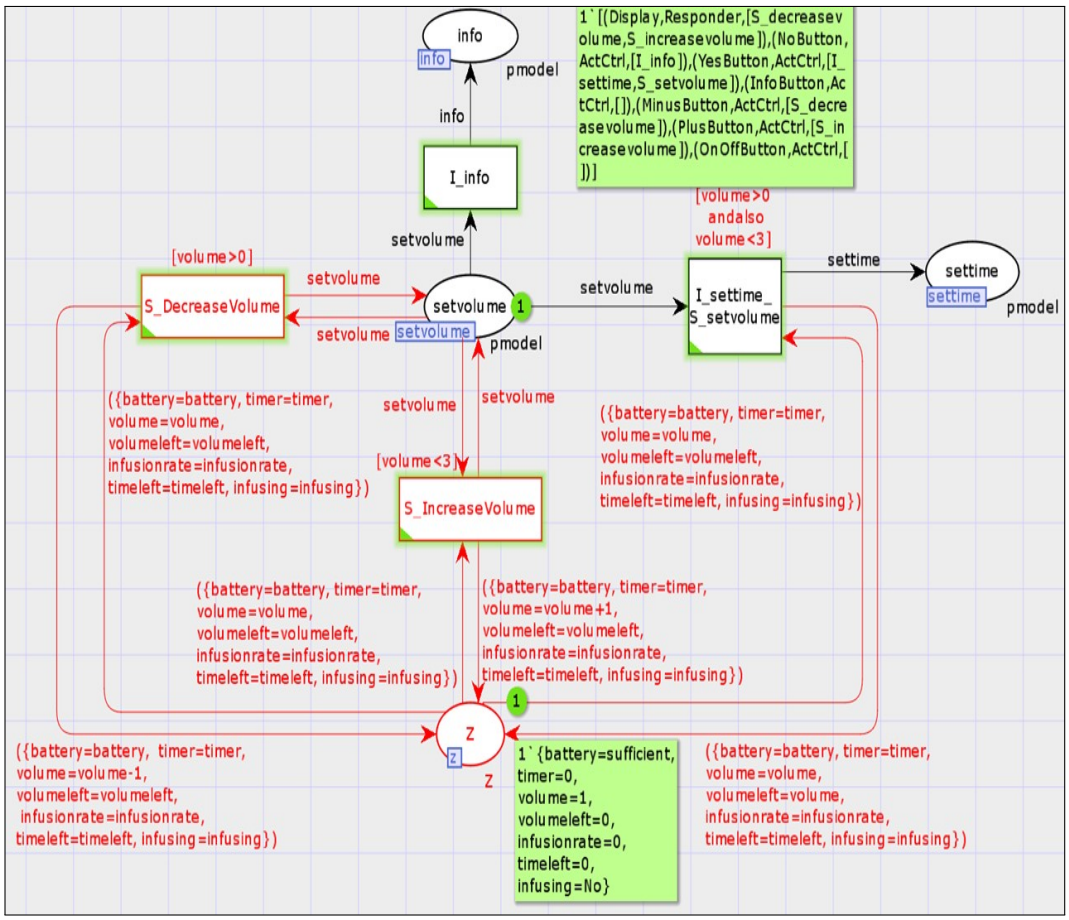

Figure 5.8: SetVolume page after occurrence of *S_IncreaseVolume* transition

In Figure 5.8, four transitions: *I_info, S_DecreaseVolume, S_IncreaseVolume*

and *I_settime_S_setvolume* are enabled concurrently. Also transitions *S_Increase Volume, S_DecreaseVolume* and *I_settime_S_setvolume* are concurrently enabled with themselves. The enabled step looks as follow:

$$
\begin{aligned}
S' = & 1`(I\_info, \langle\rangle) \ + \ 1`(S\_IncreaseVolume, \langle battery = sufficient, timer = 0, \\
& volume = 1, infusionrate = 0, timeleft = 0, volumeleft = 0, infusing = No\rangle) \\
& + \ 1`(S\_IncreaseVolume, \langle\rangle \ + \ 1`(S\_DecreaseVolume, \langle battery = sufficient, \\
& timer = 0, volume = 1, infusionrate = 0, timeleft = 0, volumeleft = 0, \\
& infusing = No\rangle) \ + \ 1`(S\_DecreaseVolume, \langle\rangle) + \\
& 1`(I\_settime\_S\_setvolume, \langle\rangle) \\
& + \ 1`(I\_settime\_S\_setvolume, \langle battery = sufficient, timer = 0, \\
& volume = 1, infusionrate = 0, timeleft = 0, volumeleft = 0, \\
& infusing = No\rangle)
\end{aligned}
$$

This means that when the transition *I_info* fires, it moves a token *info* to the place *info*. When the transition *S_IncreaseVolume* fires, the occurrence of the step S' moves a *({battery=sufficient, timer=0, volume=2, infusionrate=0, timeleft=0, volumeleft=0,infusing = No})* token to the place Z and moves a *setvolume* token to the place *setvolume*. When the transition *S_DecreaseVolume* fires, the occurrence of the step S' moves a *({battery=sufficient, timer=0, volume=0, infusionrate=0, timeleft=0, volumeleft=0,infusing = No})* token to the place Z and moves a *setvolume* token to the place *setvolume*. When the transition *I_settime_S_setvolume* fires, it moves a token *({battery=sufficient, timer=0, volume=1, infusionrate=0, timeleft=0, volumeleft=0,infusing = No})* to the place Z and also moves a token *settime* to the place *settime*. The effect of the change of marking by firing the transition *I_settime_S_setvolume* is shown in Figure 5.9.

SetTime Page

SetTime page allows user to set the duration for infusion. We will extend the model as shown in Figure 4.5. The structure of the *settime* page with

S-behaviours added to it is shown in Figure 5.9. The marking of the *settime* place shows the *settime* component presentation model which has three S-behaviours: *S_decreasetime*, *S_increasetime* and *S_settime*. According to rule 7 in section 5.2.1, *S_decreasetime* and *S_increasetime* are added as transitions with the same name. As the transitions *S_SetTime* and *I_confirmrate* occur simultaneously, we have single transition for both. Occurrence of these transitions changes the value of observations shown by the marking of the place *Z*.

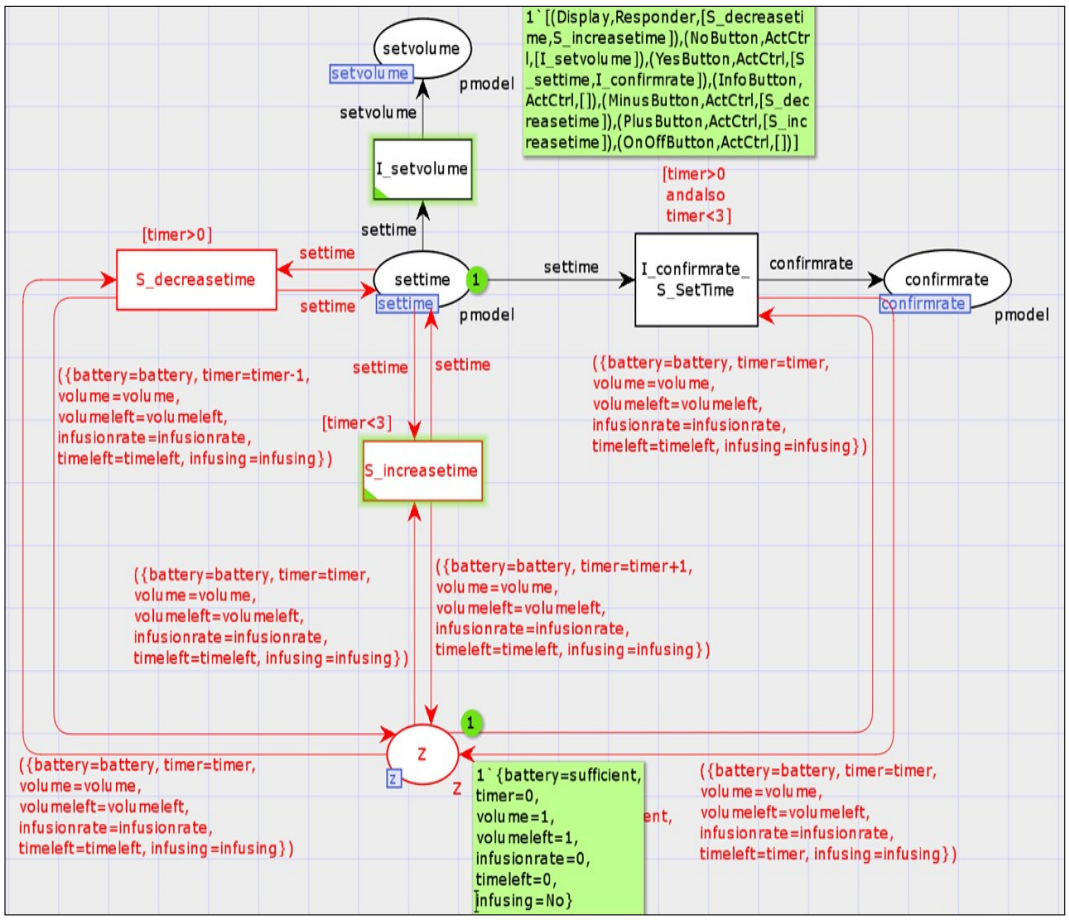

Figure 5.9: SetTime page

ConfirmRate Page

Figure 5.10 shows the structure of the *ConfirmRate* page. This is the extension of the page as shown in Figure 4.6. The marking of the place *confirmrate* represents the *confirmrate* component presentation model. The marking has just one S-behaviour named *S_setrate*. As the behaviours *S_setrate* and *I_infuse*

occur simultaneously, there is one transition with the name *I_infuse_S_SetRate*.

The Z operation schema for *S_SetRate* is:

SetRate

$\Delta SimpleInfusionPump$

$battery' = battery$

$timer' = timer$

$volume' = volume$

$volumeleft' = volumeleft$

$infusionrate' = volume' \mathsf{div} timer'$

$timeleft' = timeleft$

$infusing' = Yes$

The outgoing arc expression from the transition *I_infuse_S_SetRate* to the place *Z* is the same as the *SetRate* Z operation schema. When this transition fires, the new value of the observation *infusionrate* will be value of the observation *volume* divided by value of the observation *timer* and the value of the observation *infusing* will be changed to *Yes*. This change is shown by the marking on the place *Z*.

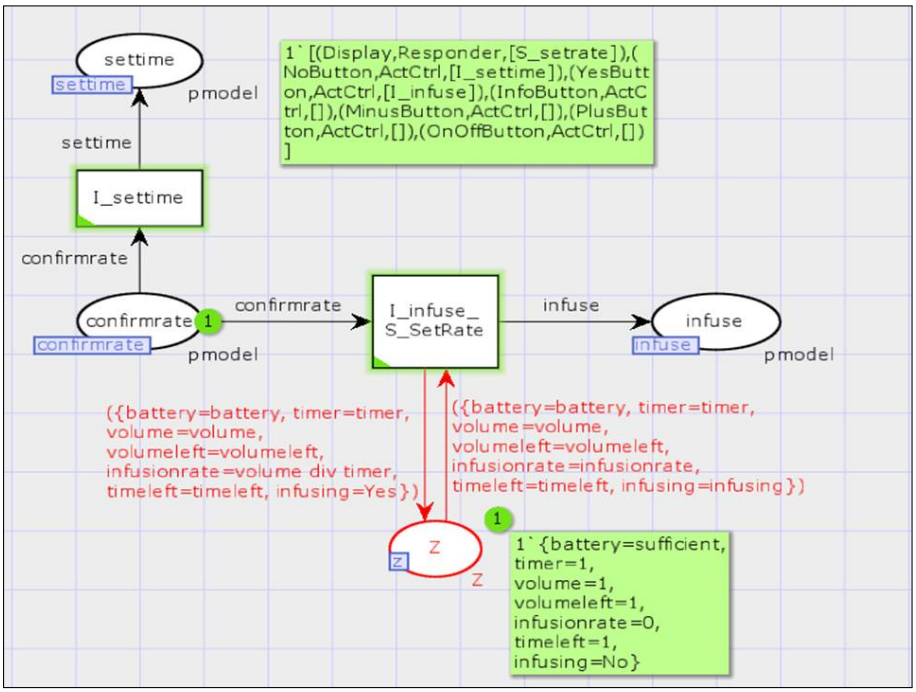

Figure 5.10: ConfirmRate page

In the *confirmrate* page also there are two transitions: *I_settime* and *I_infuse_S_SetRate* that are concurrently enabled as there exists bindings for the variables (in the guard and the surrounding arc expressions). The enabled step looks as follows:

$$S = 1`(I\_settime, \langle\rangle) \; + \; 1`(I\_infuse\_S\_SetRate, \langle battery = sufficient, timer = 1, volume = 1, volumeleft = 1, infusionrate = 0, timeleft = 0, infusing = No\rangle) \; + \; 1`(I\_infuse\_S\_SetRate, \langle\rangle)$$

This means that if the transition *I_settime* fires, the occurrence of the step S moves a *settime* token to the place *settime*. If the transition *I_infuse_S_SetRate* fires, then the occurrence of the step S moves a *({battery=sufficient, timer=1, volume=1, volumeleft=1, infusionrate=0, timeleft=0, infusing = No})* token to the place Z and moves a *infuse* token to the place *infuse*. The effect of the occurrence of the transition *I_infuse_S_SetRate* is shown in Figure 5.11.

Infuse Page

The structure of the *Infuse* page is shown in Figure 5.11. This is the last state of the simple infusion pump. The marking of the place *Infuse* shows the *Infuse* component presentation model. The marking shows that there is just one S-behaviour named *S_displayinfusingmsg* which displays the infusing message on the display.

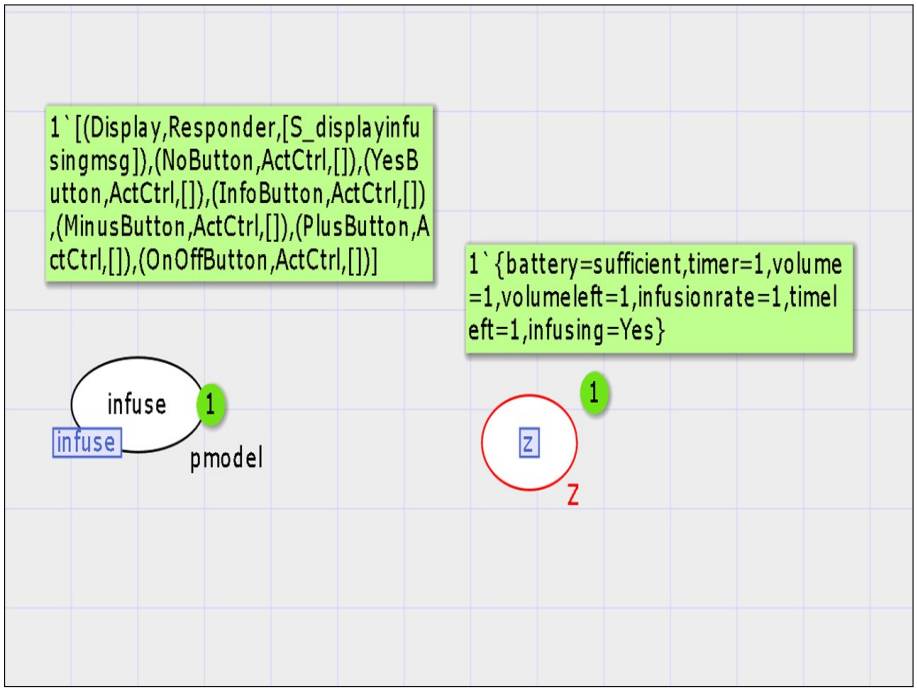

Figure 5.11: Infuse page

5.5 Analysis

In this section we will investigate the same general properties as discussed in Section 4.4 for the complete model now.

The detailed state space report of CPN model of simple infusion pump with all the three aspects (user interface, interaction and functionality) is given in Appendix A.3 and is summarized in table 5.4

State Space Graph	
Number of Nodes	654
Number of Arcs	1399
SCC Graph	
Number of Nodes	12
Number of Arcs	21
Number of Dead Markings	9 (Nodes [86,77,59,42,212,165,141,128,104])
Dead Transition Instances	None
Live Transition Instances	None

Table 5.4: statistics of the CPN model of user interface and interaction and functionality of simple infusion pump

5.5.1 Investigation of General Properties

5.5.1.1 Deadlock freedom

In this section we again investigate the CPN model for the absence of deadlock. We user the same method for checking the absence of deadlock as described in Section 4.4.2.1. Table 5.5 shows the dead markings in the model.

```
ListDeadMarkings()
output:
val it = [86,77,59,42,212,165,141,128,104]:  Node list
```

Table 5.5: Dead markings of the simple infusion pump model

As we know that infusing is the expected terminal marking in our case, we need to apply a function given in Table 5.6 that detects the dead markings that are not intended (i.e. not terminal markings) for our model.

```
fun ValidTerminalMarking n = (Mark.infuse'infuse 1 n =
1‘[(Display, Responder, [S_displayinfusingmsg]),
(NoButton, ActCtrl, []),
(YesButton, ActCtrl, []),
(InfoButton, ActCtrl, []),
(MinusButton, ActCtrl, []),
(PlusButton, ActCtrl, []),
(OnOffButton, ActCtrl, [])] )
fun InValidTerminal()=PredNodes(ListDeadMarkings(),
fn n => not (ValidTerminalMarking n),
NoLimit);
```

```
output:
val it =[]:  Node list
```

Table 5.6: Invalid terminal nodes for simple infusion pump model detecting deadlock

In this case also we have an empty list as value i.e. all the dead markings are terminal markings. Hence there are no deadlocks.

5.5.1.2 Livelock

We will use the same method as described in Section 4.4.2.2 to detect a livelock in this CPN model of a simple infusion pump.

Firstly, we need to check if there exist self-loop terminals in state space graph. We apply the same function given in Table 4.10 to the CPN model of the simple infusion pump and get an empty output list as shown in Table 5.7. This means that there are no nodes in the state space graph of the simple infusion pump that end in self loops.

```
fun SelfLoopNodes n = (OutNodes (n) = [n])
fun SelfLoopTest()=PredNodes(EntireGraph,
fn n => (SelfLoopNodes n),
NoLimit);
```

```
Output:
val it =[] :  Node list
```

Table 5.7: Detecting the absence of self loops in Simple Infusion Pump model

Secondly, we need to check if the state space graph is isomprphic to its

SCC graph. In our example, the number of nodes and arcs in the state space graph of the simple infusion are not equal to the number of nodes and arcs in the SCC graph as shown in the Table 5.4 and also the strongly connected components consist of more than one node. Hence, we cannot say that the state space graph is isomorphic to its SCC graph. Therefore we cannot prove the absence of livelock with this method. So there is a need to check the if all terminal SCCs are trivial. Now we apply the function given in Table 4.12 to the CPN model of a simple infusion pump which results in an empty list which means that all terminal SCCs are trivial as shown in Table 5.8. Hence, the model is free of livelock.

```
fun ListTerminalSCCs()=PredAllSccs(SccTerminal);
fun Livelock()=PredSccs(ListTerminalSCCs(),
fn n => not (SccTrivial n),
NoLimit);
```

```
Output:
val it=[ ] :  Scc list
```

Table 5.8: Query to find a non-trivial terminal SCC in Simple Infusion Pump

5.5.1.3 Reachability

As discussed in Section 4.4.2.3, reachability functions in CPN Tool [119] allow checking reachability of one node from another. With the help of the reachability functions we can investigate if all behaviours described in the presentation models can be obtained by a user by applying some sequence of commands at some point in their interaction.

Suppose we wish to check the CPN model of the simple infusion pump for the following:

- Is it possible to reach the *setvolume* state from the *info* state and vice versa?
- Is it possible to reach the *setvolume* state from the *infusing* state?
- Is it possible to reach the *init* state from the *info* state and vice versa?

- Is it possible to reach the *setvolume* state from the *settime* state and vice versa?
- Is it possible to reach the *confirmrate* state from the *settime* state and vice versa?
- Is it possible to reach the *infusing* state from the *confirmrate* state and vice versa?

1. **Is it possible to reach the *setvolume* state from the *info* state?** To check whether we can reach the *setvolume* state from the *info* state we first need to find out the setvolume and info nodes from the state space graph. The functions given in Tables 5.9 and 5.10 find the setvolume and info nodes respectively.

```
fun setvolumenodes ()=SearchAllNodes( fn n=>
let
fun setvolumearcs(nil)=false |
setvolumearcs(a ::  al :  Arc list) =
if ArcToTI a = TI.info'I_setvolume 1 orelse
ArcToTI a = TI.settime'I_setvolume 1 then
true
else
setvolumearcs(al);
in
setvolumearcs(InArcs(n))
end, fn n => n, [], op::);
```

```
Output:
val it=[97,96,87,84,82,78,75,74,70,68,649,637,634,619,610,608,
605,600,60,6,587,579,578,576,57,562,561,56,559,556,552,549,
546,54,539,533,532,529,516,515,514,512,505,488,487,485,480,
48,478,475,472,47,468,465,462,457,456,455,452,445,43,429,428,
427,425,420,419,417,413,403,40,4,394,393,391,39,388,387,385,
383,378,375,373,370,365,364,363,360,36,356,355,352,348,338,
330,329,328,323,322,321,319,318,316,311,31,307,306,304,30,3,
296,295,291,290,29,289,287,284,282,281,279,272,271,270,265,
264,263,261,260,26,257,252,249,248,245,238,237,232,231,230,
229,224,223,221,220,22,219,217,210,21,206,205,202,200,20,199,
197,191,189,185,184,183,182,177,176,174,173,172,17,169,163,
162,158,157,153,151,150,149,142,139,135,134,131,126,125,121,
12,119,116,114,113,112,105,102,101,10]:  Node list
```

Table 5.9: Returns all setvolume nodes in the state space graph

```
fun infonodes ()=SearchAllNodes( fn n=>
let
fun infoarcs(nil)=false |
infoarcs(a ::  al :  Arc list) =
if ArcToTI a = TI.init'I_info 1 orelse
ArcToTI a = TI.setvolume'I_info 1
then
true
else
infoarcs(al);
in
infoarcs(InArcs(n))
end, fn n => n, [], op::);
```

```
output:
val it=[95,94,92,9,83,81,73,69,67,653,648,646,641,64,636,633,
631,628,622,618,617,615,609,607,604,602,599,597,595,591,588,
586,583,577,575,574,572,568,560,558,555,551,55,548,545,543,
541,538,536,531,530,53,528,525,521,52,513,511,510,508,504,503,
501,5,498,493,486,484,482,479,477,474,471,467,464,461,46,459,
454,453,451,448,446,444,441,438,433,426,424,423,418,416,415,
412,411,409,406,402,401,399,392,390,386,384,382,380,38,377,
374,372,37,369,362,361,359,354,353,351,35,349,347,344,341,
339,337,334,327,326,320,317,315,314,310,309,305,303,302,300,
294,288,286,283,280,28,278,276,269,267,262,259,258,256,253,
251,247,246,244,241,236,235,228,227,222,218,216,215,209,204,
2,198,196,194,190,19,188,181,179,175,171,170,168,161,156,155,
15,148,147,138,133,130,124,120,118,111,109,100]:  Node list
```

Table 5.10: Returns all info nodes in the state space graph

Now to check whether the *setvolume* node is reachable from the *info* node, we evaluate the function given in Table 5.11 which evaluates to *true* which means that the *setvolume* node 97 is reachable from the *info* node 95.

```
Reachable(95,97)
```

```
output:
val it=true:  bool
```

Table 5.11: Reachability function to test reachability from *info* to *setvolume* node

If we want to know the path to reach the *setvolume* node from the *info*

node, the function given in Table 5.12 is evaluated which tells us the path from the *info* node to the *setvolume* node.

```
Reachable'(95,97)
output:
A path from node 99 to node 95 is:[95,57,31,18,32,58,97]
val it=true:  bool
```

Table 5.12: Reachability function to find the path from *info* to *setvolume*

Reachability functions provided by the CPN Tool [36] allow checking reachability of one node from another. In order to check multiple nodes, the function has to be invoked several times. This is inefficient when there are as many nodes as we are getting in this example because we have to repeat the process for all pairs. In such cases where there are large number of nodes, we take advantage of the reachability algorithm given in [125] and is explained in Appendix B.1.

Reachability from the *info* state to the *setvolume* state is verified using the CPN-ML code written in Table 5.13. The function *infotosetvolume*, returns the list of SCC nodes which contains *info* nodes that cannot reach the *setvolume* nodes. The function *setvolumenodes* (line 1 to 12) returns the SCC nodes that includes atleast one state space graph node of the occurrence of the transitions $info'I\_setvolume$ (I_setvolume transition on page info) and $settime'I\_setvolume$ (I_setvolume transition on page settime). The function *infonodes*(line 14 to 26) returns the SCC nodes that includes atleast one state space graph node of the occurrence of the transitions $init'I\_info$ (I_info transition on page init) and $setvolume'I\_info$ (I_info transition on page setvolume). Each of the function *INF* (line 31) and SV (line 32) is a predicate that checks if a SCC node is a member of the list returned by the functions *infonodes* and *setvolumenodes* respectively. The *Reachable* function (line 33) checks if the *info* nodes can reach the *setvolume* nodes. The SCC nodes, which includes at least a node that does not meet this condition, are returned in the $r1$ list (line

35). We get the empty list as an output which means that all the *info* nodes can reach the *setvolume* nodes.

```
1   fun setvolumenodes()=SearchAllNodes( fn n=>
2   let
3   fun setvolumearcs(nil)=false |
4   setvolumearcs(a ::  al :  Arc list) =
5   if ArcToTI a = TI.info'I_setvolume 1 orelse
6   ArcToTI a = TI.settime'I_setvolume 1 then
7   true
8   else
9   setvolumearcs(al);
10  in
11  setvolumearcs(InArcs(n))
12  end, fn n => NodeToScc(n), [], Insert);
13
14  fun infonodes()=SearchAllNodes( fn n=>
15  let
16  fun infoarcs(nil)=false |
17  infoarcs(a ::  al :  Arc list) =
18  if ArcToTI a = TI.init'I_info 1 orelse
19  ArcToTI a = TI.setvolume'I_info 1
20  then
21  true
22  else
23  infoarcs(al);
24  in
25  infoarcs(InArcs(n))
26  end, fn n => NodeToScc(n), [], Insert);
27
28  fun infotosetvolume()= let
29  val inf = infonodes();
30  val setvol = setvolumenodes();
31  fun INF x = ListMember(x, inf);
32  fun SV x = ListMember(x, setvol);
33  val(v1,r1,m1)=Reachable([~1], INF, SV, [], [],0);
34  in
35  r1
36  end
```

```
    output:
    val it= []:  Scc list
```

Table 5.13: Code to find the reachability from *info* state to *setvolume* state

2. **Is it possible to reach the *setvolume* state from the *infusing***

state?

We can verify this by checking the reachability from the *infusing* state to the *setvolume* state. The code to check the reachability from *infusing* to *setvolume* is given in Table 5.14.

The function *infusingtosetvolume*, returns the list of SCC nodes which contains *infusing* nodes that cannot reach the *setvolume* nodes. The function *setvolumenodes*(lines 1 to 12) returns the SCC nodes that includes atleast one state space graph node of the occurrence of the transitions *info'I_setvolume* (I_setvolume transition on page info) and *settime'I_setvolume* (I_setvolume transition on page settime). The function *infusingnodes*(lines 14 to 25) returns the SCC nodes that includes atleast one state space graph node of the occurrence of the transition *confirmrate'I_infuse_S_SetRate* (I_infuse_S_SetRate transition on page confirmrate). Each of the function *INFU* (line 30) and *SV* (line 31) is a predicate that checks if a SCC node is a member of the list returned by the functions *infusingnodes* and *setvolumenodes* respectively. The *Reachable* function (line 32) checks if the *infusing* nodes can reach the *setvolume* nodes. The SCC nodes, which includes at least a node that does not meet this condition, are returned in the $r1$ list (line 34). We get nine SCC nodes (~4, ~5, ~6, ~7, ~8, ~9. ~10, ~11, ~12) in the $r1$ list as an output which means that the *infusing* nodes cannot reach the *setvolume* nodes. This is how the system is expected to behave as while infusing we don't want to allow user to make any changes in the volume to be infused due to safety reasons.

```
1   fun setvolumenodes()=SearchAllNodes( fn n=>
2   let
3   fun setvolumearcs(nil)=false |
4   setvolumearcs(a ::  al :  Arc list) =
5   if ArcToTI a = TI.info'I_setvolume 1 orelse
6   ArcToTI a = TI.settime'I_setvolume 1 then
7   true
8   else
9   setvolumearcs(al);
10  in
11  setvolumearcs(InArcs(n))
12  end, fn n => NodeToScc(n), [], Insert);
13
14  fun infusingnodes ()=SearchAllNodes( fn n=>
15  let
16  fun infusingarcs(nil)=false |
17  infusingarcs(a ::  al :  Arc list) =
18  if ArcToTI a = TI.confirmrate'I_infuse_S_SetRate 1
19  then
20  true
21  else
22  infusingarcs(al);
23  in
24  infusingarcs(InArcs(n))
25  end, fn n => NodeToScc(n), [], Insert);
26
27  fun infusingtosetvolume()= let
28  val infuse = infusingnodes();
29  val setvol = setvolumenodes();
30  fun INFU x = ListMember(x, infuse);
31  fun SV x = ListMember(x, setvol);
32  val(v1,r1,m1)=Reachable([~1], INFU, SV, [], [],0);
33  in
34  r1
35  end
```

```
    output:
    val it= [~4, ~5, ~6, ~7, ~8, ~9.  ~10, ~11, ~12]:  Scc list
```

Table 5.14: Code to find the reachability from *infusing* state to *setvolume* state

3. **Is it possible to reach the *init* state from the *info* state and vice versa?**

   To check the reachability from the *init* state to the *info* state and vice versa, we first need to find the *info* nodes and the *init* nodes from the

state space graph. The function *initnodes* (line 1 to 11) in Table 5.15 returns the SCC nodes that includes the *init* nodes from the state space graph. The function *infonodes* (line 13 to 25) returns the SCC nodes that includes the *info* nodes from the state space graph.

The function *inittoinfonodes* (line 27-35) in Table 5.15, returns the list of SCC nodes which contains the *init* nodes that cannot reach the *info* nodes. We get an empty list as an output which means that all the *init* nodes can reach *info* nodes as expected.

The function *infotoinitnodes* (line 1-9) in Table 5.16, returns the list of SCC nodes which contains the *info* nodes that cannot reach the *init* nodes. We get an empty list as an output which means that all the *info* nodes can reach *init* nodes as expected.

```
1   fun initnodes()=SearchAllNodes( fn n=>
2   let
3   fun initarcs(nil)=false |
4   initarcs(a ::  al :  Arc list) =
5   if ArcToTI a = TI.info'I_init 1 orelse
6   true
7   else
8   initarcs(al);
9   in
10  initarcs(InArcs(n))
11  end, fn n => NodeToScc(n), [], Insert);
12
13  fun infonodes ()=SearchAllNodes( fn n=>
14  let
15  fun infoarcs(nil)=false |
16  infoarcs(a ::  al :  Arc list) =
17  if ArcToTI a = TI.init'I_info 1 orelse
18  ArcToTI a = TI.setvolume'I_info 1
19  then
20  true
21  else
22  infoarcs(al);
23  in
24  infoarcs(InArcs(n))
25  end, fn n => NodeToScc(n), [], Insert);
26
27  fun inittoinfonodes()= let
28  val init = initnodes();
29  val info = infonodes();
30  fun INIT x = ListMember(x, init);
31  fun INFO x = ListMember(x, info);
32  val(v1,r1,m1)=Reachable([~1], INIT, INFO, [], [],0);
33  in
34  r1
35  end
```

```
    output:
    val it= []:  Scc list
```

Table 5.15: Code to find the reachability from *init* state to *info* state

```
1  fun infotoinitnodes()= let
2  val init = initnodes();
3  val info = infonodes();
4  fun INIT x = ListMember(x, init);
5  fun INFO x = ListMember(x, info);
6  val(v1,r1,m1)=Reachable([~1], INFO, INIT, [], [],0);
7  in
8  r1
9  end
```

```
output:
val it= []:  Scc list
```

Table 5.16: Code to find the reachability from *info* state to *init* state

4. **Is it possible to reach the *setvolume* state from the *settime* state and vice versa?**

   To check the reachability from the *setvolume* state to the *settime* state and vice versa, we first need to find the *setvolume* nodes and *settime* nodes from the state space graph. The function *setvolumenodes* (line 1 to 12) in Table 5.17 returns the SCC nodes that includes the *setvolume* nodes from the state space graph. The function *settimenodes* (line 14 to 26) returns the SCC nodes that includes the *settime* nodes from the state space graph.

   The function *setvolumetosettimenodes* (line 28-36) in Table 5.17, returns the list of SCC nodes which contains the *setvolume* nodes that cannot reach the *settime* nodes. We get an empty list as an output which means that all the *setvolume* nodes can reach *settime* nodes as expected.

   The function *settimetosetvolumenodes* (line 1-9) in Table 5.18, returns the list of SCC nodes which contains the *settime* nodes that cannot reach the *setvolume* nodes. We get an empty list as an output which means that all the *settime* nodes can reach *setvolume* nodes as expected.

```
1   fun setvolumenodes()=SearchAllNodes( fn n=>
2   let
3   fun setvolumearcs(nil)=false |
4   setvolumearcs(a ::  al :  Arc list) =
5   if ArcToTI a = TI.info'I_setvolume 1 orelse
6   ArcToTI a = TI.settime'I_setvolume 1 then
7   true
8   else
9   setvolumearcs(al);
10  in
11  setvolumearcs(InArcs(n))
12  end, fn n => NodeToScc(n), [], Insert);
13
14  fun settimenodes()=SearchAllNodes( fn n=>
15  let
16  fun settimearcs(nil)=false |
17  settimearcs(a ::  al :  Arc list) =
18  if ArcToTI a = TI.setvolume'I_settime_S_setvolume orelse
19  ArcToTI a = TI.confirmrate'I_settime 1
20  then
21  true
22  else
23  settimearcs(al);
24  in
25  settimearcs(InArcs(n))
26  end, fn n => NodeToScc(n), [], Insert);
27
28  fun setvolumetosettimenodes()= let
29  val settime = settimenodes();
30  val setvolume = setvolumenodes();
31  fun ST x = ListMember(x, settime);
32  fun SV x = ListMember(x, setvolume);
33  val(v1,r1,m1)=Reachable([~1], SV, ST, [], [],0);
34  in
35  r1
36  end
```

```
    output:
    val it= []:  Scc list
```

Table 5.17: Code to find the reachability from *setvolume* state to *settime* state

```
1  fun settimetosetvolumenodes()= let
2  val settime = settimenodes();
3  val setvolume = setvolumenodes();
4  fun ST x = ListMember(x, settime);
5  fun SV x = ListMember(x, setvolume);
6  val(v1,r1,m1)=Reachable([~1], ST, SV, [], [],0);
7  in
8  r1
9  end
```

```
output:
val it= []:  Scc list
```

Table 5.18: Code to find the reachability from *settime* state to *setvolume* state

5. **Is it possible to reach the *confirmrate* state from the *settime* state and vice versea**

   To check the reachability from the *confirmrate* state to the *settime* state and vice versa, we first need to find the *confirmrate* nodes and the *settime* nodes from the state space graph. The function *confirmrate* (line 1 to 11) in Table 5.19 returns the SCC nodes that includes the *confirmrate* nodes from the state space graph. The function *settimenodes* (line 13 to 25) returns the SCC nodes that includes the *settime* nodes from the state space graph.

   The function *confirmratetosettimenodes* (line 27-35) in Table 5.19, returns the list of SCC nodes which contains the *confirmrate* nodes that cannot reach the *settime* nodes. We get an empty list as an output which means that all the *confirmrate* nodes can reach *settime* nodes as expected.

   The function *settimetoconfirmratenodes* (line 1-9) in Table 5.20, returns the list of SCC nodes which contains the *settime* nodes that cannot reach the *confirmrate* nodes. We get an empty list as an output which means that all the *settime* nodes can reach *confirmrate* nodes as expected.

```
1   fun confirmratenodes()=SearchAllNodes( fn n=>
2   let
3   fun confirmratearcs(nil)=false |
4   confirmratearcs(a ::  al :  Arc list) =
5   if ArcToTI a = TI.settime'I_confirmrate_S_SetTime 1
6   true
7   else
8   confirmratearcs(al);
9   in
10  confirmratearcs(InArcs(n))
11  end, fn n => NodeToScc(n), [], Insert);
12
13  fun settimenodes()=SearchAllNodes( fn n=>
14  let
15  fun settimearcs(nil)=false |
16  settimearcs(a ::  al :  Arc list) =
17  if ArcToTI a = TI.setvolume'I_settime_S_setvolume orelse
18  ArcToTI a = TI.confirmrate'I_settime 1
19  then
20  true
21  else
22  settimearcs(al);
23  in
24  settimearcs(InArcs(n))
25  end, fn n => NodeToScc(n), [], Insert);
26
27  fun confirmratetosettimenodes()= let
28  val settime = settimenodes();
29  val confirmrate = confirmratenodes();
30  fun ST x = ListMember(x, settime);
31  fun CR x = ListMember(x, confirmrate);
32  val(v1,r1,m1)=Reachable([~1], CR, ST, [], [],0);
33  in
34  r1
35  end
```

```
    output:
    val it= []:  Scc list
```

Table 5.19: Code to find the reachability from *confirmrate* state to *settime* state

```
1  fun settimetoconfirmratenodes()= let
2  val settime = settimenodes();
3  val confirmrate = confirmratenodes();
4  fun ST x = ListMember(x, settime);
5  fun CR x = ListMember(x, confirmrate);
6  val(v1,r1,m1)=Reachable([~1], ST, CR, [], [],0);
7  in
8  r1
9  end
```

```
output:
val it= []:  Scc list
```

Table 5.20: Code to find the reachability from *settime* state to *confirmrate* state

6. **Is it possible to reach the *infusing* state from the *confirmrate* state and vice versa?**

   To check the reachability from the *confirmrate* state to the *infusing* state and vice versa, we first need to find the *confirmrate* nodes and the *infusing* nodes from the state space graph. The function *confirmrate* (line 1 to 11) in Table 5.21 returns the SCC nodes that includes the *confirmrate* nodes from the state space graph. The function *infusingnodes* (line 13 to 24) returns the SCC nodes that includes the *infusing* nodes from the state space graph.

   The function *infusingtoconfirmratenodes* (line 26-34) in Table 5.21, returns the list of SCC nodes which contains the *infusing* nodes that cannot reach the *confirmrate* nodes. We get nine SCC nodes (~4, ~5, ~6, ~7, ~8, ~9. ~10, ~11, ~12) in r1 list as an output which means that all the *infusing* nodes cannot reach the *confirmrate* nodes as expected.

   The function *confirmratetoinfusingnodes* (line 1-9) in Table 5.22, returns the list of SCC nodes which contains the *confirmrate* nodes that cannot reach the *infusing* nodes. We get an empty list as an output which means that all the *confirmrate* nodes can reach the *infusing* nodes as expected.

```
1   fun confirmratenodes()=SearchAllNodes( fn n=>
2   let
3   fun confirmratearcs(nil)=false |
4   confirmratearcs(a ::  al :  Arc list) =
5   if ArcToTI a = TI.settime'I_confirmrate_S_SetTime 1
6   true
7   else
8   confirmratearcs(al);
9   in
10  confirmratearcs(InArcs(n))
11  end, fn n => NodeToScc(n), [], Insert);
12
13  fun infusingnodes ()=SearchAllNodes( fn n=>
14  let
15  fun infusingarcs(nil)=false |
16  infusingarcs(a ::  al :  Arc list) =
17  if ArcToTI a = TI.confirmrate'I_infuse_S_SetRate 1
18  then
19  true
20  else
21  infusingarcs(al);
22  in
23  infusingarcs(InArcs(n))
24  end, fn n => NodeToScc(n), [], Insert);
25
26  fun infusingtoconfirmratenodes()= let
27  val infuse = infusingnodes();
28  val confirmrate = confirmratenodes();
29  fun INFU x = ListMember(x, infuse);
30  fun CR x = ListMember(x, confirmrate);
31  val(v1,r1,m1)=Reachable([~1], INFU, CR, [], [],0);
32  in
33  r1
34  end
```

```
    output:
    val it= [~4, ~5, ~6, ~7, ~8, ~9.  ~10, ~11, ~12]:  Scc list
```

Table 5.21: Code to find the reachability from *infusing* state to *confirmrate* state

```
1  fun confirmratetoinfusingnodes()= let
2  val infuse = infusingnodes();
3  val confirmrate = confirmratenodes();
4  fun INFU x = ListMember(x, infuse);
5  fun CR x = ListMember(x, confirmrate);
6  val(v1,r1,m1)=Reachable([~1], CR, INFU, [], [],0);
7  in
8  r1
9  end
```

```
output:
val it= []:  Scc list
```

Table 5.22: Code to find the reachability from *confirmrate* state to *infusing* state

5.5.1.4 Checking Values

It is also important to check the ranges of some values in a model. For example, limits on the volume to be infused or limits on the time of the infusion. The state space method allows us to check if there exist values which violate these limits. For the simple infusion pump, we have a guard $volume > 0$ *andalso* $volume <= 3$ on the transition $I\_setime\_S\_setvolume$. This means that in the entire state space there should not exist an arc where the value of the observation *volume* is more than three. We verify this by writing the code given in Table 5.23 to query the state space to see if there is any marking where the value of *volume* is more than three (we check for $volume = 4$).

We get the empty arc list as an output. This means that there exists no such arc in the entire state space that has a binding element $volume = 4$.

```
fun checkvaluevolume(volume:NAT): Arc list =
PredAllArcs (
fn a=> case ArcToBE a of
Bind.setvolume'I_setime_S_setvolume (1,
{battery=_, infusing=_, infusionrate=_, timeleft=_,
timer=_, volume=4, volumeleft=_})
=> volume = 4 | _ => false)
```

```
Output:
val it=[]:  Arc list
```

Table 5.23: Returns all the arcs in the state space graph where the value of *volume* is 4

5.6 Summary

In this chapter we have discussed how to express Z in Coloured Petri Nets. We have also laid down rules to extend the CPN model of user interface and interaction by adding S-behaviour transitions to it. We get a complete CPN model which has all the three aspects: interface, interaction and functional in a single model. Then we have investigated the behaviours of the CPN model with the help of state space analysis method.

We have explained the technique to build a CPN model of a safety-critical interactive system using a simple infusion pump example. In the next chapter we apply this technique to a real system. We have chosen the Niki T34 syringe driver as it is an example of commonly used safety-critical medical device.

Chapter 6

Case Study: T34 Pump

6.1 Introduction

In Chapters 4 and 5 we have presented a technique to model the user interface, interaction and functionality of safety-critical interactive systems. We illustrated the approach using a simplified infusion pump example. In this chapter we show how this technique is applicable to a real world example. We present a case study on the Niki T34 Syringe driver. We model the user interface, interaction and functionality of the driver using the technique as presented in Chapters 4 and 5.

6.2 Niki T34 Syringe Driver

The T34 is a compact and lightweight syringe driver used to deliver drugs. Figure 6.1 shows an image of the Niki T34 syringe driver. More information about the device can be found in [126].

Figure 6.1: Niki T34 Syringe Driver

6.3 CPN model of a User Interface and Interaction of the Niki T34 Syringe Driver

In this section we model the user-interface and interactivity of the syringe driver using Coloured Petri Nets. We will use the technique described in chapter 4. Modelling of the user-interface and interactivity of the device is carried out using the CPN tool [37].

There are ten widgets in the T34 syringe driver shown in Figure 6.1 : *LeftFFSK, RightBackSK, OnOffButton, UpPlusSK, DownMinusSK, Display, NoStopSK, InfoSK, YesStartSK* and *Timeout*. It also has an audible alarm that start beeping during a timeout warning but that is not a part of this model.

1. Display : This shows messages and other information.

2. Timeout : This is a small operation LED that glows red when a timeout warning goes off and alarm starts beeping.

3. OnOffButton : This button switches the device on and off.

4. NoStopSK : This button either stops the pump or it takes the user one step backwards.

5. YesStartSK : This button confirms choices made.

6. DownMinusSK : This button is used to scroll between options and also to decrease values.

7. UpPlusSK : This button is used to scroll between options and also to increase values.

8. InfoSK : This button activates the technical data and battery status display.

9. RightBackSK : This button is used to move the actuator.

10. LeftBackSK : This button is used to move the actuator.

There are a total of nineteen states, of which the syringe driver can be in exactly one at any given point in time. These states are: *LoadSyringe, Init, BatteryLevel, SetVolume, SetDuration, RateSet, RateConfirm, ConfirmSettings, StartInfusingConfirm, Infusing, Paused, Inittwo, Resume, InfusionStatus, BatteryStatus, EventLog, ChangeSetUp* and *TimeOut*. We can, typically, get to know this by experimenting with the device. We might also read the user manuals, but this is not recommended since user manuals are, worryingly, notoriously unreliable [127]. The nineteen states are explained below:

1. *LoadSyringe* is the initial state of the device in which the user loads the syringe;

2. *Init* is the state after the syringe is loaded into the device and information about the syringe is displayed on the screen;

3. *Info* is the state which gives several options with which the user can change the settings and check status;

4. *BatteryLevel* state will provide information about the available battery level;

5. *SetVolume* is the state which allows the user to set the dose to be infused. The default amount is the maximum size of the syringe. A user can change the volume by pressing the *UpPlusSK* and *DownMinusSK* buttons on the pump;

6. *SetDuration* is the state in which the user can set the duration. The default duration is 24 hours. A user can change it by pressing the *UpPlusSK* and *DownMinusSK* buttons on the pump;

7. *RateSet* allows the user to set the rate at which the drug is infused;

8. *RateConfirm* state will ask the user to confirm the rate before moving further;

9. *ConfirmSettings* will display all the information about the volume, duration and rate and asks the user to confirm the values by pressing the *YesStartSK* button on the device;

10. *StartInfusingConfirm* is the state in which the user is asked for confirmation to start infusing;

11. *Infusing* is the state in which the syringe driver will start infusing the drug into the patient;

12. *Paused* state is the state in which the infusion has been paused;

13. *Inittwo* is the state where the infusion is paused and syringe information is displayed;

14. *Resume* is the state where the infusion can be resumed from where it has stopped;

15. *InfusionStatus* will display the status of the infusion on the display;

16. *BatteryStatus* state will provide information about the available battery level during infusion;

17. *EventLog* is the state in which the user can see the history of the use of the device;

18. *ChangeSetUp* is the state in which the user is able to change certain settings for the device;

19. *TimeOut* is the warning state in which the alarm starts beeping if the device is not in use for a certain amount of time.

