An Automatic Transformation for Multi-Component Systems from ALIAS to mCRL2

Pan, Y.

Award date:
2020

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computer Science

An Automatic Transformation for Multi-Component Systems from ALIAS to mCRL2

Yuanteng Pan

Supervisors:
prof.dr.ir. Jan Friso Groote
dr.ir. Ramon Schiffelers
Thomas Neele MSc

Public

Eindhoven, 15th January 2020

Abstract

In modern industry, designing complex software systems in terms of small components is a means of developing reliable software that are easy to be verified and maintained. ASD, which is specially designed for modeling and verification, is used to assist in developing software. However, the verification system of ASD suffers from some limitations. For instance, ASD verification can check whether an ASD component correctly matches each used component according to its service specification. But it cannot check whether the behavior of the resulting system matches its desired purpose. Different from ASD, mCRL2, as a robust process algebraic model checker, is capable of handling such a verification.

Under this situation, transformations from ASD to ALIAS, and then from ALIAS to mCRL2 are proposed in this document. Then the transformed mCRL2 model is used for end-to-end property verification. Until now, such a transformation has been developed for a single component. However, the verification of a single component cannot check end-to-end properties. Furthermore, a transformation based on multi-component ASD systems has been come up. This transformation relies on manual work, which may lead to many errors when performing a large system transformation.

This thesis is dedicated to implementing an automatic transformation for multi-component systems from ALIAS to mCRL2. The transformation is achieved by the model to model transformation through QVTo, a domain-specific language to express model transformations.

We use several example models to perform validation. Some simple models are analyzed by inspecting their visualized state space obtained by the corresponding labelled transition systems manually. For a complex system, we utilize modal μ-calculus formulas to verify its properties. The verification results are in line with our expectations that the behaviors of the output mCRL2 models match the expected behaviors of the input ASD models. The verification indicates that the result of the automatic transformation for multi-component systems from ALIAS to mCRL2 is correct for checking models on this scale.

Contents

Abstract i

Contents ii

1 Introduction 1
- 1.1 Approach . . . 2
- 1.2 Related work . . . 2
- 1.3 Outline . . . 3

2 Background information 5
- 2.1 Analytical Software Design . . . 5
  - 2.1.1 Basic architecture in ASD . . . 5
  - 2.1.2 Specifications in ASD . . . 6
  - 2.1.3 Verification in ASD . . . 8
- 2.2 Single thread execution in ASD . . . 8
- 2.3 ALIAS . . . 8
- 2.4 mCRL2 . . . 9
- 2.5 M2M . . . 10
- 2.6 A meta-model example . . . 11

3 Implementation 13
- 3.1 From an ALIAS model to a combined mCRL2 model . . . 13
- 3.2 ALIAS language specification . . . 14
- 3.3 The specifications of communications in mCRL2 . . . 17
- 3.4 ALIAS to mCRL2 standard specifications . . . 19
  - 3.4.1 Sort . . . 19
  - 3.4.2 Action . . . 20
  - 3.4.3 Process . . . 21
  - 3.4.4 Initialization . . . 25
  - 3.4.5 The full transformation . . . 26

4 Validation 27
- 4.1 Validation method . . . 27
- 4.2 The kettle system . . . 28
  - 4.2.1 Model description . . . 28
  - 4.2.2 State space and discussion . . . 30
- 4.3 Lamp . . . 31
  - 4.3.1 Model description . . . 31
  - 4.3.2 State space and discussion . . . 32
- 4.4 Assembly line . . . 34
  - 4.4.1 Model description . . . 34
  - 4.4.2 Properties and discussion . . . 35

5 Conclusions **39**

Bibliography **41**

Appendix **43**

A Meta-models of mCRL2 **43**

B Specifications of the assembly line model **47**

C The mCRL2 file of the kettle model **51**

D The mCRL2 file of the lamp model **55**

E The mCRL2 file of the assembly line model **59**

Chapter 1

Introduction

In the high-tech industry, model-driven engineering is playing an increasingly important role. With the help of model-driven engineering, the cost and time of developing large systems are reduced. In some domains, such large systems are extremely complex, often consisting of hundreds or even more components. ASML, as the leading provider of lithographic equipment in the world, develops very complex systems. ASML applies a tool called ASD, which is specifically designed for model-driven engineering. ASD is a powerful tool to build software models and verify them. The verification is especially crucial. Because it not only helps minimize the number of defects during the large system development but also assists developers in building a system that conforms to the specified requirements and design specifications.

In general, we call a property that exists only in one component a **local property**. However, other properties required multiple components in practical scenarios. As such, they are called **global properties**. Furthermore, some of the global properties are called **end-to-end properties** that express relations between the used and offered services of the whole system. Such **end-to-end properties** focus on inputs and outputs of the whole system. For instance, if a control component instructs the machine to turn off, every component under its control should be turned off, and the control component can never send any working instruction (like 'move robot arm') before the machine restarts again. Besides that one, sometimes we want to check internal properties that over multiple components.

ASD verification can check deadlock, livelock, race-condition, compliance, etc.. Although ASD is a robust tool, the verification of ASD suffers from many limitations. One of the limitations can be seen when ASD checks interface compliance. Precisely, it verifies whether an ASD component correctly matches each used component according to its service specification, but it cannot check whether the behavior of the resulting system matches its desired purpose. This also means that some **end-to-end properties** cannot be checked by ASD verification.

For research purposes, ASML is using a proprietary ALIAS modeling language to experiment with different forms and different means of formal verification. One of the directions researched is the use of mCRL2 to verify system level and custom (end-to-end) properties.

To use the power of mCRL2, we propose transformations from ASD to mCRL2. Precisely, the transformations consist of ASD to ALIAS and from ALIAS to mCRL2. The resulted mCRL2 model can be used for verification. Until now, such a transformation for a single component has been developed [10]. However, the verification of a single component cannot check **end-to-end properties**. Furthermore, a transformation based on multi-component ASD systems has been come up [15]. This transformation relies on manual work, which may cost engineers much time and result in deviation in the verification only because of manual errors.

Figure 1.1 depicts an overview of the whole transformation, including previous works and this work. Everything starts from an ASD model, which generally consists of multiple components. The model is translated into an ALIAS system with an existing transformation provided by ASML. And then, each component of the ALIAS system can be transformed into an mCRL2 model separately with QVTo [11]. It means that for a single ASD model based on multi-component, more than one

mCRL2 model will be generated. This workflow corresponds to the work [10]. As these mCRL2 models are independent of each other, it is only possible to verify their properties independently. Another workflow that corresponds to the work [15], it translates an ALIAS system to a combined mCRL2 model by hand.

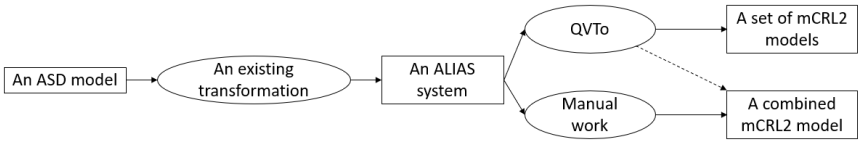

Figure 1.1: An overview of the transformation

Since the reliability and efficiency of a machine transformation will be much higher than that of a manual transformation, in this work, we aim to improve the existing transformations. The dashed line in Figure 1.1 shows the strategy, which specially focuses on the transformation from ALIAS to mCRL2. This enables the industrial application of mCRL2 for checking **end-to-end properties**, especially for systems based on multi-components. Additionally, we hope to develop an automatic way to transform models built in ALIAS into mCRL2.

1.1 Approach

Figure 1.2 shows the model to model transformation from the ALIAS side to the mCRL2 side. The ALIAS system consists of a composition model, multiple design models, and multiple interface models. They have different usages. The composition model is used to the ALIAS system at a structured level, not only including how many components exist but also including all inside connections and connections to the external environment. The design and interface models describe the behaviors of the system and clarify the communication protocols. Basically, all these models in the ALIAS system are instances of the ALIAS meta-model. With the help of QVTo, we define a transformation that results a combined mCRL2 model.

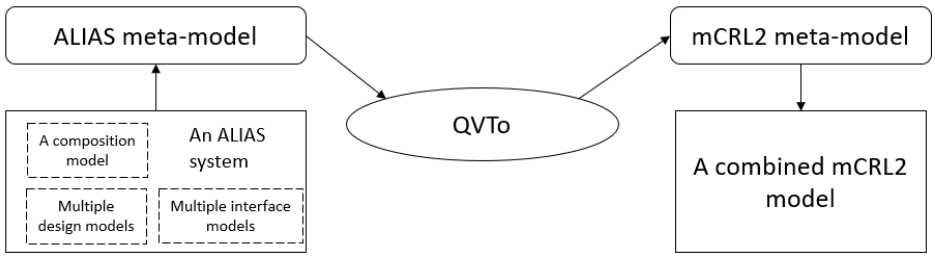

Figure 1.2: Details from the ALIAS side to the mCRL2 side

1.2 Related work

ASD developer Verum develops another tool called Dezyne. Dezyne is similar to ASD, designed for model-driven software. However, it suffers from the limitation that cannot check end-to-end properties as well. A transformation from Dezyne has been implemented in [16]. However, this transformation is also based on a single component.

A translation of a single component from ALIAS to mCRL2 has been developed by Jonk [10]. This is the first work that put the idea of converting a model from ALIAS to mCRL2 into practice. This work considers not only single-thread execution but also multi-thread execution. This work uses some prototypes to verify some single components of ASML products, but it also indicates that the prototyping work is not a finished product and improvements are possible.

A manual implementation of multiple component translation from ALIAS to mCRL2 has been put forward by Rol [15]. Execution behavior relevant differences between mCRL2 and ALIAS is proposed in this work, such as some protection measures (for example, thread control) are added to ensure that the model after transformed is consistent with the original model. This work checks a large system of ASML products manually and shows that scalability can be an issue during the verification.

mCRL2 is a powerful language that is capable of handling very large systems. For instance, in [3], the researchers applied mCRL2 to the train control system ERTMS Hybrid Level 3 and verified the system.

A tool called TReX [1] can deal with models with variables of different kinds of infinite-domain data structures and with parameters. The tool uses symbolic reachability which can generate a reachable state graph to help analyze systems.

Process algebra [2] is the study of parallel systems with communicating processes by algebraic means. It treats the behavior of a system using algebraic and axiomatic techniques. There are various process algebra theories. For instance, Calculus of Communicating Systems (CCS) [13] and Communicating Sequential Processes (CSP) [9].

1.3 Outline

The remainder of this report is organized as follows. The next chapter discusses background information related to this work, including the introductions of used languages as well as tools and the methodology of work. Chapter 3 explains the implementation of the transformation in detail, starting with the most basic principles and ending with the full transformation. Some effective models are used to validate our transformation in chapter 4. Chapter 5 concludes this report.

Chapter 2

Background information

This section briefly explains ASD, ALIAS, and mCRL2, which are the formalisms and knowledge used for the project. Also, model to model transformation is discussed.

2.1 Analytical Software Design

Analytical Software Design (ASD) [5][4] developed by Verum is a model-driven formalism that can be used to design large scale software systems. In ASD, system design relies on components. Individual components can interact by exchanging information. ASD can verify systems consisting of components at design time. After that, ASD can generate executable code automatically.