6.4 Modelling User Interface and Interaction of Niki T34 Syringe Driver in Coloured Petri Nets

In this section we model the user interface of the Niki T34 syrige driver and give meaning to the I-behaviours. The presentation model is comprised of two parts, declaration and definition, which form the top level for the CPN model of the T34 syringe driver, so we have: *Presentation Model Declaration* and *Presentation Model Definition*. Table 6.1 shows the declaration part of the presentation model of the syringe driver expressed in CPN.

```
(* ========== Presentation Model Declaration ========== *)

colset WidgetName =   with LeftFFSK | RightBackSK | OnOffButton |
                      UpPlusSK | DownMinusSK | Display |
                      NoStopSK | InfoSK | Timeout | YesStartSK;
colset Category =     with ActionControl | MValResponder |
                      System | display;
colset Behaviour =    with S_SyringeWarnings | S_MoveActuatorFwd |
                      S_ArmWarning | S_MoveActuatorBwd |
                      I_Init | I_Info | S_SyringeDisplay |
                      S_ScrollSyringeList | I_SetVolume |
                      S_SelectSyringe | Quit | I_TimeOut |
                      S_InfoList | S_BatteryLevel | S_KeypadLock |
                      S_ScrollInfoListUp | S_ScrollInfoListDown |
                      I_BatteryLevel | S_Init | I_RateSet |
                      I_EventLog | I_ChangeSetUp | S_TimeOut |
                      S_StopAlarm | S_CurrentVolume |
                      S_CurrentSyringe | S_DecreaseVTBI |
                      S_SetVTBI | I_SetDuration | S_IncorrectCode |
                      S_SelectDigit | S_RateCodeAccept |
                      S_IncreaseDuration | S_DecreaseDuration |
                      S_SetNewDuration | I_RateConfirm |
                      S_CurrentRate | I_ConfirmSettings |
                      S_SaveSettings | S_CurrentVTBI |
                      S_CurrentDuration | S_ConfirmMsg |
                      I_StartInfusingConfirm | S_StartInfusingMsg |
                      I_Infusing | S_Infusing | S_TimeRemaining |
                      S_DeliveringMsg | I_InfusionStatus |
                      S_Pause | I_Paused |
                      S_PumpStoppedWarning | S_ResumeInfusing |
                      S_SyringeStatusDisplay | S_PauseInfusion |
                      I_Pause | S_EventLogData | I_EventInfo |
                      S_NextEvent | S_PrevEvent |
                      S_EventInfoDisplay | S_SetUpCodeAccept |
                      I_Resume | S_ResumeConfirm | I_Inittwo;
colset Behaviours =   list Behaviour;
colset widgetdescr =  product WidgetName * Category * Behaviours;
colset pmodel =       list widgetdescr;
```

Table 6.1: Presentation Model Declarations for Niki T34 Syringe Driver

Now we look at the definition part of the presentation model for the Niki T34 syringe driver. As there are nineteen states, the number of component presentation models will be the same. Table 6.2 shows the definition part of

the presentation model of the T34 syringe driver.

```
(* =========== Presentation Model Definition =========== *)

val LoadSyringe =           [(Display, MValResponder,
                            [S_SyringeWarnings]),
                            (InfoSK, ActionControl, [I_Info]),
                            (UpPlusSK, ActionControl, [ ]),
                            (DownMinusSK, ActionControl, [ ]),
                            (YesStartSK, ActionControl,[ ]),
                            (NoStopSK, ActionControl, [ ]),
                            (LeftFFSK, ActionControl,
                            [S_MoveActuatorFwd, S_ArmWarning]),
                            (RightBackSK, ActionControl,
                            [S_MoveActuatorBwd, S_ArmWarning]),
                            (OnOffButton, ActionControl, [Quit]),
                            (Timeout, System, [I_Init])];

val Init =                  [(Display, MValResponder,
                            [S_SyringeDisplay]),
                            (InfoSK, ActionControl, [I_Info]),
                            (UpPlusSK, ActionControl,
                            [S_ScrollSyringeList]),
                            (DownMinusSK, ActionControl,
                            [S_ScrollSyringeList ]),
                            (YesStartSK, ActionControl,
                            [I_SetVolume, S_SelectSyringe]),
                            (NoStopSK, ActionControl, [  ]),
                            (LeftFFSK, ActionControl, [  ]),
                            (RightBackSK, ActionControl, [ ]),
                            (OnOffButton, ActionControl, [Quit]),
```

```
                                (Timeout, System, [I_TimeOut])];

val Info =                      [(Display, MValResponder,
                                [S_InfoList]),
                                (InfoSK, ActionControl,
                                [S_KeypadLock]),
                                (UpPlusSK, ActionControl,
                                [S_ScrollInfoListUp]),
                                (DownMinusSK, ActionControl,
                                [S_ScrollInfoListDown]),
                                (YesStartSK, ActionControl,
                                [I_BatteryLevel, I_Init, I_RateSet,
                                I_EventLog, I_ChangeSetUp]),
                                (NoStopSK, ActionControl, [I_Init]),
                                (LeftFFSK, ActionControl, [ ]),
                                (RightBackSK, ActionControl, [ ]),
                                (OnOffButton, ActionControl,
                                [Quit, I_Init]),
                                (Timeout, System, [ ])];

val Batterylevel =              [(Display, MValResponder,
                                [S_BatteryLevel]),
                                (InfoSK, ActionControl, [I_Info]),
                                (UpPlusSK, ActionControl, [  ]),
                                (DownMinusSK, ActionControl, [ ]),
                                (YesStartSK, ActionControl, [I_Info]),
                                (NoStopSK, ActionControl, [I_Info]),
                                (LeftFFSK, ActionControl, [ ]),
                                (RightBackSK, ActionControl, [ ]),
                                (OnOffButton, ActionControl,
                                [Quit, I_Init]),
```

```
                        (Timeout, System, [ ])];

val TimeOut =           [(Display, MValResponder, [S_TimeOut]),
                        (InfoSK, ActionControl, [I_Info]),
                        (UpPlusSK, ActionControl, [ ]),
                        (DownMinusSK, ActionControl, [ ]),
                        (YesStartSK, ActionControl,[I_Init]),
                        (NoStopSK, ActionControl, [S_StopAlarm]),
                        (LeftFFSK, ActionControl, [ ]),
                        (RightBackSK, ActionControl, [ ]),
                        (OnOffButton, ActionControl, [ ]),
                        (Timeout, System, [ ])];

val SetVolume =         [(Display, MValResponder,
                        [S_CurrentVolume, S_CurrentSyringe]),
                        (InfoSK, ActionControl, [I_Info]),
                        (UpPlusSK, ActionControl,
                        [S_IncreaseVTBI]),
                        (DownMinusSK, ActionControl,
                        [S_DecreaseVTBI]),
                        (YesStartSK, ActionControl,
                        [S_SetVTBI, I_SetDuration]),
                        (NoStopSK, ActionControl, [I_Init]),
                        (LeftFFSK, ActionControl, [ ]),
                        (RightBackSK, ActionControl, [ ]),
                        (OnOffButton, ActionControl,
                        [I_Init, Quit]),
                        (Timeout, System, [ ])];

val RateSet =           [(Display, MValResponder,
                        [S_SelectDigit, S_IncorrectCode]),
```

```
                        (InfoSK, ActionControl, [I_Info]),
                        (UpPlusSK, ActionControl,
                        [S_SelectDigit]),
                        (DownMinusSK, ActionControl,
                        [S_SelectDigit]),
                        (YesStartSK, ActionControl,
                        [S_RateCodeAccept, S_IncorrectCode]),
                        (NoStopSK, ActionControl, [I_Init]),
                        (LeftFFSK, ActionControl, [ ]),
                        (RightBackSK, ActionControl, [ ]),
                        (OnOffButton, ActionControl,
                        [Quit, I_Init]),
                        (Timeout, System, [ ])];

val SetDuration =       [(Display, MValResponder,
                        [S_CurrentDuration]),
                        (InfoSK, ActionControl, [I_Info]),
                        (UpPlusSK, ActionControl,
                        [S_IncreaseDuration]),
                        (DownMinusSK, ActionControl,
                        [S_DecreaseDuration]),
                        (YesStartSK, ActionControl,
                        [S_SetNewDuration, I_RateConfirm]),
                        (NoStopSK, ActionControl,
                        [I_SetVolume]),
                        (LeftFFSK, ActionControl, [ ]),
                        (RightBackSK, ActionControl, [ ]),
                        (OnOffButton, ActionControl,
                        [Quit, I_Init]),
                        (Timeout, System, [ ])];
```

```
val RateConfirm =           [(Display, MValResponder,
                            [S_CurrentRate]),
                            (InfoSK, ActionControl, [I_Info]),
                            (UpPlusSK, ActionControl, [ ]),
                            (DownMinusSK, ActionControl, [ ]),
                            (YesStartSK, ActionControl,
                            [I_ConfirmSettings, S_SaveSettings]),
                            (NoStopSK, ActionControl,
                            [I_SetDuration]),
                            (LeftFFSK, ActionControl, [ ]),
                            (RightBackSK, ActionControl, [ ]),
                            (OnOffButton, ActionControl,
                            [Quit, I_Init]),
                            (Timeout, System, [ ])];

val ConfirmSettings =       [(Display, MValResponder,
                            [S_CurrentVTBI, S_CurrentDuration,
                            S_CurrentRate, S_ConfirmMsg]),
                            (InfoSK, ActionControl, [I_Info]),
                            (UpPlusSK, ActionControl, [ ]),
                            (DownMinusSK, ActionControl, [ ]),
                            (YesStartSK, ActionControl,
                            [I_StartInfusingConfirm]),
                            (NoStopSK, ActionControl,
                            [I_RateConfirm]),
                            (LeftFFSK, ActionControl, [ ]),
                            (RightBackSK, ActionControl, [ ]),
                            (OnOffButton, ActionControl,
                            [Quit, I_Init]),
                            (Timeout, System, [ ])];
```

```
val StartInfusingConfirm =  [(Display, MValResponder,
                            [S_StartInfusingMsg]),
                            (InfoSK, ActionControl, [I_Info]),
                            (UpPlusSK, ActionControl, [ ]),
                            (DownMinusSK, ActionControl, [ ]),
                            (YesStartSK, ActionControl,
                            [I_Infusing, S_Infusing]),
                            (NoStopSK, ActionControl,
                            [I_ConfirmSettings]),
                            (LeftFFSK, ActionControl, [ ]),
                            (RightBackSK, ActionControl, [ ]),
                            (OnOffButton, ActionControl,
                            [Quit, I_Init]),
                            (Timeout, System, [ ])];

val Infusing =              [(Display, MValResponder,
                            [S_TimeRemaining, S_CurrentRate,
                            S_CurrentSyringe, S_DeliveringMsg]),
                            (InfoSK, ActionControl,
                            [I_InfusionStatus]),
                            (UpPlusSK, ActionControl, [ ]),
                            (DownMinusSK, ActionControl, [ ]),
                            (YesStartSK, ActionControl,[ ]),
                            (NoStopSK, ActionControl,
                            [S_Pause, I_Paused]),
                            (LeftFFSK, ActionControl, [ ]),
                            (RightBackSK, ActionControl, [ ]),
                            (OnOffButton, ActionControl, [ ]),
                            (Timeout, System, [ ])];

val Paused =                [(Display, MValResponder,
```

```
                        [S_PumpStoppedWarning]),
                        (InfoSK, ActionControl, [I_Info]),
                        (UpPlusSK, ActionControl, [ ]),
                        (DownMinusSK, ActionControl, [ ]),
                        (YesStartSK, ActionControl,
                        [S_ResumeInfusing, I_Infusing]),
                        (NoStopSK, ActionControl, [ ]),
                        (LeftFFSK, ActionControl, [ ]),
                        (RightBackSK, ActionControl, [ ]),
                        (OnOffButton, ActionControl,
                        [Quit, I_Inittwo]),
                        (Timeout, System, [ ])];

val Inittwo =           [(Display, MValResponder,
                        [S_SyringeDisplay2]),
                        (InfoSK, ActionControl, [I_Info]),
                        (UpPlusSK, ActionControl, [ ]),
                        (DownMinusSK, ActionControl, [ ]),
                        (YesStartSK, ActionControl,([I_Resume]),
                        (NoStopSK, ActionControl, [ ]),
                        (LeftFFSK, ActionControl, [ ]),
                        (RightBackSK, ActionControl, [ ]),
                        (OnOffButton, ActionControl, [Quit]),
                        (Timeout, System, [ ])];

val Resume =            [(Display, MValResponder,
                        [S_ResumeConfirm]),
                        (InfoSK, ActionControl, [ ]),
                        (UpPlusSK, ActionControl, [ ]),
                        (DownMinusSK, ActionControl, [ ]),
                        (YesStartSK, ActionControl,
```

```
                    ([I_Infusing]),
                    (NoStopSK, ActionControl,
                    [I_SetVolume]),
                    (LeftFFSK, ActionControl, [ ]),
                    (RightBackSK, ActionControl, [ ]),
                    (OnOffButton, ActionControl, [Quit]),
                    (Timeout, System, [ ])];

val InfusionStatus =    [(Display, MValResponder,
                    [S_SyringeStatusDisplay]),
                    (InfoSK, ActionControl,
                    [I_BatteryLevel]),
                    (UpPlusSK, ActionControl, [ ]),
                    (DownMinusSK, ActionControl, [ ]),
                    (YesStartSK, ActionControl,[ ]),
                    (NoStopSK, ActionControl,
                    [S_Pause, I_Pause]),
                    (LeftFFSK, ActionControl, [ ]),
                    (RightBackSK, ActionControl, [ ]),
                    (OnOffButton, ActionControl, [ ]),
                    (Timeout, System,
                    [S_TimeOut, I_Infusing])];

val BatteryStatus =     [(Display, MValResponder,
                    [S_BatteryLevel]),
                    (InfoSK, ActionControl,
                    [I_Infusing]),
                    (UpPlusSK, ActionControl, [ ]),
                    (DownMinusSK, ActionControl, [ ]),
                    (YesStartSK, ActionControl,[ ]),
                    (NoStopSK, ActionControl,
```

```
                        [S_Pause, I_Pause]),
                        (LeftFFSK, ActionControl, [ ]),
                        (RightBackSK, ActionControl, [ ]),
                        (OnOffButton, ActionControl, [ ]),
                        (Timeout, System,
                        [S_TimeOut, I_Infusing])];

val EventLog =          [(Display, MValResponder,
                        [S_EventLogData, S_EventInfoDisplay]),
                        (InfoSK, ActionControl, [I_EventInfo]),
                        (UpPlusSK, ActionControl, [S_NextEvent]),
                        (DownMinusSK, ActionControl,
                        [S_PrevEvent]),
                        (YesStartSK, ActionControl,[I_Info]),
                        (NoStopSK, ActionControl, [I_Info]),
                        (LeftFFSK, ActionControl, [ ]),
                        (RightBackSK, ActionControl, [ ]),
                        (OnOffButton, ActionControl,
                        [Quit, I_Init]),
                        (Timeout, System, [ ])];

val ChangeSetUp =       [(Display, MValResponder,
                        [S_SelectDigit, S_IncorrectCode]),
                        (InfoSK, ActionControl, [I_Info]),
                        (UpPlusSK, ActionControl,
                        [S_SelectDigit]),
                        (DownMinusSK, ActionControl,
                        [S_SelectDigit]),
                        (YesStartSK, ActionControl,
                        [S_SetUpCodeAccept, S_IncorrectCode]),
                        (NoStopSK, ActionControl, [I_Info]),
```

```
(LeftFFSK, ActionControl, [ ]),
(RightBackSK, ActionControl, [ ]),
(OnOffButton, ActionControl,
[Quit, I_Init]),
(Timeout, System, [ ])];
```

Table 6.2: Presentation Model Definition for Niki T34 Syringe Driver

There are nineteen component presentation models for the Niki T34 syringe driver, therefore, the CPN model of the user interface and interaction of the syringe driver has nineteen pages which are interconnected by fusion places.

We present the details of these pages next:

1. **LoadSyringe Page:**

   LoadSyringe is the initial state of the Niki T34 syringe driver. The structure of the *LoadSyringe* page is shown in Figure 6.2. This page shows the navigational possibility from the initial state. The three places: *LoadSyringe*, *Init* and *Info* represent the states of the system that can be reached. Each place has an associated *colour set* which determines the type of data the place may contain. In our model each place belongs to just one colour set, *pmodel* which is a list of colour set *widgetdescr* (a product colour set comprised of *WidgetName*, *Category* and *Behaviours*).

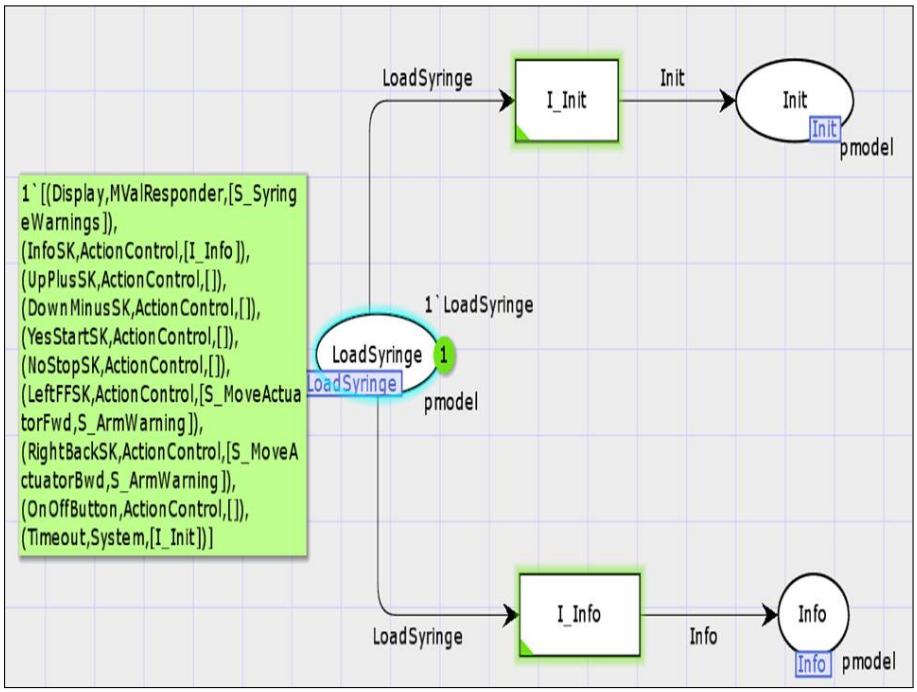

Figure 6.2: LoadSyringe Page

In Figure 6.2, the place *LoadSyringe* has $1'LoadSyringe$ as its initial marking which means that the place has one token with the value *LoadSyringe*. Initially, the remaining places do not contain any tokens. In a model, a marking of places are shown in a green box. For example, in Figure 6.2, the current marking on the place *LoadSyringe* (green box) shows the definition of the *LoadSyringe* component presentation model as described in Table 6.2 which gives information about the available widgets in that state and what behaviours are attached to those widgets in that state. The marking shows that there are two I-behaviours: *I_Init* and *I_Info*, so there are two transitions namely: *I_Init* and *I_Info* on this page. This means that from the *LoadSyringe* state, the user can go either to the *Info* state by firing the transition *I_info* or the *Init* state by firing the transition *I_Init*. As we are modelling user interface and interaction of the Niki T34 syringe driver in CPN which gives meaning to the I-behaviours, so the CPN model will contain the transitions which represent the I-behaviours.

In the initial marking, the transitions *I_Info* and *I_Init* are enabled be-

cause there are enough tokens of the right form on the place and missing guards evaluates to True. As the transitions *I_Info* and *I_Init* are enabled, they can occur. Although both the transitions are concurrently enabled, only one can occur as there is only one token. When the transition *I_Init* fires, then a token 1‘*LoadSyringe* is removed from the input place *LoadSyringe* and a token 1‘*Init* is added to the output place *Init* as shown in Figure 6.3. When the transition *I_Info* fires, then a token 1‘*LoadSyringe* is removed from the input place *LoadSyringe* and a token 1‘*Info* is added to the output place *Info* shown in Figure 6.4.

2. **Init Page**

   The structure of the *Init* page is shown in Figure 6.3. The page has four places: *Init*, *SetVolume*, *TimeOut* and *Info*. Figure 6.3 shows that from the place *Init*, a user can go to three different states by pressing the appropriate widgets.

   The marking of the place *Init* (in the green box) show what widgets are available to a user and their associated behaviours. All the arc expressions carry constants (component presentation models as shown in Table 6.2) in this model. All the three transitions: *I_SetVolume*, *I_Info* and *I_TimeOut* shown in Figure 6.3 are enabled (as per Definition 3.6.4). Although all the transitions are concurrently enabled, only one can occur as there is only one token. A user can go to any of the three states by firing an appropriate transition. When the transition *I_Info* fires, then a token is removed from the input place *Init* and another token is added to the output place *Info* shown in Figure 6.4. When the transition *I_TimeOut* fires, then a token is removed from the input place *Init* and another token is added to the output place *TimeOut* shown in Figure 6.9. Similarly, if a user goes to the *SetVolume* state by firing a *I_SetVolume* transition, then a token is removed from the input place *Init* and is added to the output place *SetVolume* as shown in Figure 6.10.

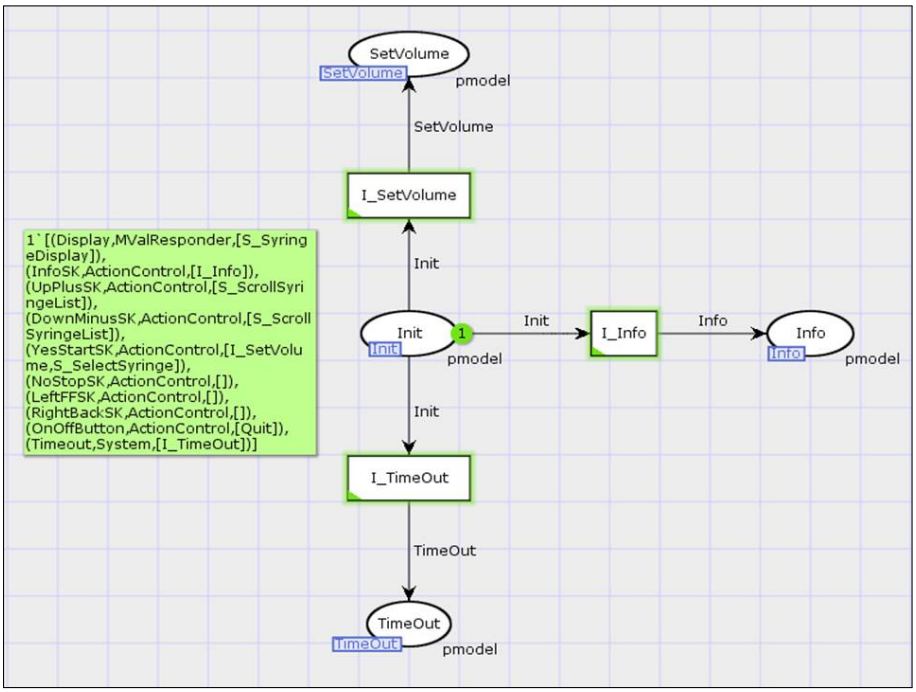

Figure 6.3: Init Page

3. **Info Page**

   Info state shows a list of options that a user can select to see status and change settings. A user is allowed to see a battery charge level by selecting BatteryLevel menu option. A user can also view the history of actions by selecting EventLog menu option. ChangeSetUp option allows a user to change the settings such as changing default time. This is done by entering the correct access code. Another option is the RateSet which allow users to set the rate at which volume is to be infused. This is also done by entering the correct access code.

   There are six places in the *Info* page: *Info*, *BatteryLevel*, *ChangeSetUp*, *Eventlog*, *RateSet* and *Init*. The marking on the place *Info* is a token showing the definition of a component presentation model *Info*.

   There are five transitions: *I_BatteryLevel*, *I_ChangeSetUp*, *I_EventLog*, *I_RateSet* and *I_Init*. All the transitions are enabled in its current marking. As these transitions does not have disjoint set of tokens, only one transition can fire. The user can go to any of the five states by firing

these transitions which updates the markings on the corresponding places as shown in Figures 6.5, 6.6, 6.7, 6.8 and 6.3.

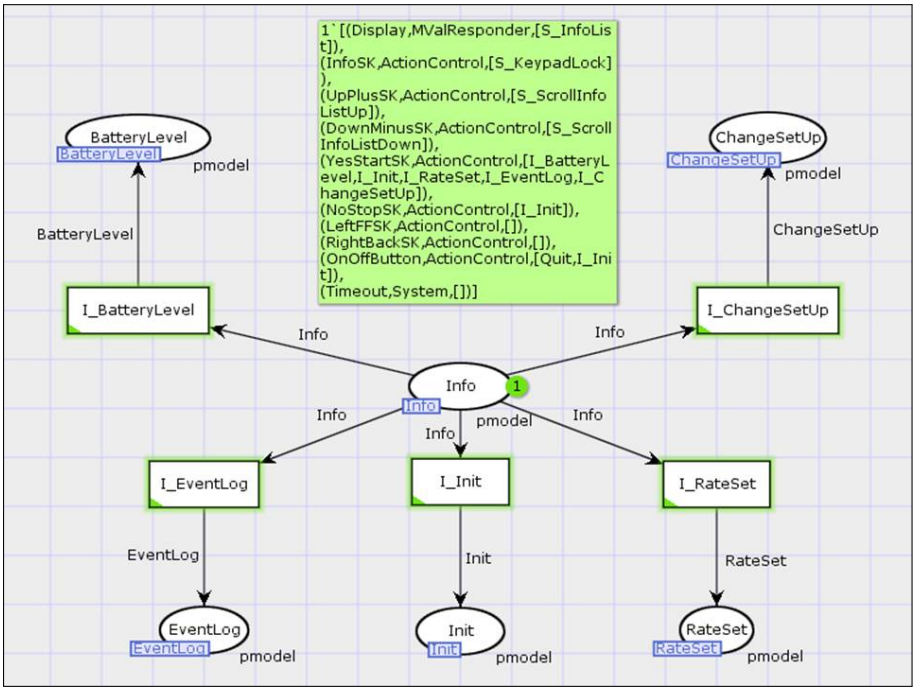

Figure 6.4: Info Page

4. **BatteryLevel page**

   There are several options in the info menu that a user can choose, such as, looking at a battery level, changing the set up, looking at a history of steps or changing the rate of infusion. The structure of the *BatteryLevel* page is shown in figure 6.5. There are two navigational possibilities from this state. A user can go back to the *Info* state by pressing *InfoSK* button or a user can go to the *Init* state by pressing the *OnffButton*. This information of available widgets and its associated behaviours is shown by the marking of the place *BatteryLevel*. When the transition *I_Info* fires, then a token is removed from the input place *BatteryLevel* and another token is added to the output place *Info* shown in Figure 6.4 and if the the transition *I_Init* fires then a token is removed from the input place *BatteryLevel* and another token is added to the output place *Init* shown in Figure 6.3.

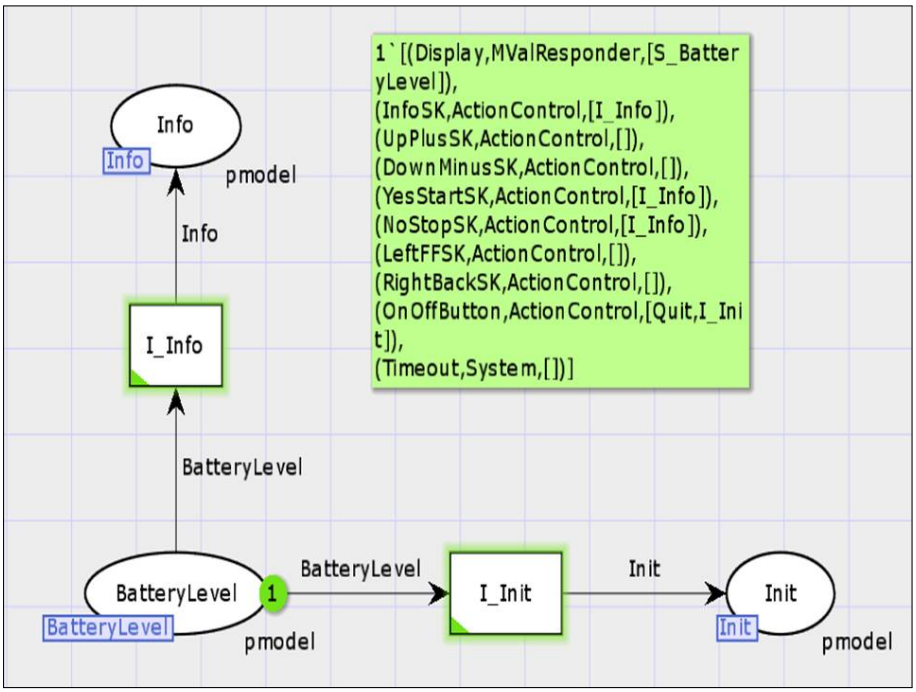

Figure 6.5: BatteryLevel Page

5. **ChangeSetUp Page**

A user is also allowed to change the settings. The structure of the *ChangeSetUp* page is shown in Figure 6.6.

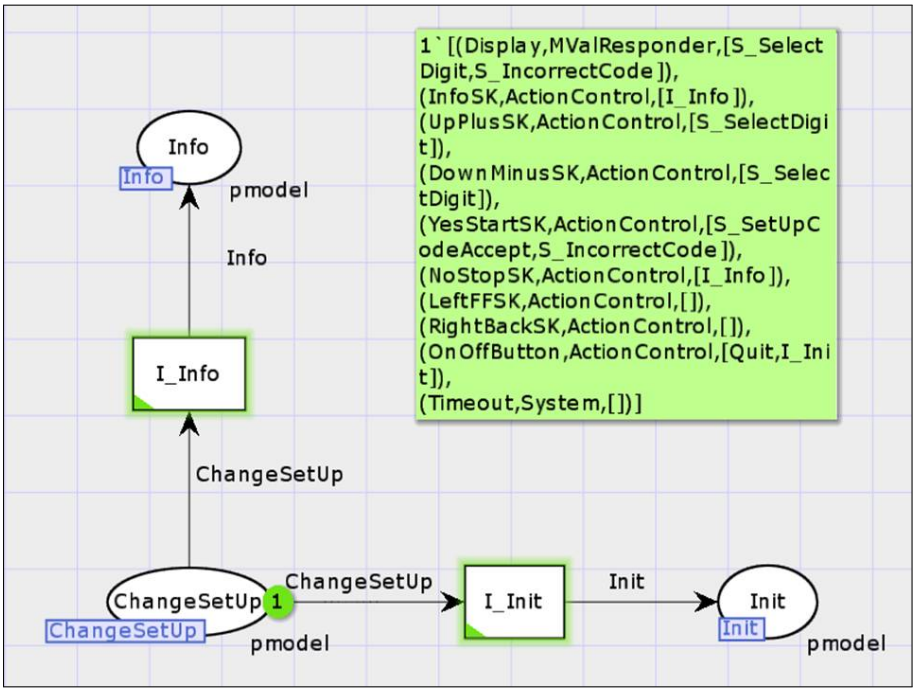

Figure 6.6: Change Set Up Page

This page is similar to the *BatteryLevel* page with two navigational possibilities: *Init* state and *Info* state. The marking of the place *ChangeSetUp* is shown in a green box in Figure 6.6.

6. **EventLog Page**

   The structure of the *EventLog* page is shown in Figure 6.7. This state gives information about all the recent activities done with the Niki T34 syringe driver. A user can get into this state from the *Info* state.

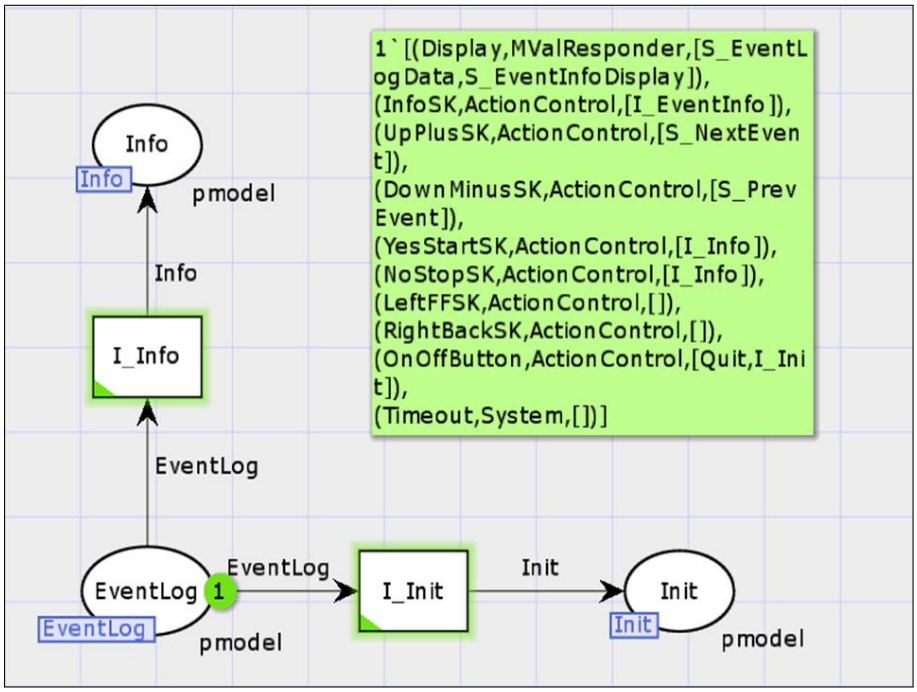

Figure 6.7: EventLog Page

7. **RateSet Page**

   The structure of the *RateSet* page is shown in Figure 6.8. This state allows the user to set the rate of infusion. A user can get into this state from the *Info* state. This page also has two navigational possibilities: *Init* and *Info*.

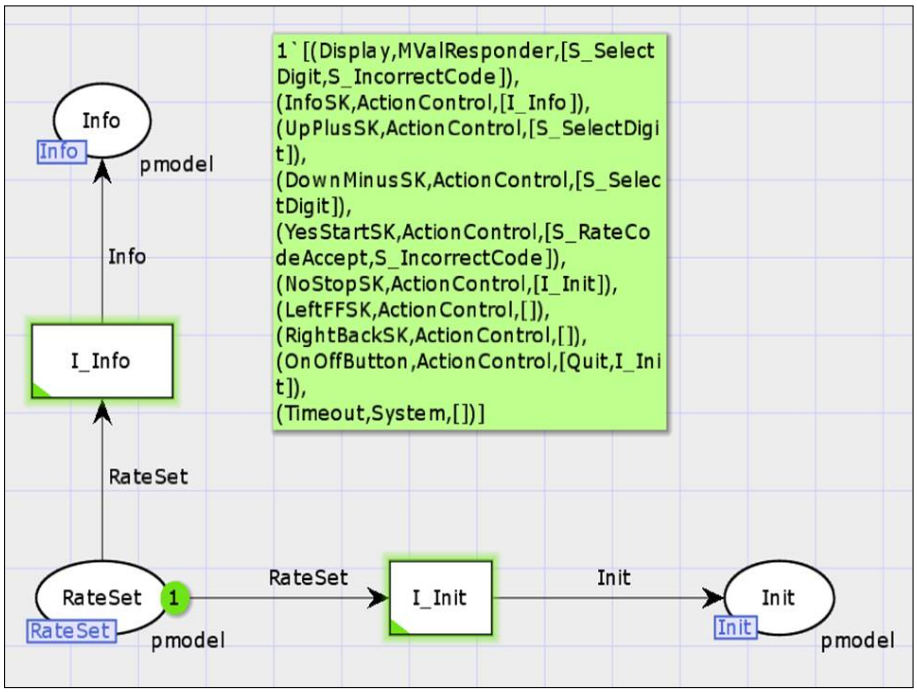

Figure 6.8: RateSet Page

8. **TimeOut Page**

Figure 6.9 shows the *TimeOut* page. *TimeOut* is the warning state in which the alarm beeps when the device is ON but no action has been taken for a certain amount of time. There are three places: *TimeOut*, *Info* and *Init*. The page shows that from *TimeOut*, a user can go to either the *Info* state or the *Init* state.

There exists a marking on the place *TimeOut* when a user is in this state as shown in green box in Figure 6.9. We can see two I-behaviours in the current marking. That is why this page has two transitions: *I_Info* and *I_Init*. Both the transitions are enabled in the current marking but only one can occur as there is only one token. A user can go to *Info* state by firing the transition *I_Info* (pressing *InfoSK* button on the pump) or can go to *Init* state by firing the transition *I_Init* (pressing *NoStopSK* button on the pump). Occurrences of transitions updates the markings on the corresponding places as shown in Figures 6.3 and 6.4.

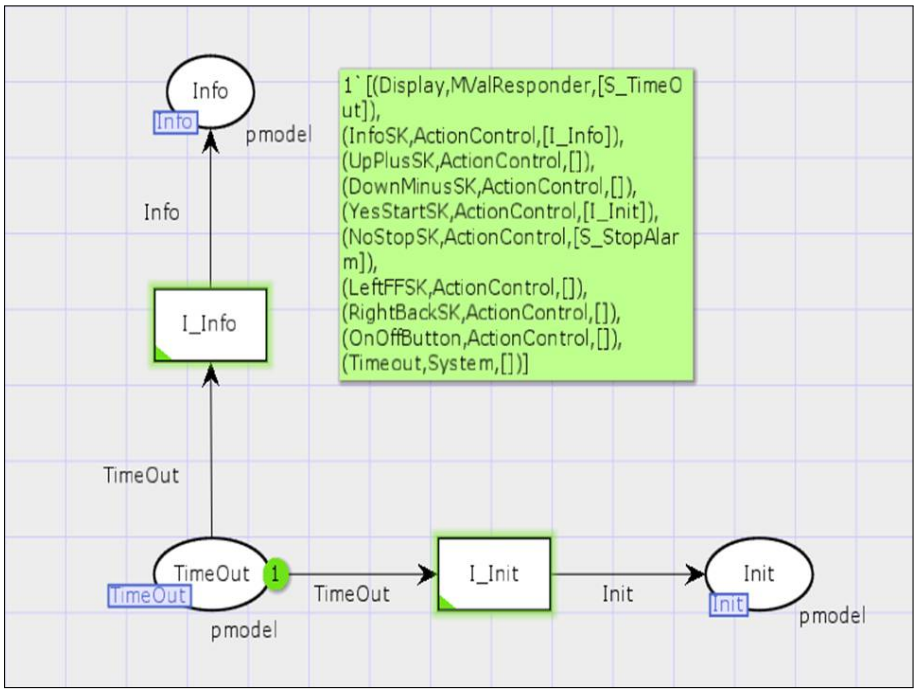

Figure 6.9: TimeOut Page

9. **SetVolume page**

   SetVolume state allows a user to set the volume to be infused. The structure of the page is as shown in Figure 6.10. The three possible states that the user can go to from *SetVolume* state are *Info*, *Init* and *SetDuration*. If a user is in this state, then the place *SetVolume* has a marking as shown in the green box in Figure 6.10 which is the definition of the *SetVolume* component presentation model.

   As the marking has three I-behaviours, this page contains three transitions: *I_Info*, *I_SetDuration* and *I_Init*. All the three transitions are enabled in its current marking and as they are enabled they may occur. Only one transition can occur as there is only one token. Occurrence of the transition will change the state and update the corresponding marking of the place. Figure 6.3 shows the effect of firing of the the *I_Init* transition. Figure 6.4 shows the effect of firing of the the *I_Info* transition and Figure 6.11 shows the effect of firing of the the *I_SetDuration* transition.

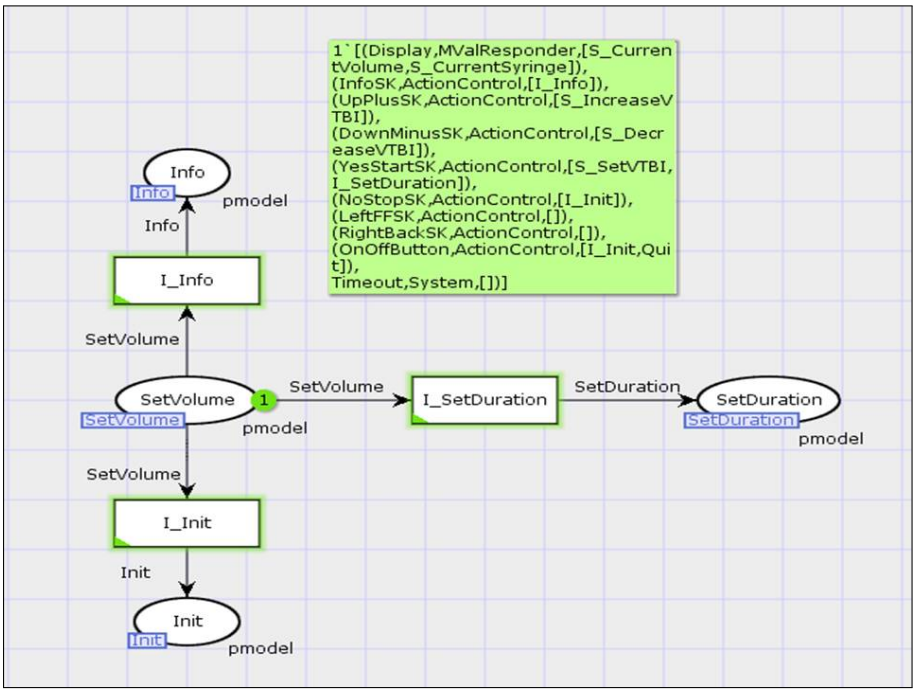

Figure 6.10: SetVolume Page

10. **SetDuration page**

The structure of the *SetDuration* page is shown in Figure 6.11. This state allow users to set the duration for infusion. User can use *UpPlusSK* and *DownMinusSK* to change the duration.

The three possible states that the user can go to from *SetDuration* state are *SetVolume*, *Info*, *Init* and *RateConfirm*. If a user is in this state, then the place *SetDuration* has a marking as shown in the green box in Figure 6.11 which is the definition of the *SetDuration* component presentation model.

As the marking has four I-behaviours, this page contains four transitions: *I_Info*, *I_SetVolume*, *I_RateConfirm*and *I_Init*. Occurrence of these transitions will change the state and update the corresponding marking of the place as shown in Figures 6.3, 6.4, 6.10 and 6.12.

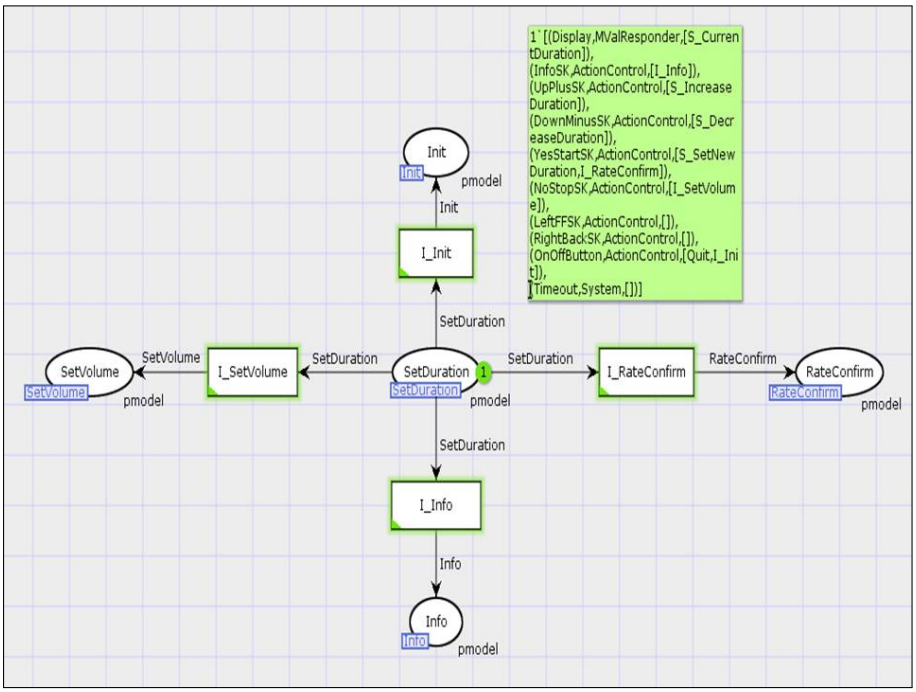

Figure 6.11: SetDuration Page

11. **RateConfirm Page**

RateConfirm state of the pump asks user to confirm the rate at which the drug is to be infused. Figure 6.12 shows the structure of the page. There are four navigational possibilities from the *RateConfirm* state. A user can go back to *SetDuration* state and update the time or can go the next state, i.e., *ConfirmSettings* to confirm all the values. The user can also go to *Init* state and *Info* state. The place *RateConfirm* has a marking as shown in the green box in Figure 6.12.

As the marking has four I-behaviours, this page contains four transitions: *I_Init*, *I_SetDuration*, *I_Info* and *I_ConfirmSettings* which represents all the I-behaviours given in the definition of the component presentation model *RateConfirm*. Occurrence of these transitions will change the state and update the corresponding marking of the place as shown in Figures 6.3, 6.4, 6.11 and 6.13.

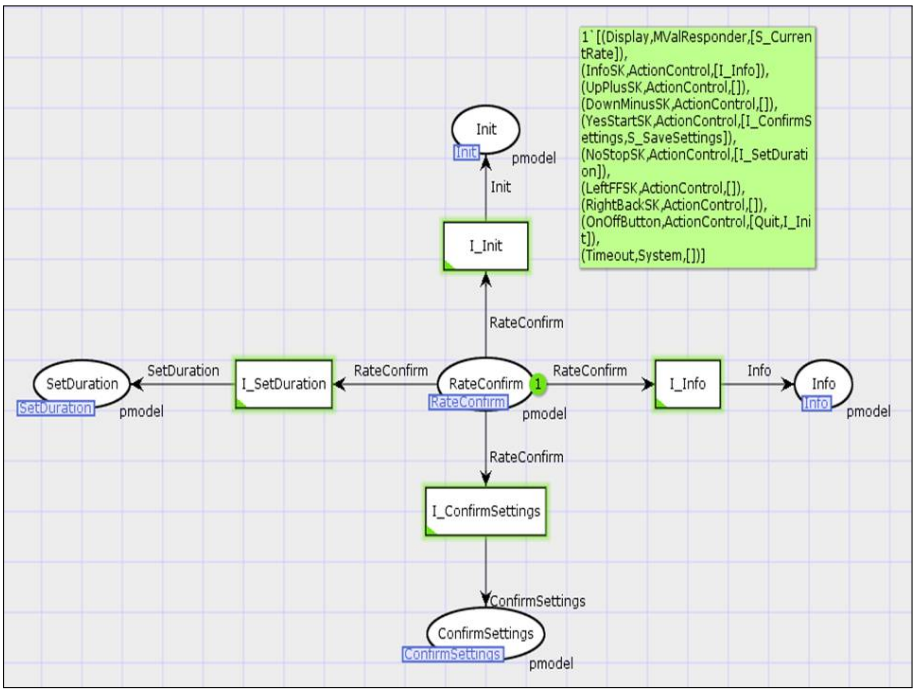

Figure 6.12: RateConfirm Page

12. **ConfirmSettings Page**

Another state that the user can be in is the *ConfirmSettings*. The structure of the page is as shown in Figure 6.13.

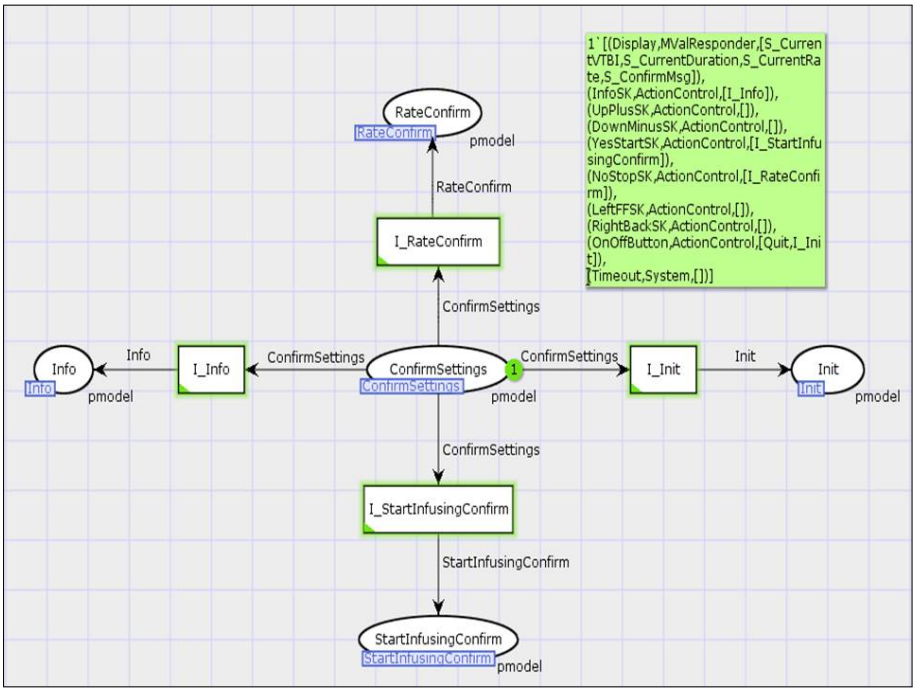

Figure 6.13: ConfirmSettings Page

This state also has four navigational possibilities. A user can go to any one of these states: *RateConfirm*, *StartInfusionConfirm*, *Init*, *Info* and *Init*. The place *ConfirmSettings* has a marking as shown in the green box in Figure 6.13. As the marking has four I-behaviours, this page contains four transitions: *I_Init*, *I_Info*, *I_StartInfusingConfirm* and *I_RateConfirm* which represents all the I-behaviours given in the definition of the component presentation model *ConfirmSettings*. Occurrence of the transition *I_RateConfirm* removes a token from the input place *ConfirmSettings* and adds another token to the output place *RateConfirm* as shown in Figure 6.12. If the transition *I_Init* fires then a token is removed from the input place *ConfirmSettings* and another token is added to the output place *Init* as shown in Figure 6.3. The effect of occurrences of the transitions *I_Info* and *I_StartInfusingConfirm* is shown in Figures 6.4 and 6.14.

13. **StartInfusingConfirm Page**

    Figure 6.14 shows the *StartInfusingConfirm* page of the Niki T34 syringe driver. This state asks user to confirm if they want to actually start infusing a drug into the patient. From this state the user can go to one of the four other states (places): *Info*, *Init*, *Infusing* or *ConfirmSettings*. If a user is in the *StartInfusingConfirm* state, then the place *StartInfusingConfirm* has the marking as shown in the green box in Figure 6.14. In its current marking all the four transitions are enabled because the place *StartInfusingConfirm* has enough tokens of the right form. As all the transitions are enabled, they might occur. The effect of the four possibilities are shown in Figures 6.3, 6.4, 6.15 and 6.13.

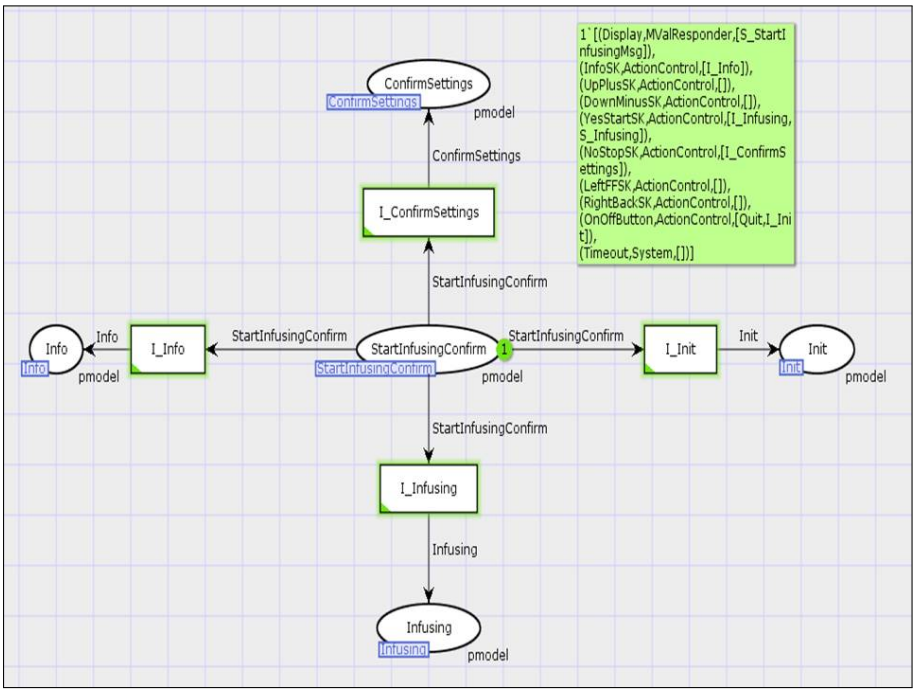

Figure 6.14: StartInfusingConfirm Page

14. **Infusing Page**

The structure of *Infusing* page is shown in Figure 6.15. This state will actually start infusing drug into the patient. This state would not allow users to change any kind of settings. This is one of the safety requirements of the infusion pumps. The only two states that a user can go to are: *InfusionStatus* and *Paused*. In Figure 6.15, the place *Infusing* has a marking that gives information about all the widgets and its associated behaviours. There are two transitions: *I_InfusionStatus* and *I_Paused*. Occurrences of these transitions will result in change of states and markings as shown in Figures 6.16 and 6.18.

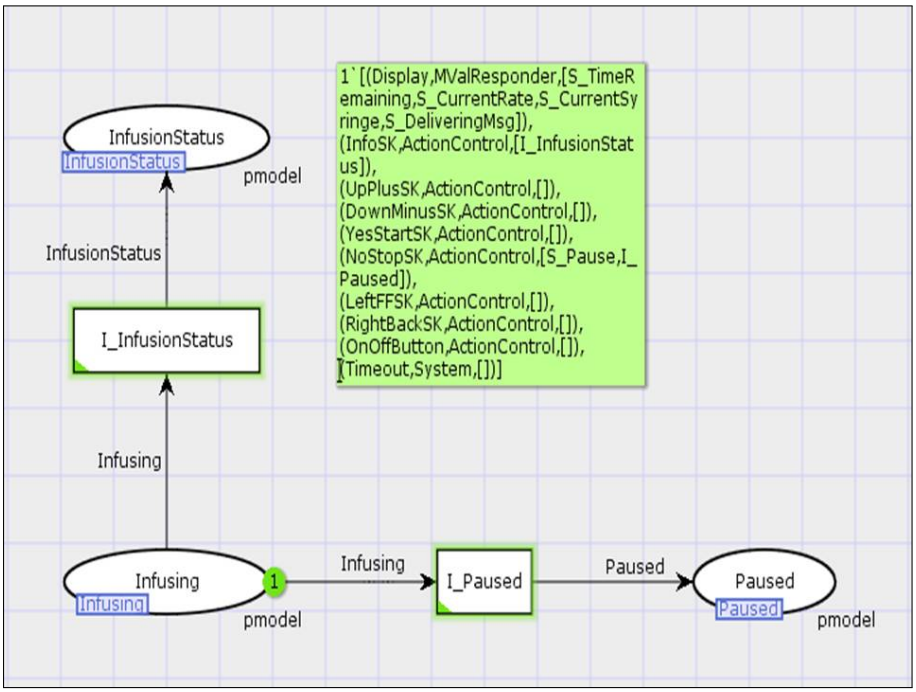

Figure 6.15: Infusing Page

15. **InfusionStatus Page**

Figure 6.16 shows the *InfusionStatus* page. This state will give information/status regarding infusion. There are three navigational possibilities from this state: *Infusing*, *BatteryStatus* or *Paused* state. The pump goes back to *Infusing* state after showing the infusion status. The user is also allowed to see how much battery charge is left by going to the *BatteryStatus* state. The third possibility is to pause the infusion which will result in the *Paused* state.

The place *InfusionStatus* has the marking as shown in the green box in Figure 6.16. The effect of the three possibilities are shown in Figures 6.15, 6.18 and 6.17.

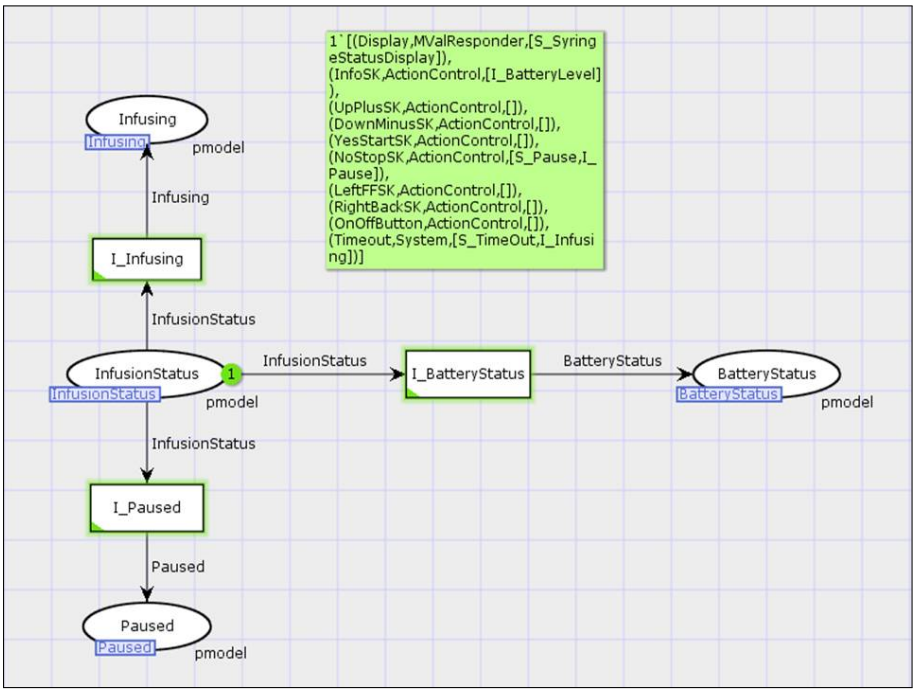

Figure 6.16: InfusionStatus Page

16. **BatteryStatus Page**

The structure of *BatteryStatus* page is shown in Figure 6.17. This state also shows the level of the battery charge similar to that shown by the *BatteryLevel* state shown in Figure 6.5. The only difference is that in this state we can see the battery level information while the driver is allowing infusion.

This page has three places: *BatteryStatus*, *Paused* and *Infusing*. There are two navigational possibilities from this state. A user can go back to either the *Infusing* state by pressing the *InfoSK* button or can pause the infusing and go to the *Paused* state by pressing the *NoStopSK* button. The current marking of the place *BatteryStatus* shows the definition of the component presentation model *BatteryStatus* which has two I-behaviours. The two transitions: *I_Paused* and *I_Infusing* represents these two I-behaviours. Occurrence of these transitions updates the marking on the corresponding places as shown in Figures 6.18 and 6.15.

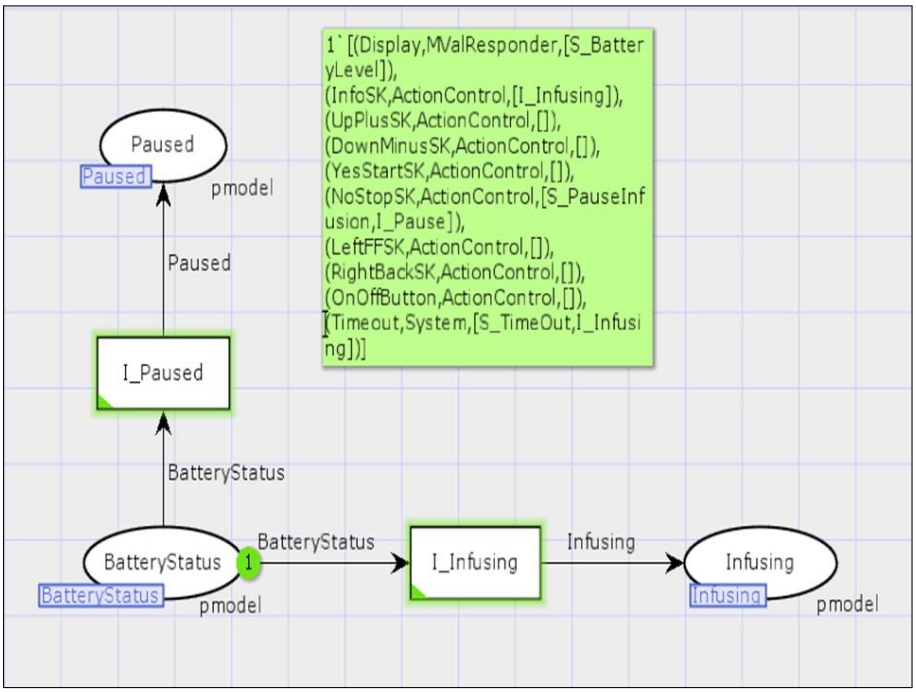

Figure 6.17: Battery Status Page

17. **Paused Page**

A user is allowed to pause the infusion. The structure of the *Paused* page is shown in Figure 6.18.

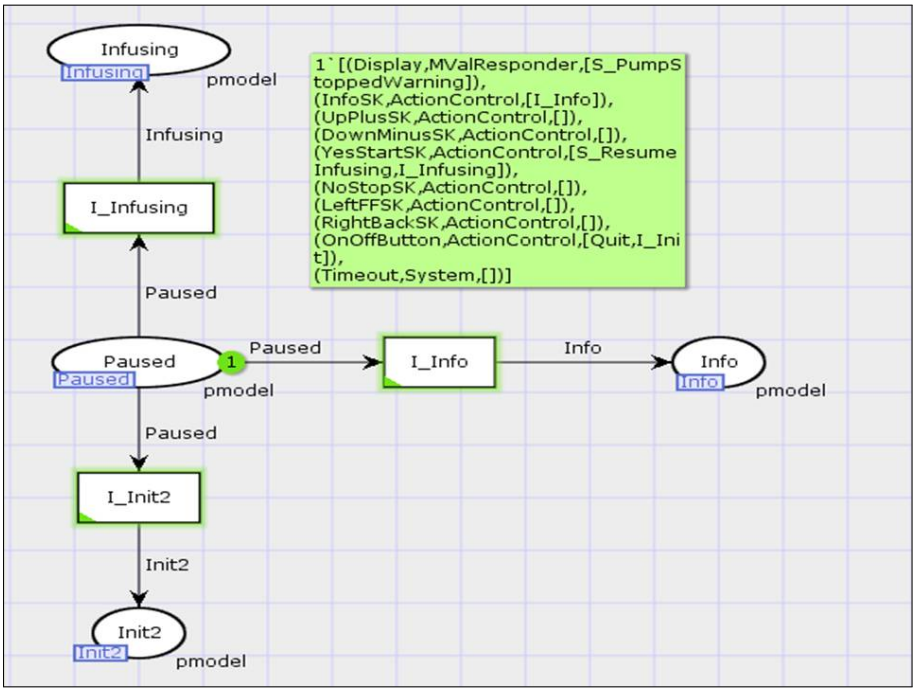

Figure 6.18: Paused Page

From the *Paused* state, a user can go to one of these three states: *Infusing*, *InitTwo* or *Info*. All the arc expressions are the constants representing the names of the component presentation models. The marking on the place *Paused* shown in the green box has three I-behaviours, therefore the model has three transitions with the same name as I-behaviours. All the transitions are enabled in the current marking. The effect of the three possibilities are shown in Figures 6.15, 6.4 and 6.19. Although *Inittwo* looks similar to *Init* state, we have made it a separate state and given the name *Inittwo* because the behaviours of the widgets are different in this state.

18. **Inittwo Page**

The structure of the *Inittwo* page is shown in Figure 6.19.

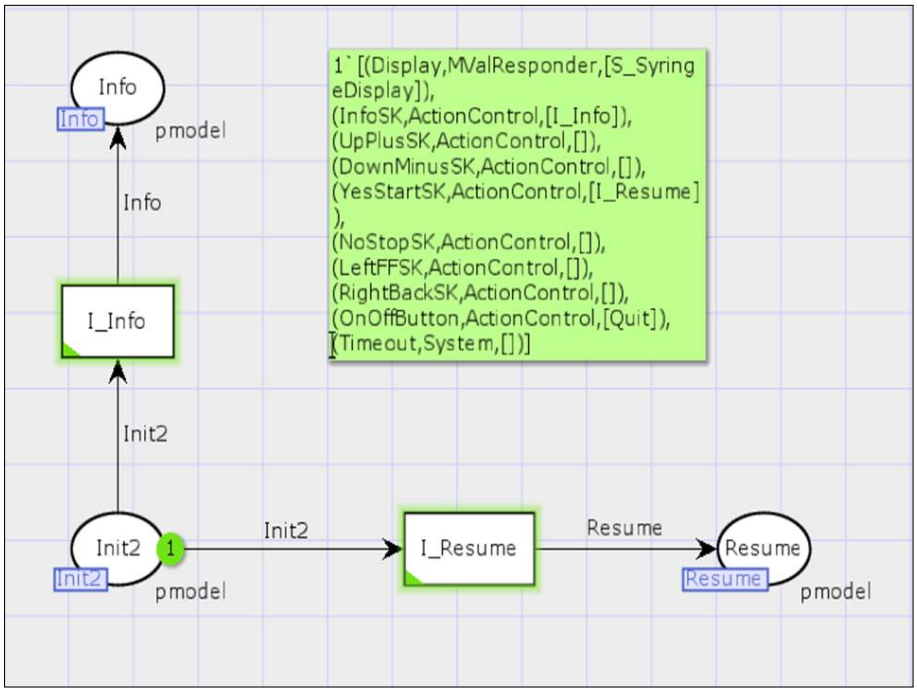

Figure 6.19: Inittwo Page

This page comprises of three places: *Info*, *Init*2 and *Resume*. There are two navigational possibilities from the *Init*2 state: a user can either go to *Info* state or a user can go to *Resume* state. The marking shown in a green box is a current marking if a user is in this state. In the

current marking both the transitions that represents two I-behaviours are enabled and may occur resulting is change in markings as shown in Figures 6.4 and 6.20.

19. **Resume Page**

Resume state asks the user if they want to resume the infusion from where it was stopped or not. A user can resume the infusion by pressing *YesStartSk* button and can go to the *Infusing* state. If the user presses *NoStopSK* button, then the system goes to the *SetVolume* state and from there a user has to set the values again. Figure 6.20 shows the structure of the *Resume* page. The information about all the available widgets in this state and their associated behaviours are shown by a marking on the place *Resume*. There are two transitions: *I_Infusing* and *I_Setvolume* repreneting two I-behaviours. Both the transitions are enabled in the current marking and occurrence of these transition updates the marking on the corresponding places as shown in Figures 6.15 and 6.10.

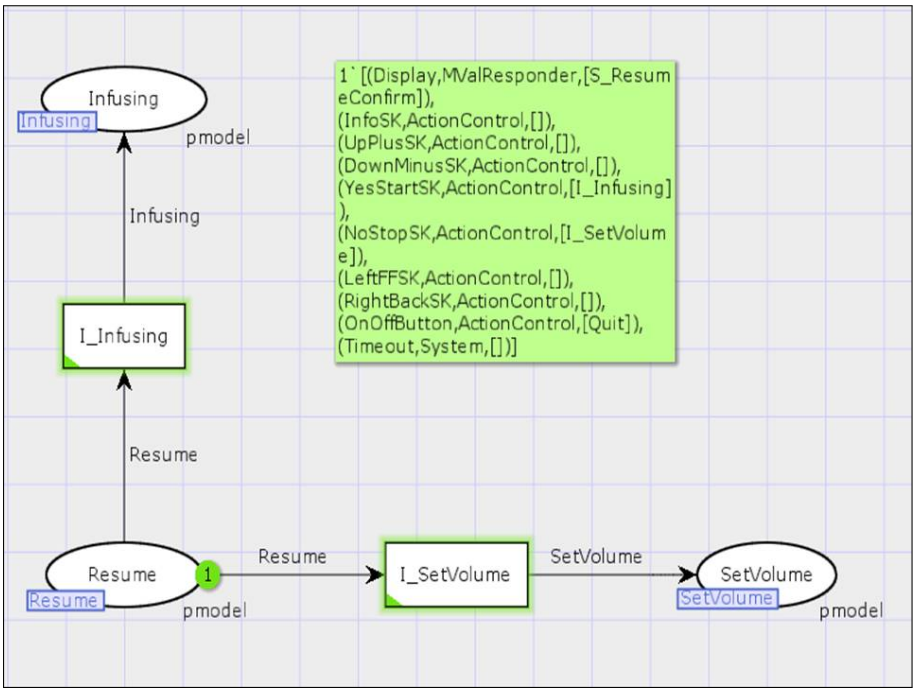

Figure 6.20: Resume Page

6.5 Modelling Functionality of Niki T34 Syringe Driver in Coloured Petri Nets

Having modelled the user interface and interaction in Coloured Petri Nets, we now move to the final part, modelling the functionality. We use the technique specified in Chapter 5 to model the functionality by extending the model from the above section by adding S-behaviours (underlying system functionality behaviours specified using Z) to it. For every S-behaviour in the presentation model of the Niki T34 syringe driver in Table 6.2, there exists a Z operation schema. As discussed in Chapter 5, to add the S-behaviours we express the Z specification of the Niki T34 syringe driver in Coloured Petri Nets. The Z specification of the Niki T34 syringe driver is given in Appendix C.

6.5.1 Complete CPN Model of Niki T34 Syringe Driver

In this section we extend the CPN model of the user interface and interaction of the Niki T34 syringe driver given above in Section 6.3 by adding S-behaviours to it according to the rules written in Section 5.2.1. According to rule 1, for every Z type used to specify the underlying functionality of a system, a corresponding colour set is created. Rule 2 states that for modelling the Z state schema, we use a record colour set.

Table 6.3 shows all the colour sets and variables of the T34 syringe driver as per rules 1 and 2.

```
colset YesNo =                with yes | no;
colset SyringeBrand =         with BDPlastipak;
colset PerCent =              int with 0..100;
colset millilitres=           int with 0..100;
colset millimeters =          int with 0..10;
colset hours =                int with 0..24;
colset minutes =              int with 0..59;
colset millilitresperhour =   int with 0..100;
```

colset Z =	record BC:PerCent * KP:YesNo *
	PL:YesNo * TL:YesNo *
	BD: SyringeBrand * SS: millilitres *
	VL: millilitres * PP : millimeters *
	SOK: YesNo * BCPOK: YesNo *
	SR: YesNo * VTBI: millilitres *
	HH: hours * MM: minutes *
	IR: millilitresperhour;
var BC:	PerCent ;
var KP:	YesNo;
var PL:	YesNo;
var TL:	YesNo;
var BD:	SyringeBrand ;
var SS:	millilitres;
var VL:	millilitres ;
var PP :	millimeters;
var SOK:	YesNo;
var BCPOK:	YesNo;
var SR:	YesNo;
var VTBI:	millilitres;
var HH:	hours;
var MM:	minutes;
var IR:	millilitresperhour;

Table 6.3: Colour sets and variables for T34 Sringe Driver

- *YesNo* is declared as the enumerated colour set that can have exactly two values *yes* or *no*.
- *SyringeBrand* is declared as enumerated colour set with one value *BDPlastipak*.
- *PerCent* is declared as integer colour set with range 0..100.
- *millilitres* is declared as integer colour set with range 0..100.

- *millimeters* is declared as integer colour set with range 0..10.
- *hours* is declared as integer colour set with range 0..24.
- *minutes* is declared as integer colour set with range 0..59.
- Z is a record colour set with a record of all the observations of the state schema with their corresponding types. It represents the $T34$ state schema[1].
- As the Z operation schemas would be expressed as arc inscriptions so we need to declare variables which could be bound to different values of their respective colour sets during simulation. There are fifteen variables BC, KP, PL, TL, BD, SS, VL, PP, SOK, $BCPOK$, SR, $VTBI$, HH, MM and IR.

According to rule 3 of Section 5.2.1, a fusion place named Z of type colour set Z is added to every page of the model that contains all the observations of the state schema. As per rule 4 of Section 5.2.1, an initial marking of a fusion place Z represents the Z *init* schema. The fusion place Z with initial marking is shown in Figure 6.21.

[1]To make the description short and easy to read we have used abbreviated names for the Z observations. In this declaration BC stands for *BatteryCharge*, KP is for *KeyPadLocked*, PL is for *ProgramLocked*, TL is for *TechMenuLocked*, BD is for *Brand*, SS is for *SyringeSize*, VL is for *VolumeLeft*, PP is for *PlungerPosition*, SOK is for *SyringeOK*, $BCPOK$ is for *BarrelOK*, *CollarOK*, *PlungerOK*, SR is for *SystemReady*, HH is for *Hours*, MM is for *Minutes* and IR is for *InfusionRate*

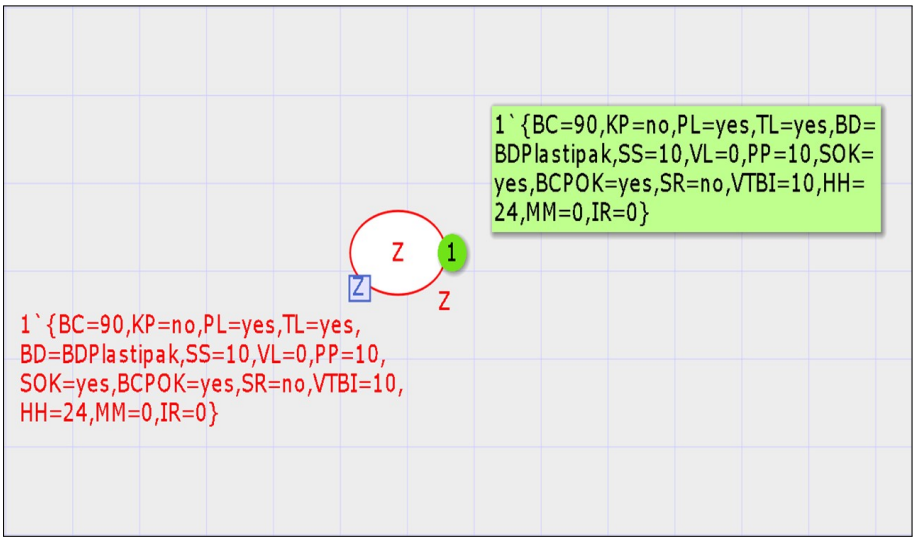

Figure 6.21: Z fusion place of Niki T34 syringe driver

This place is added to every page of the model. Now we will see each and every page in detail.

1. **LoadSyringe Page**

   The structure of the *LoadSyringe* page is shown in Figure 6.22.

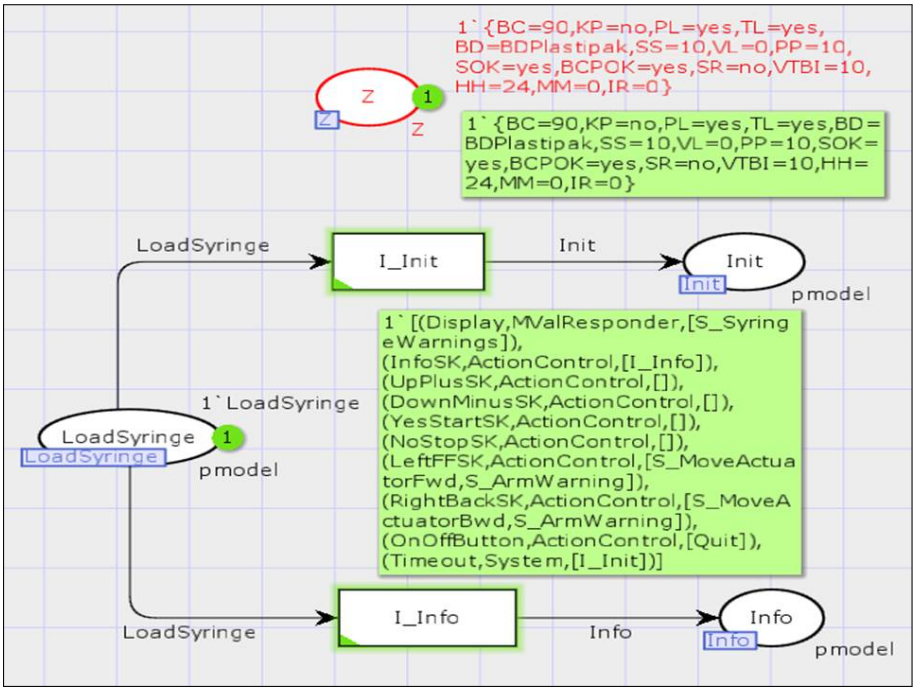

Figure 6.22: LoadSyringe Page with Z

The three places: *LoadSyringe*, *Init* and *Info* represent the states of the

system. The marking on the place *LoadSyringe* shows the definition of the *LoadSyringe* component presentation model which gives information about the available widgets. The two transitions: *I_Init* and *I_Info* represents the two I-behaviours as shown in the marking. The marking on the *LoadSyringe* page also shows that there are four S-behaviours: *S_SyringeWarnings* and *S_ArmWarning* display warning messages on the screen and *S_MoveActuatorBwd and S_MoveActuatorFwd* are the functions that move the syringe plunger forward and backward. The Z specification for these S-behaviours are not modelled here to keep the size of the state space small, so just the fusion place *Z* is added to the page and represents only the Z *Init* schema.

In Figure 6.22, there are two initial markings as shown in Table 6.4, one on place *LoadSyringe* that shows the component presentation model definition and one on *Z* that shows the Z *Init* schema. The user can go either to the *Info* state by firing the transition *I_info* or the *Init* state by firing the transition *I_Init*.

Place	Marking
LoadSyringe	1‘[(Display, MValResponder, [S_SyringeWarnings]), (InfoSK, ActionControl, [I_Info]), (UpPlusSK, ActionControl, [ ]), (DownMinusSK, ActionControl, [ ]), (YesStartSK, ActionControl,[ ]), (NoStopSK, ActionControl, [ ]), (LeftFFSK, ActionControl, [S_MoveActuatorFwd, S_ArmWarning]), (RightBackSK, ActionControl, [S_MoveActuatorBwd,S_ArmWarning]), (OnOffButton, ActionControl, [Quit]), (Timeout, System, [I_Init])];
p=Z	1‘{BC=90, KP=no, PL=yes,TL=yes,

	BD=BDPlastipak,SS=10,VL=0, PP=10, SOK=yes,BCPOK=yes,SR=no, VTBI=10, HH=24,MM=0,IR=0}

Table 6.4: Initial Markings for T34 pump CPN model

2. **Init Page**

The basic structure of the *Init* page with S-behaviours is shown in Figure 6.23. There are three S-behaviours as shown by the marking of the place *Init*: *S_SyringeDisplay*, *S_ScrollSyringeList* and *S_SelectSyringe*. The real device supports two types of syringe brands. A user can select any of them by scrolling through the list. But to make the model simple, we here assumed that the pump has only one syringe brand. So these S-behaviours are not a part of the model. Only fusion place *Z* is added to the page as per rule 3.

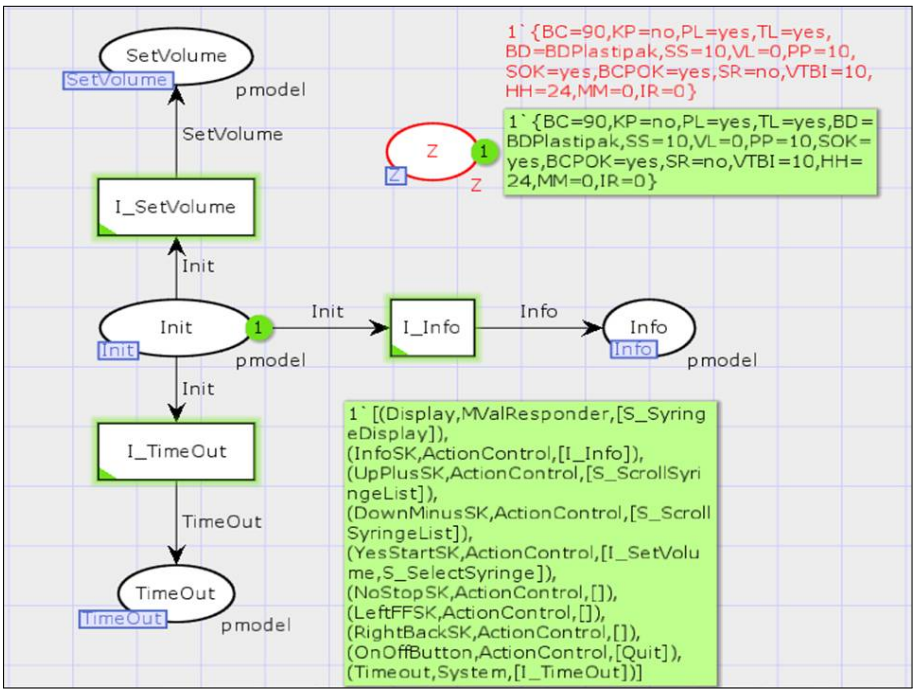

Figure 6.23: Init Page with Z

3. **Info Page**

Figure 6.24 shows the structure of the *Info* page. The *Info* state shows

a list of options that a user can select to see status and change settings. There are seven places in the *Info* page: *Info*, *BatteryLevel*, *Init*, *ChangeSetUp*, *Eventlog*, *RateSet* and *Z*. The marking on the place *Info* is a token showing the definition of a component presentation model *Info* which provides information about the available widgets and tells us which button press results in what state. The marking shows that there are five I-behaviours and four S-behaviours. As we are not modelling the S-behaviours which just display messages on the screen, the page *Info* has just one S-behaviour transition *S_KeyPadLocked*. There are six transitions: *I_BatteryLevel*, *I_ChangeSetUp*, *I_Init*, *I_EventLog*, *I_RateSet* and *S_KeyPadLocked*. These transitions give meaning to the I-behaviours and S-behaviours given for the definition of the component presentation model *Info*.

If the user gives a long press to the *Info* button on the device, the keypad gets locked or unlocked. This behaviour changes the value of the observation *KP* from *no* to *yes* and vice-versa. In the CPN model, the transition *S_KeyPadLocked* represents this behaviour. There are two arcs needed to model this (going to and from the place *Z*). The arc from *Z* to *S_KeyPadLocked* simply contains assignments which set each variable to its current value (where the variables are the ones that model the observations from the Z operation schema *KeyPadLocked* where they will appear on the left of each equation in primed form). This set of assignments "picks up" the current values of the variables ready to be used by the second arc. This second arc, the one from *S_KeyPadLocked* to *Z* as shown in Table 6.5, assigns each variable to its *new* value, as given by the right-hand side of each equation in the *KeyPadLocked* operation schema. Taken together these two arcs express the intent of the equations in the operation schema *KeyPadLocked*. Users can go to any of the five states by firing the I-behaviour transitions which will update the markings on the corresponding places. If the value of *KP* is *yes*, i.e., if the keypad is locked, then a user cannot go to any further possible states. For

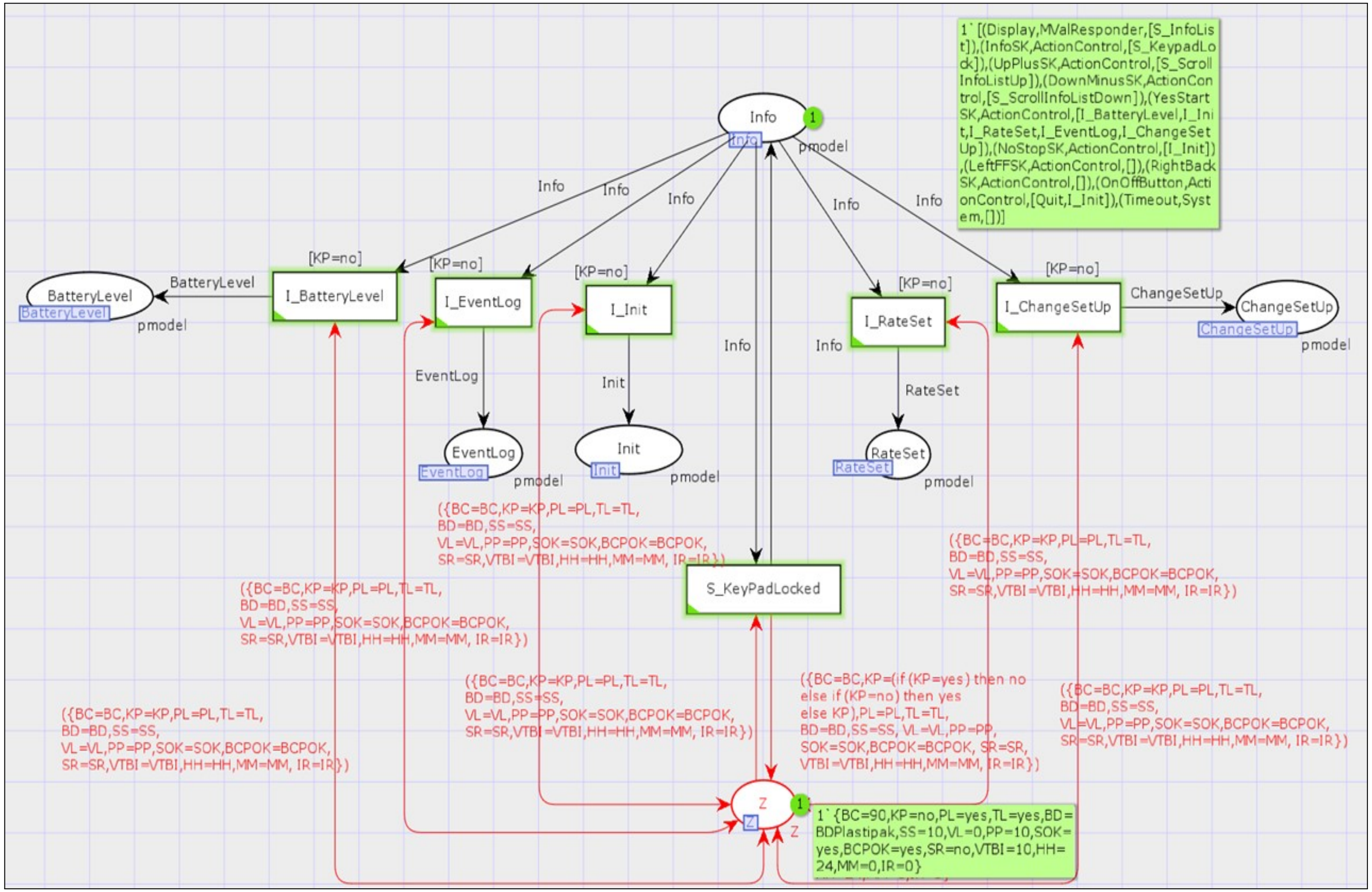

Figure 6.24: Info Page with Z

that reason, we have a guard $[KP = no]$ on transitions *I_BatteryLevel*, *I_EventLog*, *I_Init*, *I_RateSet* and *I_ChangeSetUp*. These transitions would not be enabled if the keypad is locked.

Arc	Arc Expression
S_KeyPadLocked to Z	({BC=BC, KP=(if (KP=yes) then no else if (KP=no) then yes else KP), PL=PL, TL=TL, BD=BD, SS=SS, VL=VL, PP=PP, SOK=SOK, BCPOK=BCPOK, SR=SR, VTBI=VTBI, HH=HH, MM=MM, IR=IR})

Table 6.5: Arc Expression S_KeyPadLocked to Z

4. **TimeOut Page**

In *TimeOut* state, when no action is being taken, the alarm starts beeping and a timeout warning is displayed on the screen.

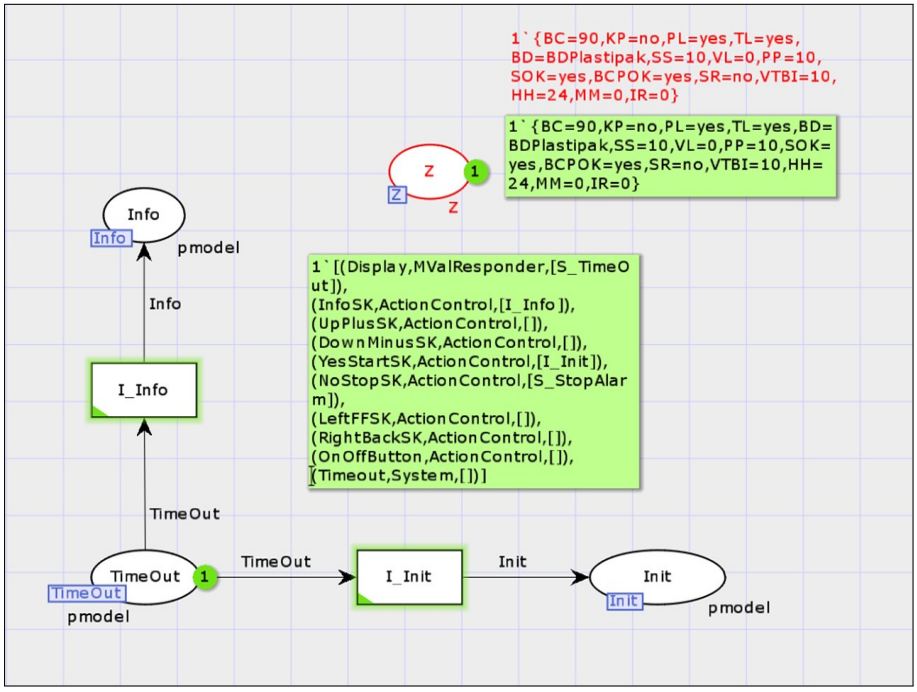

Figure 6.25: TimeOut Page with Z

Fusion place *Z* is added to the page which shows the current values of the observations. This is shown in Figure 6.25. As we are abstracting

the display messages, so S-behaviours are not modelled for this state. Hence, there is no transition representing S-behaviour in this page.

5. **SetVolume page**

   Figure 6.26 shows the basic structure of the *SetVolume* page.

   There are total five S-behaviours as shown by the marking of the place *SetVolume*. *S_CurrentSyringe* and *S_CurrentVolume* are what is displayed on the screen which will not be the part of this model. The other three S-behaviours: *S_IncreaseVTBI*, *S_DecreaseVTBI* and *S_SetVTBI* are drawn as transitions as shown in Figure 6.26. According to rule 7 in Section 5.2.1, if two two behaviours occurs simultaneously, then there will be one transition with both the behaviour names. In *SetVolume* state, as *S_SetVolume* and *I_SetDuration* occurs together, so there is one transition named *S_SetVTBI_I_SetDuration*.

   The arcs from *Z* to the three transitions *S_IncreaseVTBI, S_DecreaseVTBI* and *S_SetVTBI_I_SetDuration* simply contains assignments which set each variable to its current value. These arc expressions are shown in Figure 6.26 and Table 6.6. These arcs, the ones from tranistions *S_IncreaseVTBI*, *S_DecreaseVTBI* and *S_SetVTBI_I_SetDuration* to *Z* assigns each variable to its *new* value, as given by the right-hand side of each equation in the corresponding Z operation schemas.

Arc	Arc Expression
S_IncreaseVTBI to Z	({BC=BC, KP=KP, PL=PL, TL=TL, BD=BD, SS=SS, VL=VL, PP=PP, SOK=SOK, BCPOK=BCPOK, SR=SR, VTBI=VTBI+1, HH=HH, MM=MM, IR=IR})
S_DecreaseVTBI to Z	({BC=BC, KP=KP, PL=PL, TL=TL, BD=BD, SS=SS, VL=VL, PP=PP, SOK=SOK, BCPOK=BCPOK, SR=SR, VTBI=VTBI-1, HH=HH, MM=MM, IR=IR})
S_SetVTBI_I_SetDuration to Z	({BC=BC, KP=KP, PL=PL, TL=TL,

	BD=BD, SS=SS, VL=VTBI, PP=PP, SOK=SOK, BCPOK=BCPOK, SR=SR, VTBI=VTBI, HH=HH, MM=MM, IR=IR})

Table 6.6: Arc Expression setvolume

Firing of transition *S_IncreaseVTBI* increases the *VTBI* by one and firing of transition *S_DecreaseVTBI* decreases the *VTBI* by one. This is reflected in a marking of the place *Z*. As we have assumed the syringe size to be ten millilitres, so VTBI can never be greater than ten. VTBI can be increased by firing the transition *S_IncreaseVTBI* only if VTBI is less than ten. This is represented by the guard $[VTBI < 10]$ on transition *S_IncreaseVTBI* and also VTBI can never be negative which is represented by the guard $[VTBI > 0]$. Transition *S_SetVTBI_I_SetDuration* changes the current value of *VL* to the current value of *VTBI* and moves to the next state which is *SetDuration*.

6. **SetDuration page**

   The structure of the *SetDuration* page is shown in Figure 6.27. The three S-behaviours: *S_IncreaseDuration, S_DecreaseDuration* and *S_SetNew Duration* are drawn as transitions.

   The arc expressions from *Z* to the three transitions *S_IncreaseDuration, S_DecreaseDuration* and *S_SetNewDuration* are shown in Figure 6.27 and Table6.7. These arcs, the ones from tranistions *S_IncreaseDuration, S_DecreaseDuration* and *S_SetNewDuration* to *Z* assigns each variable to its *new* value, as given by the right-hand side of each equation in the corresponding Z operation schemas. The arc expressions that represents Z operation schemas are shown in table 6.7.