2.1.1 Basic architecture in ASD

Since a software system in ASD is a component-based design, a component is an important concept here. A component is either a standard component or a foreign component. A standard component usually contains an interface model and a design model. The interface model provides a visible view of the behavior of the component. The design model, which refines its interface model, is invisible to other components. A foreign component, as the name implies, is generated in a non-ASD environment and usually consists of hand-written code. In real systems, standard components have to communicate with foreign components. Hence, the behavior of foreign components must be specified and then be integrated into the ASD system. In this project, we only consider integrated foreign components, generally in the form of interface models.

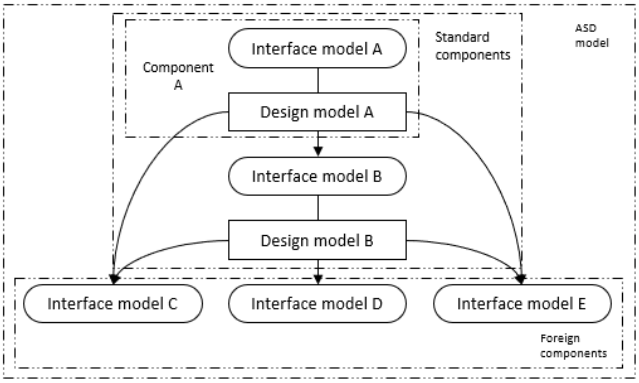

Figure 2.1: An ASD model example, consisting of two standard components and three foreign components

Figure 2.1 depicts a system in ASD, consisting of two standard components (A and B) and three foreign components (C, D, and E). Within the standard components block, it is not hard to see that component A communicates with component B. As component A requires the behavior of component B, we say A is a client of B, and B is a server of A. The concept of client-server is significant in this project. Furthermore, both components A and B rely on components C and E, which shows that different components can use the same interface.

In a standard component, although one design model corresponds to one interface model, the interface model can have multiple interfaces. Note that interfaces are different from interface models. Generally, an interface model has multiple application interfaces (API) and notification interfaces (NI). The example shown in Figure 2.2 is used to illustrate such a case. The interface model A consists of three interfaces 1, 2, and 3. These three interfaces can be API or NI. Interfaces 1 and 3 provide services to the components B and C, respectively. The interface 2, which is shared by the two components, is an attractive point in this system. Usually, for separations of concern, specifying interfaces separately is preferred. Nevertheless, if the interfaces share state behavior or have dependencies, it is better to combine the services that the interfaces provide. So, the existence of interface 2 is useful.

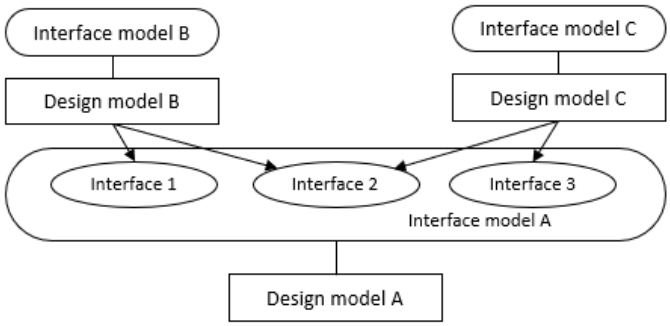

Figure 2.2: An ASD model illustrating that an interface model can contain multiple interfaces, and one interface can be shared by different components

2.1.2 Specifications in ASD

Each component designed in ASD is modeled by a state machine, which consists of a set of state variables and multiple states.

The states are described by a sequence-based-specification (SBS), which is a key concept in ASD. SBS explains when the model should perform what task and what it should do after the end of the task.

The state variables contain state information. Usually, they are used to remember previous actions and to make choices based on previous actions. ASD provides boolean, integer, enumeration, and used service reference types for state variables. For a boolean state variable, the values must be true or false. For an integer state variable, it is possible to set it to a fixed range. For an enumeration type variable, the values are taken from a user-defined finite set. A used service reference type state variable is a vector containing instances of used service reference. Its possible values are deduced from the use in the SBS. Furthermore, every state variable has an initial value. Figure 2.3 gives an example of types of state variables.

Figure 2.4 presents an SBS of a simple ASD model, which shows part of the behavior of an assembly line model.

Each line in an SBS is a transition that is composed of a trigger, a guard, a set of actions, a set of state variable updates, and a target state.

The guard is an expression, similar to an 'if-statement'. It usually uses the state variables to determine a condition. Besides, the guard always returns true or false.

	State Variable	Type	Constraint	Initial Value
1	hasError	Boolean		false
2	Counter	Integer	[0:5]	0
3	Mode	Enumeration		On
4	DeactivatedSensors	Used Service Reference	typeof(WindowSensor)	<>

Figure 2.3: Types of the state variables

Both the trigger and the action can be a call event, a notification event, or a reply event. The difference is that the trigger is an input to a component, while the action is an output from the component. From a client-server perspective, the relationship between the trigger and the action is more explicit. An execution launched by the client is the action, and this action received by the server is considered as the trigger.

The state variable updates are used to assign new values to the state variables. An update takes place when the transition turns to the target state.