   Firing of transition *S_IncreaseDuration* affects both *HH* and *MM* observations. If $MM <= 58$, then value of *HH* remains same and the value of *MM* is increased by one. If $MM = 59$ and $HH <= 23$, then the value of *HH* is increased by one. If $HH = 24$, then the value of *HH* becomes

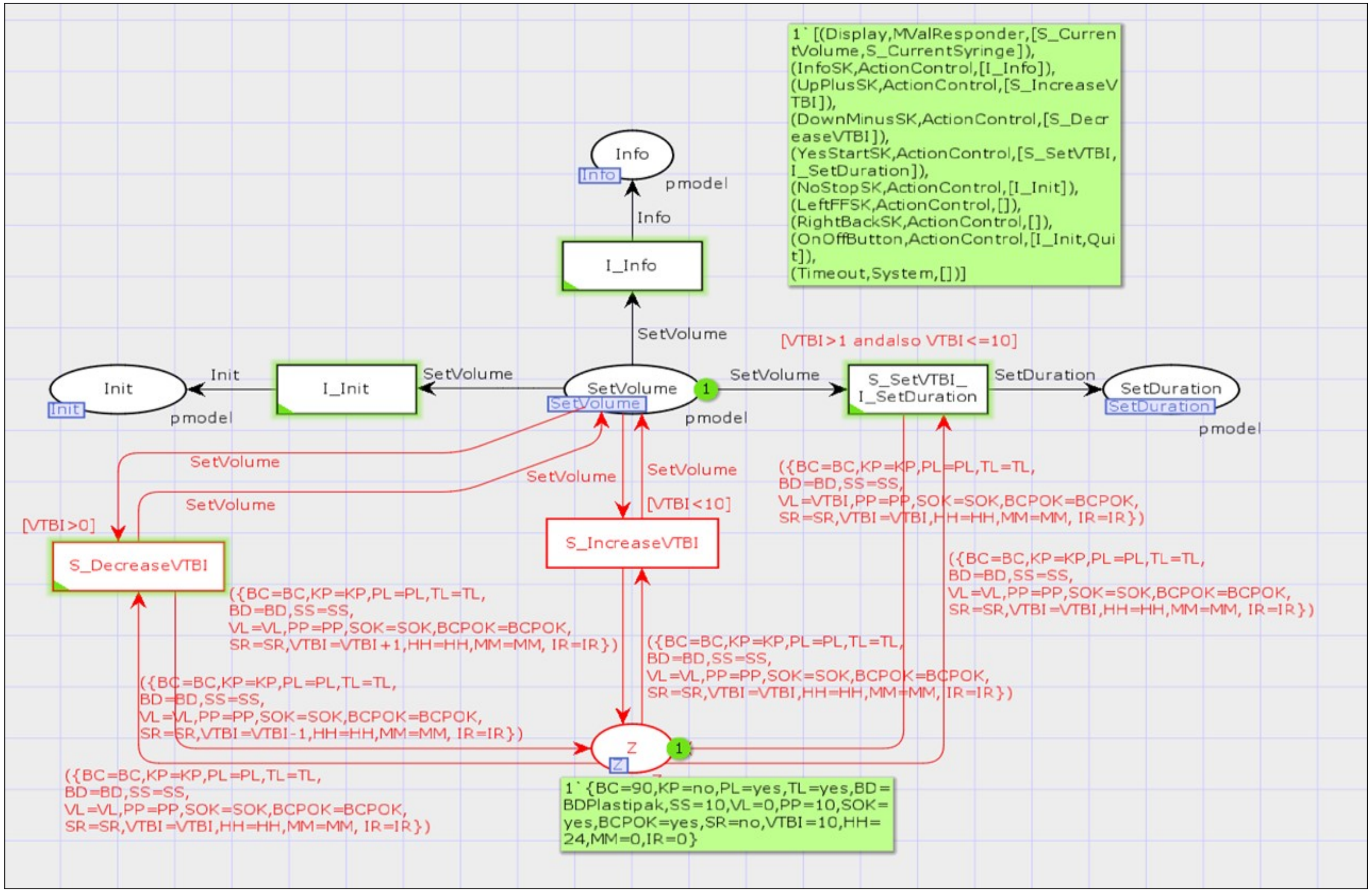

Figure 6.26: SetVolume Page with Z

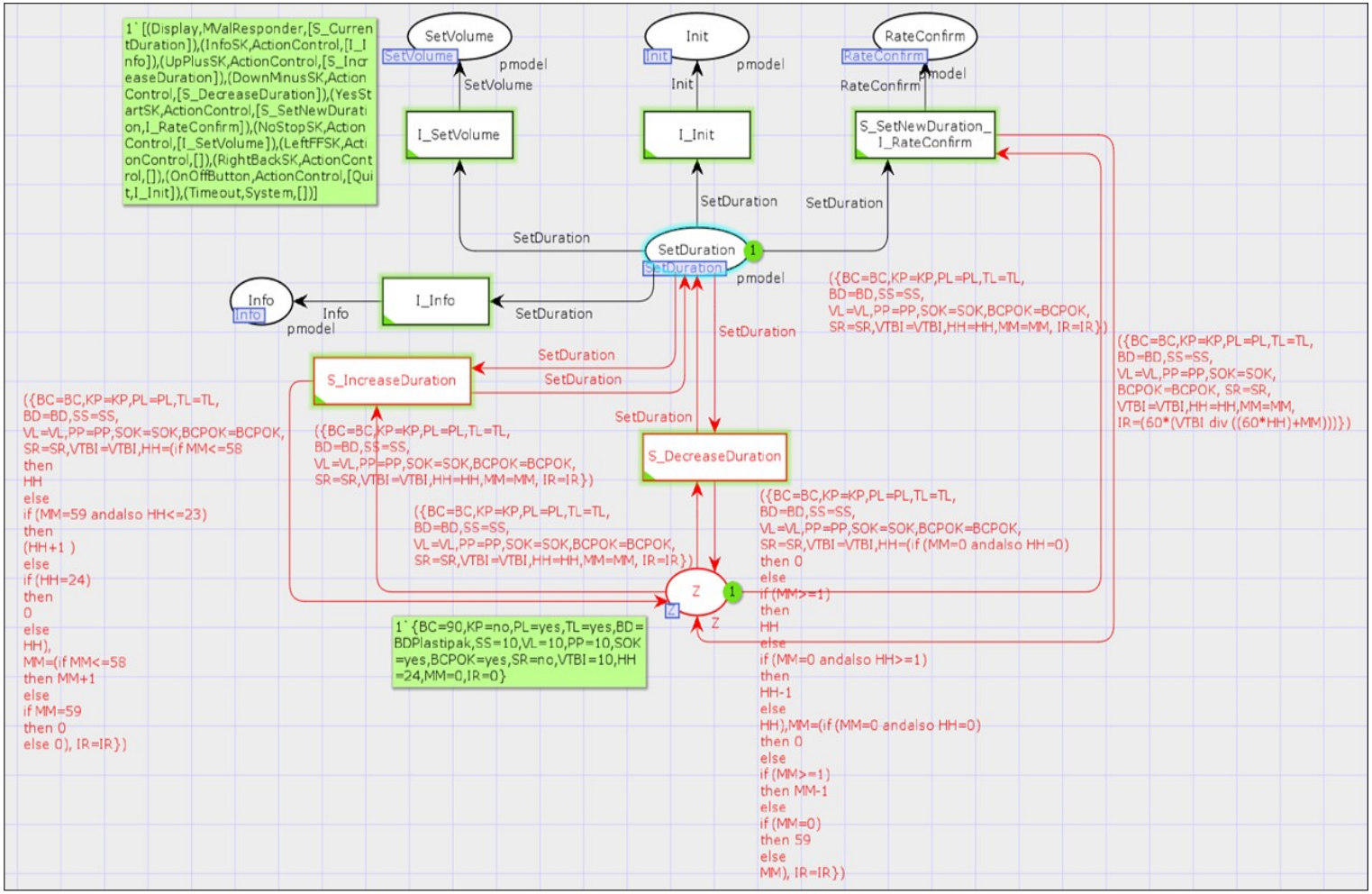

Figure 6.27: SetDuration Page with Z

zero. If $MM = 59$, then firing of transition *S_IncreaseDuration* changes the value of MM to zero.

Arc	Arc Expression
S_IncreaseDuration to Z	({BC=BC, KP=KP, PL=PL, TL=TL, BD=BD, SS=SS, VL=VL, PP=PP, SOK=SOK, BCPOK=BCPOK, SR=SR, VTBI=VTBI, HH=(if MM<=58 then HH else if (MM=59 andalso HH<=23) then (HH+1) else if (HH=24) then 0 else HH), MM=(if MM<=58 then MM+1 else if MM=59 then 0 else 0), IR=IR})
S_DecreaseDuration to Z	({BC=BC, KP=KP, PL=PL, TL=TL, BD=BD, SS=SS, VL=VL, PP=PP, SOK=SOK, BCPOK=BCPOK, SR=SR, VTBI=VTBI, HH=(if (MM=0 andalso HH=0) then 0 else if (MM>=1) then HH else if (MM=0 andalso HH>=1) then HH-1 else HH), MM=(if (MM=0 andalso HH=0) then 0 else if (MM>=1) then MM-1 else if (MM=0) then 59 else MM), IR=IR})
S_SetNewDuration_I_RateConfirm to Z	({BC=BC, KP=KP, PL=PL, TL=TL, BD=BD, SS=SS, VL=VL, PP=PP,

	SOK=SOK, BCPOK=BCPOK, SR=SR, VTBI=VTBI, HH=HH, MM=MM, IR=(60*(VTBI div ((60*HH)+ MM)))})

Table 6.7: Arc Expression setduration

An arc expression of the arc *S_DecreaseDurationtoZ* represents *Decrease-Duration* Z operation schema. Firing of transition *S_DecreaseDuration* changes both *HH* and *MM* observations. If *MM* and *HH* both are zero, then firing of the transition will not decrease the values further as negative values are not acceptable. If $MM \geq 1$, then *HH* will remain the same and the value of *MM* will decrease by one. If $MM = 0$ and $HH \geq 1$, then *HH* is reduced by one. If $MM = 0$, then firing of the transition will change the value of *MM* to 59.

Transition *S_SetNewDuration_I_RateConfirm* sets the new duration and changes the state to *RateConfirm*. Both behaviours occurs simultaneously. An arc expression of the arc *S_SetNewDuration_I_RateConfirmtoZ* shows the *SetNewDuration* Z operation schema. It changes the value of the observation *IR* according to the formula $(60 * (VTBIdiv((60 * HH) + MM)))$.

7. **RateConfirm Page**

   RateConfirm page is shown in Figure 6.28. The marking of the place *RateConfirm* shows that there is just one S-behaviour which displays the current infusion rate on the screen. So only fusion place *Z* is added to the page.

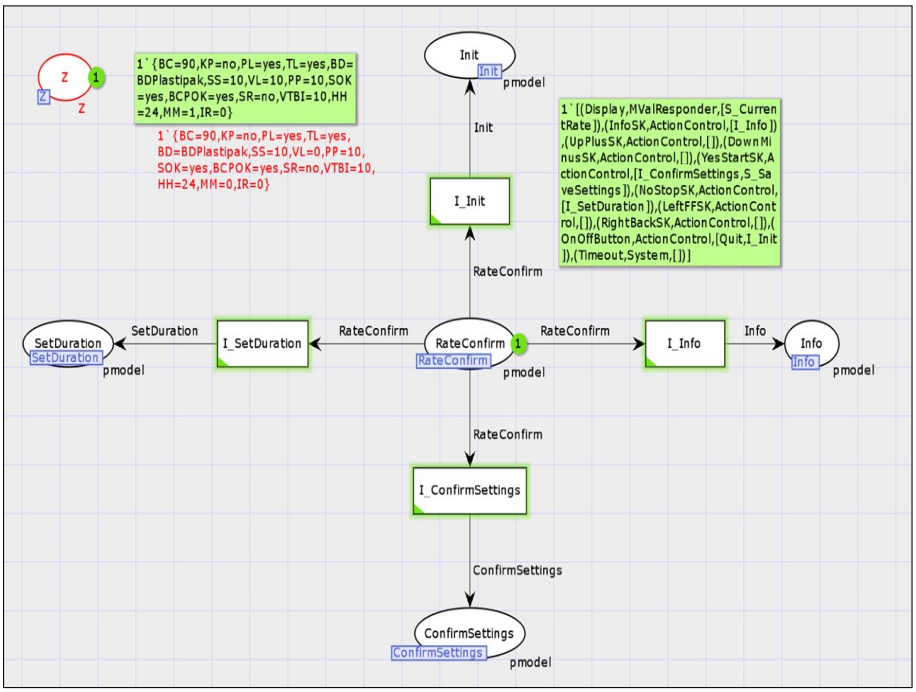

Figure 6.28: RateConfirm Page with Z

8. **ConfirmSettings Page**

   Figure 6.29 shows the structure of the *ConfirmSettings* page. This state has four S-behaviours: *S_CurrentVTBI, S_CurrentDuration*, *S_CurrentRate* and *S_ConfirmMsg*. These S-behaviours displays the information about the current volume to be infused, current duration, current infusion rate and a message asking user to confirm these settings. As it is only displaying information on the screen and not changing any underlying functionality we do not model it in CPN. So only fusion place *Z* is added to the page.

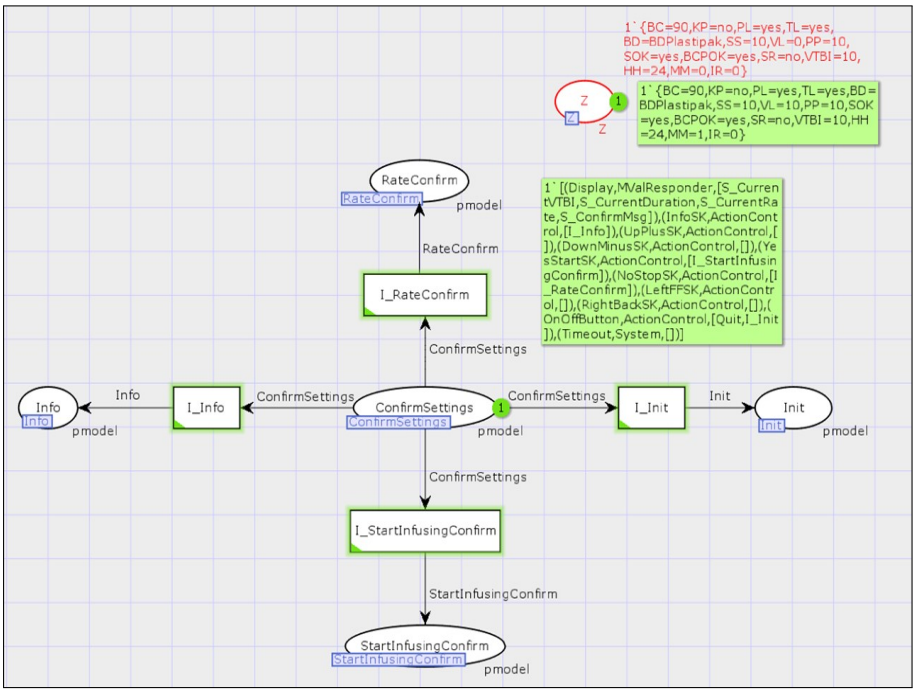

Figure 6.29: ConfirmSettings Page with Z

9. **StartInfusingConfirm Page**

The structure of the page *StartInfusingConfirm* is shown in Figure 6.30. Marking of the place *StartInfusingConfirm* shows two S-behaviours: *S_StartInfusingMsg* and *S_Infusing*. *S_StartInfusingMsg* displays a message on the screen which is not a part of this model. *S_Infusing* starts the infusion and changes that value of the variable *SR* from *no* to *yes*. As *S_Infusing* and *I_Infusing* occurs simultaneously, we have one transition named *S_Infusing_I_Infusing*. An arc expression of the arc *S_Infusing_I_Infusing* shown in Table 6.8 represents the *Infusing* Z operation schema. The arc from *Z* to *S_Infusing_I_Infusing* simply contains assignments which set each variable to its current value. This set of assignments picks up the current values of the variables ready to be used by the second arc. This second arc, the one from *S_Infusing_I_Infusing* to *Z* as shown in Table 6.8, assigns each variable to its *new* value, as given by the right-hand side of each equation in the *Infusing* operation schema. Taken together these two arcs express the intent of the equations in the operation schema *Infusing*.

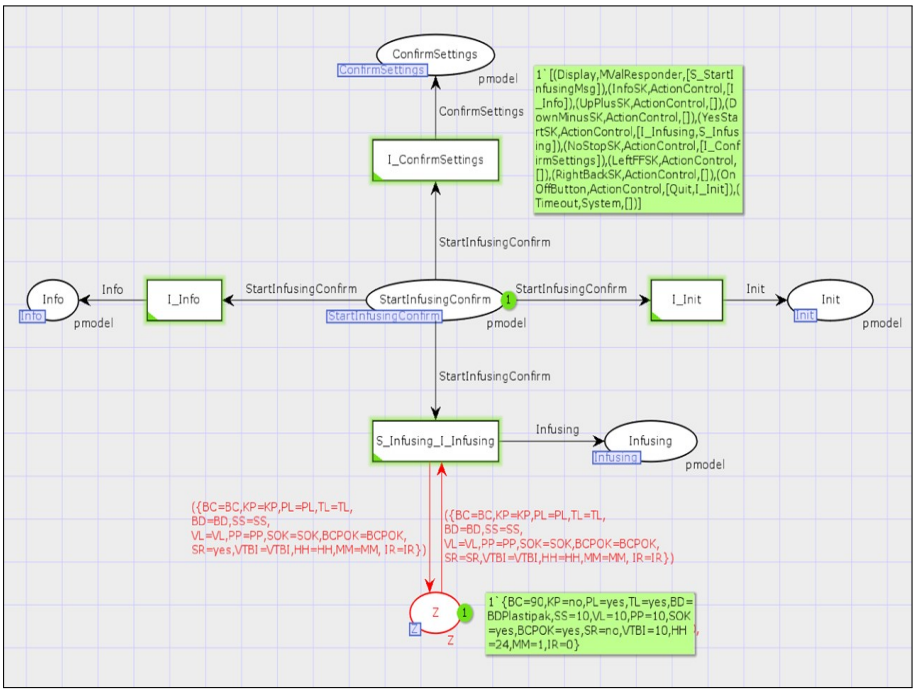

Figure 6.30: StartInfusingConfirm Page with Z

Arc	Arc Expression
S_Infusing_I_Infusing to Z	({BC=BC, KP=KP, PL=PL, TL=TL, BD=BD, SS=SS, VL=VL, PP=PP, SOK=SOK, BCPOK=BCPOK, SR=yes, VTBI=VTBI, HH=HH, MM=MM, IR=IR})

Table 6.8: Arc Expression StartInfusingConfirm

10. **Infusing Page**

Infusing state with S-behaviours is shown in Figure 6.31.

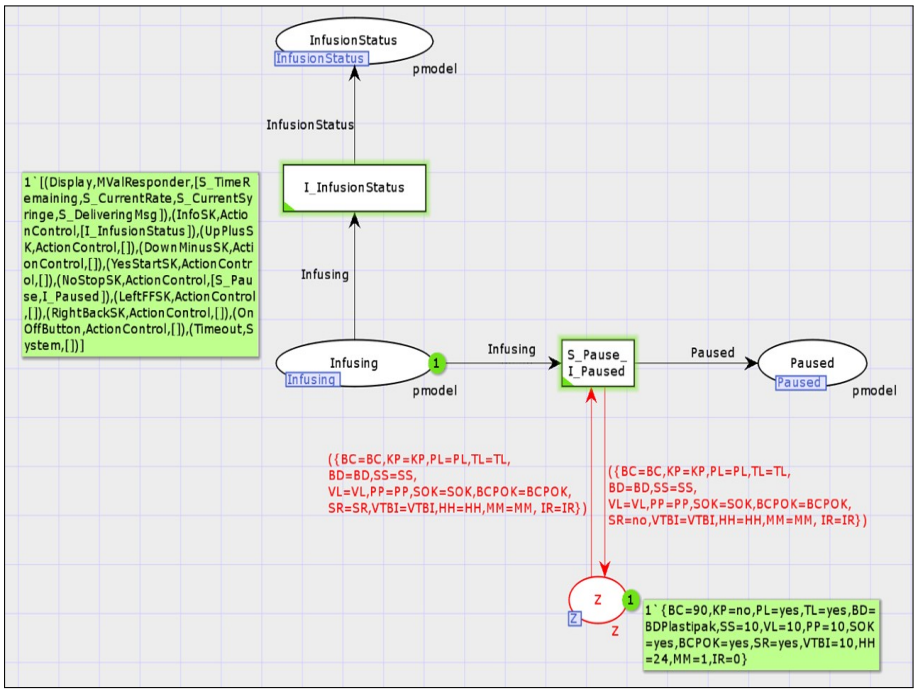

Figure 6.31: Infusing Page with Z

The marking of the place *Infusing* shows that the widget *Display* has four S-behaviours that shows the information about the syringe, remaining time, current infusing rate and delivering message which is not a part of this model. If a user presses *NoStopSk*, then infusion is paused which is expressed by the *S_Pause* behaviour. *S_Pause* pauses the infusion and changes that value of the variable *SR* from *yes* to *no*. As *S_Pause* and *I_Pause* occur simultaneously, we have one transition named *S_Pause_I_Pause*. An arc expression of the arc *S_Pause_I_Pause* shown in Table 6.9 represents the *Paused* Z operation schema.

Arc	Arc Expression
S_Pause_I_Paused to Z	({BC=BC, KP=KP, PL=PL, TL=TL, BD=BD, SS=SS, VL=VL, PP=PP, SOK=SOK, BCPOK=BCPOK, SR=no, VTBI=VTBI, HH=HH, MM=MM, IR=IR})

Table 6.9: Arc Expression StartInfusingConfirm

11. **InfusionStatus Page**

Figure 6.32 shows the *InfusionStatus* page with S-behaviours. While in *InfusionStatus* state, if a user presses *NoStopSk* button, then the system goes to *Paused* state which changes the value of the observation *SR* to *no*. An arc expression of the arc *S_Pause_I_Paused* to *Z* is the same as shown in Table 6.9.

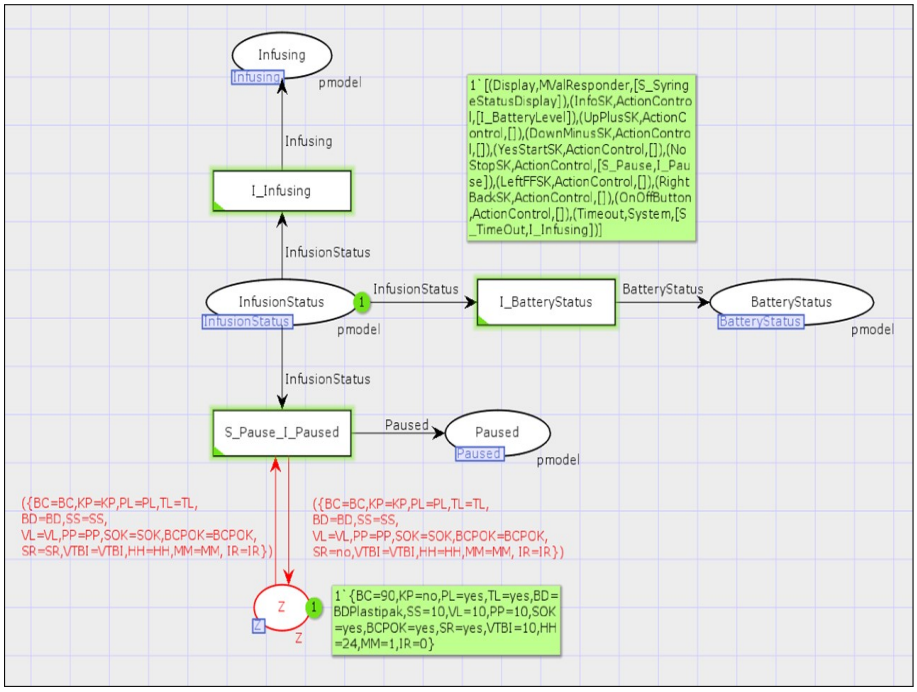

Figure 6.32: InfusionStatus Page with Z

12. **BatteryStatus Page**

The *BatteryStatus* state displays the information about the battery and is shown in Figure 6.33. It also has *S_Pause* behaviour which changes the value of the observation *SR* to *no*. An arc expression of the arc *S_Pause_I_Paused* to *Z* is the same as shown in Table 6.9.

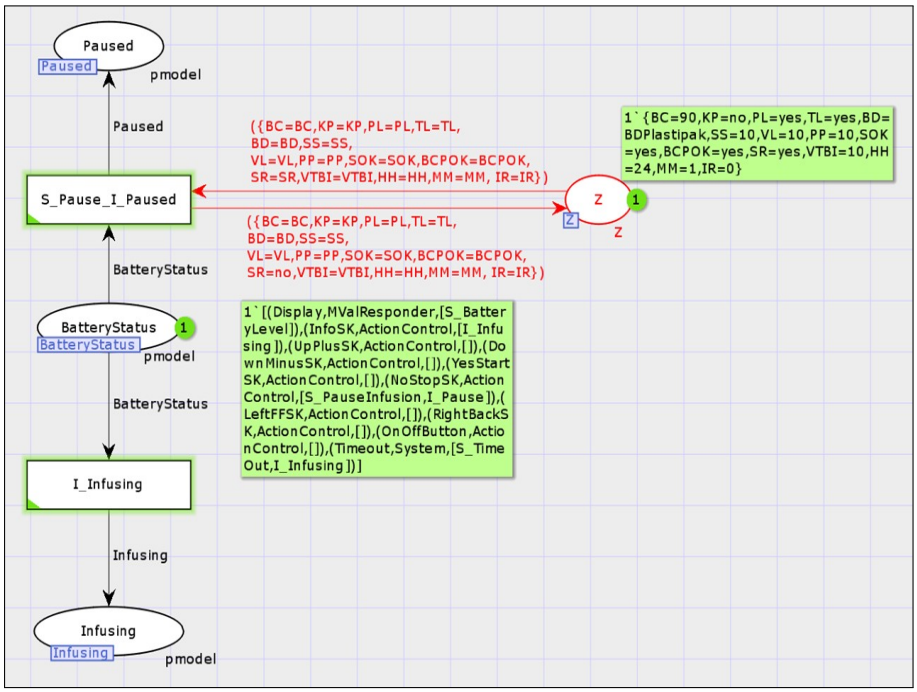

Figure 6.33: Battery Status Page with Z

13. **Paused Page**

Figure 6.34 shows the *Paused* state with S-behaviours. The marking of the place *Paused* shows that the widget *Display* has one S-behaviour *S_PumpStoppedWarning* that displays a message on the screen which is not a part of this model. If a user presses *YesStartSK* button in this state, then infusion is resumed which is expressed by the *S_ResumeInfusing* behaviour which is represented as a transition. The transition *S_Resume Infusing_I_Infusing* resumes the infusion and changes that value of the variable *SR* to *yes*. An arc expression of the arc *S_ResumeInfusing _I_Infusing* to *Z* is shown in Table 6.10 represents the *ResumeInfusing* Z operation schema.

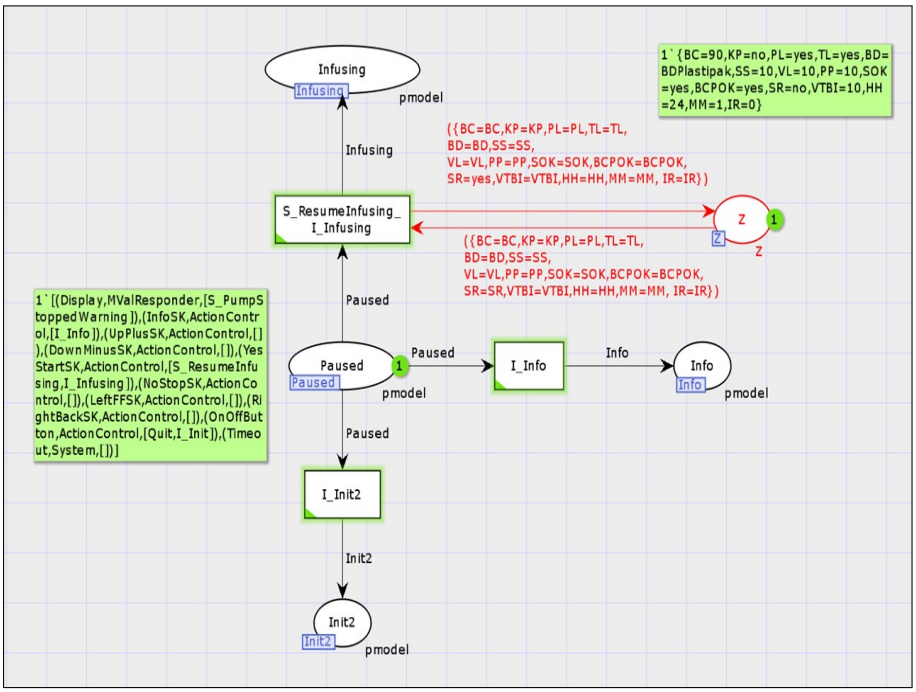

Figure 6.34: Paused Page with Z

Arc	Arc Expression
S_ResumeInfusing_I_Infusing to Z	({BC=BC, KP=KP, PL=PL, TL=TL, BD=BD, SS=SS, VL=VL, PP=PP, SOK=SOK, BCPOK=BCPOK, SR=yes, VTBI=VTBI, HH=HH, MM=MM, IR=IR})

Table 6.10: Arc Expression Paused

14. **Inittwo Page**

Inittwo is the state when a user go to the initial state when the device is paused. Then the device gives an option to a user to resume infusion from the point it was paused. The structure of the *Inittwo* page is shown in Figure 6.35. Fusion place *Z* is added to the page as there is only one S-behaviour *S_SyringeDisplay* which displays the information about the syringe on the screen.

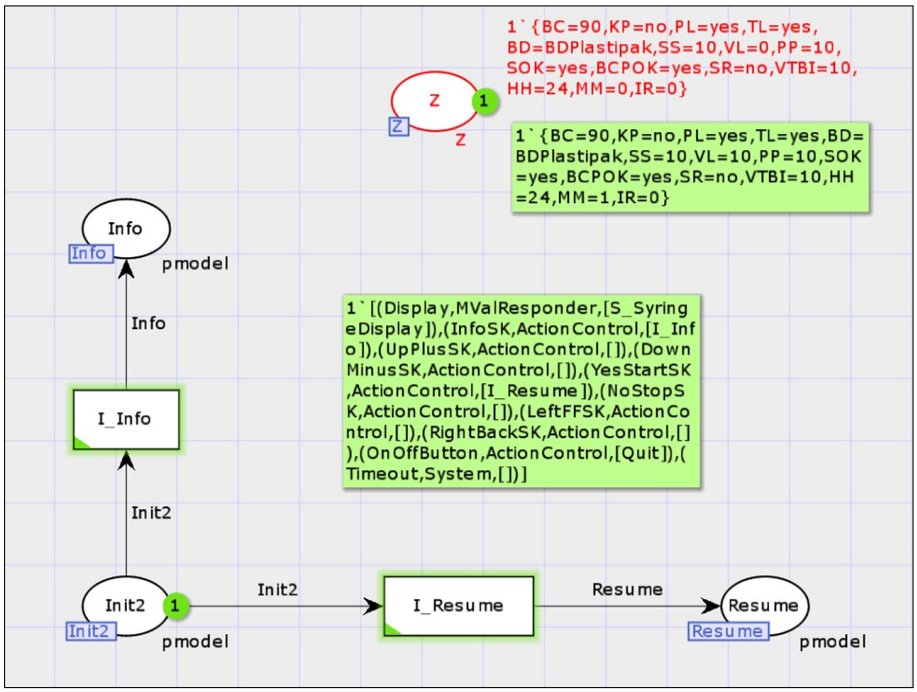

Figure 6.35: Inittwo Page with Z

15. **Resume Page**

Figure 6.36 shows the *Resume* page.

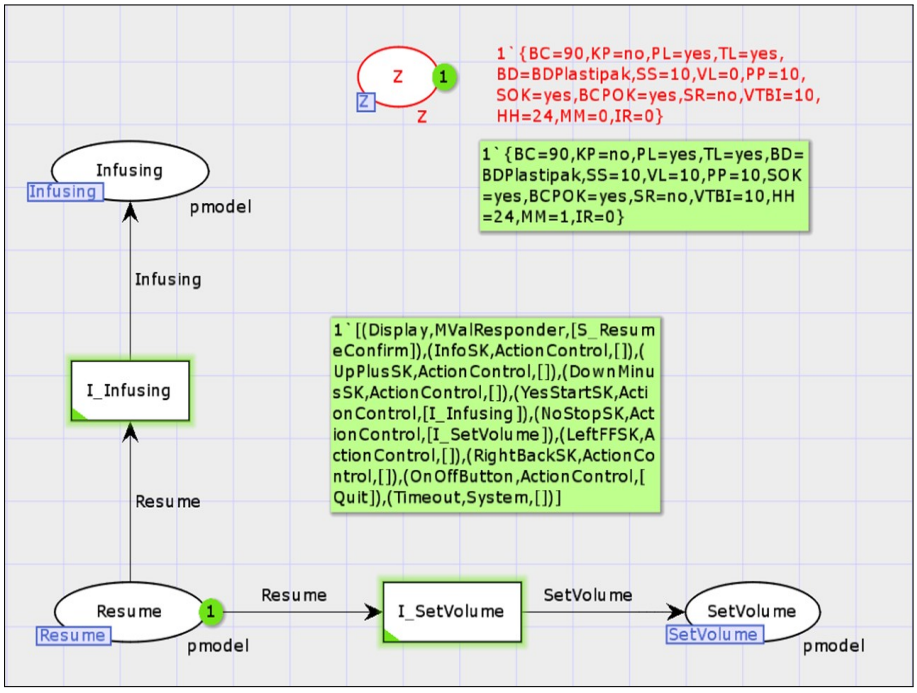

Figure 6.36: Resume Page with Z

The marking of the place *Resume* shows that there is one S-behaviour *S_ResumeConfirm* which shows the message on the screen. So the fusion place *Z* is added to the page.

16. **BatteryLevel page**

Figure 6.37 shows the *BatteryLevel* page. The marking of the place *BatteryLevel* shows that there is one S-behaviour *S_BatteryLevel* which shows the message on the screen. So the fusion place *Z* is added to the page.

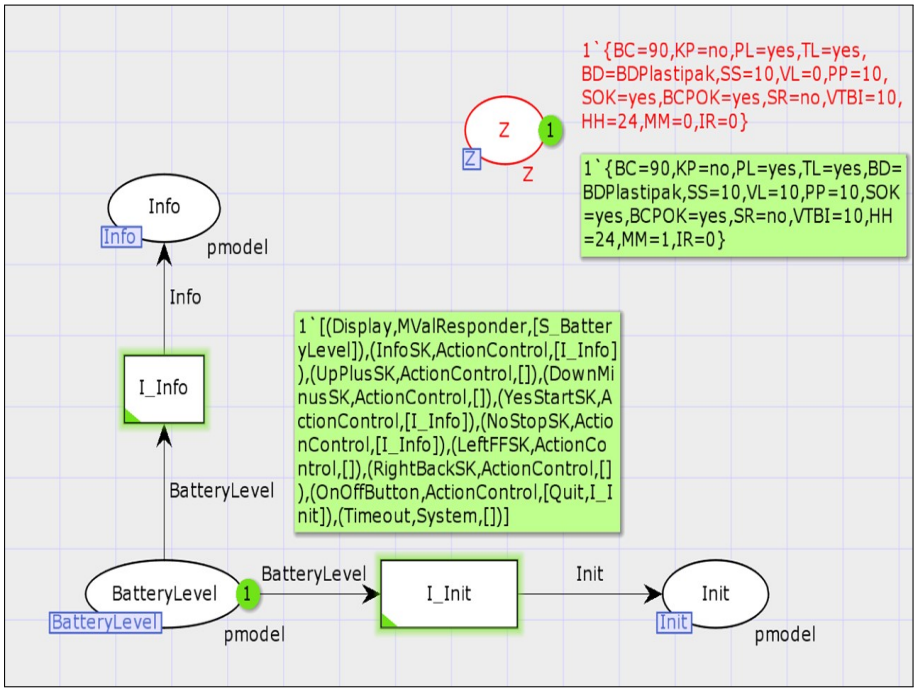

Figure 6.37: BatteryLevel Page with Z

17. **ChangeSetUp Page**

Figure 6.38 shows the *ChangeSetUp* page. The fusion place *Z* is added to this page.

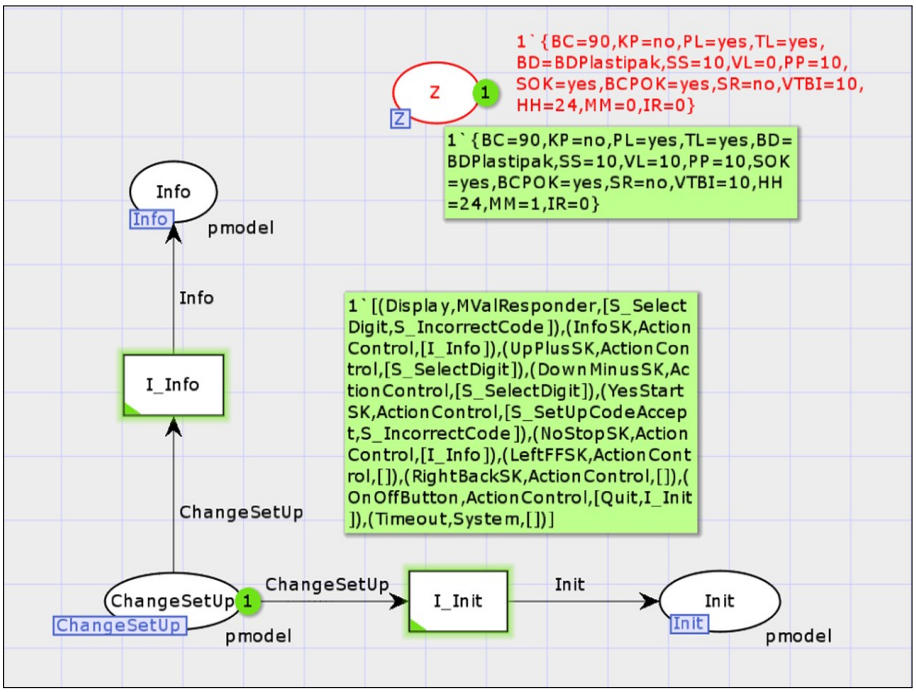

Figure 6.38: Change Set Up Page with Z

18. **EventLog Page**

Figure 6.39 shows the *EventLog* page. The fusion place *Z* is added to this page.

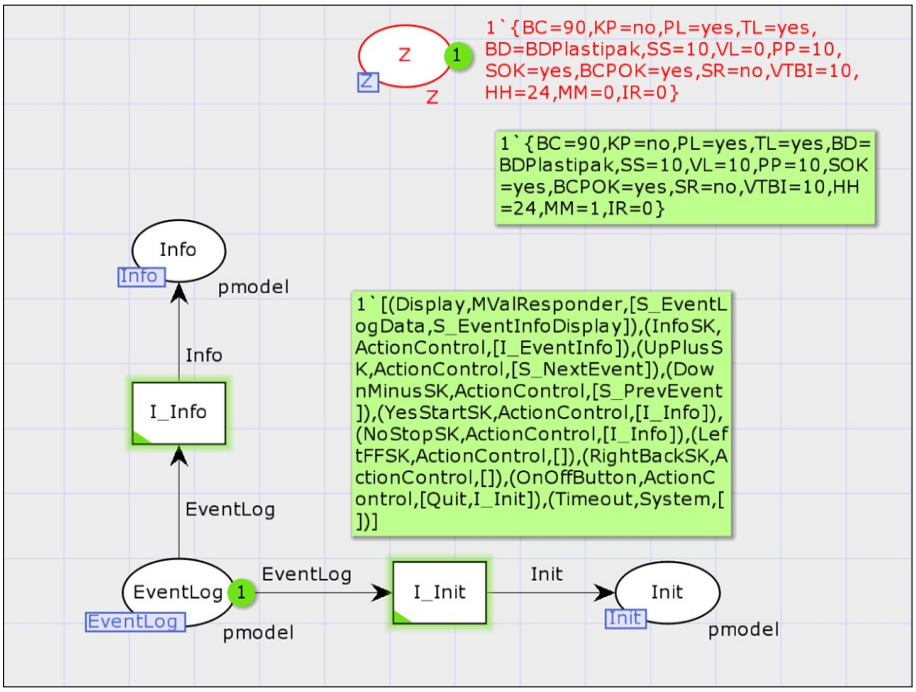

Figure 6.39: EventLog Page with Z

19. **RateSet Page**

Figure 6.40 shows the *RateSet* page. The fusion place *Z* is added to this page.

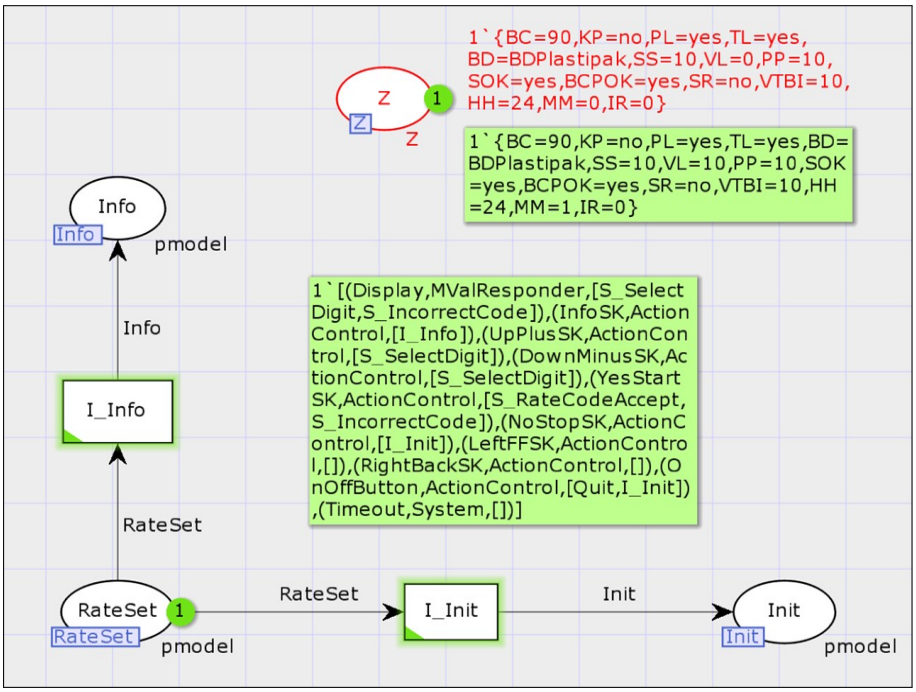

Figure 6.40: RateSet Page with Z

We have now the complete CPN model of the Niki T34 syringe driver which includes all the three aspects of interest: user interface, interaction and functionality. We now analyze the model and investigate its behaviour.

6.6 Analysis

In this section we will investigate the same general properties as discussed in Chapter 4 and Chapter 5 for the complete CPN model of the Niki T34 syringe driver. The real device works for upto 24 hours and can infuse upto 10ml. If we try to do the model checking using these values then we come across the state explosion problem. So we use some straightforward ways of shrinking the state space, for the purposes of model checking. For example, a duration might stretch over 24 hours in a real device, but in this research we model check on a duration that has five hours as its maximum. This makes the state space

far smaller, but does not mean we miss checking things like upper bounds and lower bounds of duration.

To prevent the state space explosion, for analysis we assume the following:

- The maximum value of the volume to be infused (VTBI) is 3.
- The maximum value of the duration for infusion is 5 hours. (We assume that instead of 60 minutes in an hour, there are 9 minutes in an hour)

The detailed state space report of CPN model of T34 syringe driver with all three aspects (user interface, interaction and functionality) is given in [2] and is summarized in Table 6.11

State Space Graph	
Number of Nodes	19213
Number of Arcs	52944
SCC Graph	
Number of Nodes	3
Number of Arcs	4
Number of Dead Markings	None
Dead Transition Instances	None

Table 6.11: statistics of the CPN model of user interface and interaction and functionality of T34 syringe driver

6.6.1 Investigation of General Properties

6.6.1.1 Livelock

It is important to detect a livelock (i.e. a cycle in which no progress is being made) in medical infusion pumps. This requirement is related to identifying hazards occurring from the "failures due to normal or abnormal use during operational use, including sequences of use resulting in failure" as specified by U.S. FDA in[3] or error in the user interface design. For example, if a user is stuck in a loop and not making any progress.

[2] See `https://github.com/sapnajaidka/Niki-T34/blob/master/T34withZ-SSReport` for complete details

[3] `https://www.fda.gov/media/78369/download`

We will use the same method as described in Section 4.4.2.2 to detect a livelock in the CPN model of a T34 syringe driver.

Firstly, we need to check if there exists self-loops in the state space graph. We apply the same function given in Table 4.10 to the CPN model of the T34 syringe driver and gets an empty output list as shown in Table 6.12. This means that there are no nodes in the state space graph of the T34 syringe driver that end in self loops.

```
fun SelfLoopNodes n = (OutNodes (n) = [n])
fun SelfLoopTest()=PredNodes(EntireGraph,
fn n => (SelfLoopNodes n),
NoLimit);
```

```
Output:
val it =[ ] :  Node list
```

Table 6.12: Detecting the absence of self loops in T34 syringe driver model

Secondly, we need to check if the state space graph is isomorphic to its SCC graph. In our example, the number of nodes and arcs in the state space graph of the T34 syringe driver are not equal to the number of nodes and arcs in the SCC graph as shown in the Table 6.11 and also the strongly connected components consists of more than one node. Hence, we can not say that the state space graph is isomorphic to its SCC graph. Therefore we can not prove the absence of livelock with this method. So there is a need to check the non-trivial terminal SCCs. Now we apply the function given in Table 4.12 to the CPN model of a T34 syringe driver. Table 6.13 shows that ~3 is the non-trivial SCC node, i.e., Node ~3 is a terminal SCC node that contains more than one node and also has arcs. Hence, the model contains a livelock.

```
fun ListTerminalSCCs()=PredAllSccs(SccTerminal);
fun Livelock()=PredSccs(ListTerminalSCCs(),
fn n => not (SccTrivial n),
NoLimit);
```

```
Output:
val it=[~3] :  Scc list
```

Table 6.13: Query to find a non-trivial terminal SCC in T34 syringe driver

After investigating our model we found that it was obvious that the model has livelock because in the *infusing* state, we haven't modelled the *S_tick* behaviour that will decrease the volume and duration as the time passes. In the current model, once the values of the observations *VolumeLeft*(*VL*), *Hours*(*HH*) and *Minutes*(*MM*) are set in *SetVolume* and *SetDuration* states, they remain constant and does not change while the device is infusing in the *infusion* state. This creates confusion for a user as a user might believe that infusion is not occurring, but it is.