Initial	Initial (initial state)					
	Interface	Event	Guard	Actions	State Variable Updates	Target State
1	**Initial (initial state)**					
4	AI	Initialize		MachineA:AI.Initialize; MachineB:AI.Initialize; AI.VoidReply		Ready
11	**Ready**					
15	AI	MakeA+	countA < 2	MachineA:AI.Make; AI.OKReply	countA++	Ready
16	AI	MakeA+	otherwise	AI.FAILReply		Ready
17	AI	MakeB+	countB < 3	MachineB:AI.Make; AI.OKReply	countB++	Ready
18	AI	MakeB+	otherwise	AI.FAILReply		Ready
19	MachineA:NI	Failed		NI.Error		Error
20	MachineA:NI	Done		NI.MadeA	countA--	Ready
21	MachineB:NI	Failed		NI.Error		Error
22	MachineB:NI	Done		NI.MadeB	countB--	Ready
23	**Error**					
27	AI	MakeA+		AI.FAILReply		Error
28	AI	MakeB+		AI.FAILReply		Error
29	MachineA:NI	Failed		NoOp		Error
30	MachineA:NI	Done		NoOp		Error
31	MachineB:NI	Failed		NoOp		Error
32	MachineB:NI	Done		NoOp		Error

Figure 2.4: Specifications of an assemble line model

In this figure, the three lines with a blue background (lines 1, 11, and 23) represent the three states of this component. Two state variables '$countA$' and '$countB$' are used to remember the number of A products and B products, respectively. Taking line 15 as an example, we see how a transition works. Firstly, the transition is triggered by an event '$MakeA+$', then the guard expression '$countA < 2$' checks how many A products are produced. Only if this number is smaller than 2, it sends '$Make$' and '$OKReply$' actions to corresponding interfaces. After that, this component processes the notifications stored in its queue in order. Finally, the state variable '$countA$' is updated, and the component moves to the next state.

2.1.3 Verification in ASD

Model verification is an essential element in ASD. Once a system is built in ASD, the correctness can be checked by the model verification in ASD. The correctness of a component design is always relative to its specification. The model verification confirms that a design model, together with its used interface models correctly implement the specified service behavior. The verification includes whether the interface models are well-formed ASD models, whether the design models make proper use of their used components, whether the servers correctly implement the behavior of their clients. With the verification, the system can be confirmed that it is well-formed, free of livelocks and deadlocks. Furthermore, it checks range errors, freedom, race-conditions, absence of illegal actions.

2.2 Single thread execution in ASD

There are two distinct execution models in ASD: the multi-threaded and single-threaded execution models. In this project, only the single-threaded execution model is considered. In the single-threaded execution model, only one thread can be activated at a time. Besides, the single-threaded execution model processes events stored in the scheduled (First-In-First-Out) notification queue [10].

Because events cannot happen in parallel in the single-threaded execution model, the sequence of processing events is important. Run-to-completion semantics are used to precisely clarify the sequence in a transition from one state to another. With the help of run-to-completion semantics, we can guarantee that the guard expression is evaluated, the trigger as well as all actions are executed, all notifications are processed, and all state variables are updated before entering the next state. Based on [14], we conclude the run-to-completion semantics in ASD. Whenever a component receives a request from a client, the following steps are taken consecutively.

1. Process all actions in an SBS rule.
2. Update the state variables.
3. Move to the target state.
4. Process notifications in the queue one by one. Except for the events caused by the notifications, no events can take place before the queue has been emptied.
5. Send a void or valued reply to the client.

2.3 ALIAS

ALIAS is a Domain Specific Language (DSL) developed by ASML, aimed at reducing software system complexity. By using ALIAS, it is easy to decompose a large system into smaller pieces (components), which can be engineered simultaneously and independently.

An ALIAS system consists of design models and interface models. The design models refine the behaviors of their interface models. ALIAS also applies the same concept of client-server relation as ASD. Furthermore, behaviors of a component in ALIAS are expressed in a state machine, which is a type of Mealy machine [12]. However, in ALIAS, the state machine is distinguished into two different types of state machines: protocol and realization state machines. Protocol state machines are used to specify the communications between two components, and they correspond to interface models. Realization state machines correspond to design models. They realize the specified protocols of their interface models. Although protocol and realization state machines are different in some ways, they are implemented as same as the state machines in ASD, see section 2.1.2.

In ALIAS, components in a system communicate with each other in three ways: **control operations** (which are the same as the call events in ASD), **notifications**, and **replies** (both

the notifications and the replies are the same terms in ASD). They have a clear distinction. The **control operations** are run on the server until completion: the client is blocked until the server replies. The **notifications** are realized by means of the queue, which also means this is asynchronous communication. The **replies** are used to indicate the server has finished the job. A valued reply may carry values. When it carries some values, we say it is a valued reply. Otherwise, it is a void reply.

The most important concept in ALIAS different from ASD is **composition**. **Composition** depicts how a system is structured, and it contains all information that can form the system in a tree structure. The information consists of two parts: **composite system** and **decomposition**. The composite system contains provided interface(s) and required interface(s) of the system, and these interfaces explain how the system provides services to the higher-level environment and requests corresponding services from the lower-level environment. The decomposition is more complex because it draws all connections of the system. It instantiates the components in the system, along with their interfaces.

2.4 mCRL2

mCRL2 stands for micro Common Representation Language 2. It is a communicating process algebraic language extended with data, time, and multi-actions [7]. As the target language in this automatic transformation, a system modeled in mCRL2 can be converted to a labelled transition system (LTS), and this LTS can be visualized via some extensive tools developed for mCRL2 [6]. Additionally, mCRL2 is robust to verify the behavior of models against their requirements. We take an example to explain the basic behaviors of mCRL2.

$$
\begin{aligned}
&\textbf{act} \quad \mathit{insert}, \mathit{collect}, \mathit{addition} : \mathit{Nat};\\
&\qquad \mathit{offSend}, \mathit{offReceive}, \mathit{off};\\
&\textbf{proc} \ \mathit{User} = \mathit{insert}(1).\mathit{User} + \mathit{offReceive}.\mathit{delta};\\
&\qquad \mathit{Mach}(n : \mathit{Int}) = (n < 10) \rightarrow \mathit{collect}(1).\mathit{Mach}(n+1) \diamond \mathit{offSend}.\mathit{Mach}(0);\\
&\textbf{init} \quad \textbf{allow}(\{\mathit{addition}, \mathit{off}\},\\
&\qquad\quad \textbf{comm}(\{\mathit{insert} \mid \mathit{collect} \rightarrow \mathit{addition}, \mathit{offSend} \mid \mathit{offReceive} \rightarrow \mathit{off}\},\\
&\qquad\qquad \mathit{User} \parallel \mathit{Mach}(0)));
\end{aligned}
$$

Basically, this example describes a simple counter. This system has two processes, *User* and *Mach*. The process *Mach* uses an integer number n to store the current number. The process *User* can choose to execute action *insert(1)*. If the integer number n in *Mach* is smaller than 10, it must perform action *collect*, and then increase the counter by adding '1' to n. If the integer number n is already equal to or larger than 10, *Mach* sends an *offsend* to terminate the system. In the **init** part, **allow** expresses the actions that can be performed in the system. **comm** constraints what actions must happen synchronously, and the remaining part represents the parallel processes in the system.