To fix this, we made changes in the *infusion* page of the CPN model of the T34 syringe driver as shown in Figure 6.41. We have added two more transitions: *S_tick* and *Exit* to the page. The Z operation schema *tick* is given below:

tick

$\Delta T34$

$BatteryCharge' = BatteryCharge$

$KeyPadLocked' = KeyPadLocked$

$ProgramLocked' = ProgramLocked$

$TechMenuLocked' = TechMenuLocked$

$Brand' = Brand$

$SyringeSize' = SyringeSize$

$VolumeLeft' = VolumeLeft - InfusionRate$

$PlungerPosition' = PlungerPosition$

$SyringeOK' = SyringeOK$

$BarrelOK' = BarrelOK$

$CollarOK' = CollarOK$

$PlungerOK' = PlungerOK$

$SystemReady' = SystemReady$

$VTBI' = VTBI$

$(Minutes = 0 \land Hours = 0) \Rightarrow (Minutes' = 0 \land Hours' = 0)$

$Minutes \geq 1 \Rightarrow (Minutes' = Minutes - 1 \land Hours' = Hours)$

$Minutes = 0 \Rightarrow (Minutes' = 59 \land ((Hours \leq 1 \Rightarrow$

$Hours' = Hours - 1) \land$

$(Hours = 0 \Rightarrow Hours' = 0)))$

$InfusionRate' = InfusionRate$

The tick operation represents infusion of the drug in a patient's body every minute as per the infusion rate. After every minute the value of the observation *VolumeLeft* is updated and duration is decreased by one minute which updates the values of the observations *HH* and *MM*. The *tick* operation is represented by the *S_tick* transition in Figure 6.41. The arcs from *Z* to the transition *S_tick* contains assignments which set each variable to its current value. This set of assignments "picks up" the current values of the variables ready to be

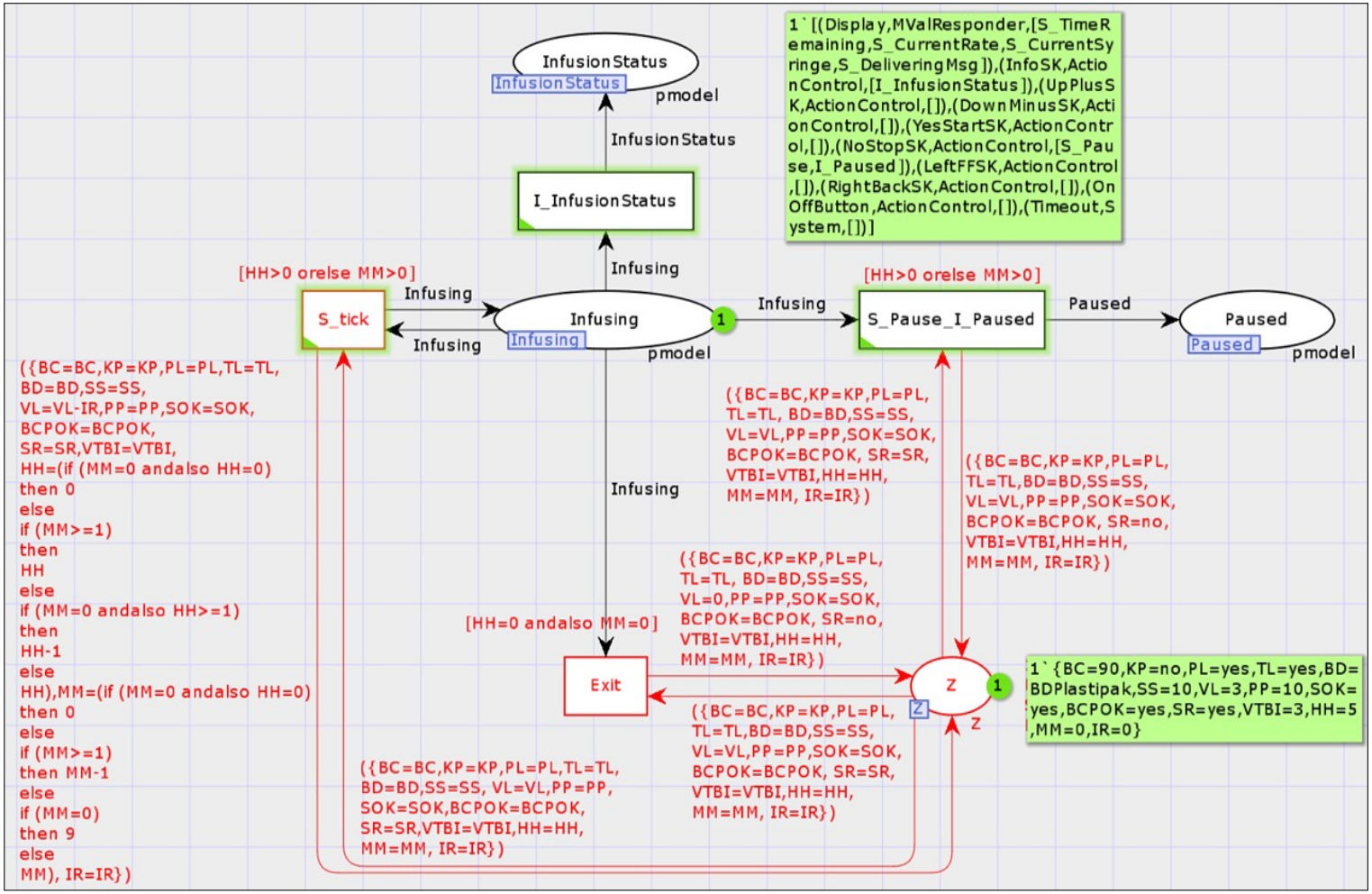

Figure 6.41: Modified Infusion Page with Z

used by the arcs going from this S-behaviour transition to the place Z. The arc expression from the place *S_tick* to the place Z is shown in Figure 6.41 and Table6.14. This assigns each variable to its *new* value, as given by the right-hand side of each equation in the *tick* Z operation schema. There is a guard $[HH > 0$ *orelse* $MM > 0]$ on the transition *S_tick* which means that this transition will only be enabled if the values of the observation *HH* or *MM* is greater than zero.

Arc	Arc Expression
S_tick to Z	({BC=BC, KP=KP, PL=PL, TL=TL, BD=BD, SS=SS, VL=VL-IR, PP=PP, SOK=SOK, BCPOK=BCPOK, SR=SR, VTBI=VTBI, HH=(if (MM=0 andalso HH=0) then 0 else if (MM>=1) then HH else if (MM=0 andalso HH>=1) then HH-1 else HH), MM=(if (MM=0 andalso HH=0) then 0 else if (MM>=1) then MM-1 else if (MM=0) then 59 else MM), IR=IR})

Table 6.14: Arc Expression infusing modified

The *Exit* transition is added so that the pump stops when the remaining duration is zero i.e. when the values of the observations *HH* and *MM* both are zero. This breaks the livelock as after completing the infusion pump stops. There is a guard $[HH = 0$ *andalso* $MM = 0]$ on the transition *Exit* which means that the transition will only be enabled if the values of the observation *HH* or *MM* is zero.

A state space graph is calculated again for a new modified model and can

be found in [4] and summarized below:

State Space Graph	
Number of Nodes	16663
Number of Arcs	46032
SCC Graph	
Number of Nodes	9
Number of Arcs	18
Number of Dead Markings	6 [5696, 5651, 5625, 5338, 5298, 5212]
Dead Transition Instances	None

Table 6.15: statistics of the modified CPN model of the T34 infusion pump

As from Table 6.15 we can see that number of state space nodes and SCC nodes are different which means the state space graph is not isomorphic to its SCC graph. So there is a need to check the non-trivial terminal SCCs. Now we check for livelock again by applying the same function as in Table 6.13 which results in an empty list as shown in Table 6.16. Hence, the model is free from livelock.

```
fun ListTerminalSCCs()=PredAllSccs(SccTerminal);
fun Livelock()=PredSccs(ListTerminalSCCs(),
fn n => not (SccTrivial n),
NoLimit);
```

```
Output:
val it=[ ] :  Scc list
```

Table 6.16: Query to find a non-trivial terminal SCC in modified T34 syringe driver

6.6.1.2 Deadlock freedom

In this section we investigate the CPN model of the device for the absence of deadlock. To ensure good usability, it is important for medical infusion pumps to be deadlock-free because we don't want users to enter into a state in which no action can be taken. We use the same method for checking the absence of

[4] See `https://github.com/sapnajaidka/Niki-T34/blob/master/T34Modified-SSReport` for complete details

deadlock as described in Section 4.4.2.1. Table 6.17 shows the dead marking in the model.

ListDeadMarkings()
output: val it = [5696, 5651, 5625, 5338, 5298, 5212]: Node list

Table 6.17: Dead markings of the modified T34 syringe driver model

As we now know that when the *Exit* transition is fired then there is no other enabled transition in the model because the pump has completed the infusion. We need to apply a function given in Table 5.6 that detects the dead markings that are not intended (i.e. not terminal markings) for our model.

```
fun ValidTerminalMarking n = (Mark.Infusing'Z 1 n =
1‘BC=90,KP=no,PL=yes,TL=yes,BD=BDPlastipak,SS=10,VL=0,PP=10,
SOK=yes,BCPOK=yes,SR=no,VTBI=2,HH=0,MM=0,IR=2) orelse
(Mark.Infusing'Z 1 n = 1‘BC=90,KP=no,PL=yes,TL=yes,
BD=BDPlastipak,SS=10,VL=0,PP=10,SOK=yes,BCPOK=yes,SR=no,
VTBI=2,HH=0,MM=0,IR=1) orelse
(Mark.Infusing'Z 1 n = 1‘BC=90,KP=no,PL=yes,TL=yes,
BD=BDPlastipak,SS=10,VL=0,PP=10,SOK=yes,BCPOK=yes,SR=no,
VTBI=3,HH=0,MM=0,IR=0) orelse
(Mark.Infusing'Z 1 n = 1‘BC=90,KP=no,PL=yes,TL=yes,
BD=BDPlastipak,SS=10,VL=0,PP=10,SOK=yes,BCPOK=yes,SR=no,
VTBI=2,HH=0,MM=0,IR=0) orelse
(Mark.Infusing'Z 1 n = 1‘BC=90,KP=no,PL=yes,TL=yes,
BD=BDPlastipak,SS=10,VL=0,PP=10,SOK=yes,BCPOK=yes,SR=no,
VTBI=3,HH=0,MM=0,IR=3) orelse
(Mark.Infusing'Z 1 n = 1‘BC=90,KP=no,PL=yes,TL=yes,
BD=BDPlastipak,SS=10,VL=0,PP=10,SOK=yes,BCPOK=yes,SR=no,
VTBI=3,HH=0,MM=0,IR=1)
fun InValidTerminal()=PredNodes(ListDeadMarkings(),
fn n => not (ValidTerminalMarking n),
NoLimit);
```

```
output:
val it =[]:  Node list
```

Table 6.18: Invalid terminal nodes for the modified T34 syringe driver model detecting deadlock

We have an empty list as value i.e. all the dead markings are terminal markings. Hence there are no deadlocks.

6.6.1.3 Checking Boundary Values

One notorious place where models go wrong is at the boundary values of the data values. It is important to check whether upper and lower bounds on values are properly respected and handled. For example, limits on the volume to be infused or limits on the time of the infusion. This relates to one of the safety requirement "for small-volume pumps (i.e., pumps that provide microinfusion flows as low as 0.1 ml/hr), VTBI (Volume to be Infused) settings shall cover the range from 0.1 to 999 ml" specified in [128] for generic infusion pumps for approval by the FDA.

The state space method allows us to check if there exist values which violate these limits. In Niki T34 syringe driver we have assumed that the pump operates for a maximum of five hours and the time can not be negative. Also one hour has nine minutes in it. So we need to check the boundary values of variables *HH* that represent hours and *MM* that represents minutes. The question we need to answer is: *Does time work properly during the set duration state?* For this we need the following to be true when a user is increasing a duration: when the value of the variable *HH* is maximum, i.e., 5, then the value of the variable *MM* should be 0.

We verify this by writing the code given in Table 6.19 to query the state space to see if there is any marking where the value of *MM* is greater than zero when the value of *HH* is 5.

```
fun checkvaluehours(HH:hours, MM:minutes):  Arc list =
PredAllArcs (
fn a=> case ArcToBE a of
Bind.SetDuration'S_IncreaseDuration(1,{BC=_,KP=_,PL=_,
TL=_,BD=_,SS=_,VL=_,PP=_,SOK=_,BCPOK=_,SR=_,VTBI=_,
HH=5,MM=1,IR=_}) => HH = 5 andalso MM= 1| _ => false)
```

```
Output:
val it=[48568, 47646, 46693, 46591, 45673, 38, 152]:  Arc list
```

Table 6.19: Returns all the arcs in the state space graph where the value of *HH* is 5 and *MM* is 1

We get an arc list: [48568, 47646, 46693, 46591, 45673, 38, 152] as an

output. This means that there exist arcs in the entire state space that has a binding element $MM = 1$ when $HH = 5$. This means that user is allowed to set the duration that is more than 5 hours which violates the boundary values which needs to be fixed.

To solve this problem, we need to fix the arc expression from the transition *S_IncreaseDuration* to the place Z. The current arc expression is:

```
({BC=BC, KP=KP, PL=PL, TL=TL, BD=BD, SS=SS, VL=VL, PP=PP,
SOK=SOK, BCPOK=BCPOK, SR=SR, VTBI=VTBI, HH=(if MM<=8 then HH
else if (MM=9 andalso HH<=4) then (HH+1) else if (HH=5) then 0
else HH), MM=(if (MM<=8) then MM+1 else if MM=9 then 0 else 0),
IR=IR})
```

In the current expression we can see that if MM is greater than equal to 8 then its value is increased by 1. It does not have any relation with the value of HH. That is why even after having $HH = 5$, it increases the value of MM. One way of fixing it is using the arc expression as written below:

```
({BC=BC, KP=KP, PL=PL, TL=TL, BD=BD, SS=SS, VL=VL, PP=PP,
SOK=SOK, BCPOK=BCPOK, SR=SR, VTBI=VTBI, HH=(if MM<=8 then HH
else if (MM=9 andalso HH<=4) then (HH+1) else if (HH=5) then 0
else HH), MM=(if (MM<=8 andalso HH<5) then MM+1 else if MM=9
then 0 else 0), IR=IR})
```

After fixing we again apply the function written in Table 6.19 and we get the empty arc list as an output as shown in Table 6.20. This means that there exists no such arc in the entire state space that has a binding element $MM = 1$ when $HH = 5$. This means that a user can not increase the duration after 5 hours.

```
fun checkvaluehours(HH:hours, MM:minutes):  Arc list =
PredAllArcs (
fn a=> case ArcToBE a of
Bind.SetDuration'S_IncreaseDuration(1,{BC=_,KP=_,PL=_,
TL=_,BD=_,SS=_,VL=_,PP=_,SOK=_,BCPOK=_,SR=_,VTBI=_,
HH=5,MM=1,IR=_}) => HH = 5 andalso MM= 1| _ => false)
```

```
Output:
val it=[ ]:  Arc list
```

Table 6.20: Returns all the arcs in the state space graph of modified model where the value of *HH* is 5 and *MM* is 1

As per our assumption, an hour consists of 9 minutes. So, we also need to check that the value of the variable *MM* should not exceed 9 in the entire state space. We can check this by applying the function written in Table 6.21. We get the empty arc list as an output which means that there exists no such arc in the entire state space that has a binding element $MM = 10$.

```
fun checkvalueminutes(MM:minutes):  Arc list =
PredAllArcs (
fn a=> case ArcToBE a of
Bind.SetDuration'S_IncreaseDuration(1,{BC=_,KP=_,PL=_,
TL=_,BD=_,SS=_,VL=_,PP=_,SOK=_,BCPOK=_,SR=_,VTBI=_,
HH=_,MM=10,IR=_}) => MM = 10| _ => false)
```

```
Output:
val it=[ ]:  Arc list
```

Table 6.21: Returns all the arcs in the state space graph of modified model where the value of *MM* is 10

We have checked the maximum duration that a user can set for the device. The other boundary concerns the minimum duration for an infusion. Here a question to ask is: when the duration for an infusion is being set up by a user, is it possible to go below zero minutes?

We can check this by applying the function written in Table 6.22. We get the empty arc list as an output which means that there exists no such arc in the entire state space that has a binding element $MM = \tilde{}1$.

```
fun checkvalueminutes(MM:minutes):  Arc list =
PredAllArcs (
fn a=> case ArcToBE a of
Bind.SetDuration'S_DecreaseDuration(1,{BC=_,KP=_,PL=_,
TL=_,BD=_,SS=_,VL=_,PP=_,SOK=_,BCPOK=_,SR=_,VTBI=_,
HH=_,MM=(~1),IR=_}) => MM = (~1)| _ => false)
```

```
Output:
val it=[ ]:  Arc list
```

Table 6.22: Returns all the arcs in the state space graph of modified model where the value of MM is ~1)

We have checked the boundary values that a user is allowed for setting the duration.

Similarly, we can check the boundary values for the volume to be infused. According to our assumption, the pump can infuse up to 3ml of dose and a minimum dose allowed is 1ml. This means a user can set the value of the variable $VTBI$ ranging from 1 to 3.

We can check if there exists any such arc in the entire state space that has a binding element $VTBI = 0$ or $VTBI = 4$. This is done by running the functions given in Table 6.23 and 6.24. Both the table results in an empty arc list. This means that there exists no binding element $VTBI = 0$ or $VTBI = 4$ in the entire state space. So our model is respecting the upper and lower bounds of the volume to be infused.

```
fun checkvaluevtbi(VTBI:millilitres):  Arc list =
PredAllArcs (
fn a=> case ArcToBE a of
Bind.SetVolume'S_SetVTBI_I_SetDuration(1,{BC=_,KP=_,PL=_,
TL=_,BD=_,SS=_,VL=_,PP=_,SOK=_,BCPOK=_,SR=_,VTBI=0,
HH=_,MM=_,IR=_}) => VTBI = 0| _ => false)
```

```
Output:
val it=[ ]:  Arc list
```

Table 6.23: Returns all the arcs in the state space graph of modified model where the value of $VTBI$ is 0

```
fun checkvaluevtbi(VTBI:millilitres):  Arc list =
PredAllArcs (
fn a=> case ArcToBE a of
Bind.SetVolume'S_SetVTBI_I_SetDuration(1,{BC=_,KP=_,PL=_,
TL=_,BD=_,SS=_,VL=_,PP=_,SOK=_,BCPOK=_,SR=_,VTBI=4,
HH=_,MM=_,IR=_}) => VTBI =4 | _ => false)
```

```
Output:
val it=[ ]:  Arc list
```

Table 6.24: Returns all the arcs in the state space graph of modified model where the value of *VTBI* is 4

6.7 Summary

In this chapter we have used CPNs to model a Niki T34 syringe driver. This not only shows the navigational properties of the device, but also allows us to include the underlying system functionality in the model as well. By means of simulation we can actually see what widgets are available and what happens when the user interacts with them. Also we can actually *see* how the behaviours change the underlying system functionality. By these means we can check to see if the model is working as expected.

In this research we have used the state space analysis method to investigate the behaviours of the CPN model to ensure if it is working as expected. If it seems that it is not then we can look deeper and see what the flaws are in the model and what changes should be made to make the model work correct in all situations: this is the most important thing for safety-critical devices. This method provides information about the dynamic properties of a system, for example, dead transitions, and dead markings. It also gives information about the fairness and liveness properties of a modelled system. Therefore it is possible to investigate the behaviour of the system in sophisticated and useful ways: this includes the safety requirements of the system. With the state space method, in conjunction with suitable queries, it is possible to verify that queries hold, so safety requirements (like detecting livelocks, total reachability, desired terminal states etc.) for safety-critical systems can be proved. In this

chapter, we have investigated the behaviours of a Niki T34 syringe driver and found livelock in the model. Then the model is modified and investigated again to check if it is working as expected.

Chapter 7

Case Study: Nuclear Power Plant System

7.1 Introduction

In previous chapters we have described how to model a user interface, interaction and functionality of a safety-critical interactive system from an existing method (PM/PIM/Z) and re-express in a new way using Coloured Petri Nets. The main technical point was to show carefully that each formalism in the original method can be expressed in Coloured Petri Nets, thus showing that there is no loss of expressiveness in moving from the established PM/PIM/Z method to Coloured Petri Nets. Knowing that we have retained all the expressiveness that we need, we can now use the Coloured Petri Nets alone to model user interface, interaction and functionality of safety-critical interactive systems.

In this chapter we present a case study on the reactor control system of a nuclear power plant. We model and investigate the behaviour of the user interface, interaction and functionality of the reactor control system of a nuclear power plant using Coloured Petri Nets alone.

7.2 Case Study: Reactor Control System of a Nuclear Power Plant

The reactor is the core of the power plant. The boiling water reactor is shown in Figure 7.1 [6]. There are three main components of the boiling water reactor: the *reactor vessel* on the left hand side of Figure 7.1 which contains uranium fuel and the control rods, the *turbine* which is connected to a generator that generates electrical energy and the *condenser* which condenses the steam back to water.

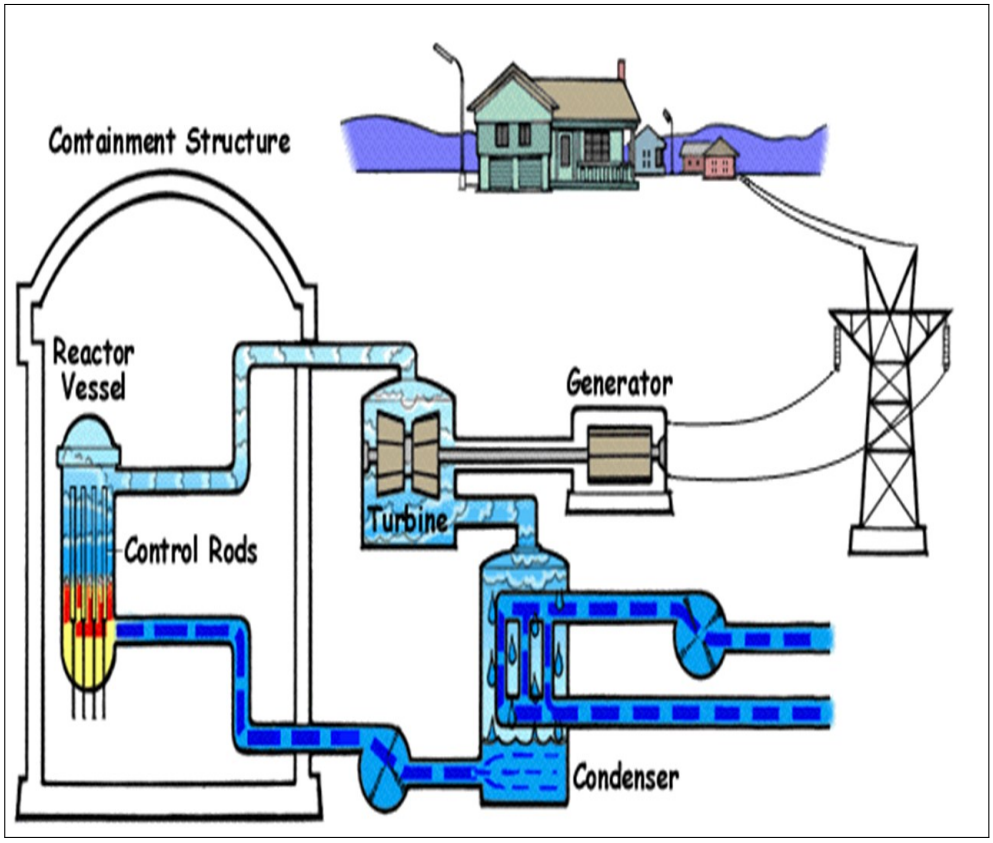

Figure 7.1: Boiling Water Reactor [6]

Water pumps are responsible for transporting water into the reactor vessel. The water level can be controlled by opening and closing water valves. Uranium fuel is filled into the reactor vessel and its atoms break apart and release heat energy. The amount of generated heat energy, and therefore steam, is controlled by the amount of water pumped into the reactor vessel and the

position of the control rods since they control the atomic decay and therefore the temperature. This steam is then transported by pipes to the turbine by opening the steam valves where the thermal energy stored in the steam is transformed into kinetic energy, which is then transformed to electrical energy in a connected generator. As the reactor is responsible for generating heat, it is considered safe or in stable state when the reactor pressure is no more than 70 bar and the temperature remains below 286 degree Celsius.

A high degree of automation is involved in the control of the nuclear power plant system to support the operator's tasks. The operator's tasks include the start up and shut down of the reactor, adjusting control rods which affect how much electricity a reactor generates, monitoring turbines, generators and cooling systems and making adjustments as necessary and responding to abnormalities and taking corrective action. The operator's main task is to observe the process because the safety system takes all the controls from the operator when the state becomes dangerous until it is returned to its stable state.

Figure 7.2 shows the nuclear power plant control interface. There are a total of twenty two widgets in the interface: *PowerDisplay, ReactorWaterLevelDisplay, ReactorPressureDisplay, CondenserWaterLevelDisplay, CondenserPressureDisplay, ControlRodLevel, WP1Control, WP2Control, CPControl, SV1Open, SV1Close, SV1Status, SV2Open, SV2Close, SV2Status, WV1Open, WV1Close, WV1Status, WV2Open, WV2Close, RStatusDisplay* and *WV2Status*. For brevity, we are not including the top set of status lights in our model.

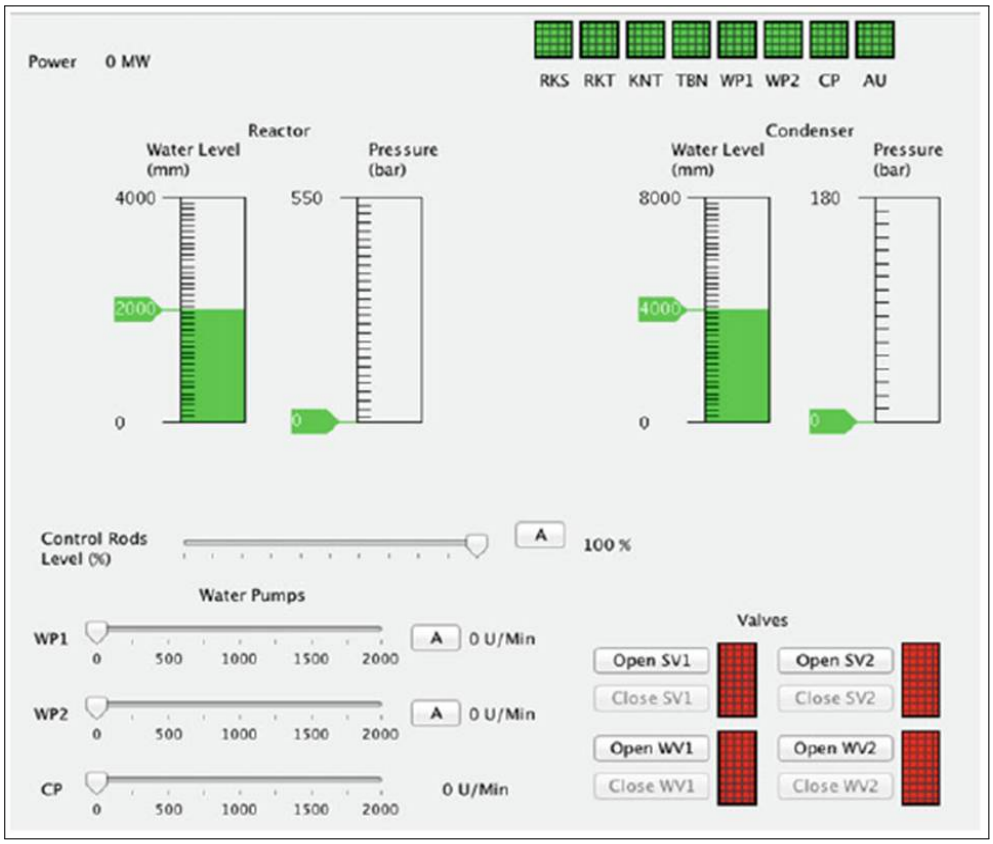

Figure 7.2: Example Interface of Nuclear Power Plant Control [7]

1. PowerDisplay: It displays the power being generated by the reactor.

2. ReactorWaterLevelDisplay: It displays the current level of water in a reactor vessel which can range from 0 upto 4000 mm.

3. ReactorPressureDisplay: It displays the current pressure in the reactor vessel which can range from 0 upto 550 bar.

4. CondenserWaterLevelDisplay: It displays the current level of water in a condenser vessel which can range from 0 upto 8000 mm.

5. CondenserPressureDisplay: It displays the current pressure in the condenser which can range from 0 upto 180 bar.

6. ControlRodLevel: This widget allows the operator to increase or decrease the level of control rods in the reactor vessel.

7. WP1Control: This widget allows the operator to adjust the speed of the water pump 1, thus influencing the amount of water sent through the pipes.

8. WP2Control: This widget allows the operator to adjust the speed of the water pump 2, thus influencing the amount of water sent through the pipes.

9. CPControl: This widget allows the operator to adjust the speed of the condenser pump.

10. SV1Open: This widget allows the operator to open the steam valve 1.

11. SV1Close: This widget allows the operator to close the steam valve 1.

12. SV1Status: It displays the status (open or closed) of the steam valve 1.

13. SV2Open: This widget allows the operator to open the steam valve 2.

14. SV2Close: This widget allows the operator to close the steam valve 2.

15. SV2Status: It displays the status (open or closed) of the steam valve 2.

16. WV1Open: This widget allows the operator to open the water valve 1.

17. WV1Close: This widget allows the operator to close the water valve 1.

18. WV1Status: It displays the status (open or closed) of the water valve 1.

19. WV2Open: This widget allows the operator to open the water valve 2.

20. WV2Close: This widget allows the operator to close the water valve 2.

21. WV2Status: It displays the status (open or closed) of the water valve 2.

22. RStatusDisplay : It displays the status of the reactor (stable, scram, accident or abnormal).

There are a total of four states, one of which the reactor can be in at any given point which are: *stable, accident, abnormal* and *scram*. The interface

shown in Figure 7.2 provides full control to the operator when it is in *stable* mode. However, when the system moves into the *accident* or *abnormal* modes, then the operator has limited control, and no control at all when the system is in *scram* state. In this thesis we model only two states: *stable* and *scram*. In *stable* state the operator has full control and in *scram* the operator has no control at all. Note that this is the simplified model of a reactor control system as fully describing the actual working of the system is out of scope of this thesis.

7.3 Modelling of the user interface and interaction of the reactor control system of a nuclear power plant using Coloured Petri Nets

As mentioned in the previous section, we model two states in this thesis, so there would be two pages named: *stable* and *scram* of the CPN model which are interconnected via fusion places.

We start with the modelling of the user interface of the system in Coloured Petri Nets. Figure 7.2 shows the interface of the reactor control of the nuclear power plant in the stable state. Table 7.1 shows the declaration of all the widgets available on the interface, categories associated with the widgets and behaviours associates with the widgets.

- *WidgetName* is an enumeration type and represents the names of the widgets of the interface. There are twenty two widgets in the interface: *PowerDisplay, ReactorWaterLevelDisplay, ReactorPressureDisplay, CondenserWaterLevelDisplay, CondenserPressureDisplay, ControlRodLevel, WP1 Control, WP2Control, CPControl, SV1Open, SV1Close, SV1Status, SV2 Open, SV2Close, SV2Status, WV1Open, WV1Close, WV1Status, WV2 Open, WV2Close, RStatusDisplay* and *WV2Status*.

- *Category* is an enumeration type that describes the categories of the widgets. All the widgets fall in one of the three categories: *SystemControl*,

ActionControl and *Responder*. If a widget is displaying the status, then it falls under the *Responder* category. If the operator has control over a widget, then it falls under the *ActionControl* category and if the system is automatically controlling the widgets and the operator has no control, then it falls under the *SystemControl* category.

- *Behaviour* is an enumeration type that represents a set of all behaviours.
- *Behaviours* is of type list, so a widget can have more than one behaviour. *Behaviours* is a list of behaviours where the names of the behaviours is taken from the *Behaviour* colour set.
- *widgetdescr* is of type product. It represents a triple (*WidgetName*, *Category*, [*Behaviours*]).
- *pmodel* is a list colour set that can contains a set of triples.

```
colset WidgetName =   with PowerDisplay | ReactorWaterLevelDisplay |
                      ReactorPressureDisplay |
                      CondenserWaterLevelDisplay|
                      CondenserPressureDisplay | ControlRodLevel |
                      WP1Control | WP2Control | CPControl | SV1Open |
                      SV1Close | SV1Status | SV2Open | SV2Close |
                      SV2Status | WV1Open | WV1Close | WV1Status |
                      WV2Open | WV2Close | WV2Status;
colset Category =     with SystemControl | ActionControl |
                      Responder;
colset Behaviour =    with S_OutputPower |
                      S_OutputReactorWaterLevel |
                      S_OutputReactorPressure |
                      S_OutputCondenserWaterLevel |
                      S_OutputCondenserPressure | S_RaiseControlRods |
                      S_LowerControlRods |
                      S_IncreaseWP1Speed |
                      S_DecreaseWP1Speed |
                      S_IncreaseWP2Speed |
                      S_DecreaseWP2Speed | S_IncreaseCPspeed |
                      S_DecreaseCPspeed | S_OpenSV1 | S_CloseSV1 |
                      S_OutputSV1Status | S_OpenSV2 | S_CloseSV2 |
                      S_OutputSV2Status | S_OpenWV1 | S_CloseWV1 |
                      S_OutputWV1Status | S_OpenWV2 | S_CloseWV2 |
                      S_OutputWV2Status | I_scram | S_RStatusDisplay;
colset Behaviours =   list Behaviour;
colset widgetdescr =  product WidgetName * Category * Behaviours;
colset pmodel =       list widgetdescr;
```

Table 7.1: Declarations of widget names, categories and behaviours for the Reactor Control System of a Nuclear Power Plant

Now we look at what widgets are available to the operator in which state and in what category that widgets fall in and what behaviours are associated with the widgets in each state. As we are modelling two states of the reactor control system, we have two constants declared in CPN: *stable* and *sram* as shown in Table 7.2.

```
val stable =  [(PowerDisplay, Responder, [S_OutputPower]),
               ( ReactorWaterLevelDisplay, Responder,
               [S_OutputReactorWaterLevel]),
```

```
                ( ReactorPressureDisplay, Responder,
                [S_OutputReactorPressure]),
                ( CondenserWaterLevelDisplay, Responder,
                [S_OutputCondenserWaterLevel]),
                ( CondenserPressureDisplay, Responder,
                [S_OutputCondenserPressure]),
                ( ControlRodLevel, ActionControl,
                [S_RaiseControlRods, S_LowerControlRods]),
                ( WP1Control, ActionControl,
                [S_IncreaseWP1Speed, S_DecreaseWP1Speed]),
                ( WP2Control, ActionControl,
                [S_IncreaseWP2Speed, S_DecreaseWP2Speed]),
                ( CPControl, ActionControl,
                [S_IncreaseCPspeed, S_DecreaseCPspeed]),
                ( SV1Open, ActionControl,[S_OpenSV1]),
                ( SV1Close, ActionControl,[S_CloseSV1]),
                ( SV1Status, Responder,[S_OutputSV1Status]),
                ( SV2Open, ActionControl,[S_OpenSV2]),
                ( SV2Close, ActionControl,[S_CloseSV2]),
                ( SV2Status, Responder,[S_OutputSV2Status]),
                ( WV1Open, ActionControl,[S_OpenWV1]),
                ( WV1Close, ActionControl,[S_CloseWV1]),
                ( WV1Status, Responder,[S_OutputWV1Status]),
                ( WV2Open, ActionControl,[S_OpenWV2]),
                ( WV2Close, ActionControl,[S_CloseWV2]),
                ( WV2Status, Responder, [S_OutputWV2Status]),
                ( RStatusDisplay, SystemControl,
                [S_RStatusDisplay, I_scram])];

val scram =     [(PowerDisplay, Responder, [S_OutputPower]),
                ( ReactorWaterLevelDisplay, Responder,
```

```
[S_OutputReactorWaterLevel]),
( ReactorPressureDisplay, Responder,
[S_OutputReactorPressure]),
( CondenserWaterLevelDisplay, Responder,
[S_OutputCondenserWaterLevel]),
( CondenserPressureDisplay, Responder,
[S_OutputCondenserPressure]),
( ControlRodLevel, Responder,
[S_RaiseControlRods, S_LowerControlRods]),
( WP1Control, Responder,
[S_IncreaseWP1Speed, S_DecreaseWP1Speed]),
( WP2Control, Responder,
[S_IncreaseWP2Speed, S_DecreaseWP2Speed]),
( CPControl, Responder,
[S_IncreaseCPspeed, S_DecreaseCPspeed]),
( SV1Open, Responder,[S_OpenSV1]),
( SV1Close, Responder,[S_CloseSV1]),
( SV1Status, Responder,[S_OutputSV1Status]),
( SV2Open, Responder,[S_OpenSV2]),
( SV2Close, Responder,[S_CloseSV2]),
( SV2Status, Responder,[S_OutputSV2Status]),
( WV1Open, Responder,[S_OpenWV1]),
( WV1Close, Responder,[S_CloseWV1]),
( WV1Status, Responder,[S_OutputWV1Status]),
( WV2Open, Responder,[S_OpenWV2]),
( WV2Close, Responder,[S_CloseWV2]),
( WV2Status, Responder, [S_OutputWV2Status]),
( RStatusDisplay, SystemControl, [S_RStatusDisplay])];
```

Table 7.2: Constants describing widgets, categories and associated behaviours in each state of the reactor control system of a nuclear power plant

The constant *stable* is comprised of a list of tuples containing name of the widget, its category and associated behaviours in the *stable* state. In the *stable* state, the operator has full control of the interface. That is why the widgets like *ControlRodLevel, WP1Control, WP2Control, CPControl, etc.* falls under *ActionControl* category.

The constant *scram* is comprised of list of tuples containing names of the widget, its category and associated behaviours in the *scram* state. The control of a nuclear power plant involves a high degree of automation. In the *scram* state the entire control is given to the system, i.e., the operator has no control of the interface in the *scram* state. Now the widgets which were in the *ActionControl* category fall in the *Responder* category because they show the operator what the automated system is doing rather than enabling the operator to make changes themself. The widgets are still associated with the same behaviours, but instead of generating these behaviours (as action controls do), they now respond to them. In addition, an automated behaviour is included, described as a 'SystemControl', which leads to the interface behaviours of changing the display.

The CPN model of user interface and interaction is made by creating a single page for each state and creating the transitions between states based on the relevant I-behaviours. As we are modelling two states of the system, the CPN model of a user interface and interaction of the system has two pages which are interconnected by fusion places. The names of the fusion places and their tags are the same as the name of the states. Each page represents the individual state that shows the navigational possibilities of a user-interface for the system.

Let us take a closer look at both the pages:

1. **Stable Page**:

   Figure 7.3 shows the *stable* state of the reactor control system. The two places: *stable* and *scram* represent the states of the system. Each place has an associated colour set which determines the type of data the place may contain. In our model each place belongs to just one colour set,

pmodel which is a list of colour set widgetdescr (a product colour set comprised of WidgetName, Category and Behaviours).

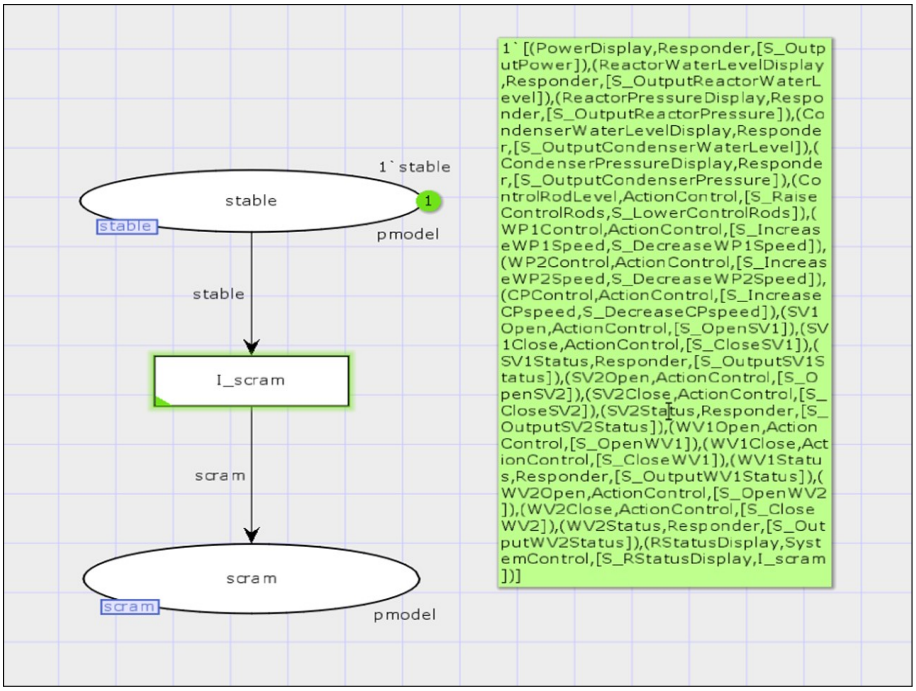

Figure 7.3: Stable page of the reactor control system

In Figure 7.3, the place *stable* has 1‘*stable* as its initial marking which means that the place has one token with the value *stable*. In a model, a marking of place is shown in a green box. For example, in Figure 7.3, the current marking on the place *stable* (green box) shows the value of the constant *stable* as written in Table 7.2 which gives information about the available widgets in that state and what behaviours are attached to those widgets in that state. The marking shows that there is just one I-behaviour: I_scram, so there is only transition namely: I_scram on this page. This means that from the *stable* state, the system can go into the *scram* state via transition I_scram if something goes wrong, for example, if the reactor pressure goes to more than 70 bar or the temperature goes above 286 degrees Celsius. As we are modelling the user interface and interaction of the reactor control system in CPN which gives meaning to the I-behaviours, so the CPN model will contain the transitions which represent the I-behaviours only.

In our model, each arc expression is a constant which is the name of the constants written in Table 7.2. The expressions are written next to arcs and determine which tokens are removed or added to the places. An arc expression is evaluated by binding data values to variables. A transition can occur if it is enabled. For a transition to be enabled in the current marking, it must be possible to bind data values to the variables appearing on the surrounding arc expressions. In the initial marking, the transition *I_scram* is enabled because there are enough tokens of the right form on the place. As the transition *I_scram* is enabled, it can occur. When the transition *I_scram* fires, then a token 1‘*stable* is removed from the input place *stable* and a token 1‘*stable* is added to the output place *scram* as shown in Figure 7.4.

2. **Scram Page**:

   Figure 7.4 shows the structure of the *scram* page of the user interface and interaction of the reactor control system.

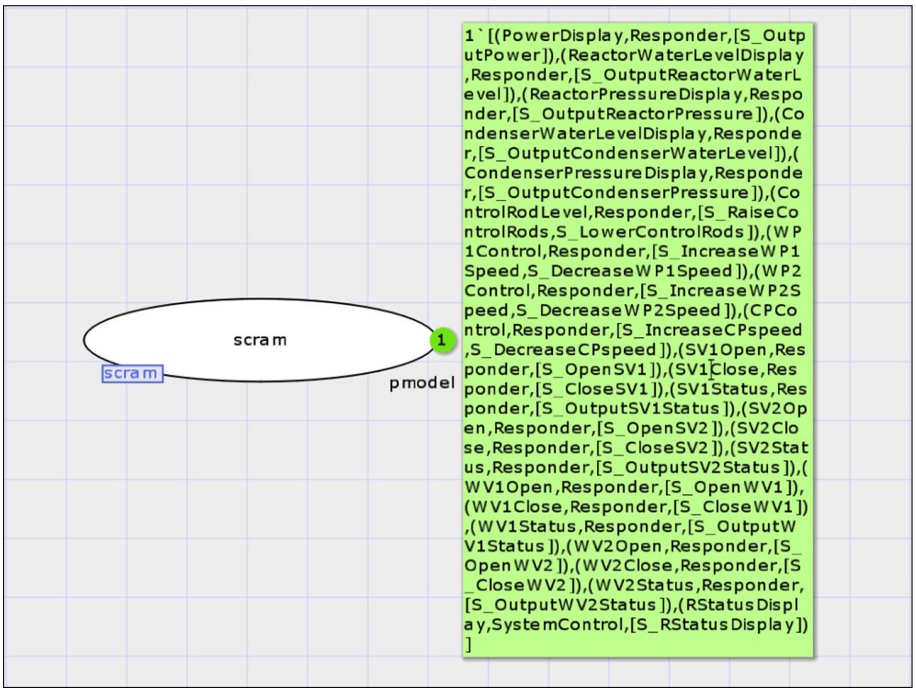

Figure 7.4: Scram page of the reactor control system

The marking shows that there is no I-behaviour, so there is no transition

on this page. This means that we can never get to a stable state and the system will automatically shut down.

7.4 Modelling Functionality of the Reactor Control System of a Nuclear Power Plant

Having modelled a user interface and interaction in Coloured Petri Nets, we now move to the final part, modelling the functionality. We will first have a look at the observations that need to be monitored while the reactor control system is running. It is important to note that these observations, values and the operations are based on simplified assumptions taken from [129].

There are a total of nineteen observations that are monitored during the working of the reactor control system which are:

1. Steam valve 1 status (SV1): This observation can have two values: *Open* or *Closed*. If the steam valve 1 is open then the value of this observation is *Open* else *Closed*.

2. Steam valve 1 flow rate (SV1FR): This observation is used to record the flow rate of the steam valve 1 which can be an integer number between 0 up to 2000.

3. Steam valve 2 status (SV2): This observation can have two values: *Open* or *Closed*. If the steam valve 1 is open then the value of this observation is *Open* else *Closed*.

4. Steam valve 2 flow rate (SV2FR): This observation is used to record the flow rate of the steam valve 2 which can be an integer number between 0 up to 2000.

5. Water valve 1 status (WV1): This observation can have two values: *Open* or *Closed*. If the water valve 1 is open then the value of this observation is *Open* else *Closed*.