The above example only mentions the most basic process expressions of mCRL2. More expressions are listed in Table 2.1. There are several process expressions worth mentioning. The first one is Process instantiation, which is used to declare processes. The second one is 'delta'. It is used to terminate a process or generate a deadlock. The process expression 'tau' represents a hidden action or an empty multi-action. The choice operator '+' expresses that either the behavior of the left-hand side 'procExp' or the right-hand side 'procExp' can be chosen. Processes in the parallel operator can execute actions simultaneously. The sum operator is a generalization of the choice operator. We assume that the process expression in the right 'procExp' is $p(v)$, with the help of its variable v with a type D, it is easy to obtain an expression in terms of a choice operator, $p(v_0) + p(v_1) + ... + p(v_n)$, for all $n \geq 0$. In the if-then-else operator, only if the guard expression g is true, the left-hand side process expression can execute. Otherwise, the right-hand side process

procExp	:=	Action	Action
	|	P	Process instantiation
	|	delta	Delta
	|	tau	Tau
	|	procExp + procExp	Choice operator
	|	procExp ‖ procExp	Parallel operator
	|	(procExp)	Brackets
	|	**sum** $v : D.$ procExp	Sum operator
	|	g $\rightarrow$ procExp	If-then operator
	|	g $\rightarrow$ procExp $\diamond$ procExp	If-then-else operator
	|	**allow**({MultiAction}, procExp)	Allow operator
	|	**comm**({CommExpr}, procExp)	Communication operator

Table 2.1: Relevant expressions of mCRL2

expression performs. The allow operator is used to declare what multi-actions (MultiAction) are enforced to perform, so it has a set of multi-actions. The communication operator with a set of communication expressions (CommExpr) lets actions communicate or synchronize. We do not explain the contents of the MultiAction and CommExpr here, to find more details about mCRL2 expressions, see [7].

2.5 M2M

A model defined in a modeling language can be characterized by an abstract representation of a system. The model language contains many elements, and from a certain perspective, the model is built by these elements. A meta-model defines the specification that describes the model. If the meta-model can be imagined as a formal language, the model is an article described in the language, where the elements in the meta-model are the vocabulary of the language, and the relationship between the elements is the grammar of the language. In short, the model is an instance of the modeling language, and the meta-model defines the modeling language.

With the help of model and modeling language, a fundamental concept follows, which is a model transformation. The model transformation is a process of converting a source model into a target model according to a set of transformation rules. Such a model transformation is known as a model to model (M2M) transformation.

Figure 2.5 depicts an M2M transformation that could be useful in this project. As a source model, the model A is an instance of the meta-model of A, with the help of a model transformation language, it is possible to create a new model B, which is an instance of the meta-model of B.

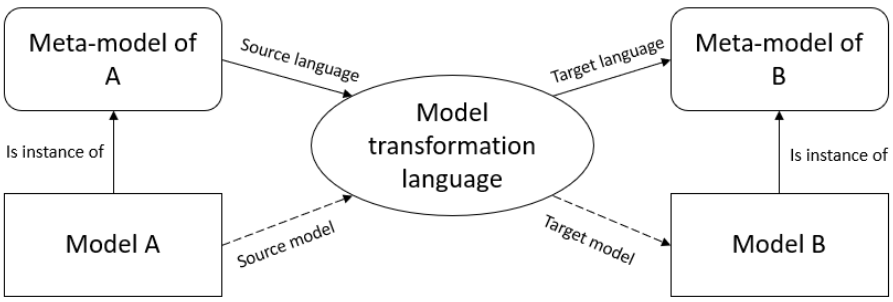

Figure 2.5: Model to model transformation

2.6 A meta-model example

As stated in the previous section, the meta-model is an important concept of this transformation. We can convert one model into another with the help of the corresponding meta-model. The following Figure 2.6 is a class diagram in UML. It shows an example of the meta-model, which describes the **Sorts** in the mCRL2 language.