6. Water valve 1 flow rate (WV1FR): This observation is used to record the flow rate of the water valve 1 which can be an integer number between 0 up to 2000.

7. Water valve 2 status (WV2): This observation can have two values: *Open* or *Closed*. If the water valve 2 is open then the value of this observation is *Open* else *Closed*.

8. Water valve 2 flow rate (WV2FR): This observation is used to record the flow rate of the water valve 2 which can be an integer number between 0 up to 2000.

9. Water pump 1 speed (WP1): This observation is used to store the current speed of the water pump 1 which can be an integer number between 0 up to 2000.

10. Water pump 2 speed (WP2): This observation is used to store the current speed of the water pump 2 which can be an integer number between 0 up to 2000.

11. Condenser pump speed (CP): This observation is used to store the current speed of the condenser pump which can be an integer number between 0 up to 2000.

12. Reactor water level status (RWLevel): This observation is used to store the current level of water in the reactor vessel which can be an integer number between 0 up to 4000

13. Reactor pressure (RPressure): This observation is used to store the current pressure in the reactor vessel which can be an integer number between 0 up to 550.

14. Condenser water level status (CWLevel): This observation is used to store the current level of water in the condenser vessel which can be an integer number between 0 up to 8000.

15. Condenser pressure (CPressure): This observation is used to store the current pressure in the condenser vessel which can be an integer number between 0 up to 180.

16. Rod position (RodPos): This observation is used to store the current position of the rods which can be an integer number from 0 up to 100.

17. Boiling water temperature (BWTEMP): This observation is used to save the current temperature of a boiling water in the reactor vessel which can be an integer number between 0 up to 1000.

18. Power (POWEROUT): This observation is used to record the value of the current power being generated which can be an integer number between 0 up to 1000.

19. Reactor status (RSTATUS): This observation gives the current status of the reactor control system which can be *STABLE* or *SCRAM*.

To write these observations in CPN-ML, we first need to declare the colour sets or types that would be associated with each of the above mentioned observations. Table 7.3 shows all the colour sets and variables of the reactor control system.

- Colour set *VALVESTATUS* is declared as the enumerated colour set that can have exactly two values *Open* or *Closed*.

- Colour set *FLOWRATE* is declared as an integer colour set with range 0..2000.

- Colour set *RWL* is declared as an integer colour set with range 0..4000.

- Colour set *RPR* is declared as an integer colour set with range 0..550.

- Colour set *CWL* is declared as an integer colour set with range 0..8000.

- Colour set *CPR* is declared as an integer colour set with range 0..180.

- Colour set *RODPOS* is declared as an integer colour set with range 0..100.

- Colour set *SPEED* is declared as an integer colour set with range 0..2000.
- Colour set *DEGREE* is declared as an integer colour set with range 0..1000.
- Colour set *RSTATUS* is declared as the enumerated colour set that can have exactly two values *STABLE* or *SCRAM*.
- Colour set *POWER* is declared as an integer colour set with range 0..1000.
- Colour set *NPP* is a record colour set with a record of all the above mentioned nineteen observations with their corresponding types. In this declaration $SV1$ stands for $SteamValve1$, $SV1FR$ is for $SteamValve1FlowRate$, $SV2$ is for $SteamValve2$, $SV2FR$ is for $SteamValve2FlowRate$, $WV1$ stands for $WaterValve1$, $WV1FR$ is for $WaterValve1FlowRate$, $WV2$ is for $WaterValve2$, $WV2FR$ is for $WaterValve2FlowRate$, $WP1$ is for $WaterPump1$, $WP2$ is for $WaterPump2$, CP is for $CondenserPump$, $RWLevel$ is for $ReactorWaterLevel$, $RPressure$ is for $ReactorPressure$, $CWLevel$ is for $CondenserWaterLevel$, $CPressure$ is for $CondenserPressure$, $RodPos$ is for $RodPosition$, $BWTEMP$ is for $BoilingWaterTemperature$, $POWEROUT$ is for $PowerOutput$, and $RSTATUS$ is for $ReactorStatus$.
- As operations would be expressed as arc inscriptions so we need to declare variables which could be bound to different values of their respective colour sets during simulation. There are nineteen variables *SV1, SV1FR, SV2, SV2FR, WV1, WV1FR, WV2, WV2FR, WP1, WP2, CP, RWLevel, RPressure, CWLevel, CPressure, RodPos, BWTEMP, POWEROUT* and *RSTATUS*.

```
colset VALVESTATUS =  with Open |Closed;
colset FLOWRATE =     int with 0..2000;
colset RWL=           int with 0..4000;
colset RPR =          int with 0..550;
```

```
colset CWL =          int with 0..8000;
colset CPR =          int with 0..180;
colset RODPOS =       int with 0..100;
colset SPEED =        int with 0..2000;
colset DEGREE =       int with 0..1000;
colset RSTATUS =      with STABLE | SCRAM;
colset POWER =        int with 0..1000;
colset NPP =          record SV1:VALVESTATUS *
                      SV1FR:FLOWRATE * SV2:VALVESTATUS *
                      SV2FR:FLOWRATE * WV1:VALVESTATUS *
                      WVIFR:FLOWRATE * WV2:VALVESTATUS *
                      WV2FR:FLOWRATE * WP1:SPEED *
                      WP2:SPEED * CP:SPEED *
                      RWLevel:RWL * RPressure:RPR *
                      CWLevel:CWL * CPressure:CPR *
                      RodPos:RODPOS * BWTEMP:DEGREE *
                      POWEROUT:POWER * RSTATUS:RSTATUS;
var SV1:              VALVESTATUS;
var SV1FR:            FLOWRATE;
var SV2:              VALVESTATUS;
var SV2FR:            FLOWRATE;
var WV1:              VALVESTATUS;
var WV1FR:            FLOWRATE;
var WV2:              VALVESTATUS;
var WV2FR:            FLOWRATE;
var WP1:              SPEED;
var WP2:              SPEED;
var CP:               SPEED;
var RWLevel:          RWL;
var RPressure:        RPR;
var CWLevel:          CWL;
```

var CPressure:	CPR;
var RodPos:	RODPOS;
var BWTEMP:	DEGREE;
var POWEROUT:	POWER;
var RSTATUS:	RSTATUS;

Table 7.3: Colour sets and variables for Reactor Control System

A fusion place named *NPP* of type colour set *NPP* is added to every page of the model that contains record of all the observations. The fusion place *NPP* with initial marking is shown in Figure 7.5.

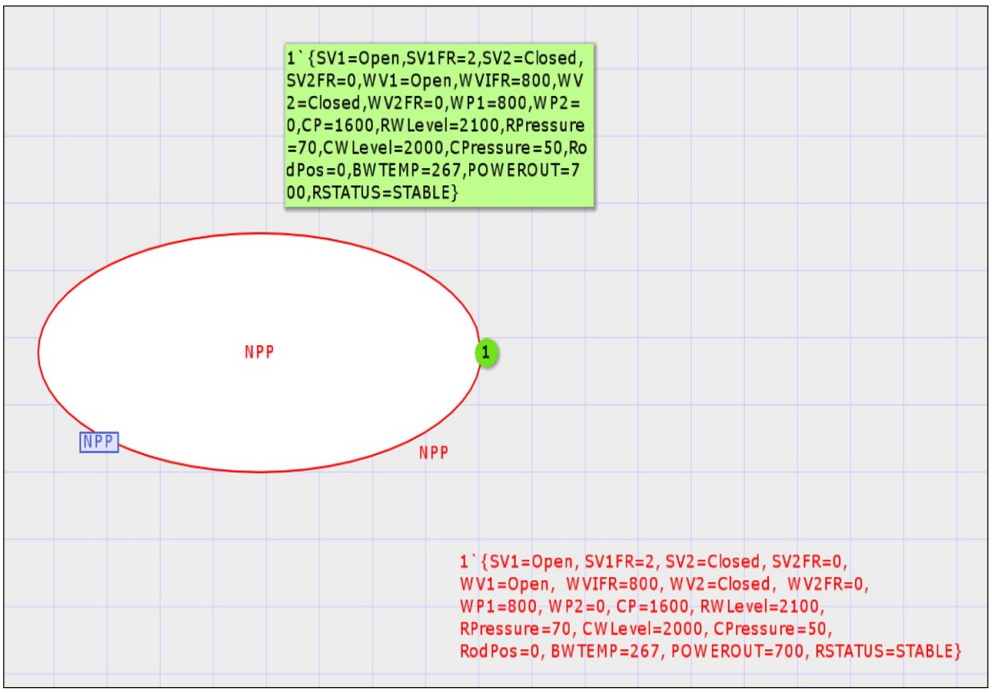

Figure 7.5: NPP fusion place of Reactor Control System

This place is added to every page of the model. An initial marking of a fusion place *NPP* has all the initial values of the observations. We are assuming that the reactor control is in the *stable* state. An initial marking *1‘{SV1=Open, SV1FR=2, SV2=Closed, SV2FR=0, WV1=Open, WVIFR=800, WV2=Closed, WV2FR=0, WP1=800, WP2=0, CP=1600, RWLevel =2100, RPressure=70, CWLevel=2000, CPressure=50, RodPos=0, BWTEMP*

=267, POWEROUT=700, RSTATUS=STABLE} shown in Figure 7.5 shows the initial values of the observations.

We will now extend the model given in the previous section by adding S-behaviours to it. As mentioned earlier in this work we are abstracting away from (i.e. hiding) the S-behaviours associated with the *Responder* category to keep the model simple and reduce the state space. Now we will see both the pages/state in detail and explain the underlying system functionality in each state. Note that as the model is too large for a single page, we present it in a number of images. The upcoming images are the snippets from one page. The full model can be seen in [1].

1. **Stable Page**

   We will first explain the underlying system functionality in the *stable* state. In the *stable* state, there are a total of sixteen underlying system operations that we are going to model so the *stable* page will have sixteen S-behaviour transitions. The operations and its modelling using Coloured Petri Nets is explained below:

   (a) **S_RaiseControlRods:** This operation allows the operator to raise the level of control rods in the reactor vessel. When the operator scrolls the *ControlRodLevel* widget in the forward direction each time, it raises the level of control rods in the reactor vessel and the value of the variable *RodPos* is increased by 1. As the rods enter the reactor vessel, the reaction affects the temperature of the boiling water in the vessel and increases the value of the variable *BWTEMP* by one. If the boiling water temperature goes beyond 286 degree Celsius and if the reactor pressure goes beyond 70 bars then the system goes to the *SCRAM* state, the value of the variable *RSTATUS* changes to *SCRAM* and all control is taken from the operator. Figure 7.6 shows the modelling of this operation using Coloured Petri Nets.

[1] See `https://github.com/sapnajaidka/NPP/blob/master/ReactorNPP.cpn` for complete details

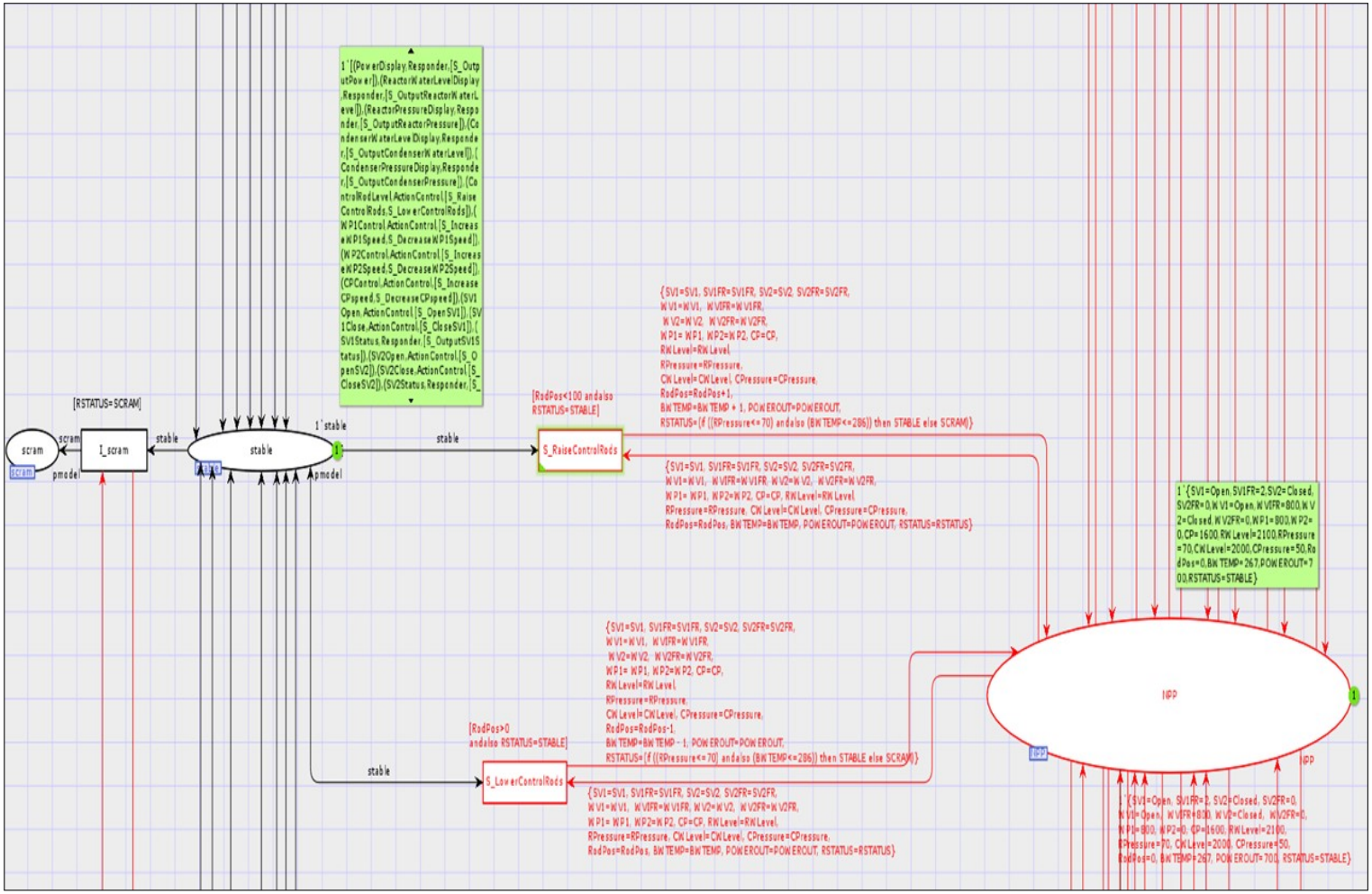

Figure 7.6: S_RaiseControlRods and S_LowerControlRods operations of Reactor Control System

The two places: *stable* and *scram* represent the states of the system. The marking on the place *stable* gives information about the available widgets and behaviours associated with those widgets. The transitions: *I_scram* represents the I-behaviour as shown in the marking. The marking on the *stable* place also shows that there are many S-behaviours which represents the underlying system functionality. To keep the size of the state space small, we are not modelling the S-behaviours which display messages or statuses on the screen. The fusion place *NPP* is added to the page which has an initial marking that shows all the initial values of the variables.

In Figure 7.6, we see a transition *S_RaiseControlRods*. Input and output arcs are added to and from the fusion place *NPP* to the *S_RaiseControlRods* transition. The expression from the fusion place *NPP* to the transition *S_RaiseControlRods* simply contains assignments which set each variable to its current value. This set of assignments "picks up" the current values of the variables ready to be used by the second arc, i.e. an input arc to the place *NPP* from the same *S_RaiseControlRods* transition. This second arc, the one to the place *NPP* from the same *S_RaiseControlRods* transition, assigns each variable to its new value as per the operation. When the transition *S_RaiseControlRods* is enabled and fired (as per Definitions 3.6.4 and 3.6.6), the new value of the variable *RodPos* will be old value of the variable *RodPos* plus 1.

The guard *[RodPos<100 andalso RSTATUS=STABLE]* on the transition *S_RaiseControlRods* means that this transition will only be enabled if the position of the control rods is less than 100 and the system is in the *STABLE* state.

(b) **S_LowerControlRods:** This operation allows the operator to lower the level of control rods in the reactor vessel. When the operator scrolls the *ControlRodLevel* widget in the reverse direction each time, it lowers the level of control rods and the value of the variable

RodPos is decreased by 1. As the rods are pushed out of the reactor vessel, the reaction affects the temperature of a boiling water in the vessel and decreases the value of the variable *BWTEMP* by one. If the boiling water temperature goes beyond 286 degree Celsius and if the reactor pressure goes beyond 70 bars then the system goes to the *SCRAM* state and the value of the variable *RSTATUS* changes to *SCRAM* and all the controls are taken from the operator. Figure 7.6 shows the modelling of this operation using Coloured Petri Nets.

In Figure 7.6, we see a transition *S_LowerControlRods*. Input and output arcs are added to and from the fusion place *NPP* to the *S_LowerControlRods* transition. The expression from the fusion place *NPP* to the transition *S_LowerControlRods* simply contains assignments which set each variable to its current value. This set of assignments "picks up" the current values of the variables ready to be used by the second arc, i.e. an input arc to the place *NPP* from the same *S_LowerControlRods* transition. This second arc, the one to the place *NPP* from the same *S_LowerControlRods* transition, assigns each variable to its new value as per the operation.

The guard *[RodPos>0 andalso RSTATUS=STABLE]* on the transition *S_RaiseControlRods* means that this transition will only be enabled if the position of the control rods is greater than 0 and the system is in the *STABLE* state.

(c) **S_IncreaseWP1Speed:** This operation allows the operator to increase the speed of the water pump 1 by scrolling the *WP*1*Control* widget in the forward direction and the value of the variable *WP*1 is increased by 1. When the speed of the water pump 1 increases, it increases the flow rate of the water valve 1 which is connected to the pump. In Figure 7.7 we can see a transition *S_IncreaseWP*1*Speed* which represents this operation.

There are two arcs (in red) needed to model this (going to and

from the place *NPP*). The arc from *NPP* to *S_IncreaseWP1Speed* simply contains assignments which set each variable to its current value. This set of assignments "picks up" the current values of the variables ready to be used by the second arc. This second arc, the one from *S_IncreaseWP1Speed* to *NPP* as shown in Table 7.4, assigns each variable to its *new* value. When this transition is fired, then the the value of the variable $WP1$ is increased by 1. When the speed of the water pump 1 increases, it increases the flow rate of the water valve 1 which is connected to the pump. If the value of $WV1$ is *Open*, i.e., if the water valve 1 is open, then a water valve 1 flow rate ($WV1FR$) will be the current speed of water pump 1 plus one.

Arc	Arc Expression
S_IncreaseWP1Speed to NPP	({SV1=SV1, SV1FR=SV1FR, SV2=SV2, SV2FR=SV2FR, WV1=WV1, WVIFR=(if (WV1=Open) then (WP1+1) else 0), WV2=WV2, WV2FR=WV2FR, WP1= WP1+1, WP2=WP2, CP=CP, RWLevel=RWLevel, RPressure=RPressure, CWLevel=CWLevel, CPressure=CPressure, RodPos=RodPos, BWTEMP=BWTEMP, POWEROUT=POWEROUT, RSTATUS=(if ((RPressure<=70) andalso (BWTEMP<=286)) then STABLE else SCRAM)})

Table 7.4: Arc Expression S_IncreaseWP1Speed to NPP

The guard *[WP1<2000 andalso WP1>0 andalso RSTATUS=STABLE]* on the transition *S_IncreaseWP1Speed* means that this transition will only be enabled if the speed of the water pump 1 is less than 2000 and the system is in the *STABLE* state.

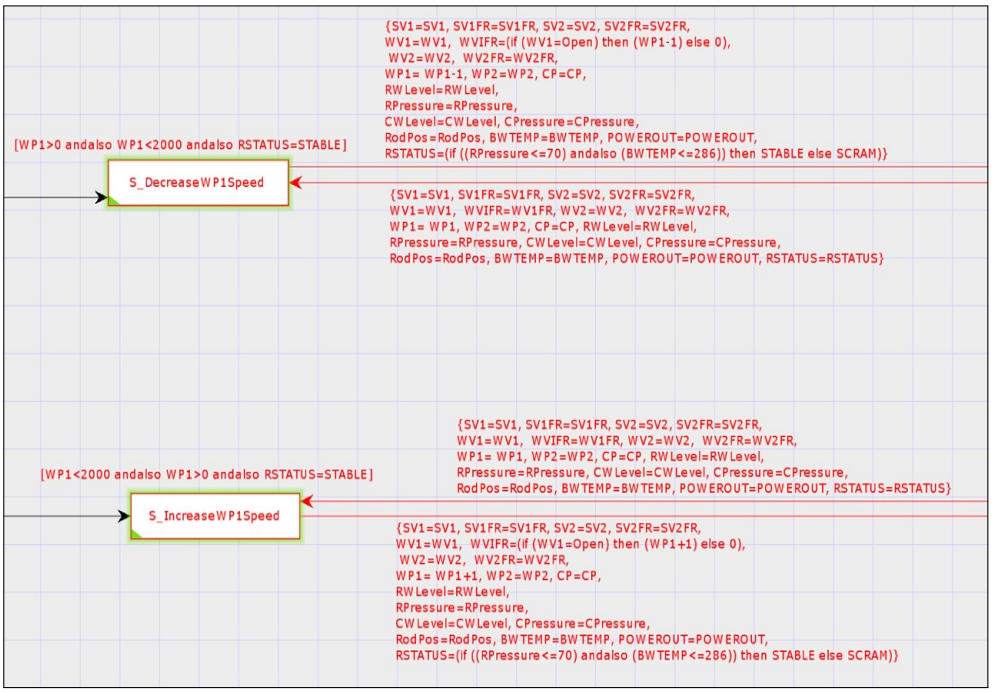

Figure 7.7: S_IncreaseWP1Speed and S_DecreaseWP1Speed operations of Reactor Control System

(d) **S_DecreaseWP1Speed:** This operation allows the operator to decrease the speed of the water pump 1 by scrolling the $WP1Control$ widget in the backward direction and the value of the variable $WP1$ is decreased by 1. When the speed of the water pump 1 decreases, it decreases the flow rate of the water valve 1 which is connected to the pump. In Figure 7.7 we can see a transition $S\_DecreaseWP1Speed$ which represents this operation.

Similar to other operations being modelled, we again need two arcs (in red) to model this (going to and from the place NPP). The arc from NPP to $S\_DecreaseWP1Speed$ simply contains assignments which set each variable to its current value. This set of assignments "picks up" the current values of the variables ready to be used by the second arc. This second arc, the one from $S\_DecreaseWP1Speed$ to NPP as shown in Table 7.5, assigns each variable to its *new* value. When this transition is fired, then the the value of the variable $WP1$ is decreased by 1. When the speed of the water pump 1 decreases,

it decreases the flow rate of the water valve 1 which is connected to the pump. If the value of the variable $WV1$ is *Open*, i.e., if the water valve 1 is open, then a water valve 1 flow rate ($WV1FR$) will be the current speed of water pump 1 minus one.

Arc	Arc Expression
S_DecreaseWP1Speed to NPP	({SV1=SV1, SV1FR=SV1FR, SV2=SV2, SV2FR=SV2FR, WV1=WV1, WVIFR=(if (WV1=Open) then (WP1-1) else 0), WV2=WV2, WV2FR=WV2FR, WP1= WP1-1, WP2=WP2, CP=CP, RWLevel=RWLevel, RPressure=RPressure, CWLevel=CWLevel, CPressure=CPressure, RodPos=RodPos, BWTEMP=BWTEMP, POWEROUT=POWEROUT, RSTATUS=(if ((RPressure<=70) andalso (BWTEMP<=286)) then STABLE else SCRAM)})

Table 7.5: Arc Expression S_DecreaseWP1Speed to NPP

The guard *[WP1>0 andalso WP1<2000 andalso RSTATUS=STABLE]* on the transition *S_DecreaseWP1Speed* means that this transition will only be enabled if the speed of the water pump 1 is greater than 0 and the system is in the *STABLE* state.

(e) **S_IncreaseWP2Speed:** Figure 7.8 shows the modelling of this operation in Coloured Petri Nets. This operation allows the operator to increase the speed of the water pump 2. It works same as *S_IncreaseWP1Speed* operation, but this operation affects the value of the variables $WP2$ and $WV2FR$.

When this transition is fired, then the the value of the variable $WP2$ is increased by 1. When the speed of the water pump 2 increases, it increases the flow rate of the water valve 2 which is connected to

the pump. If the value of $WV2$ is *Open*, i.e., if the water valve 2 is open, then a water valve 2 flow rate ($WV1FR$) will be the current speed of water pump 2 plus one.

The guard *[WP2<2000 andalso WP2>0 andalso RSTATUS=STABLE]* on the transition *S_IncreaseWP2Speed* means that this transition will only be enabled if the speed of the water pump 2 is less than 2000 and the system is in the *STABLE* state.

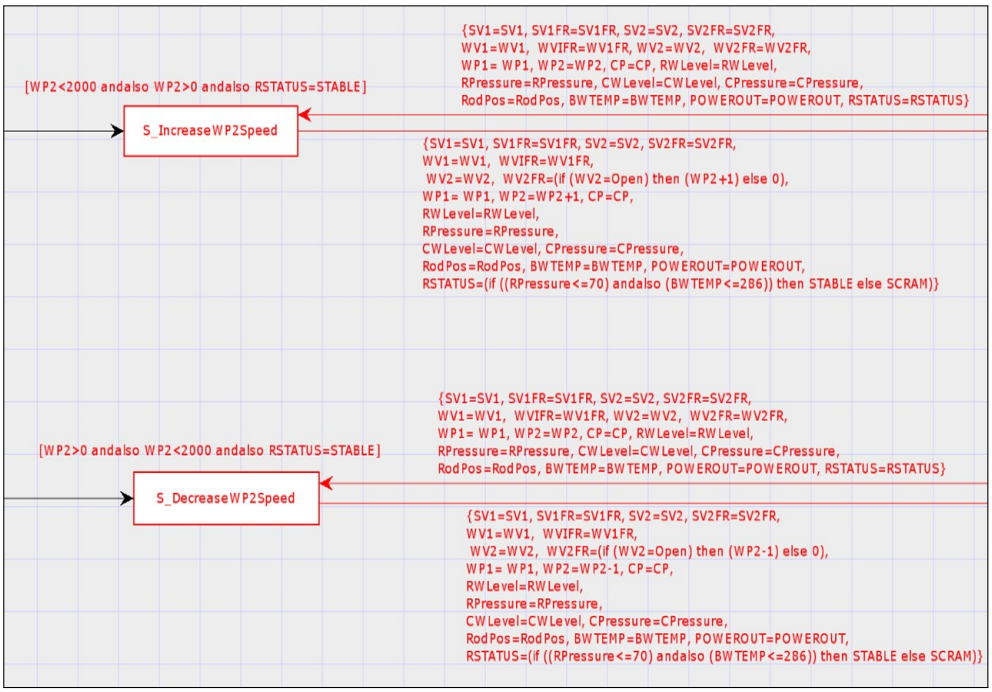

Figure 7.8: S_IncreaseWP2Speed and S_DecreaseWP2Speed operations of Reactor Control System

(f) **S_DecreaseWP2Speed:** This operation allows the operator to decrease the speed of the water pump 2 by scrolling the *WP2Control* widget in the backward direction and the value of the variable $WP2$ is decreased by 1. When the speed of the water pump 2 decreases, it decreases the flow rate of the water valve 2 which is connected to the pump. In Figure 7.8 we can see a transition *S_DecreaseWP2Speed* which represents this operation.

When this transition is fired, then the the value of the variable $WP2$ is decreased by 1. When the speed of the water pump 2 decreases,

it decreases the flow rate of the water valve 2 which is connected to the pump. If the value of the variable $WV2$ is *Open*, i.e., if the water valve 2 is open, then a water valve 2 flow rate ($WV1FR$) will be the current speed of water pump 1 minus one.

The guard *[WP2>0 andalso WP2<2000 andalso RSTATUS= STABLE]* on the transition *S_DecreaseWP2Speed* means that this transition will only be enabled if the speed of the water pump 2 is greater than 0 and the system is in the *STABLE* state.

(g) **S_IncreaseCPspeed:** This operation allows the operator to increase the speed of the condenser pump by scrolling the *CPControl* widget in the forward direction and the value of the variable *CP* is increased by 1. In Figure 7.9 we can see a transition *S_IncreaseCPspeed* which represents this operation.

There are two arcs (in red) needed to model this (going to and from the place *NPP*). The arc from *NPP* to *S_IncreaseCPspeed* simply contains assignments which set each variable to its current value. This set of assignments "picks up" the current values of the variables ready to be used by the second arc.

Arc	Arc Expression
S_IncreaseCPSpeed to NPP	({SV1=SV1, SV1FR=SV1FR, SV2=SV2, SV2FR=SV2FR, WV1=WV1, WVIFR=WV1FR, WV2=WV2, WV2FR=WV2FR, WP1= WP1, WP2=WP2, CP=CP+1, RWLevel=RWLevel, RPressure=RPressure, CWLevel=CWLevel, CPressure=CPressure, RodPos=RodPos, BWTEMP=BWTEMP, POWEROUT=POWEROUT, RSTATUS=(if ((RPressure<=70) andalso (BWTEMP<=286)) then STABLE else SCRAM)})

Table 7.6: Arc Expression S_IncreaseCPSpeed to NPP

This second arc, the one from *S_IncreaseCPspeed* to *NPP* as shown in Table 7.6, assigns each variable to its *new* value. When this transition is fired, then the the value of the variable *CP* is increased by 1.

The guard *[CP<2000 andalso RSTATUS=STABLE]* on the transition *S_IncreaseCPSpeed* means that this transition will only be enabled if the speed of the condenser is less than 2000 and the system is in the *STABLE* state.

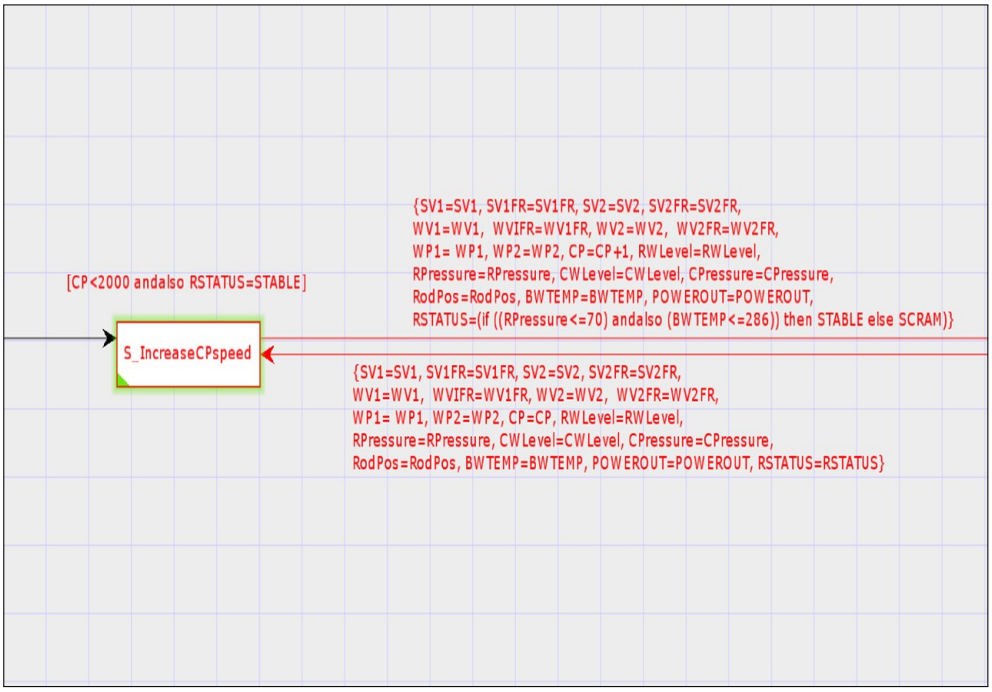

Figure 7.9: S_IncreaseCPSpeed operations of Reactor Control System

(h) **S_DecreaseCPspeed:** This operation allows the operator to decrease the speed of the condenser pump by scrolling the *CPControl* widget in the backward direction and the value of the variable *CP* is decreased by 1. In Figure 7.10 we can see a transition *S_DecreaseCPspeed* which represents this operation.

There are two arcs (in red) needed to model this (going to and from the place *NPP*). The arc from *NPP* to *S_DecreaseCPspeed* simply contains assignments which set each variable to its current value. This set of assignments "picks up" the current values of the variables

ready to be used by the second arc. This second arc, the one from *S_DecreaseCPspeed* to *NPP* as shown in Table 7.7, assigns each variable to its *new* value. When this transition is fired, then the the value of the variable *CP* is decreased by 1.

Arc	Arc Expression
S_DecreaseCPSpeed to NPP	({SV1=SV1, SV1FR=SV1FR, SV2=SV2, SV2FR=SV2FR, WV1=WV1, WVIFR=WV1FR, WV2=WV2, WV2FR=WV2FR, WP1= WP1, WP2=WP2, CP=CP-1, RWLevel=RWLevel, RPressure=RPressure, CWLevel=CWLevel, CPressure=CPressure, RodPos=RodPos, BWTEMP=BWTEMP, POWEROUT=POWEROUT, RSTATUS=(if ((RPressure<=70) andalso (BWTEMP<=286)) then STABLE else SCRAM)})

Table 7.7: Arc Expression S_DecreaseCPSpeed to NPP

The guard *[CP>0 andalso RSTATUS=STABLE]* on the transition *S_DecreaseCPSpeed* means that this transition will only be enabled if the speed of the condenser is greater than 0 and the system is in the *STABLE* state.

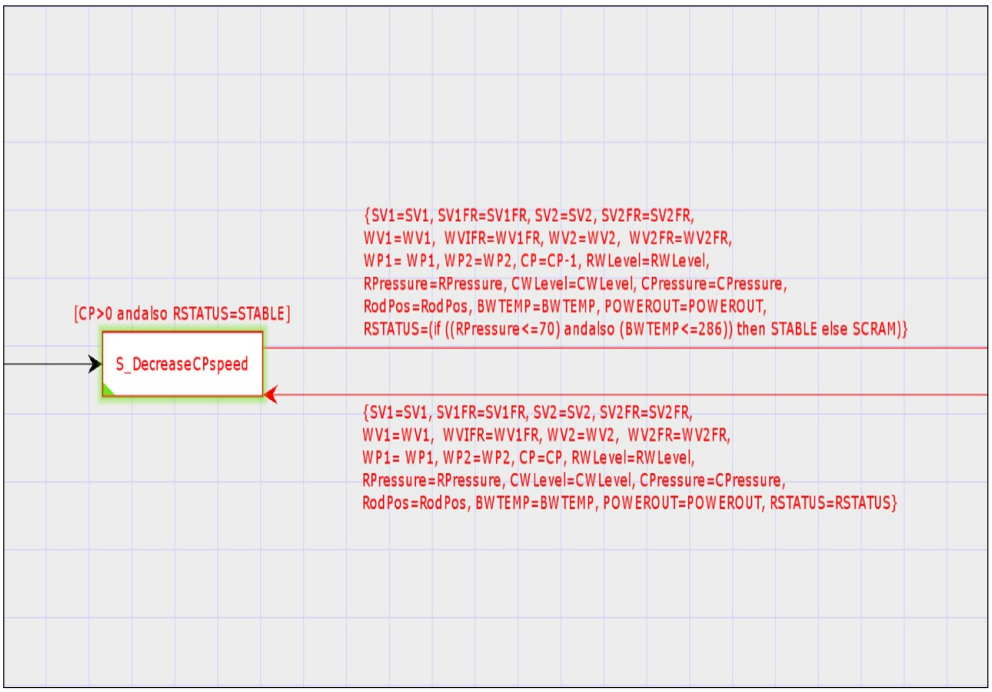

Figure 7.10: S_DecreaseCPSpeed operations of Reactor Control System

(i) **S_OpenSV1:** This operation allows the operator to open the steam valve 1 by pressing the $OpenSV1$ button on the interface. This operation changes the value of the variable $SV1$ to $Open$. The opening of a steam valve 1 allows to release the steam from the reactor vessel. This will decrease the reactor pressure and decreases the value of the variable $RPressure$ by 1. It also affects the flow rate of the steam valve 1. If Steam valve 1 is $Open$, then the value of the variable $SV1FR$ will be increased by 1. This operation is shown in Figure 7.11. The arc expression from the transition $S\_OpenSV1$ to the place NPP assigns each variable to its new value.

The guard *[SV1=Closed andalso RSTATUS=STABLE]* on the transition $S\_OpenSV1$ means that this transition will only be enabled if the steam valve was closed and the system is in the $STABLE$ state.

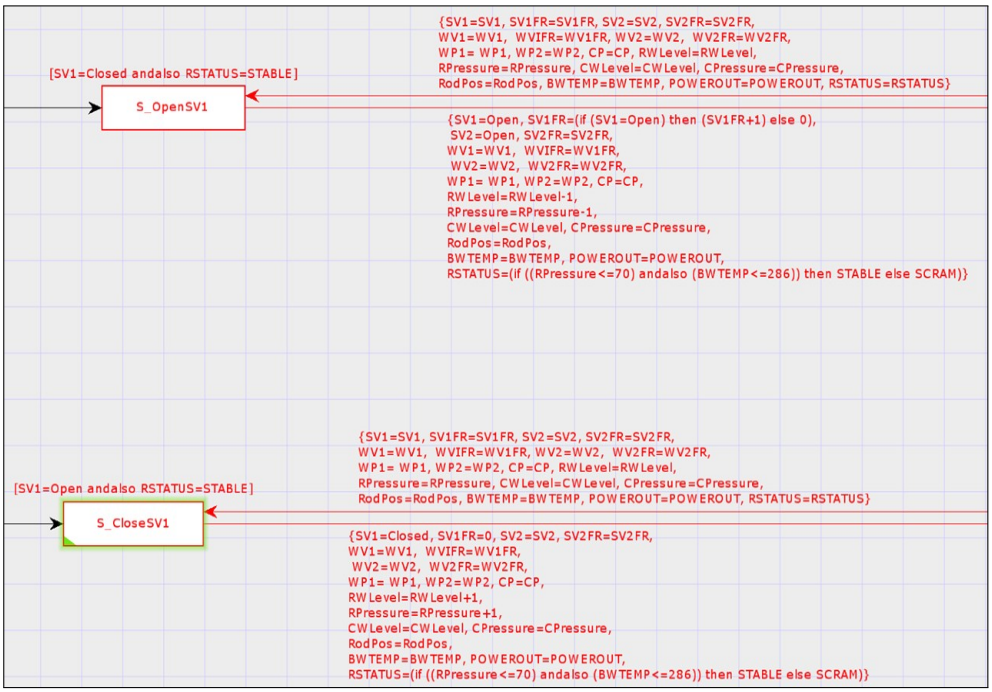

Figure 7.11: S_OpenSV1 and S_CloseSV1 operations of Reactor Control System

(j) **S_CloseSV1:** This operation allows the operator to close the steam valve 1 by pressing the $CloseSV1$ button on the interface. This operation changes the value of the variable $SV1$ to *Closed*. The closing of a steam valve 1 will increase the reactor pressure and increases the value of the variable *RPressure* by 1. It also affects the flow rate of the steam valve 1. If Steam valve 1 is *Closed*, then the value of the variable $SV1FR$ will be 0. This operation is shown in Figure 7.11. The arc expression from the transition $S\_CloseSV1$ to the place *NPP* assigns each variable to its *new* value.

The guard *[SV1=Open andalso RSTATUS=STABLE]* on the transition $S\_CloseSV1$ means that this transition will only be enabled if the steam valve was open and the system is in the *STABLE* state.

(k) **S_OpenSV2:** This operation allows the operator to open the steam valve 2 by pressing the $OpenSV2$ button on the interface. This operation changes the value of the variable $SV2$ to *Open*. This will decrease the reactor pressure and decreases the value of the vari-

able *RPressure* by 1. It also affects the flow rate of the steam valve 2. If Steam valve 2 is *Open*, then the value of the variable $SV2FR$ will be increased by 1. This operation is shown in Figure 7.12. The arc expression from the transition $S\_OpenSV2$ to the place *NPP* assigns each variable to its *new* value.

The guard *[SV2=Closed andalso RSTATUS=STABLE]* on the transition $S\_OpenSV2$ means that this transition will only be enabled if the steam valve 2 was closed and the system is in the *STABLE* state.

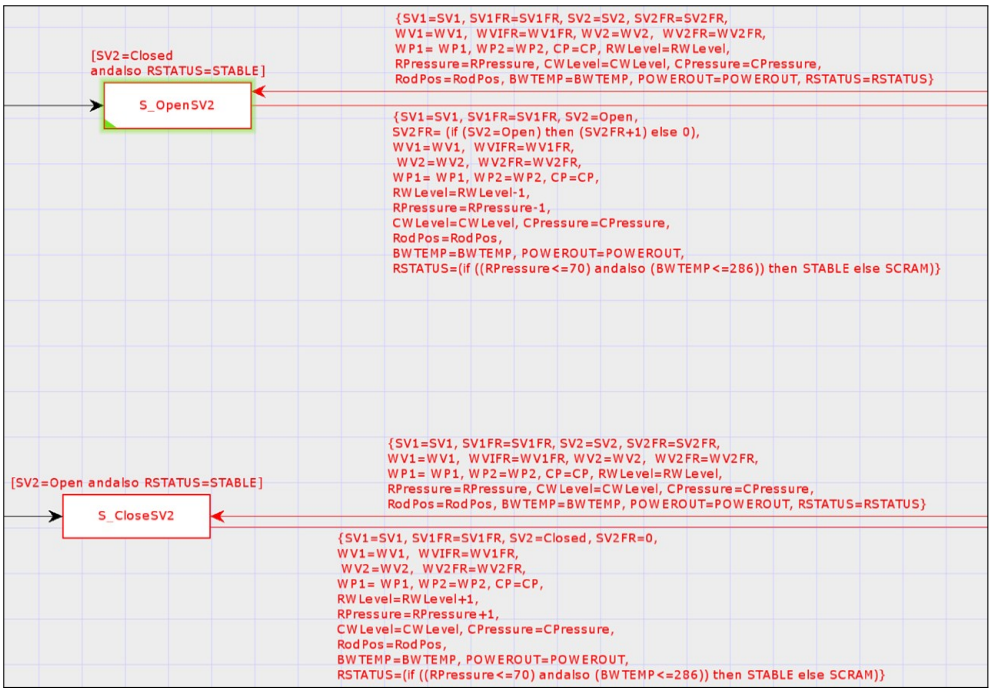

Figure 7.12: S_OpenSV2 and S_CloseSV2 operations of Reactor Control System

(l) **S_CloseSV2:** This operation allows the operator to close the steam valve 2 by pressing the $CloseSV2$ button on the interface. This operation changes the value of the variable $SV2$ to *Closed*. The closing of a steam valve 2 will increase the reactor pressure and increases the value of the variable *RPressure* by 1. It also affects the flow rate of the steam valve 2. If Steam valve 2 is *Closed*, then the value of the variable $SV2FR$ will be 0. This operation is shown

in Figure 7.12. The arc expression from the transition $S\_CloseSV2$ to the place NPP assigns each variable to its *new* value.

The guard *[SV2=Open andalso RSTATUS=STABLE]* on the transition $S\_CloseSV2$ means that this transition will only be enabled if the steam valve 2 was open and the system is in the $STABLE$ state

(m) **S_OpenWV1:** This operation allows the operator to open the water valve 1 by pressing the $OpenWV1$ button on the interface. This operation changes the value of the variable $WV1$ to *Open*. When the water valve gets open, the flow rate of water valve 1 i.e., the value of the variable $WV1FR$ will be equal to the spedd of the water pump 1, i.e., value of the variable $WP1$. As opening of the water valve 1 allows water to enter the reactor vessel, so the reactor water level will also raise. This will change the value of the variable $RWLevel$ to $(RWLevel + WP1 + WV2FR)$. This operation is shown in Figure 7.13. The arc expression from the transition $S\_OpenWV1$ to the place NPP assigns each variable to its *new* value.

The guard *[WV1=Closed andalso RSTATUS=STABLE andalso RW Level<2100]* on the transition $S\_OpenWV1$ means that this transition will only be enabled if the water valve 1 was closed, the system is in the $STABLE$ state and the reactor water level is less than 2100.

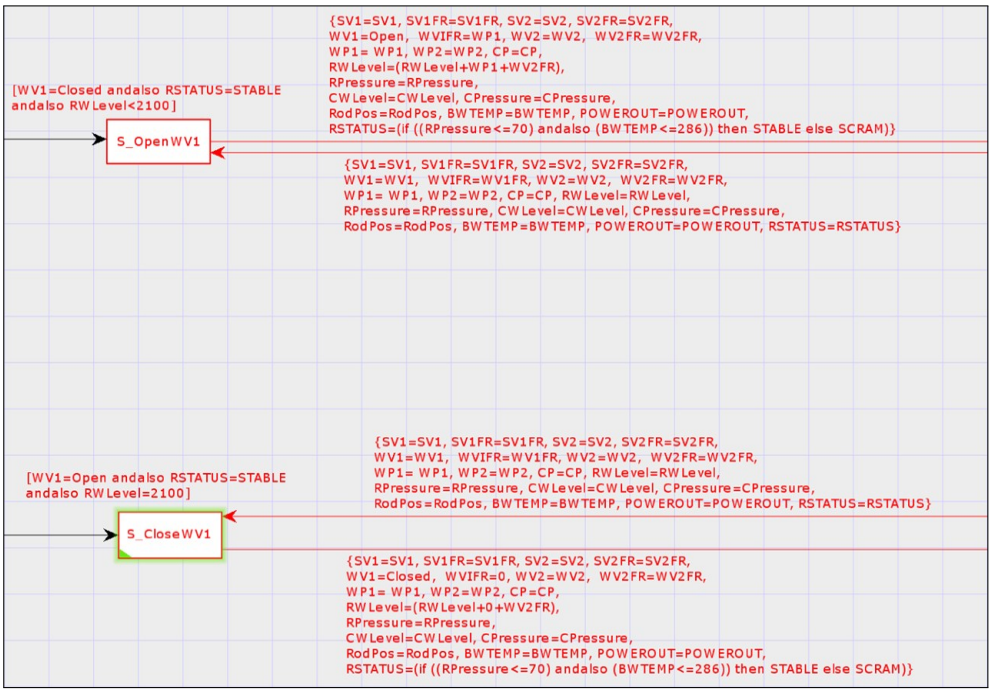

Figure 7.13: S_OpenWV1 and S_CloseWV1 operations of Reactor Control System

(n) **S_CloseWV1:** This operation allows the operator to close the water valve 1 by pressing the $CloseWV1$ button on the interface. This operation changes the value of the variable $WV1$ to *Closed*. When the water valve gets closed, the flow rate of water valve 1 i.e., the value of the variable $WV1FR$ will be equal to 0. As closing water valve 1 stops water from entering the reactor vessel, so the reactor water level will also change. This will change the value of the variable $RWLevel$ to $(RWLevel + 0 + WV2FR)$. This operation is shown in Figure 7.13. The arc expression from the transition $S\_CloseWV1$ to the place NPP assigns each variable to its *new* value.