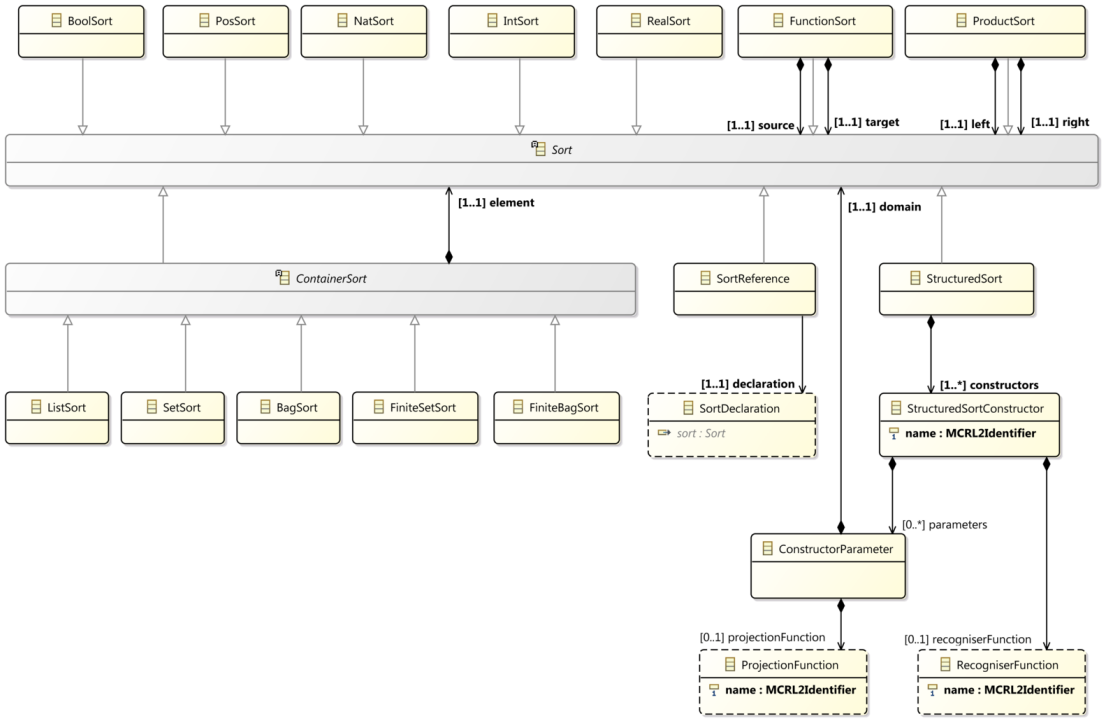

Figure 2.6: The meta-model of sorts in mCRL2

In the middle of this picture is a class **Sort**, which is connected with many sub-classes. For instance, **BoolSort**, **NatSort**, **IntSort**. They all use generalization arrows to point to the class **Sort**, which means that the relation between these sub-classes and the class **Sort** are inheritance relations. In the figure, we can also find some composition arrows, for example, between the class **ConstructorParameter** and the class **Sort**. The relation between them emphasizes the concepts of overall and partial inclusion.

For both the source language ALIAS and the target language mCRL2, we need to clarify their meta-models. The complete meta model for mCRL2 can be found in Appendix A.

Chapter 3

Implementation

In the previous chapter, background information has been introduced. We discuss specific details about transforming an ALIAS model into an mCRL2 model in this chapter. In particular, we focus on the abstract aspects.

3.1 From an ALIAS model to a combined mCRL2 model

One of the challenges in this project is how to convert an ALIAS model to an mCRL2 model at the structured level. As stated previously, there are interface models in the ASD model that provide a visible view of the behavior of the components. Similarly, in the ALIAS model, a concept named protocol provides the same property.

In previous work [10], a transformation has been made from a single design model with its associated interface model to mCRL2. However, in this project, our purpose is to integrate multiple design models and interfaces models to an entire system, and then verify its properties. In such a case, we might want to make some changes to make the ALIAS model and the mCRL2 model achieve the same purpose with different structures. The solution to this problem is replacing the interface model with the corresponding design model since the design model refines its interface model. We call this step **combination**. And the model after the **combination** is called **combined model**. It is worth mentioning that the interface models at the lowest level contain behaviors of the external environment, so they will not be removed.

It is worth mentioning that the resulted ALIAS model from ASD to ALIAS (this transformation is achieved by an ASML tool) keeps the same structure as the one in ASD. So, it is not essential to distinguish between ASD and ALIAS at the structured level. Figure 3.1 shows the combined result of an original ASD model (left) to an mCRL2 model (right) at the structured level. It is not hard to see that at the structure level, the interface model A and the interface model B are removed, or we say they are replaced by their design models. Moreover, the interface models C, D, E still exist, because they are placed at the lowest level.

However, to solve this problem, only changes at the structure level are not enough. In the following sections, more changes will be discussed.

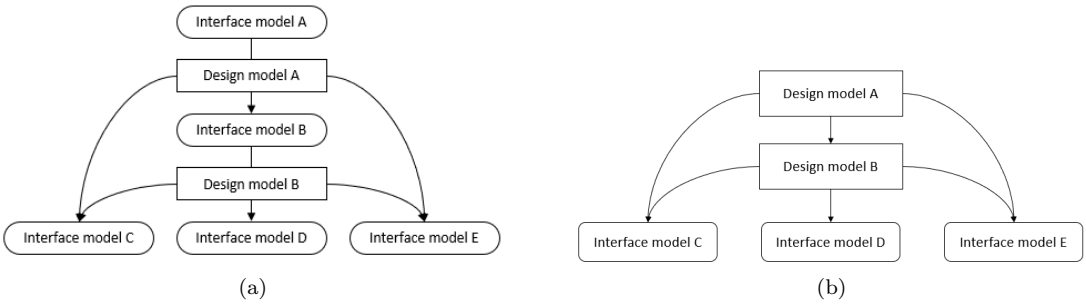

Figure 3.1: Combination result from an original ASD model to an mCRL2 model at the structure level

3.2 ALIAS language specification

At the structured level, we have figured out what the structure of the resulting system should be.

It is obvious that to transform a model composed of the ALIAS language into the corresponding mCRL2 model, for which both of them have the same behavior, only taking the combination step is far from enough. The main work we do is to transform ALIAS models to mCRL2 models via QVTo, which applies a model to model transformation. In this section, we abstractly define the source language ALIAS using mathematical notations and syntax. Moreover, to better define this language, some universes that contain important concepts of ALIAS are introduced. In the following, all letters using calligraphic fonts represent a universe, all uppercase letters represent a set or a sequence, and all lowercase letters represent a single term.