The guard *[WV1=Open andalso RSTATUS=STABLE andalso RW Level<2100]* on the transition $S\_CloseWV1$ means that this transition will only be enabled if the water valve 1 is open, the system is in the $STABLE$ state and the reactor water level is equal to 2100.

(o) **S_OpenWV2:** This operation allows the operator to open the

water valve 2 by pressing the $OpenWV2$ button on the interface. This operation changes the value of the variable $WV2$ to $Open$. When the water valve is open, the flow rate of water valve 2 i.e., the value of the variable $WV2FR$ will be equal to the speed of the water pump 2, i.e., value of the variable $WP2$. As opening of the water valve 2 allows water to enter the reactor vessel, so the reactor water level will also raise. This will change the value of the variable $RWLevel$ to $(RWLevel + WV1FR + WP2)$. This operation is shown in Figure 7.14. The arc expression from the transition $S\_OpenWV2$ to the place NPP assigns each variable to its new value.

The guard $[WV2{=}Closed\ andalso\ RSTATUS{=}STABLE\ andalso\ RW\ Level{<}2100]$ on the transition $S\_OpenWV2$ means that this transition will only be enabled if water valve 2 was closed, the system is in the $STABLE$ state and the reactor water level is less than 2100.

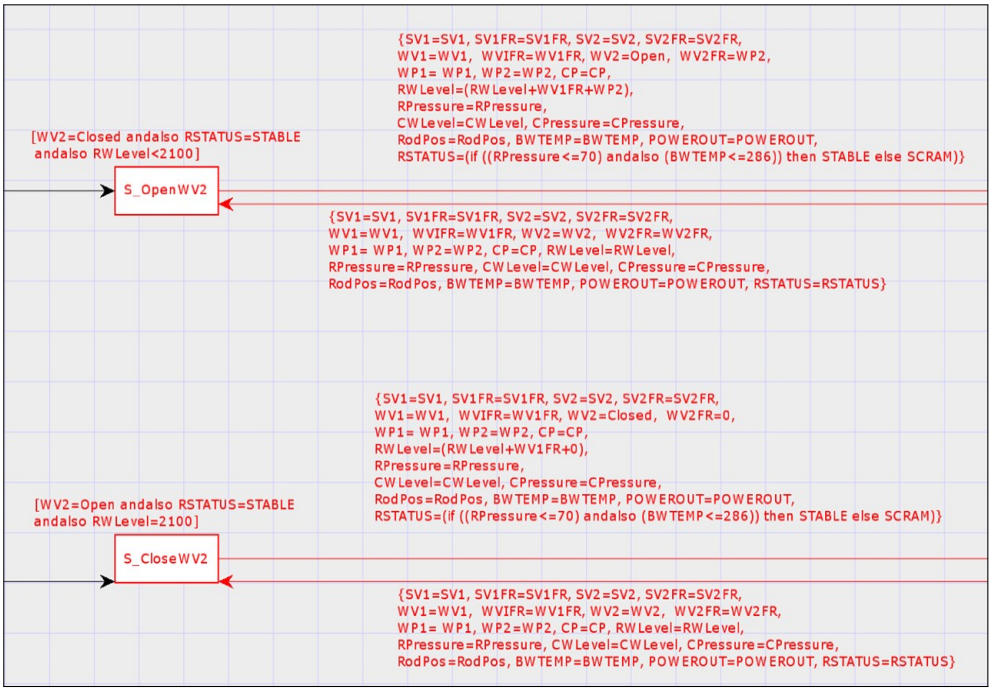

Figure 7.14: S_OpenWV2 and S_CloseWV2 operations of Reactor Control System

(p) **S_CloseWV2:** This operation allows the operator to close the water valve 2 by pressing the $CloseWV2$ button on the interface.

This operation changes the value of the variable $WV2$ to *Closed*. When the water valve is closed, the flow rate of water valve 2 i.e., the value of the variable $WV2FR$ will be equal to 0. As closing of the water valve 2 stops water from entering the reactor vessel, so the reactor water level will also change. This will change the value of the variable *RWLevel* to $(RWLevel + WV1FR + 0)$. This operation is shown in Figure 7.14. The arc expression from the transition $S\_CloseWV2$ to the place *NPP* assigns each variable to its *new* value.

The guard *[WV2=Open andalso RSTATUS=STABLE andalso RW Level=2100]* on the transition $S\_CloseWV2$ means that this transition will only be enabled if the water valve 2 is open, the system is in the *STABLE* state and the reactor water level is equal to 2100.

2. **Scram Page**

Figure 7.15 shows the scram page of the reactor control system. If the reactor pressure goes beyond 286 degree Celsius or the reactor pressure goes above 70 bars, then the system enters the *scram* state. In the *scram* state no control is given to the operator. The system automatically handles and shuts down the system. All the widgets in the scram state are unavailable to the operator and act as *Responder*. In this work as we are not modelling the S-behaviours associated with the *Responder* category, so this page has no S-behaviour transitions. Only one place *NPP* is added to the page which will show the current values of all the variables.

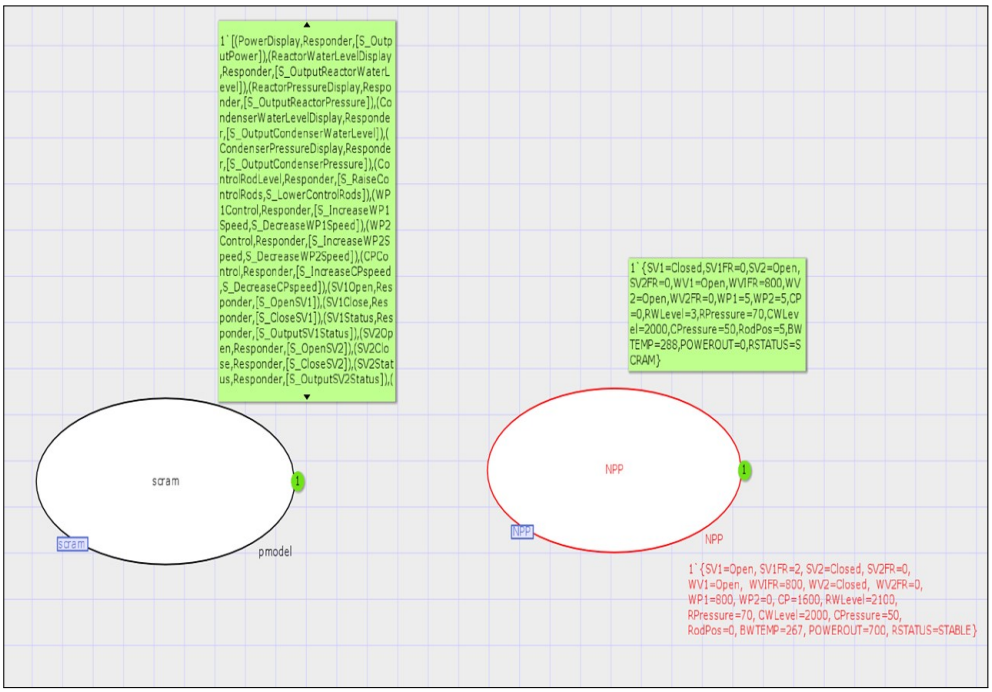

Figure 7.15: Scram Page of Reactor Control System

7.5 Analysis

In this section we will investigate the general properties for the complete CPN model of the reactor control system. To prevent the state space explosion, for analysis we assume the following:

- The value of the steam valve 1 flow rate (SV1FR) can be an integer number between 0 up to 5.
- The value of the steam valve 2 flow rate (SV2FR) can be an integer number between 0 up to 5.
- The value of the water valve 1 flow rate (WV1FR) can be an integer number between 0 up to 5.
- The value of the water valve 2 flow rate (WV2FR) can be an integer number between 0 up to 5.
- The value of the water pump 1 speed (WP1) can be an integer number between 0 up to 5.

- The value of the water pump 2 speed (WP2) can be an integer number between 0 up to 5.
- The value of the condenser pump speed (CP) can be an integer number between 0 up to 5.
- The value of the reactor water level status (RWLevel) can be an integer number between 0 up to 8
- The values of the reactor pressure (RPressure) can be an integer number between 0 up to 5.
- The value of the condenser water level status (CWLevel) can be an integer number between 0 up to 10.
- The value of the condenser pressure (CPressure) can be an integer number between 0 up to 5.
- The value of the rod position (RodPos) can be an integer number from 0 up to 5.
- The value of the boiling water temperature (BWTEMP) can be an integer number between 0 up to 10.
- The value of the power (POWEROUT) can be an integer number between 0 up to 10.
- The system is in stable state and an initial marking is *1‘{SV1=Open, SV1FR=2, SV2=Closed, SV2FR=0, WV1=Open, WVIFR=2, WV2 = Closed, WV2FR=0, WP1= 2, WP2=0, CP=4, RWLevel=3, RPressure=4, CWLevel=4, CPressure=2, RodPos=0, BWTEMP=5, POWEROUT=4, RSTATUS=STABLE}*.
- If the value of the variable *BWTEMP* and *RPRESSURE* gows beyond 5, then the system enters the *Scram* state.

The detailed state space report of the CPN model of the reactor control system is given in [2] and is summarized in Table 7.8

State Space Graph	
Number of Nodes	33756
Number of Arcs	108778
SCC Graph	
Number of Nodes	27754
Number of Arcs	82942
Number of Dead Markings	102 [892, 891, 890, 889, 871, ...]
Dead Transition Instances	None

Table 7.8: statistics of the CPN model of the reactor control system

7.5.1 Investigation of General Properties

7.5.1.1 Livelock

We will use the same method as described in Section 4.4.2.2 to detect a livelock in the CPN model of a reactor control system. In our example, the number of nodes and arcs in the state space graph are not equal to the number of nodes and arcs in the SCC graph as shown in the Table 7.8 and also the strongly connected components consists of more than one node. Hence, we can not say that the state space graph is isomorphic to its SCC graph. Therefore we apply the function given in Table 4.12 to the CPN model of a reactor control system. Table 7.9 shows an empty list as an output. Hence, the model contains no livelock.

```
fun ListTerminalSCCs()=PredAllSccs(SccTerminal);
fun Livelock()=PredSccs(ListTerminalSCCs(),
fn n => not (SccTrivial n),
NoLimit);
```

```
Output:
val it=[ ] :  Scc list
```

Table 7.9: Query to find a non-trivial terminal SCC in the CPN model of a reactor control system

[2] See `https://github.com/sapnajaidka/NPP` for complete details

7.5.1.2 Deadlock freedom

In this section we investigate the CPN model of the system for the absence of deadlock. We use the same method for checking the absence of deadlock as described in Section 4.4.2.1. Table 7.10 shows the dead marking in the model.

```
ListDeadMarkings()
output:
val it = [892, 891, 890, 889, 871, 870,...]:  Node list
```

Table 7.10: Dead markings of the CPN model of the reactor control system

As we now know that when the reactor pressure goes beyond 5 bar and boiling water temperature goes beyond 5 degrees, then the system enters the *SCRAM* state. In the *SCRAM* state entire control is taken from the operator and is automatically handled by a system, therefore, our model does not have any transitions in the *scram* page. Hence, the *SCRAM* state is the expected terminal marking. Now we need to check whether the dead markings in the output of Table 7.10 are the intended dead markings or not. We need to apply a function given in Table 7.11 that detects the dead markings that are not intended (i.e. not terminal markings) for our model.

```
fun ValidTerminalMarking n = (Mark.stablestate'scram 1 n =
1`[(PowerDisplay, Responder, [S_OutputPower]),
(ReactorWaterLevelDisplay, Responder, [S_OutputReactorWaterLevel]),
(ReactorPressureDisplay, Responder,[S_OutputReactorPressure]),
(CondenserWaterLevelDisplay, Responder, [S_OutputCondenserWaterLevel]),
(CondenserPressureDisplay, Responder,[S_OutputCondenserPressure]),
(ControlRodLevel, Responder, [S_RaiseControlRods, S_LowerControlRods]),
(WP1Control, Responder, [S_IncreaseWP1Speed, S_DecreaseWP1Speed]),
(WP2Control, Responder, [S_IncreaseWP2Speed, S_DecreaseWP2Speed]),
(CPControl, Responder, [S_IncreaseCPspeed, S_DecreaseCPspeed]),
(SV1Open, Responder,[S_OpenSV1]),
(SV1Close, Responder,[S_CloseSV1]),
(SV1Status, Responder,[S_OutputSV1Status]),
```

```
(SV2Open, Responder,[S_OpenSV2]),
(SV2Close, Responder,[S_CloseSV2]),
(SV2Status, Responder,[S_OutputSV2Status]),
(WV1Open, Responder,[S_OpenWV1]),
(WV1Close, Responder,[S_CloseWV1]),
(WV1Status, Responder,[S_OutputWV1Status]),
(WV2Open, Responder,[S_OpenWV2]),
(WV2Close, Responder,[S_CloseWV2]),
(WV2Status, Responder, [S_OutputWV2Status]),
(RStatusDisplay, SystemControl, [S_RStatusDisplay])]
orelse Mark.scram'scram 1 n =
1`[(PowerDisplay, Responder, [S_OutputPower]),
(ReactorWaterLevelDisplay, Responder,[S_OutputReactorWaterLevel]),
(ReactorPressureDisplay, Responder,[S_OutputReactorPressure]),
(CondenserWaterLevelDisplay, Responder, [S_OutputCondenserWaterLevel]),
(CondenserPressureDisplay, Responder,[S_OutputCondenserPressure]),
(ControlRodLevel, Responder, [S_RaiseControlRods, S_LowerControlRods]),
(WP1Control, Responder, [S_IncreaseWP1Speed, S_DecreaseWP1Speed]),
(WP2Control, Responder, [S_IncreaseWP2Speed, S_DecreaseWP2Speed]),
(CPControl, Responder, [S_IncreaseCPspeed, S_DecreaseCPspeed]),
(SV1Open, Responder,[S_OpenSV1]),
(SV1Close, Responder,[S_CloseSV1]),
(SV1Status, Responder,[S_OutputSV1Status]),
(SV2Open, Responder,[S_OpenSV2]),
(SV2Close, Responder,[S_CloseSV2]),
(SV2Status, Responder,[S_OutputSV2Status]),
(WV1Open, Responder,[S_OpenWV1]),
(WV1Close, Responder,[S_CloseWV1]),
(WV1Status, Responder,[S_OutputWV1Status]),
(WV2Open, Responder,[S_OpenWV2]),
(WV2Close, Responder,[S_CloseWV2]),
```

```
(WV2Status, Responder, [S_OutputWV2Status]),
(RStatusDisplay, SystemControl, [S_RStatusDisplay])] )
fun InValidTerminal()=PredNodes(ListDeadMarkings(),
fn n => not (ValidTerminalMarking n), NoLimit);
```

```
output:
val it =[]:  Node list
```

Table 7.11: Invalid terminal nodes for the CPN model of a reactor control system

We have an empty list as value i.e. all the dead markings are terminal markings. Hence there are no deadlocks.

7.5.1.3 Checking Boundary Values

It is important to check whether upper and lower bounds on values are properly respected and handled. For example, the position of rods cannot be more than 5.

We verify this by writing the code given in Table 7.12 to query the state space to see if there is any marking where the value of *RodPos* is greater than 5.

```
fun checkvaluerodpos(RodPos:RODPOS): Arc list =
PredAllArcs (fn a=> case ArcToBE a of
Bind.stablestate'S_RaiseControlRods(1,{RPressure=_,
RodPos=6, CP=_, RWLevel=_, WV2=_, BWTEMP=_, WV1=_,
CPressure=_, SV2=_, SV1=_, WP2=_,WP1=_, WV2FR=_,
WV1FR=_, RSTATUS=_,POWEROUT=_, SV2FR=_, SV1FR=_,
CWLevel=_}) => RodPos = 6| _ => false))
```

```
Output:
val it=[ ]:  Arc list
```

Table 7.12: Returns all the arcs in the state space graph where the value of *RodPos* is 6

we get the empty arc list as an output as shown in Table 7.12. This means that there exists no such arc in the entire state space that has a binding

element $RodPos = 6$. This means that in the entire state space, the value of the $RodPos$ is never greater than 5.

We also assume in our model that if the value of the $BWTEMP$ goes above 5, then the system enters the $SCRAM$ state. We need to make sure that in our model, when the value of the variable $BWTEMP$ is greater than 5, then the value of $RSTATUS$ should be $SCRAM$. We verify this by writing the code given in Table 7.13. In this code, we are checking whether there exist any arc with bindings $BWTEMP = 6$ and $RSTATUS = STABLE$.

```
fun checkvaluebwtemp(BWTEMP:DEGREE, RSTATUS:RSTATUS): Arc list =
PredAllArcs (fn a=> case ArcToBE a of
Bind.stablestate'S_RaiseControlRods(1,{RPressure=_,
RodPos=_, CP=_, RWLevel=_, WV2=_, BWTEMP=6, WV1=_,
CPressure=_, SV2=_, SV1=_, WP2=_,WP1=_, WV2FR=_,
WV1FR=_, RSTATUS=STABLE,POWEROUT=_, SV2FR=_, SV1FR=_,
CWLevel=_}) => BWTEMP = 6 andalso RSTATUS=STABLE| _ => false)
```

```
Output:
val it=[99992, 99966, 99951, 9995, 99945, 99910,...]:  Arc list
```

Table 7.13: Returns all the arcs in the state space graph where the value of $BWTEMP$ is 6 and $RSTATUS$ is STABLE

we get the arc list: [99992, 99966, 99951, 9995, 99945, 99910,...] as shown in Table 7.13. This means that there exists such arc that has a binding elements $BWTEMP = 6$ and $RSTATUS = STABLE$. This means that the safety condition is violated in our model.

To solve this problem, we need to fix the arc expression from the transition $S\_RaiseControlRods$ to the place NPP. Current arc expression is:

```
({SV1=SV1, SV1FR=SV1FR, SV2=SV2, SV2FR=SV2FR, WV1=WV1,
WVIFR=WV1FR, WV2=WV2, WV2FR=WV2FR, WP1= WP1, WP2=WP2, CP=CP,
RWLevel=RWLevel, RPressure=RPressure, CWLevel=CWLevel,
CPressure=CPressure, RodPos=RodPos+1, BWTEMP=BWTEMP + 1,
POWEROUT=POWEROUT, RSTATUS=(if ((RPressure<=5) andalso
(BWTEMP<=5)) then STABLE else SCRAM)})
```

In the current expression we can see that the value of the variable $BWTEMP$

taken by *RSTATUS* is the old value of *BWTEMP* and not the new increased value. So we fix the current expression by using the arc expression as written below:

```
({SV1=SV1, SV1FR=SV1FR, SV2=SV2, SV2FR=SV2FR, WV1=WV1,
WVIFR=WV1FR, WV2=WV2, WV2FR=WV2FR, WP1= WP1, WP2=WP2, CP=CP,
RWLevel=RWLevel, RPressure=RPressure, CWLevel=CWLevel,
CPressure=CPressure, RodPos=RodPos+1, BWTEMP=BWTEMP + 1,
POWEROUT=POWEROUT, RSTATUS=(if ((RPressure<=5) andalso
((BWTEMP+1)<=5)) then STABLE else SCRAM)})
```

After fixing we again apply the function written in Table 7.13 and we get the empty arc list as an output as shown in Table 7.14. This means that there exists no such arc in the entire state space that has a binding element $BWTEMP = 6$ when $RSTATUS = STABLE$.

```
fun checkvaluebwtemp(BWTEMP:DEGREE, RSTATUS:RSTATUS): Arc list =
PredAllArcs (fn a=> case ArcToBE a of
Bind.stablestate'S_RaiseControlRods(1,{RPressure=_,
RodPos=_, CP=_, RWLevel=_, WV2=_, BWTEMP=6, WV1=_,
CPressure=_, SV2=_, SV1=_, WP2=_,WP1=_, WV2FR=_,
WV1FR=_, RSTATUS=STABLE,POWEROUT=_, SV2FR=_, SV1FR=_,
CWLevel=_}) => BWTEMP = 6 andalso RSTATUS=STABLE| _ => false)
```

```
Output:
val it=[ ]:  Arc list
```

Table 7.14: Returns all the arcs in the state space graph where the value of *BWTEMP* is 6 and *RSTATUS* is STABLE

7.6 Summary

In this chapter we have used the Coloured Petri Net to model a reactor control system of nuclear power plant. The main aim of doing this was to show that we can use the Coloured Petri Nets alone to model safety-critical interactive systems and the scope of the method presented is not just limited to medical infusion pumps. We have used the same method and rules as defined in Chapter 4 and Chapter 5. We have used the state space analysis method to

investigate the behaviours of the CPN model to ensure that the system works as intended. We are able to find some flaws in the system with the help of the queries and modified the model and investigated the model again to check it is working.

Chapter 8

Conclusions

8.1 Introduction

In this research a new approach for modelling and analyzing a safety-critical interactive system using Coloured Petri Nets is presented which has all the parts (user interface, interaction and functional) in a single model. The achieved model not only shows the navigational properties of the system, but also allows us to include the underlying system functionality in the model as well. By means of simulation we can actually see what widgets are available and what happens when the user interacts with them. Also we can actually see how the behaviours change the underlying system functionality. We have used the state space analysis method to investigate the behaviour of a system. By these means we can check to see if the model is working as expected.

8.2 Contributions

The contributions of this thesis are summarised as:

- A technique to model user interface and interaction of a safety-critical interactive system (chapter 4).
- A technique to model functionality of a safety-critical interactive system (chapter 5).

- A technique to combine the user interface, interaction and functional part of a safety-critical interactive system in a single model (chapter 5).
- Investigating the behaviour of a model to see if the model is working as expected (chapters 4 and 5).

We set out to answer the following research questions:

1. How can we create formal models of safety-critical interactive systems that have all the aspects (user interface, interaction and functional) in a single model, in a way which allows us to verify properties and formally reason about it?
2. How can we combine existing accepted formalisms to synthesize another formal method which builds model as in 1?
3. What are the benefits of the new method?

In chapter 4, we presented a technique to model the user interface and interaction of a safety-critical interactive system using Coloured Petri Nets based on the existing formalisms (PM and PIMs). Using the state space analysis method, we are able to investigate the behaviour of the model to see if it is working as expected. In chapter 5, we presented a technique to model the functionality of a safety-critical system using Coloured Petri Nets based on an existing formalism (Z specification). We have also described how to extend the model of a user interface and interaction by adding the functional part to it. The new technique results in a single model capturing all three aspects and all the connections between them. Using the state space analysis method, we investigated the behaviour of a single combined model and verified properties. This effectively addresses questions one and two.

Simulations are used for checking the model and help finding errors such as errors in inscriptions on arcs and errors in sequences of events. If it seems that the model is not working as expected then we can look deeper and see what the flaws are in the model and what changes should be made. A simulation is similar to single-step debugging. It provides a way to walk through a CPN

model, investigating different scenarios in detail and checking whether the model works as expected. It is possible to observe the effects of the individual steps directly on the graphical representation of the CPN model.

We have chosen CPNs mainly because of the state space analysis-based methods made possible within the CPN Tool, based on support for the state space graph and the strongly connected components graph to be automatically generated. Once we have these, then functions can be written in the SML-subset available in CPN Tools which allow many useful further checking and testing mechanisms. This way of investigating and model-checking properties by writing SML functions to define a process for computing a check contrasts with the more usual (for ProB users, for example) method of writing temporal logic statements which are statements of properties. It may be that this more "procedural" way of expressing properties is more attractive to "conventional" programmers, compared with having to learn and then express temporal logic statements in a "declarative" way.

Some of the general modelling approaches (as for instance in [110] [42]) discussed in chapter 2 results in three separate models and they have to tackle the problem of separation of concerns between user interface, interaction and functional elements, while at the same time managing the relationship between them. A lot of work is required to do the coupling of functional behaviour with interactive elements to ensure consistency. Combining all the three parts of a system in a single model addresses this issue and bridges the gap between the user interface, interaction and underlying system functionality. The benefit of having all aspects in a single model is that there is no work required to combine them for analysis. We have given three examples of modelling and analysis of safety-critical interactive systems and presented the results; simplified infusion pump example in chapters 4 and 5, Niki T34 syringe driver in chapter 6 and reactor control system of nuclear power plant in chapter 7. We believe that using this technique can contribute to the safer use of interactive systems. This addresses question three.

8.3 Limitations and Future Work

There are some limitations to this work. To prevent the state space explosion we have constrained the limit of the values of the variables. For example, in chapter six for the Niki T34 syringe driver, we have constrained the time to five hours instead of 24 hours to control the state space. This makes the state space far smaller, but does not mean we miss checking things like upper bounds and lower bounds of duration. Also we have not modelled the behaviours which display messages on the screen, just to control the state space. In future it may be possible to explore different ways of controlling state space explosion.

In chapter four we present a technique to model the user interface of a system. Each state is represented as a set of triples comprised of widget name, its category and set of behaviours associated with that widget in that particular state. Before actually modelling these sets of triples, we declare the names of the widgets, categories and all the behaviours that a system have. This way we can have consistent names in the triple. We also say for our technique that the name of the transitions are the same as the name of the behaviours. In the current state of this work, we need to manually check that the names of the transitions are the same as the name of the behaviours. In future it might be worth looking at automating this check to ensure consistency.

As presented in chapter six, the S-behaviour transitions in the CPN model represent the underlying system functionality and there are two arcs needed to model the functionality going to and from the place Z to the S-behaviour transition. The arc from Z to the S-behaviour transition simply contains assignments which set each variable to its current value. This set of assignments "picks up" the current values of the variables ready to be used by the second arc. This second arc, the one from the S-behaviour to the place Z, assigns each variable to its *new* value. Taken together these two arcs express the particular operation of a system. The drawback is that if the operation is complex, then the arc expression tends to become very large and the model looks very complicated. In future it may be possible to explore creating functions in CPN

SML and use only the functions' names in the arc expression instead of the whole description in the arc to make the arc inscription shorter.

In this thesis we have focused on Coloured Petri Nets. We chose to use CPN because of its popularity and we were most familiar with this formalism. Coloured Petri Nets provides a graphical representation and hierarchical structuring mechanism, and a state space verification technique, which allows querying the state space to investigate behaviours of a system. There are several tools that support Coloured Petri Nets including the CPN Tool which helps in building CPN models and allows simulation and analysis using state spaces. Comparing this with other possibilities, there is a tool called RENEW for Reference Nets but it does not generate the state space graph [130]. Also we find in the literature that the Reference Net models or even Object Petri Nets models get transformed into behaviourally equivalent CPNs for adapting CPN analysis techniques, as mentioned in [131]. Despite this, we believe that the underlying principles of our work can be applied to other formal languages.

We believe that this technique is useful for developers and user interface designers, but knowledge of formal methods, especially Coloured Petri Nets, is required to use this technique. In future it may be possible to build a tool incorporating the techniques developed in this thesis so that any user without having knowledge of formal methods would be able to model and analyse the system.

The Niki T34 syringe driver and the reactor control system of a nuclear power plant are used as examples in this thesis. Concrete investigation into the applicability of the developed technique to different types of interactive systems is required. It is intended that the developed technique is applicable to all interactive systems. This would require future work in order to ensure this applicability to other types of interactive systems and to other safety critical domains.

Bibliography

[1] S. Jaidka, S. Reeves, and J. Bowen, "Modelling safety-critical devices: coloured petri nets and z," in *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pp. 51–56, ACM, 2017.

[2] S. Jaidka, S. Reeves, and J. Bowen, "A coloured petri net approach to model and analyze safety-critical interactive systems," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 347–354, IEEE, 2019.

[3] A. J. Dix, *Formal methods for interactive systems*, vol. 16. Academic Press London, 1991.

[4] D. J. Duke and M. D. Harrison, "Event model of human-system interaction," *Software Engineering Journal*, vol. 10, no. 1, pp. 3–12, 1995.

[5] J. C. Campos and M. D. Harrison, "Interaction engineering using the IVY tool," in *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, pp. 35–44, ACM, 2009.

[6] U. S. N. R. Commission *et al.*, "Boiling water reactors," *URL: http://www. nrc. gov/reactors/bwrs. html*, 2016.

[7] J. Bowen and S. Reeves, "Combining models for interactive system modelling," in *The Handbook of Formal Methods in Human-Computer Interaction*, pp. 161–182, Springer, 2017.

[8] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, and T. Carey, *Human-computer interaction*. Addison-Wesley Longman Ltd., 1994.

[9] N. R. Storey, *Safety critical computer systems*. Addison-Wesley Longman Publishing Co., Inc., 1996.

[10] H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, and J. Raman, "Analysis of safety-critical computer failures in medical devices," *IEEE Security & Privacy*, vol. 11, no. 4, pp. 14–26, 2013.

[11] CNBC, "Uber suspends self-driving car program after arizona crash.." Retrieved September 2019 from: `https://www.cnbc.com/2017/03/26/uber-self-driving-car-arizona-crash-suspended.html`.

[12] D. M. Reporter, "Mother dies after nurse makes error administering drug." Retrieved September 2019 from: `https://www.dailymail.co.uk/health/article-1359778/Mother-dies-nurse-administers-TEN-times-prescribed-drug.html`.

[13] H. Thimbleby, P. Oladimeji, and P. Cairns, "Unreliable numbers: error and harm induced by bad design can be reduced by better design," *Journal of The Royal Society Interface*, vol. 12, no. 110, p. 20150685, 2015.

[14] A. Degani and M. Heymann, "Pilot-autopilot interaction: A formal perspective," *Abbott et al.[1]*, pp. 157–168, 2000.

[15] H. Thimbleby, "Think! interactive systems need safety locks," *Journal of computing and information technology*, vol. 18, no. 4, pp. 349–360, 2010.

[16] D. Arney, R. Jetley, P. Jones, I. Lee, and O. Sokolsky, "Formal methods based development of a PCA infusion pump reference model: Generic infusion pump (GIP) project," in *2007 Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability (HCMDSS-MDPnP 2007)*, pp. 23–33, IEEE, 2007.

[17] E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys (CSUR)*, vol. 28, no. 4, pp. 626–643, 1996.

[18] J. Bowen and S. Reeves, "Generating obligations, assertions and tests from UI models," *Proceedings of the ACM on Human-Computer Interaction*, vol. 1, no. EICS, p. 5, 2017.

[19] J. D. Turner, *Supporting interactive system testing with interaction sequences*. PhD thesis, The University of Waikato, 2019.

[20] H. Garavel and S. Graf, "Formal methods for safe and secure computer systems," *Federal Office for Information Security*, 2013.

[21] K. Jensen, *Coloured Petri Nets: basic concepts, analysis methods and practical use*, vol. 1. Springer Science & Business Media, 2013.

[22] R. J. Jacob, "Using formal specifications in the design of a human-computer interface," *Communications of the ACM*, vol. 26, no. 4, pp. 259–264, 1983.

[23] A. J. Dix and C. Runciman, "Abstract models of interactive systems," *People and Computers: Designing the interface*, pp. 13–22, 1985.

[24] J. Bowen and S. Reeves, "Formal models for user interface design artefacts," *Innovations in Systems and Software Engineering*, vol. 4, no. 2, pp. 125–141, 2008.

[25] A. M. Porrello, "Death and denial: The failure of the THERAC-25, A Medical Linear Accelerator," *Death and Denial: The Failure of the THERAC-25, AMedical Linear Accelerator*, 2012.

[26] B. Kim, A. Ayoub, O. Sokolsky, I. Lee, P. Jones, Y. Zhang, and R. Jetley, "Safety-assured development of the gpca infusion pump software," in *Proceedings of the ninth ACM international conference on Embedded software*, pp. 155–164, ACM, 2011.

[27] A. Degani and M. Heymann, "Formal verification of human-automation interaction," *Human factors*, vol. 44, no. 1, pp. 28–43, 2002.

[28] K. Jensen, *Coloured Petri Nets: basic concepts, analysis methods and practical use*, vol. 2. Springer Science & Business Media, 2013.

[29] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems," *International Journal on Software Tools for Technology Transfer*, vol. 9, no. 3-4, pp. 213–254, 2007.

[30] W. Reisig and G. Rozenberg, *Lectures on Petri nets i: basic models: advances in Petri nets*. Springer Science & Business Media, 1998.

[31] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[32] D. Vernez, D. Buchs, and G. Pierrehumbert, "Perspectives in the use of coloured Petri nets for risk analysis and accident modelling," *Safety science*, vol. 41, no. 5, pp. 445–463, 2003.

[33] A. Kleyner and V. Volovoi, "Application of Petri nets to reliability prediction of occupant safety systems with partial detection and repair," *Reliability Engineering & System Safety*, vol. 95, no. 6, pp. 606–613, 2010.

[34] D. A. Zaitsev and T. R. Shmeleva, "Modeling with Colored Petri Nets: Specification, Verification, and Performance Evaluation of Systems," in *Automated Systems in the Aviation and Aerospace Industries*, pp. 378–404, IGI Global, 2019.

[35] D. Navarre, P. Palanque, J.-F. Ladry, and E. Barboni, "ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 16, no. 4, p. 18, 2009.

[36] M. Westergaard, "CPN Tools 4: Multi-formalism and extensibility," in *International Conference on Applications and Theory of Petri Nets and Concurrency*, pp. 400–409, Springer, 2013.

[37] M. Westergaard and H. Verbeek, "CPN Tools," 2012.

[38] X. He, "PZ nets - a formal method integrating Petri nets with Z," *Information and Software Technology*, vol. 43, no. 1, pp. 1–18, 2001.

[39] M. Benjamin, "A message passing system. an example of combining CSP and Z," in *Z User Workshop*, pp. 221–228, Springer, 1990.

[40] P. Kars, "Experience using Spin and Promela in the design of a storm surge barrier control system," *Gr egoire Gr e95*, 1995.

[41] R. Duke and G. Smith, "Temporal logic and Z specifications," *Australian Computer Journal*, vol. 21, no. 2, pp. 62–66, 1989.

[42] J. Bowen and S. Reeves, "Using formal models to design user interfaces: a case study," in *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI... but not as we know it-Volume 1*, pp. 159–166, British Computer Society, 2007.

[43] L. J. Chubb, *Data Refinement Models for Program and User Interfaces*. PhD thesis, New South Wales, Australia, Australia, 1994. Not available from Univ. Microfilms Int.

[44] M. Heiner and M. Heisel, "Modeling safety-critical systems with Z and Petri nets," in *Computer Safety, Reliability and Security*, pp. 361–374, Springer, 1999.

[45] P. A. Palanque and A. Schyn, "A model-based approach for engineering multimodal interactive systems.," in *INTERACT*, vol. 3, pp. 543–550, 2003.

[46] B. Weyers, "FILL: formal description of executable and reconfigurable models of interactive systems," in *Proceedings of the workshop on formal methods in human computer interaction*, pp. 1–6, 2015.

[47] J. L. Silva, O. R. Ribeiro, J. M. Fernandes, J. C. Campos, and M. D. Harrison, "The APEX framework: prototyping of ubiquitous environments based on Petri nets," in *International Conference on Human-Centred Software Engineering*, pp. 6–21, Springer, 2010.

[48] W. Reisig, *Petri nets: an introduction*, vol. 4. Springer Science & Business Media, 2012.

[49] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1, pp. 134–152, 1997.

[50] J. Woodcock and J. Davies, *Using Z: specification, refinement, and proof*. Prentice-Hall, Inc., 1996.

[51] J.-R. Abrial and J.-R. Abrial, *The B-book: assigning programs to meanings*. Cambridge University Press, 2005.

[52] J. Nielsen and R. Molich, "Heuristic evaluation of user interfaces," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 249–256, ACM, 1990.

[53] B. Weyers, J. Bowen, A. Dix, and P. Palanque, *The handbook of formal methods in human-computer interaction*. Springer, 2017.

[54] H. Thimbleby, "Interaction walkthrough: Evaluation of safety critical interactive systems," in *Interactive Systems. Design, Specification, and Verification* (G. Doherty and A. Blandford, eds.), (Berlin, Heidelberg), pp. 52–66, Springer Berlin Heidelberg, 2007.

[55] G. de Haan, G. C. van der Veer, and J. C. van Vliet, "Formal modelling techniques in human-computer interaction," *Acta Psychologica*, vol. 78, no. 1-3, pp. 27–67, 1991.

[56] J. Bowen, "Formal methods in safety-critical standards," in *Proceedings 1993 Software Engineering Standards Symposium*, pp. 168–177, Aug 1993.

[57] J. Bowen and V. Stavridou, "Safety-critical systems, formal methods and standards," *Software Engineering Journal*, vol. 8, no. 4, pp. 189–209, 1993.

[58] P. Palanque, R. Bastide, and F. Paternò, "Formal specification as a tool for objective assessment of safety-critical interactive systems," in *Human-Computer Interaction INTERACT'97*, pp. 323–330, Springer, 1997.

[59] L. M. Ma, A. Houser, K. M. Feigh, and M. Bolton, "An analysis of air traffic management concepts of operation using simulation and formal verification," in *AIAA Scitech 2019 Forum*, p. 0984, 2019.

[60] J. Lahtinen, J. Valkonen, K. Björkman, J. Frits, I. Niemelä, and K. Heljanko, "Model checking of safety-critical software in the nuclear engineering domain," *Reliability Engineering & System Safety*, vol. 105, pp. 104–113, 2012.

[61] A. Fantechi, "Twenty-five years of formal methods and railways: what next?," in *International Conference on Software Engineering and Formal Methods*, pp. 167–183, Springer, 2013.

[62] T. Lecomte, T. Servat, G. Pouzancre, *et al.*, "Formal methods in safety-critical railway systems," in *10th Brasilian symposium on formal methods*, pp. 29–31, 2007.

[63] M. Gaied, A. M'halla, D. Lefebvre, and K. Ben Othmen, "Robust control for railway transport networks based on stochastic p-timed petri net models," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, p. 0959651818823583, 2019.

[64] P. E. Barbosa, M. Morais, K. Galdino, M. F. Andrade, L. Gomes, F. Moutinho, J. C. de Figueiredo, *et al.*, "Towards medical device behavioural validation using Petri nets," in *Computer-Based Medical Systems (CBMS), 2013 IEEE 26th International Symposium on*, pp. 4–10, IEEE, 2013.

[65] N. Majma and S. M. Babamir, "Specification and verification of medical monitoring system using petri-nets," *Journal of medical signals and sensors*, vol. 4, no. 3, p. 181, 2014.

[66] M. D. Harrison, L. Freitas, M. Drinnan, J. C. Campos, P. Masci, C. di Maria, and M. Whitaker, "Formal techniques in the safety analysis of software components of a new dialysis machine," *Science of Computer Programming*, vol. 175, pp. 17–34, 2019.

[67] G. D. Abowd, *Formal aspects of human-computer interaction*. University of Oxford, 1991.

[68] M. D. Harrison, J. C. Campos, and P. Masci, "Reusing models and properties in the analysis of similar interactive devices," *Innovations in Systems and Software Engineering*, vol. 11, no. 2, pp. 95–111, 2015.

[69] D. Navarre, P. Palanque, and S. Basnyat, "A formal approach for user interaction reconfiguration of safety critical interactive systems," in *International Conference on Computer Safety, Reliability, and Security*, pp. 373–386, Springer, 2008.

[70] J. Bowen, "Creating models of interactive systems with the support of lightweight reverse-engineering tools," in *Proceedings of the 7th ACM SIGCHI symposium on engineering interactive computing systems*, pp. 110–119, ACM, 2015.

[71] J. C. Knight and S. S. Brilliant, "Preliminary evaluation of a formal approach to user interface specification," in *International Conference of Z Users*, pp. 329–346, Springer, 1997.

[72] F. Paternò, C. Santoro, and L. D. Spano, "Improving support for visual task modelling," in *International Conference on Human-Centred Software Engineering*, pp. 299–306, Springer, 2012.

[73] J. Jacky, *The way of Z: practical programming with formal methods*. Cambridge University Press, 1997.

[74] F. Paterno, M. S. Sciacchitano, and J. Lowgren, "A user interface evaluation mapping physical user actions to task-driven formal specifications," in *Design, specification and verification of interactive systems' 95*, pp. 35–53, Springer, 1995.

[75] T. Bolognesi and E. Brinksma, "Introduction to the ISO specification language LOTOS," *Computer Networks and ISDN systems*, vol. 14, no. 1, pp. 25–59, 1987.

[76] G. Doherty and M. D. Harrison, "A representational approach to the specification of presentations," in *Design, Specification and Verification of Interactive Systems' 97*, pp. 273–290, Springer, 1997.

[77] N. Plat and P. G. Larsen, "An overview of the ISO/VDM-SL standard," *ACM Sigplan Notices*, vol. 27, no. 8, pp. 76–82, 1992.

[78] E. Barboni, S. Conversy, D. Navarre, and P. Palanque, "Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification," in *Interactive Systems. Design, Specification, and Verification* (G. Doherty and A. Blandford, eds.), (Berlin, Heidelberg), pp. 25–38, Springer Berlin Heidelberg, 2007.

[79] H. Thimbleby, "Safer user interfaces: A case study in improving number entry," *IEEE Transactions on Software Engineering*, vol. 41, no. 7, pp. 711–729, 2015.

[80] C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[81] J. C. Silva, J. C. Campos, and J. Saraiva, "Combining formal methods and functional strategies regarding the reverse engineering of interactive applications," in *Interactive Systems. Design, Specification, and Verification* (G. Doherty and A. Blandford, eds.), (Berlin, Heidelberg), pp. 137–150, Springer Berlin Heidelberg, 2007.

[82] J. C. Campos and M. Harrison, "Modelling and analysing the interactive behaviour of an infusion pump," *Electronic Communications of the EASST*, vol. 45, 2011.

[83] J. Bowen and S. Reeves, "Modelling safety properties of interactive medical systems," in *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*, pp. 91–100, ACM, 2013.

[84] K. Loer and M. Harrison, "Towards usable and relevant model checking techniques for the analysis of dependable interactive systems," in *Proceedings 17th IEEE International Conference on Automated Software Engineering,*, pp. 223–226, IEEE, 2002.

[85] G. D. Abowd, H.-M. Wang, and A. F. Monk, "A formal technique for automated dialogue development," in *Proceedings of the 1st conference on Designing interactive systems: processes, practices, methods, & techniques*, pp. 219–226, ACM, 1995.

[86] D. R. Olsen Jr, "Propositional production systems for dialog description," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 57–64, ACM, 1990.

[87] H.-M. Wang and G. Abowd, "A tabular interface for automated verification of event-based dialogs," tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1994.

[88] A. J. Dix, "Abstract, generic models of interactive systems.," in *BCS HCI*, vol. 1988, pp. 63–77, 1988.

[89] D. J. Duke and M. D. Harrison, "Abstract interaction objects," in *Computer Graphics Forum*, vol. 12, pp. 25–36, Wiley Online Library, 1993.

[90] C. A. Middelburg, "VVSL: A language for structured VDM specifications," *Formal aspects of computing*, vol. 1, no. 1, pp. 115–135, 1989.

[91] B. Fields, P. Wright, and M. Harrison, "Applying formal methods for human error tolerant design," in *Workshop on Software Engineering and Human-Computer Interaction*, pp. 185–195, Springer, 1994.

[92] J. C. Campos and M. D. Harrison, "Automated deduction and usability reasoning," in *Encyclopedia of Human Computer Interaction*, pp. 45–52, IGI Global, 2006.

[93] J. C. Campos and M. D. Harrison, "Model checking interactor specifications," *Automated Software Engineering*, vol. 8, no. 3-4, pp. 275–310, 2001.

[94] M. Reynolds, "An axiomatization of full computation tree logic," *The Journal of Symbolic Logic*, vol. 66, no. 3, pp. 1011–1057, 2001.

[95] J. C. Campos and M. D. Harrison, "Systematic analysis of control panel interfaces using formal tools," in *International Workshop on Design, Specification, and Verification of Interactive Systems*, pp. 72–85, Springer, 2008.

[96] J. C. Campos and M. D. Harrison, "Considering context and users in interactive systems analysis," in *IFIP International Conference on Engineering for Human-Computer Interaction*, pp. 193–209, Springer, 2007.

[97] M. Sousa, J. C. Campos, M. Alves, and M. D. Harrison, "Formal verification of safety-critical user interfaces: a space system case study," in *2014 AAAI Spring Symposium Series*, 2014.

[98] S. Owre, J. M. Rushby, and N. Shankar, "PVS: A prototype verification system," in *International Conference on Automated Deduction*, pp. 748–752, Springer, 1992.

[99] M. D. Harrison, P. Masci, J. C. Campos, and P. Curzon, "Automated theorem proving for the systematic analysis of an infusion pump," *Electronic Communications of the EASST*, vol. 69, 2014.

[100] M. D. Harrison, P. Masci, and J. C. Campos, "Verification templates for the analysis of user interface software design," *IEEE Transactions on Software Engineering*, 2018.

[101] M. D. Harrison, P. Masci, J. C. Campos, and P. Curzon, "Demonstrating that medical devices satisfy user related safety requirements," in *Software Engineering in Health Care*, pp. 113–128, Springer, 2014.

[102] P. A. Palanque and R. Bastide, "Petri net based design of user-driven interfaces using the interactive cooperative objects formalism," in *Interactive systems: Design, specification, and verification*, pp. 383–400, Springer, 1995.

[103] R. Bastide, D. Navarre, and P. Palanque, "A model-based tool for interactive prototyping of highly interactive applications," in *CHI'02 extended abstracts on Human factors in Computing Systems*, pp. 516–517, ACM, 2002.

[104] P. Palanque, R. Bastide, and V. Sengès, "Validating interactive system design through the verification of formal task and system models," in *IFIP International Conference on Engineering for Human-Computer Interaction*, pp. 189–212, Springer, 1995.

[105] C. Martinie, P. Palanque, D. Navarre, M. Winckler, and E. Poupart, "Model-based training: an approach supporting operability of critical interactive systems," in *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, pp. 53–62, ACM, 2011.

[106] R. Bastide, D. Navarre, P. Palanque, A. Schyn, and P. Dragicevic, "A model-based approach for real-time embedded multimodal systems in military aircrafts," in *ICMI*, vol. 4, pp. 243–250, 2004.

[107] J. Bowen and S. Reeves, "Formal models for informal GUI designs," *Electronic Notes in Theoretical Computer Science*, vol. 183, pp. 57–72, 2007.

[108] J. A. Bowen, *Formal specification of user interface design guidelines*. PhD thesis, Citeseer, 2005.

[109] J. Bowen and S. Reeves, "Refinement for user interface designs," *Formal Aspects of Computing*, vol. 21, no. 6, pp. 589–612, 2009.

[110] J. Bowen and S. Reeves, "A simplified Z semantics for Presentation Interaction Models," in *FM 2014: Formal Methods*, pp. 148–162, Springer, 2014.

[111] G. Smith, *The Object-Z specification language*, vol. 1. Springer Science & Business Media, 2012.

[112] J. Derrick and E. A. Boiten, "Refinement in Z and Object-Z: Foundations and advanced applications," 2014.

[113] M. Saaltink, "The Z/EVES 2.0 Mathematical Toolkit," *Z/EVES distribution document*, 1999.

[114] C. A. Petri, "Kommunikation mit automaten," 1962.

[115] J. Wang, "Petri nets for dynamic event-driven system modeling," *Handbook of Dynamic System Modeling*, pp. 1–17, 2007.

[116] K. Jensen, *High-level Petri nets*. Springer, 1983.

[117] H. J. Genrich and K. Lautenbach, "System modelling with high-level Petri nets," *Theoretical computer science*, vol. 13, no. 1, pp. 109–135, 1981.

[118] R. Harper, N. Rothwell, and K. Mitchell, *Introduction to standard ML*. Laboratory for Foundations of Computer Science, 1989.

[119] K. Jensen, S. Christensen, and L. M. Kristensen, "CPN Tools state space manual," *Department of Computer Science, Univerisity of Aarhus*, 2006.

[120] L. M. Kristensen, S. Christensen, and K. Jensen, "The practitioner's guide to Coloured Petri Nets," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 2, no. 2, pp. 98–132, 1998.

[121] A. Valmari, "The state explosion problem," in *Advanced Course on Petri Nets*, pp. 429–528, Springer, 1996.

[122] P. Katsaros, "A roadmap to electronic payment transaction guarantees and a Colored Petri Net model checking approach," *Information and Software Technology*, vol. 51, no. 2, pp. 235–257, 2009.

[123] A. Cheng, S. Christensen, and K. H. Mortensen, "Model checking Coloured Petri Nets-exploiting strongly connected components," *DAIMI report series*, no. 519.

[124] J. M. Spivey and J. Abrial, *The Z notation*. Prentice Hall Hemel Hempstead, 1992.

[125] M. E. V. Blanco, *Modelling and Analysis of the Resource Reservation Protocol Using Coloured Petri Nets*. PhD thesis, Citeseer, 2003.

[126] C. M. Electronics, *Niki T34 syringe pump instruction manual*. ref. 100-090SS Edition (2008).

[127] J. Bowen and S. Reeves, "Modelling user manuals of modal medical devices and learning from the experience," in *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, pp. 121–130, ACM, 2012.

[128] D. E. Arney, R. Jetley, P. Jones, I. Lee, A. Ray, O. Sokolsky, and Y. Zhang, "Generic infusion pump hazard analysis and safety requirements version 1.0," *Technical Reports (CIS)*, p. 893, 2009.

[129] B. Weyers, M. D. Harrison, J. Bowen, A. Dix, and P. Palanque, "Case studies," in *The Handbook of Formal Methods in Human-Computer Interaction*, pp. 89–121, Springer, 2017.

[130] O. Kummer, F. Wienberg, M. Duvigneau, M. Köhler, D. Moldt, and H. Rölke, "Renew–the reference net workshop," in *Tool Demonstrations, 21st International Conference on Application and Theory of Petri Nets,*

Computer Science Department, Aarhus University, Aarhus, Denmark, pp. 87–89, 2000.

[131] C. Lakos, "From Coloured Petri Nets to Object Petri Nets," in *International Conference on Application and Theory of Petri Nets*, pp. 278–297, Springer, 1995.

Appendix A

State Space Reports

A.1 State Space Report for example 3.7

CPN Tools state space report for:
/cygdrive/C/Users/sj113/Downloads/simple infusion pump.cpn
Report generated: Sun Dec 23 07:51:35 2018

Statistics

State Space

Nodes: 3
Arcs: 3
Secs: 0
Status: Full

Scc Graph

Nodes: 2
Arcs: 1
Secs: 0

Boundedness Properties

Best Integer Bounds

	Upper	Lower
smd'Confirm_Settings 1	1	0
smd'settime 1	1	0
smd'setvolume 1	1	0

Best Upper Multi-set Bounds

smd'Confirm_Settings 1
1`{VOLUME=4,TIME=2,RATE=2}
smd'settime 1
1`{VOLUME=4,TIME=2,RATE=0}
smd'setvolume 1
1`{VOLUME=4,TIME=2,RATE=0}

Best Lower Multi-set Bounds

smd'Confirm_Settings 1
empty
smd'settime 1
empty
smd'setvolume 1
empty

Home Properties

Home Markings

[3]

Liveness Properties

Dead Markings

[3]

Dead Transition Instances

None

```
  Live Transition Instances
     None


 Fairness Properties
-------------------------------------------------------------------------

  Impartial Transition Instances
     smd'I_settime 1
     smd'I_setvolume 1

  Fair Transition Instances
     None

  Just Transition Instances
     smd'I_confirmsettings 1

  Transition Instances with No Fairness
     None
```

A.2 State Space Report of user interface and interaction of Simple Infusion Pump

CPN Tools state space report for:
/cygdrive/C/Users/sj113/Desktop/FINAL MODELS/simple infusion pump/PIMSIP/PIMSIP.cpn
Report generated: Sat Jul 27 11:20:36 2019

Statistics

State Space
- Nodes: 6
- Arcs: 9
- Secs: 0
- Status: Full

Scc Graph
- Nodes: 2
- Arcs: 1
- Secs: 0

Boundedness Properties

Best Integer Bounds

	Upper	Lower
confirmrate'confirmrate 1	1	
	1	0
confirmrate'infuse 1	1	0
confirmrate'settime 1	1	0
info'info 1	1	0
info'init 1	1	0
info'setvolume 1	1	0
infuse'infuse 1	1	0
init'info 1	1	0
init'init 1	1	0
settime'confirmrate 1	1	0
settime'settime 1	1	0
settime'setvolume 1	1	0
setvolume'info 1	1	0
setvolume'settime 1	1	0
setvolume'setvolume 1	1	0

Best Upper Multi-set Bounds

confirmrate'confirmrate 1

```
1`[(Display,Responder,[S_setrate]),(NoButton,ActCtrl,[I_settime]),(YesBut
ton,ActCtrl,[I_infuse]),(InfoButton,ActCtrl,[]),(MinusButton,ActCtrl,[]),
(PlusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
```

confirmrate'infuse 1

```
1`[(Display,Responder,[S_displayinfusingmsg]),(NoButton,ActCtrl,[]),(YesB
utton,ActCtrl,[]),(InfoButton,ActCtrl,[]),(MinusButton,ActCtrl,[]),(PlusB
utton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
```

confirmrate'settime 1

```
1`[(Display,Responder,[S_decreasetime,S_increasetime]),(NoButton,ActCtrl,
[I_setvolume]),(YesButton,ActCtrl,[S_settime,I_confirmrate]),(InfoButton,
```

```
ActCtrl,[]),(MinusButton,ActCtrl,[S_decreasetime]),(PlusButton,ActCtrl,[S
_increasetime]),(OnOffButton,ActCtrl,[])]
     info'info 1
1`[(Display,Responder,[S_displaybatterylife]),(NoButton,ActCtrl,[I_init])
,(YesButton,ActCtrl,[I_setvolume]),(InfoButton,ActCtrl,[]),(MinusButton,A
ctCtrl,[]),(PlusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     info'init 1
1`[(Display,Responder,[S_displaystartmessage]),(NoButton,ActCtrl,[]),(Yes
Button,ActCtrl,[]),(InfoButton,ActCtrl,[I_info]),(MinusButton,ActCtrl,[])
,(PlusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     info'setvolume 1
1`[(Display,Responder,[S_decreasevolume,S_increasevolume]),(NoButton,ActC
trl,[I_info]),(YesButton,ActCtrl,[I_settime,S_setvolume]),(InfoButton,Act
Ctrl,[]),(MinusButton,ActCtrl,[S_decreasevolume]),(PlusButton,ActCtrl,[S_
increasevolume]),(OnOffButton,ActCtrl,[])]
     infuse'infuse 1
1`[(Display,Responder,[S_displayinfusingmsg]),(NoButton,ActCtrl,[]),(YesB
utton,ActCtrl,[]),(InfoButton,ActCtrl,[]),(MinusButton,ActCtrl,[]),(PlusB
utton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     init'info 1
1`[(Display,Responder,[S_displaybatterylife]),(NoButton,ActCtrl,[I_init])
,(YesButton,ActCtrl,[I_setvolume]),(InfoButton,ActCtrl,[]),(MinusButton,A
ctCtrl,[]),(PlusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     init'init 1
1`[(Display,Responder,[S_displaystartmessage]),(NoButton,ActCtrl,[]),(Yes
Button,ActCtrl,[]),(InfoButton,ActCtrl,[I_info]),(MinusButton,ActCtrl,[])
,(PlusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     settime'confirmrate 1

1`[(Display,Responder,[S_setrate]),(NoButton,ActCtrl,[I_settime]),(YesBut
ton,ActCtrl,[I_infuse]),(InfoButton,ActCtrl,[]),(MinusButton,ActCtrl,[]),
(PlusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     settime'settime 1
1`[(Display,Responder,[S_decreasetime,S_increasetime]),(NoButton,ActCtrl,
[I_setvolume]),(YesButton,ActCtrl,[S_settime,I_confirmrate]),(InfoButton,
ActCtrl,[]),(MinusButton,ActCtrl,[S_decreasetime]),(PlusButton,ActCtrl,[S
_increasetime]),(OnOffButton,ActCtrl,[])]
     settime'setvolume 1
1`[(Display,Responder,[S_decreasevolume,S_increasevolume]),(NoButton,ActC
trl,[I_info]),(YesButton,ActCtrl,[I_settime,S_setvolume]),(InfoButton,Act
Ctrl,[]),(MinusButton,ActCtrl,[S_decreasevolume]),(PlusButton,ActCtrl,[S_
increasevolume]),(OnOffButton,ActCtrl,[])]
     setvolume'info 1
1`[(Display,Responder,[S_displaybatterylife]),(NoButton,ActCtrl,[I_init])
,(YesButton,ActCtrl,[I_setvolume]),(InfoButton,ActCtrl,[]),(MinusButton,A
ctCtrl,[]),(PlusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     setvolume'settime 1
1`[(Display,Responder,[S_decreasetime,S_increasetime]),(NoButton,ActCtrl,
[I_setvolume]),(YesButton,ActCtrl,[S_settime,I_confirmrate]),(InfoButton,
ActCtrl,[]),(MinusButton,ActCtrl,[S_decreasetime]),(PlusButton,ActCtrl,[S
_increasetime]),(OnOffButton,ActCtrl,[])]
     setvolume'setvolume 1

1`[(Display,Responder,[S_decreasevolume,S_increasevolume]),(NoButton,ActC
trl,[I_info]),(YesButton,ActCtrl,[I_settime,S_setvolume]),(InfoButton,Act
Ctrl,[]),(MinusButton,ActCtrl,[S_decreasevolume]),(PlusButton,ActCtrl,[S_
increasevolume]),(OnOffButton,ActCtrl,[])]
```

```
  Best Lower Multi-set Bounds
     confirmrate'confirmrate 1
                          empty
     confirmrate'infuse 1
                          empty
     confirmrate'settime 1
                          empty
     info'info 1          empty
     info'init 1          empty
     info'setvolume 1     empty
     infuse'infuse 1      empty
     init'info 1          empty
     init'init 1          empty
     settime'confirmrate 1
                          empty
     settime'settime 1    empty
     settime'setvolume 1  empty
     setvolume'info 1     empty
     setvolume'settime 1  empty
     setvolume'setvolume 1
                          empty


 Home Properties
-------------------------------------------------------------------------

  Home Markings
     [6]


 Liveness Properties
-------------------------------------------------------------------------

  Dead Markings
     [6]

  Dead Transition Instances
     None

  Live Transition Instances
     None


 Fairness Properties
-------------------------------------------------------------------------

  Impartial Transition Instances
     None

  Fair Transition Instances
     confirmrate'I_settime 1
     init'I_info 1

  Just Transition Instances
     confirmrate'I_infuse 1
     info'I_init 1
     info'I_setvolume 1
     settime'I_confirmrate 1
```

```
   settime'I_setvolume 1
   setvolume'I_info 1
   setvolume'I_settime 1

Transition Instances with No Fairness
   None
```

A.3 State Space Report of user interface, interaction and functionality of Simple Infusion Pump with Z

CPN Tools state space report for:
/cygdrive/C/Users/Dell/Downloads/ZSIPssreport.cpn
Report generated: Sun Aug 18 15:58:15 2019

Statistics

State Space

Nodes: 654
Arcs: 1399
Secs: 0
Status: Full

Scc Graph

Nodes: 12
Arcs: 21
Secs: 0

Boundedness Properties

Best Integer Bounds

	Upper	Lower
confirmrate'Z 1	1	1
confirmrate'confirmrate 1	1	0
confirmrate'infuse 1	1	0
confirmrate'settime 1	1	0
info'Z 1	1	1
info'info 1	1	0
info'init 1	1	0
info'setvolume 1	1	0
infuse'Z 1	1	1
infuse'infuse 1	1	0
init'Z 1	1	1
init'info 1	1	0
init'init 1	1	0
settime'Z 1	1	1
settime'confirmrate 1	1	0
settime'settime 1	1	0
settime'setvolume 1	1	0
setvolume'Z 1	1	1
setvolume'info 1	1	0
setvolume'settime 1	1	0
setvolume'setvolume 1	1	0

Best Upper Multi-set Bounds

```
     confirmrate'Z 1
1`{battery=sufficient,timer=0,volume=0,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
```

```
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
```

1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=1,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++

```
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=2,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=3,tim
eleft=1,infusing=Yes}++
```

1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++

1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=1,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=1,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++

1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++

```
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=1,tim
eleft=3,infusing=Yes}
     confirmrate'confirmrate 1

1`[(Display,Responder,[S_setrate]),(NoButton,ActCtrl,[I_settime]),(Yes
```

```
Button,ActCtrl,[I_infuse]),(InfoButton,ActCtrl,[]),(MinusButton,ActCtr
l,[]),(PlusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     confirmrate'infuse 1

1`[(Display,ActCtrl,[S_displayinfusingmsg]),(NoButton,ActCtrl,[]),(Yes
Button,ActCtrl,[]),(InfoButton,ActCtrl,[]),(MinusButton,ActCtrl,[]),(P
lusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     confirmrate'settime 1

1`[(Display,Responder,[S_decreasetime,S_increasetime]),(NoButton,ActCt
rl,[I_setvolume]),(YesButton,ActCtrl,[S_settime,I_confirmrate]),(InfoB
utton,ActCtrl,[]),(MinusButton,ActCtrl,[S_decreasetime]),(PlusButton,A
ctCtrl,[S_increasetime]),(OnOffButton,ActCtrl,[])]
     info'Z 1
1`{battery=sufficient,timer=0,volume=0,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
```

1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++

1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=1,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++

```
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=2,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
```

1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=3,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++

1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=1,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++

1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=1,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++

1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++

```
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=1,tim
eleft=3,infusing=Yes}
     info'info 1
1`[(Display,Responder,[S_displaybatterylife]),(NoButton,ActCtrl,[I_ini
t]),(YesButton,ActCtrl,[I_setvolume]),(InfoButton,ActCtrl,[]),(MinusBu
tton,ActCtrl,[]),(PlusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     info'init 1
1`[(Display,Responder,[S_displaystartmessage]),(NoButton,ActCtrl,[]),(
YesButton,ActCtrl,[]),(InfoButton,ActCtrl,[I_info]),(MinusButton,ActCt
rl,[]),(PlusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     info'setvolume 1
1`[(Display,Responder,[S_decreasevolume,S_increasevolume]),(NoButton,A
ctCtrl,[I_info]),(YesButton,ActCtrl,[I_settime,S_setvolume]),(InfoButt
on,ActCtrl,[]),(MinusButton,ActCtrl,[S_decreasevolume]),(PlusButton,Ac
tCtrl,[S_increasevolume]),(OnOffButton,ActCtrl,[])]
     infuse'Z 1
1`{battery=sufficient,timer=0,volume=0,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
```

1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++

1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++

1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=1,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=2,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++

1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=3,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++

1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=1,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++

1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=1,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++

```
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
```

```
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=1,tim
eleft=3,infusing=Yes}
     infuse'infuse 1
1`[(Display,ActCtrl,[S_displayinfusingmsg]),(NoButton,ActCtrl,[]),(Yes
Button,ActCtrl,[]),(InfoButton,ActCtrl,[]),(MinusButton,ActCtrl,[]),(P
lusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     init'Z 1
1`{battery=sufficient,timer=0,volume=0,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
```

```
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
```

1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++

1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=1,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=2,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++

1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=3,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++

1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=1,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++

1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=1,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++

1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++

```
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=1,tim
eleft=3,infusing=Yes}
     init'info 1
1`[(Display,Responder,[S_displaybatterylife]),(NoButton,ActCtrl,[I_ini
t]),(YesButton,ActCtrl,[I_setvolume]),(InfoButton,ActCtrl,[]),(MinusBu
tton,ActCtrl,[]),(PlusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     init'init 1
1`[(Display,Responder,[S_displaystartmessage]),(NoButton,ActCtrl,[]),(
YesButton,ActCtrl,[]),(InfoButton,ActCtrl,[I_info]),(MinusButton,ActCt
rl,[]),(PlusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     settime'Z 1
1`{battery=sufficient,timer=0,volume=0,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
```

1`{battery=sufficient,timer=0,volume=1,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++

1`{battery=sufficient,timer=0,volume=3,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++

```
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=1,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=2,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
```

1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=3,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++

1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=1,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++

1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=1,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++

1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++

```
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=1,tim
eleft=3,infusing=Yes}
     settime'confirmrate 1

1`[(Display,Responder,[S_setrate]),(NoButton,ActCtrl,[I_settime]),(Yes
Button,ActCtrl,[I_infuse]),(InfoButton,ActCtrl,[]),(MinusButton,ActCtr
l,[]),(PlusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     settime'settime 1
1`[(Display,Responder,[S_decreasetime,S_increasetime]),(NoButton,ActCt
rl,[I_setvolume]),(YesButton,ActCtrl,[S_settime,I_confirmrate]),(InfoB
utton,ActCtrl,[]),(MinusButton,ActCtrl,[S_decreasetime]),(PlusButton,A
ctCtrl,[S_increasetime]),(OnOffButton,ActCtrl,[])]
     settime'setvolume 1
1`[(Display,Responder,[S_decreasevolume,S_increasevolume]),(NoButton,A
ctCtrl,[I_info]),(YesButton,ActCtrl,[I_settime,S_setvolume]),(InfoButt
on,ActCtrl,[]),(MinusButton,ActCtrl,[S_decreasevolume]),(PlusButton,Ac
tCtrl,[S_increasevolume]),(OnOffButton,ActCtrl,[])]
     setvolume'Z 1
1`{battery=sufficient,timer=0,volume=0,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
```

1`{battery=sufficient,timer=0,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++

1`{battery=sufficient,timer=0,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=0,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=0,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++

1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=1,infusionrate=1,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=2,infusionrate=2,tim
eleft=1,infusing=Yes}++

1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=1,volume=3,volumeleft=3,infusionrate=3,tim
eleft=1,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++

1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=2,infusionrate=1,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++

1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=2,volume=3,volumeleft=3,infusionrate=1,tim
eleft=2,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++

```
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=0,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=1,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=Yes}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
```

```
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=2,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=1,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=2,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=0,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=1,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=2,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=0,tim
eleft=3,infusing=No}++
1`{battery=sufficient,timer=3,volume=3,volumeleft=3,infusionrate=1,tim
eleft=3,infusing=Yes}
     setvolume'info 1
1`[(Display,Responder,[S_displaybatterylife]),(NoButton,ActCtrl,[I_ini
t]),(YesButton,ActCtrl,[I_setvolume]),(InfoButton,ActCtrl,[]),(MinusBu
tton,ActCtrl,[]),(PlusButton,ActCtrl,[]),(OnOffButton,ActCtrl,[])]
     setvolume'settime 1
1`[(Display,Responder,[S_decreasetime,S_increasetime]),(NoButton,ActCt
rl,[I_setvolume]),(YesButton,ActCtrl,[S_settime,I_confirmrate]),(InfoB
utton,ActCtrl,[]),(MinusButton,ActCtrl,[S_decreasetime]),(PlusButton,A
ctCtrl,[S_increasetime]),(OnOffButton,ActCtrl,[])]
     setvolume'setvolume 1

1`[(Display,Responder,[S_decreasevolume,S_increasevolume]),(NoButton,A
ctCtrl,[I_info]),(YesButton,ActCtrl,[I_settime,S_setvolume]),(InfoButt
on,ActCtrl,[]),(MinusButton,ActCtrl,[S_decreasevolume]),(PlusButton,Ac
tCtrl,[S_increasevolume]),(OnOffButton,ActCtrl,[])]

  Best Lower Multi-set Bounds
     confirmrate'Z 1     empty
     confirmrate'confirmrate 1
                         empty
     confirmrate'infuse 1
                         empty
```

```
    confirmrate'settime 1
                        empty
    info'Z 1            empty
    info'info 1         empty
    info'init 1         empty
    info'setvolume 1    empty
    infuse'Z 1          empty
    infuse'infuse 1     empty
    init'Z 1            empty
    init'info 1         empty
    init'init 1         empty
    settime'Z 1         empty
    settime'confirmrate 1
                        empty
    settime'settime 1   empty
    settime'setvolume 1 empty
    setvolume'Z 1       empty
    setvolume'info 1    empty
    setvolume'settime 1 empty
    setvolume'setvolume 1
                        empty


 Home Properties
-------------------------------------------------------------------------
--

  Home Markings
     None


 Liveness Properties
-------------------------------------------------------------------------
--

  Dead Markings
     9 [90,64,50,263,26,...]

  Dead Transition Instances
     None

  Live Transition Instances
     None


 Fairness Properties
-------------------------------------------------------------------------
--

  Impartial Transition Instances
     None
```

```
Fair Transition Instances
   confirmrate'I_settime 1
   init'I_info 1

Just Transition Instances
   confirmrate'I_infuse_S_SetRate 1
   info'I_init 1
   info'I_setvolume 1
   settime'S_decreasetime 1
   settime'S_increasetime 1
   setvolume'S_DecreaseVolume 1
   setvolume'S_IncreaseVolume 1

Transition Instances with No Fairness
   settime'I_confirmrate__S_SetTime 1
   settime'I_setvolume 1
   setvolume'I_info 1
   setvolume'I_settime__S_setvolume 1
```

Appendix B

Reachability Algorithm

B.1 Reachability Algotithm

B.1.1 Definition of a Reachability Predicate

Definition B.1.1. Let A denote the set of nodes that satisfy the predicate function A, and let B be the set of nodes that satisfy the predicate function B. We define a reachability predicate:

$$\forall\, M \in M_A, \exists\, M' \in M_B : M' \in [M\rangle$$

B.1.2 The Reachable Algorithm

This section describes a reachability algorithm that outputs two lists and a flag, based on whether or not the reachability predicate defined above is satisfied. The Reachable function is a recursive function implemented in CPN-ML and is explained as follows. Reachable is invoked with the following parameters:

- *sl* is a list of SCC graph nodes.
- *PredA* is the predicate function A.
- *PredB* is the predicate function B.
- *visited* is the list of pairs, (s, col), where s is a visited node and *col* is an indication whether it is a B-node or not.

- *rejected* is a list of A-nodes, which cannot reach any B-node.
- *gmark* is a global flag indicating that a B-node can be reached from the current visited node.

The function Reachable returns the following results:

- A list of pairs, which are formed by the node number and an indication if it is a B-node (1) or not (0).
- A list of nodes, which have been rejected. In other words, they are A-nodes, which cannot reach any B-node.
- A global flag (gmark) defined above.

Reachable invokes the following functions:

- *ChangeVisitorColour(n,nl,col)* function updates the mark of a given node n to a given mark *col* if it is in the list of pairs.

```
Fun ChangeVisitorColour (s, nil,col)= []|
ChangeVisitorColour (s, (s1,c)::sl,col) =
if s=s1
then (s1,col)::sl
else
(s1,c)::ChangeVisitorColour (s,sl,col);
```

- *ColouredVisitor(n,nl)* function returns *true* if the node n has been already marked.

```
Fun ColouredVisitor (n, nil)= false|
ColouredVisitor (n, (n1,c)::nl) =
if n=n1
then c=1
else
ColouredVisitor (n,nl);
```

- The function $next(s)$ returns the successors of the node s. The node s is a SCC node.

```
fun next (s) = SccOutNodes s;
```

- The function *PairListMember*(s, sl) returns true if s is a member of the list of pairs sl.

```
fun PairListMember (s, nil)=false|
PairListMember (s, (s1,c)::sl) =
if s=s1
then true
else
PairListMember (s,sl);
```

The following values are defined in the function Reachable:

- *val vis* = *PairListMember*$(s,$ *visited*$)$. The variable *vis* will be *true* if the node s has been visited.

- *val (v,r,m)* =
  if vis then (visited,rejected,gmark)
  else
  let
  val (v1,r1,m1) = Reachable (next (s),predA,predB,visited,rejected,cmark);
  in
  if predB s then ((s,1)::v1,r1,1)
  else
  ((s,m1)::v1,r1,m1)
  end;
  If the current node has been visited, the successors are not visited; otherwise they are. The variables: $v1$ has the visited nodes, $r1$ the rejected nodes and $m1$ the global mark indicating if there is any B-node reachable from the current visited node s. If *predB* is true for s, it marks the node as B-node and inserts it in the visited list with the corresponding marking. Otherwise, the predicate function returns *false* and marks the node according to the global mark, ml.

- *val (v2,r2,m2) = Reachable(xs,predA,predB,v,r, gmark)*

  The rest of the nodes of the list, xs, are visited. The lists of visited nodes, v, and rejected nodes, r, have been updated while visiting the successors of the current node, s (as indicated previously).

The body of the Reachable function is as follows:

```
if vis then (v2,r2,if ColouredVisitor(s,v2) then 1 else m2)
else
if predB x then (ChangeVisitorColour(x,v2,1),r2,1)
else
if predA x then
if m=1 then (ChangeVisitorColour(x,v2,1),r2, m)
else
(ChangeVisitorColour(x,v2,0),x::r2, m2)
else
(ChangeVisitorColour(x,v2,if m=1 then 1 else 0),r2,if m=1 then m else m2)
end;
```

The function checks if the current node has been visited. If that is the case, no further analysis is required. The function returns the global mark, which has been set based on the visit to the non-successor nodes or the mark assigned to the current node. If the node has not been visited and is a B-node, the function updates its mark and the global mark to 1. Otherwise, if the node is a A-node, it marks the current node as a B-node and returns a global flag of 1 if the mark, m, returned after visiting the successors of the current node is 1. If m is not 1, the current node is marked as a non B-node and the global flag is set based on the values of the global flag returned after visited the successors of the current node and the other nodes. Also, the current node is included in the rejected list, which means that it cannot reach a B-node.

If the current node is neither a B-node nor an A-node, it is marked based on the mark returned after visiting its successors (i.e. m). The global flag is set based on the values of the global flag returned after visiting the successors of the current node and the other nodes.

Appendix C

Z Specification of Niki T34 Syringe Driver

C.1 Z specification of the Niki T34 syringe driver

There are three definitions that introduce three types: *YesNo*, *SyringeBrand*, and *CHAR*. *YesNo* has two values in it, *SyringeBrand* has one value in it and *CHAR* has fifteen values in it.

$$YesNo ::= yes \mid no$$

$$SyringeBrand ::= BDPlastipak$$

$$CHAR ::= CheckPlungerSensor \mid PumpPausedForTooLong\_Beep \mid PumpPausedForTooLong\_NoBeep \mid Volume \mid Enter\_Code \mid Incorrect\_Code \mid Code\_Accepted \mid StartInfusion? \mid ConfirmSettings\_PressYes \mid PumpDelivering\_10mlBDPlastipak \mid PumpStopped\_PressYesToResume \mid Resume? \mid ConfirmToResume \mid InfusionSummary \mid SetUpCode\_Accepted$$

The axiomatic definition written below introduces six global constants *PerCent*, *millilitres*, *millimeters*, *hours*, *minutes*, and *millilitresperhour* as powersets of natural numbers and the predicate part specifies a constraint on their values

$$
\begin{array}{|l}
PerCent : \mathbb{P}\,\mathbb{N} \\
millilitres : \mathbb{P}\,\mathbb{N} \\
millimeters : \mathbb{P}\,\mathbb{N} \\
hours : \mathbb{P}\,\mathbb{N} \\
minutes : \mathbb{P}\,\mathbb{N} \\
millilitresperhour : \mathbb{P}\,\mathbb{N} \\
\hline
millilitres = 0 \,..\, 100 \\
PerCent = 0 \,..\, 100 \\
millimeters = 0 \,..\, 10 \\
hours = 0 \,..\, 24 \\
minutes = 0 \,..\, 59 \\
millilitresperhour = 0 \,..\, 100
\end{array}
$$

The $T34$ schema defines the state space of the model. It says that in each state in the state space there are fifteen observations: *BatteryCharge* of type *PerCent*, the six observations: *KeyPadLocked*, *ProgramLocked*, *TechMenuLocked*, *SyringeOk*, *BarrelOK*, *CollarOK*, *PlungerOK* and *SystemReady* of type *YesNo*, *Brand* of type *SyringeBrand*, *SyringeSize*, *VTBI* and *VolumeLeft* of type *millilitres*, *PlungerPosition* of type *millimeters*, *Hours* of type *hours*, *Minutes* of type *minutes*, *InfusionRate* of type *millilitresperhour*.

$T34$

$BatteryCharge : PerCent$

$KeyPadLocked : YesNo$

$ProgramLocked : YesNo$

$TechMenuLocked : YesNo$

$Brand : SyringeBrand$

$SyringeSize : millilitres$

$VolumeLeft : millilitres$

$PlungerPosition : millimeters$

$SyringeOK : YesNo$

$BarrelOK, CollarOK, PlungerOK : YesNo$

$SystemReady : YesNo$

$VTBI : millilitres$

$Hours : hours$

$Minutes : minutes$

$InfusionRate : millilitresperhour$

In the initial state of the specification, the initial values of the T34 syringe driver observations are as below:

Init

$T34$

$BatteryCharge = 90$

$KeyPadLocked = no$

$ProgramLocked = yes$

$TechMenuLocked = yes$

$Brand = BDPlastipak$

$SyringeSize = 10$

$VolumeLeft = 0$

$PlungerPosition = 10$

$SyringeOK = yes$

$BarrelOK = CollarOK = PlungerOK = yes$

$SystemReady = no$

$VTBI = 10$

$Hours = 0$

$Minutes = 0$

$InfusionRate = 0$

The *SyringeWarnings* schema displays the output *CheckPlungerSensor* on the screen of the device. None of the other values of the state space schema change as represented by the declaration $\Xi T34$.

SyringeWarnings

$\Xi T34$

$display! : CHAR$

$display! = CheckPlungerSensor$

The schemas *MoveActuatorFwd*, *ArmWarning* and *MoveActuatorBwd* are related to hardware movements which do not make changes to the values of the state space schema, as represented by the declaration $\Xi T34$. The Z spec-

ifications for these S-behaviours are not modelled to keep the size of the state space of the CPN model small.

MoveActuatorFwd

$\Xi T34$

ArmWarning

$\Xi T34$

MoveActuatorBwd

$\Xi T34$

The *SyringeDisplay* schema displays information about the size of the syringe and its brand. None of the other values of the state space schema change as represented by the declaration $\Xi T34$.

SyringeDisplay

$\Xi T34$

$display! : millilitres \times SyringeBrand$

$display! = (SyringeSize, Brand)$

The schemas *ScrollSyringeList, InfoList, ScrollInfoListUp* and *ScrollInfoListDown* do not make changes to the values of the state space schema as represented by the declaration $\Xi T34$. In our case we are just considering one type of syringe, i.e., BDPlastipak, so the schema *SelectSyringe* also does not make any changes to the values of the state space schema. The Z specification for these S-behaviours are not modelled here to keep the size of the state space of a CPN model small.

ScrollSyringeList

$\Xi T34$

SelectSyringe

$\Xi T34$

InfoList

$\Xi T34$

ScrollInfoListUp

$\Xi T34$

ScrollInfoListDown

$\Xi T34$

If the user gives a long press to the *InfoSK* button on the device, the keypad gets locked or unlocked. This behaviour changes the value of the observation *KP* from *no* to *yes* and vice-versa as specified by the *keypadLock* schema.

keypadLock

$\Delta T34$

$BatteryCharge' = BatteryCharge$

$KeyPadLocked = no \Rightarrow KeyPadLocked' = yes$

$KeyPadLocked = yes \Rightarrow KeyPadLocked' = no$

$ProgramLocked' = ProgramLocked$

$TechMenuLocked' = TechMenuLocked$

$Brand' = Brand$

$SyringeSize' = SyringeSize$

$VolumeLeft' = VTBI'$

$PlungerPosition' = PlungerPosition$

$SyringeOK' = SyringeOK$

$BarrelOK' = BarrelOK$

$CollarOK' = CollarOK$

$PlungerOK' = PlungerOK$

$SystemReady' = SystemReady$

$VTBI' = VTBI$

$Hours' = Hours$

$Minutes' = Minutes$

$InfusionRate' = InfusionRate$

The *BatteryLevel* schema displays the current status of the battery as a percentage.

BatteryLevel

$\Xi T34$

$display! : PerCent$

$display! = BatteryCharge$

The *TimeOut* schema displays the message *Pump paused for too long* on

the screen of the device and starts the beep. None of the other values of the state space schema change as represented by the declaration $\Xi T34$.

TimeOut

$\Xi T34$

$display! : CHAR$

$display! = PumpPausedForTooLong\_Beep$

The *StopAlarm* schema displays the message *Pump paused for too long* on the screen of the device and stops the beep. None of the other values of the state space schema change as represented by the declaration $\Xi T34$.

StopAlarm

$\Xi T34$

$display! : CHAR$

$display! = PumpPausedForTooLong\_NoBeep$

The schema *CurrentVolume* displays the current value of the observation *VTBI* on the screen.

The schema *CurrentVolume*

CurrentVolume

$\Xi T34$

$display! : CHAR \times millilitres$

$display! = (Volume, VTBI)$

The schema *CurrentSyringe* displays the information about the size of the syringe and its brand. None of the other values of the state space schema change as represented by the declaration $\Xi T34$.

CurrentSyringe

$\Xi T34$

$display! : millilitres \times SyringeBrand$

$display! = (SyringeSize, Brand)$

The operations *IncreaseVTBI* and *DecreaseVTBI* allow the user to change the value of a dose to be infused. With every press of the buttons *UpPlusSK* and *DownMinusSK*, the value of the observation VTBI increases and decrease by one respectively. The maximum value of VTBI can be ten and the value of VTBI must be greater than zero.

IncreaseVTBI

$\Delta T34$

$BatteryCharge' = BatteryCharge$

$KeyPadLocked' = KeyPadLocked$

$ProgramLocked' = ProgramLocked$

$TechMenuLocked' = TechMenuLocked$

$Brand' = Brand$

$SyringeSize' = SyringeSize$

$VolumeLeft' = VolumeLeft$

$PlungerPosition' = PlungerPosition$

$SyringeOK' = SyringeOK$

$BarrelOK' = BarrelOK$

$CollarOK' = CollarOK$

$PlungerOK' = PlungerOK$

$SystemReady' = SystemReady$

$VTBI \leq 10$

$VTBI' = VTBI + 1$

$Hours' = Hours$

$Minutes' = Minutes$

$InfusionRate' = InfusionRate$

DecreaseVTBI

$\Delta T34$

$BatteryCharge' = BatteryCharge$

$KeyPadLocked' = KeyPadLocked$

$ProgramLocked' = ProgramLocked$

$TechMenuLocked' = TechMenuLocked$

$Brand' = Brand$

$SyringeSize' = SyringeSize$

$VolumeLeft' = VolumeLeft$

$PlungerPosition' = PlungerPosition$

$SyringeOK' = SyringeOK$

$BarrelOK' = BarrelOK$

$CollarOK' = CollarOK$

$PlungerOK' = PlungerOK$

$SystemReady' = SystemReady$

$VTBI > 0$

$VTBI' = VTBI - 1$

$Hours' = Hours$

$Minutes' = Minutes$

$InfusionRate' = InfusionRate$

The *SetVTBI* schema sets the VTBI (volume to be infused) and changes the value of the *VolumeLeft* to the new value of the *VTBI*.

SetVTBI

$\Delta T34$

$BatteryCharge' = BatteryCharge$

$KeyPadLocked' = KeyPadLocked$

$ProgramLocked' = ProgramLocked$

$TechMenuLocked' = TechMenuLocked$

$Brand' = Brand$

$SyringeSize' = SyringeSize$

$VolumeLeft' = VTBI'$

$PlungerPosition' = PlungerPosition$

$SyringeOK' = SyringeOK$

$BarrelOK' = BarrelOK$

$CollarOK' = CollarOK$

$PlungerOK' = PlungerOK$

$SystemReady' = SystemReady$

$VTBI > 1$

$VTBI' = VTBI$

$Hours' = Hours$

$Minutes' = Minutes$

$InfusionRate' = InfusionRate$

The *SelectDigit* schema displays *Enter_Code* on the screen of the device. This schema asks the user to enter the code by selecting the digits.

SelectDigit

$\Xi T34$

$display! : CHAR$

$display! = Enter\_Code$

If a user enters the wrong code then the *IncorrectCode* schema displays a

message *Incorrect_Code* on the screen. None of the other observations of the state space schema changes.

IncorrectCode

$\Xi T34$

$display! : CHAR$

$display! = Incorrect\_Code$

If the code gets accepted, then the *RateCodeAccept* schema displays a message *Code_Accepted* on the screen.

RateCodeAccept

$\Xi T34$

$display! : CHAR$

$display! = Code\_Accepted$

The operations *IncreaseDuration* and *DecreaseDuration* allow the user to change the value of the observations *Hours* and *Minutes*.

IncreaseDuration

$\Delta T34$

$BatteryCharge' = BatteryCharge$

$KeyPadLocked' = KeyPadLocked$

$ProgramLocked' = ProgramLocked$

$TechMenuLocked' = TechMenuLocked$

$Brand' = Brand$

$SyringeSize' = SyringeSize$

$VolumeLeft' = VolumeLeft$

$PlungerPosition' = PlungerPosition$

$SyringeOK' = SyringeOK$

$BarrelOK' = BarrelOK$

$CollarOK' = CollarOK$

$PlungerOK' = PlungerOK$

$SystemReady' = SystemReady$

$VTBI' = VTBI$

$Minutes \leq 58 \Rightarrow (Minutes' = Minutes + 1 \land Hours' = Hours)$

$Minutes = 59 \Rightarrow$

$(Minutes' = 0 \land ((Hours \leq 23 \Rightarrow Hours' = Hours + 1)$

$\land (Hours = 24 \Rightarrow Hours' = 0)))$

$InfusionRate' = InfusionRate$

DecreaseDuration

$\Delta T34$

$BatteryCharge' = BatteryCharge$

$KeyPadLocked' = KeyPadLocked$

$ProgramLocked' = ProgramLocked$

$TechMenuLocked' = TechMenuLocked$

$Brand' = Brand$

$SyringeSize' = SyringeSize$

$VolumeLeft' = VolumeLeft$

$PlungerPosition' = PlungerPosition$

$SyringeOK' = SyringeOK$

$BarrelOK' = BarrelOK$

$CollarOK' = CollarOK$

$PlungerOK' = PlungerOK$

$SystemReady' = SystemReady$

$VTBI' = VTBI$

$(Minutes = 0 \land Hours = 0) \Rightarrow (Minutes' = 0 \land Hours' = 0)$

$Minutes \geq 1 \Rightarrow (Minutes' = Minutes - 1 \land Hours' = Hours)$

$Minutes = 0 \Rightarrow (Minutes' = 59 \land ((Hours \leq 1 \Rightarrow Hours' = Hours - 1)$

$\land (Hours = 0 \Rightarrow Hours' = 0)))$

$InfusionRate' = InfusionRate$

The operation schema *SetNewDuration* sets the duration for infusion. This schema also calculates and changes the value of the *InfusionRate* observation. Value of the observations *Hours* and *Minutes* must be greater than one.

SetNewDuration

$\Delta T34$

$BatteryCharge' = BatteryCharge$

$KeyPadLocked' = KeyPadLocked$

$ProgramLocked' = ProgramLocked$

$TechMenuLocked' = TechMenuLocked$

$Brand' = Brand$

$SyringeSize' = SyringeSize$

$VolumeLeft' = VolumeLeft$

$PlungerPosition' = PlungerPosition$

$SyringeOK' = SyringeOK$

$BarrelOK' = BarrelOK$

$CollarOK' = CollarOK$

$PlungerOK' = PlungerOK$

$SystemReady' = SystemReady$

$VTBI' = VTBI$

$Hours > 0 \lor Minutes > 0$

$InfusionRate' = (60 * (VTBI \mathsf{div} ((60 * Hours) + Minutes)))$

The *CurrentRate* schema displays the current value of the observation *InfusionRate*.

CurrentRate

$\Xi T34$

$display! : millilitresperhour$

$display! = InfusionRate$

SaveSettings

$\Xi T34$

The schema *ConfirmMsg* displays the message *ConfirmSettings_PressYes* to ask user to the confirm the settings by pressing the *YesSK* button. None of the other values of the state space schema change as represented by the declaration $\Xi T34$.

ConfirmMsg

$$\Xi T34$$
$$display! : CHAR$$

$$display! = ConfirmSettings\_PressYes$$

The *CurrentVTBI* schema displays the current value of the observation *VTBI*.

CurrentVTBI

$$\Xi T34$$
$$display! : millilitres$$

$$display! = VTBI$$

The *CurrentDuration* schema displays the current value of the observations *Hours* and *Minutes*.

CurrentDuration

$$\Xi T34$$
$$display! : hours \times minutes$$

$$display! = (Hours, Minutes)$$

The *StartInfusingMsg* schema displays the message *StartInfusion*? on the screen to ask user if they want to start infusion.

StartInfusingMsg

$\Xi T34$

$display! : CHAR$

$display! = StartInfusion?$

Infusing schema changes the value of the observation *SystemReady* from *no* to *yes*.

Infusing

$\Delta T34$

$BatteryCharge' = BatteryCharge$

$KeyPadLocked' = KeyPadLocked$

$ProgramLocked' = ProgramLocked$

$TechMenuLocked' = TechMenuLocked$

$Brand' = Brand$

$SyringeSize' = SyringeSize$

$VolumeLeft' = VolumeLeft$

$PlungerPosition' = PlungerPosition$

$SyringeOK' = SyringeOK$

$BarrelOK' = BarrelOK$

$CollarOK' = CollarOK$

$PlungerOK' = PlungerOK$

$SystemReady = no$

$SystemReady' = yes$

$VTBI' = VTBI$

$Hours' = Hours$

$Minutes' = Minutes$

$InfusionRate' = InfusionRate$

The *TimeRemaining* schema displays the current value of the observations

Hours and *Minutes*.

TimeRemaining

$\Xi T34$

$display! : hours \times minutes$

$display! = (Hours, Minutes)$

The *DeliveringMsg* schema displays the message *PumpDelivering_10ml BDPlastipak* on the screen.

DeliveringMsg

$\Xi T34$

$display! : CHAR$

$display! = PumpDelivering\_10mlBDPlastipak$

The *Pause* schema pauses the infusion and change the value of the observation *SytemReady* from *yes* to *no*.

Pause

$\Delta T34$

$BatteryCharge' = BatteryCharge$

$KeyPadLocked' = KeyPadLocked$

$ProgramLocked' = ProgramLocked$

$TechMenuLocked' = TechMenuLocked$

$Brand' = Brand$

$SyringeSize' = SyringeSize$

$VolumeLeft' = VolumeLeft$

$PlungerPosition' = PlungerPosition$

$SyringeOK' = SyringeOK$

$BarrelOK' = BarrelOK$

$CollarOK' = CollarOK$

$PlungerOK' = PlungerOK$

$SystemReady = yes$

$SystemReady' = no$

$VTBI' = VTBI$

$Hours' = Hours$

$Minutes' = Minutes$

$InfusionRate' = InfusionRate$

The *PumpStoppedWarning* schema displays the message *PumpStopped_PressYesToResume* on the screen of the pump.

PumpStoppedWarning

$\Xi T34$

$display! : CHAR$

$display! = PumpStopped\_PressYesToResume$

The *ResumeInfusing* schema resumes the paused infusion and change the

value of the observation *SytemReady* from *no* to *yes*.

ResumeInfusing

$\Delta T34$

$BatteryCharge' = BatteryCharge$

$KeyPadLocked' = KeyPadLocked$

$ProgramLocked' = ProgramLocked$

$TechMenuLocked' = TechMenuLocked$

$Brand' = Brand$

$SyringeSize' = SyringeSize$

$VolumeLeft' = VolumeLeft$

$PlungerPosition' = PlungerPosition$

$SyringeOK' = SyringeOK$

$BarrelOK' = BarrelOK$

$CollarOK' = CollarOK$

$PlungerOK' = PlungerOK$

$SystemReady = no$

$SystemReady' = yes$

$VTBI' = VTBI$

$Hours' = Hours$

$Minutes' = Minutes$

$InfusionRate' = InfusionRate$

The schema *SyringeDisplay*2 displays *Resume*? on the screen to ask the user if they want to resume the infusion.

SyringeDisplay2

$\Xi T34$

$display! : CHAR$

$display! = Resume?$

The schema *ResumeConfirm* displays *ConfirmToResume* on the screen to ask the user to confirm that they want to resume the infusion.

ResumeConfirm

$\Xi T34$

$display! : CHAR$

$display! = ConfirmToResume$

The schema *SyringeStatusDisplay* displays the infusion summary on the screen.

SyringeStatusDisplay

$\Xi T34$

$display! : CHAR$

$display! = InfusionSummary$

The four schemas *EventLogData*, *EventInfoDisplay*, *NextEvent* and *PrevEvent* doe not make changes to the observations of the state space schema as represented by the declaration $\Xi T34$.

EventLogData

$\Xi T34$

EventInfoDisplay

$\Xi T34$

NextEvent

$\Xi T34$

PrevEvent

$\Xi T34$

The schema *SetUpCodeAccept* displays the message *SetUpCode_Accepted* on the screen of the pump.

SetUpCodeAccept

$\Xi T34$

$display! : CHAR$

$display! = SetUpCode\_Accepted$