- $\mathcal{M}$ is the universe that contains all the combined models (all models after taking the combination step).
- $\mathcal{M}_d \subseteq \mathcal{M}$ is the universe that contains all the design models.
- $\mathcal{M}_i \subseteq \mathcal{M}$ is the universe that contains all the interface models.
- $\mathcal{X}$ is the universe that contains all the control systems.
- $\mathcal{R}$ is the universe that contains all the reply values.
- $\mathcal{O}$ is the universe that contains all the control operations.
- $\mathcal{N}$ is the universe that contains all the notifications.
- $\mathcal{T}_r \subseteq \mathcal{R} \cup \mathcal{O} \cup \mathcal{N}$ is the universe that contains all the triggers.
- $\mathcal{A} \subseteq \mathcal{R} \cup \mathcal{O} \cup \mathcal{N}$ is the universe that contains all the actions.
- $\mathcal{S}$ is the universe that contains all the states.
- $\mathcal{D}$ is the universe that contains all the types.
- $\mathcal{V}$ is the universe that contains all the state variables, which all $v \in \mathcal{V}$ have a type $d \in \mathcal{D}$.
- $\mathcal{I}$ is the universe that contains all the interfaces.
- $\mathcal{C}$ is the universe that contains all the connections.

A transition, as already described in section 2.1.2, is a significant concept. A transition is composed of a trigger, a guard, a set of actions, a set of updates, and a target state. It is worth mentioning that the guard is an expression over state variables used to determine whether to execute the corresponding actions. We define $\mathcal{T}$ as the universe that contains all the transitions.

Definition 1 *A transition $t \in \mathcal{T}$ is given by $t = (tr, g, A, U, s)$, where*

- *$tr \in \mathcal{T}_r$ is a trigger.*
- *g is a guard over state variables.*
- *$A \in (\mathcal{R} \cup \mathcal{O} \cup \mathcal{N})^*$ is a finite sequence of actions.*
- *U is a set of updates on state variables, in terms of pairs $(v, p) \in U$, for which the v is a state variable to update, and the p is a value or an expression to be assigned to the state variable.*
- *$s \in \mathcal{S}$ is the target state of this transition.*

In the above definition, the finite sequence of actions A has the restriction that A contains one reply $r \in \mathcal{R}$ at the end if and only if the trigger $tr \in \mathcal{O}$.

We denote states in an ALIAS model by S. The states $S \subseteq \mathcal{S}$ consist of **an initial state**, **normal states**, and **evaluation states**. Then $S = S_{normal} \cup S_{eval}$ and $S_{normal} \cap S_{eval} = \emptyset$ and $S \neq \emptyset$. It is worth mentioning that an initial state is a special type of a normal state, which is used to specify where the state machine starts.

In a state, there is a state invariant which contains a boolean expression, similar to a guard. It is a run-time constraint, mainly used for verification within a model. Although the state invariant is significant to verification, it is not considered in this project.

Besides, every state must have at least one transition.

Definition 2 *A state $s \in \mathcal{S}$ is given by the tuple $s = (inv, T)$, where*

- *inv is a state invariant.*
- *$T \subseteq \mathcal{T}$ is a non-empty set of transitions.*

The state machine talked about in section 2.1.2 is a critical concept. Like models designed in ASD, ALIAS models are also realized by state machines. We define $\mathcal{SM}$ to be the the universe that contains all the behavior state machines. More precisely, the set $\mathcal{SM}$ is partitioned into two types of state machines: realization state machines $\mathcal{SM}_r$ and protocol state machines $\mathcal{SM}_p$. Then $\mathcal{SM} = \mathcal{SM}_r \cup \mathcal{SM}_p$.

Definition 3 *A state machine $sm \in \mathcal{SM}$ is given by the tuple $sm = (S_{normal}, S_{eval}, V, s_{init}, v_{init})$, where*

- *$S_{normal} \subseteq \mathcal{S}$ is a set of normal states.*
- *$S_{eval} \subseteq \mathcal{S}$ is a set of evaluation states.*
- *$V \subseteq \mathcal{V}$ is a sequence of state variables.*
- *$s_{init} \in S_{normal}$ is the initial state.*
- *v_{init} is a set of initial values of state variables.*

An ALIAS system must be comprised of three types of interfaces: **control interfaces** (I_c), **provide interfaces** (I_p) and **required interfaces** (I_r). For each of them, we define universes: $\mathcal{I}_c$, $\mathcal{I}_p$, and $\mathcal{I}_r$, respectively. Then $I_c \subseteq \mathcal{I}_c$, $I_p \subseteq \mathcal{I}_p$, $I_r \subseteq \mathcal{I}_r$. And the interface universe $\mathcal{I}$ is defined by $\mathcal{I} = \mathcal{I}_c \cup \mathcal{I}_p \cup \mathcal{I}_r$.

Definition 4 *A control interface $i_c \in \mathcal{I}_c$ is given by the tuple $i_c = (O, N, R)$, where*

- *$O \subseteq \mathcal{O}$ is a set of control operations.*
- *$N \subseteq \mathcal{N}$ is a set of notifications.*

- $R \subseteq \mathcal{R}$ *is a set of reply values.*

Definition 5 *An interface model* $m_i \in \mathcal{M}_i$ *is given by the tuple* $m_i = (I_c, sm_p)$*, where*

- $I_c \subseteq \mathcal{I}_c$ *is a set of control interfaces.*
- $sm_p \in \mathcal{SM}_p$ *is a protocol state machine.*

Definition 6 *A design model* $m_d \in \mathcal{M}_d$ *is given by the tuple* $m_d = (I_c, sm_r)$*, where*

- $I_c \subseteq \mathcal{I}_c$ *is a set of control interfaces.*
- $sm_r \in \mathcal{SM}_r$ *is a realization state machine.*

Definition 7 *A combined model* $m \in \mathcal{M}$ *is given by the tuple* $m = (I_c, sm)$*, where*

- $I_c \subseteq \mathcal{I}_c$ *is a set of control interfaces.*
- $sm \in \mathcal{SM}$ *is a state machine.*

Three different types of connections make up the universe $\mathcal{C}$: **incomings** ($\mathcal{C}_{in}$), **outgoings** ($\mathcal{C}_{out}$), and **connects** ($\mathcal{C}_c$). So, the universe of connections $\mathcal{C} = \mathcal{C}_{in} \cup \mathcal{C}_{out} \cup \mathcal{C}_c$. For each incoming $c_{in} \in \mathcal{C}_{in}$, it represents a connection from the extended environment to the combined system, while each outgoing $c_{out} \in \mathcal{C}_{out}$ means a connection from the combined system to the extended environment. And each connect $c_c \in \mathcal{C}_c$ stands for a connection between different models in the combined system.

Definition 8 *An* ***incoming*** *connection* $c_{in} \in \mathcal{C}_{in}$ *is given by the tuple* $c_{in} = (i_p, i_c)$*, where*

- $i_p \in \mathcal{I}_p$ *is a provided interface.*
- $i_c \in \mathcal{I}_c$ *is a control interface.*

Definition 9 *A* ***connect*** *connection* $c_c \in \mathcal{C}_c$ *is given by the tuple* $c_{in} = (i_c, i_c^{'})$*, where*

- $i_c, i_c^{'} \in \mathcal{I}_c$ *are control interfaces.*

Definition 10 *An* ***outgoing*** *connection* $c_{out} \in \mathcal{C}_{out}$ *is given by the tuple* $c_{in} = (i_c, i_r)$*, where*

- $i_c \in \mathcal{I}_c$ *is a control interface.*
- $i_r \in \mathcal{I}_r$ *is a required interface.*

After we have defined all the above elements, a whole system is available to be defined.

Definition 11 *A complete ALIAS system consists of multiple control systems, and some provided, required interfaces as well as connections are used to describe the structure of the system. It is given by the tuple* $S_{complete} = (M_d, M_i, I_p, I_r, C)$*, where*

- $M_d \subseteq \mathcal{M}_d$ *is a set of design models.*
- $M_i \subseteq \mathcal{M}_i$ *is a set of interface models.*
- $M_i \cap M_d = \emptyset$
- $I_p \subset \mathcal{I}_p$ *is a set of provided interfaces.*
- $I_r \subset \mathcal{I}_r$ *is a set of required interfaces.*
- $C \subseteq \mathcal{C}$ *is a set of connections.*

3.3 The specifications of communications in mCRL2

When a model is transformed from ALIAS to mCRL2, some process synchronization mechanisms are required to ensure that the output mCRL2 model is consistent with the original model. Before we introduce these synchronization mechanisms, we define a function to help construct specifications in mCRL2 at first.

Definition 12 *A function given by $\mathcal{G}$: $\varepsilon \mapsto \mathcal{L}$ generates the globally unique name of the input, where*

- *ε contains elements which can be a combined model, a control system, a control interface, a control operation, a notification, a reply or a variable. Concretely, $\varepsilon = \mathcal{M} \cup \mathcal{X} \cup \mathcal{I} \cup \mathcal{O} \cup \mathcal{N} \cup \mathcal{R} \cup \mathcal{V}$.*
- *$\mathcal{L}$ is a set of string.*

In the following text, we generally write G_n for $\mathcal{G}(n)$, where $n \in \varepsilon$. As mentioned before, the work is constrained in single-thread execution mode, which means only one thread can be activated, and thus only one component can be processed at a time. It requires some mechanisms to ensure tasks in a single-component model execute in compliance with run-to-completion semantics. The paper [10] introduces a process blocking mechanism in queues. However, this solution alone is not enough in a multi-component model. Because [10] focuses on a single-component model, it ignores the server-client relation (see 2.1.1). For instance, when a component as a server tries to send a reply to its client, the solution does not know which component the client is of this server component. To solve the problem for systems based on multi-component, [15] proposes some other mechanisms of thread control. These mechanisms are all summarized as communications listed below.

Note that we use abbreviations for these specifications. For instance, $LockQ\_s\_G_m$ stands for the combination of the string '$LockQ\_s\_$' with the unique name of a model m 'G_m'. To see more accurately how they are transformed, refer to section 3.4.2.

- $LockQ\_s\_G_m \mid LockQ\_r\_G_m \rightarrow LockQ\_G_m$
  $unLockQ\_s\_G_m \mid unLockQ\_r\_G_m \rightarrow unLockQ\_G_m$
  These actions are used to (un)lock the queue of the client.
- $qEmpty\_s\_G_m \mid qEmpty\_r\_G_m \rightarrow qEmpty\_G_m$
  $qNonEmpty\_s\_G_m \mid qNonEmpty\_r\_G_m \rightarrow qNonEmpty\_G_m$
  These actions are used to check if the queue is empty. If the queue is empty, $qEmpty\_G_m$ is executed.
- $emptyQ\_s \mid emptyQ\_r\_G_m \rightarrow emptyQ$
  These actions are used to check if all queues in a complete system are empty.
- $lock\_p\_thread \mid lock\_t\_thread \rightarrow lock\_thread$
  $unlock\_p\_thread \mid unlock\_t\_thread \rightarrow unlock\_thread$
  These actions are used to (un)lock the thread of a model which executes as a single-thread.
- $queue\_lock\_p\_thread \mid queue\_lock\_t\_thread \rightarrow queue\_lock\_thread$
  $queue\_unlock\_p\_thread \mid queue\_unlock\_t\_thread \rightarrow queue\_unlock\_thread$
  These actions are used to (un)lock the thread of a queue. A notification can be processed if the thread is not locked.

Figure 3.2 presents the communications among the framework, two components (A and B), and one foreign component (C). In this case, because A and B send messages to and request replies from B, we say A and B are the clients of C, and C is the server of A and B. For the two clients A and B, both of them own a queue to store notifications. Each communication is labeled with a number, and the same number stands for the same behavior. The following explains each communication one by one in detail.

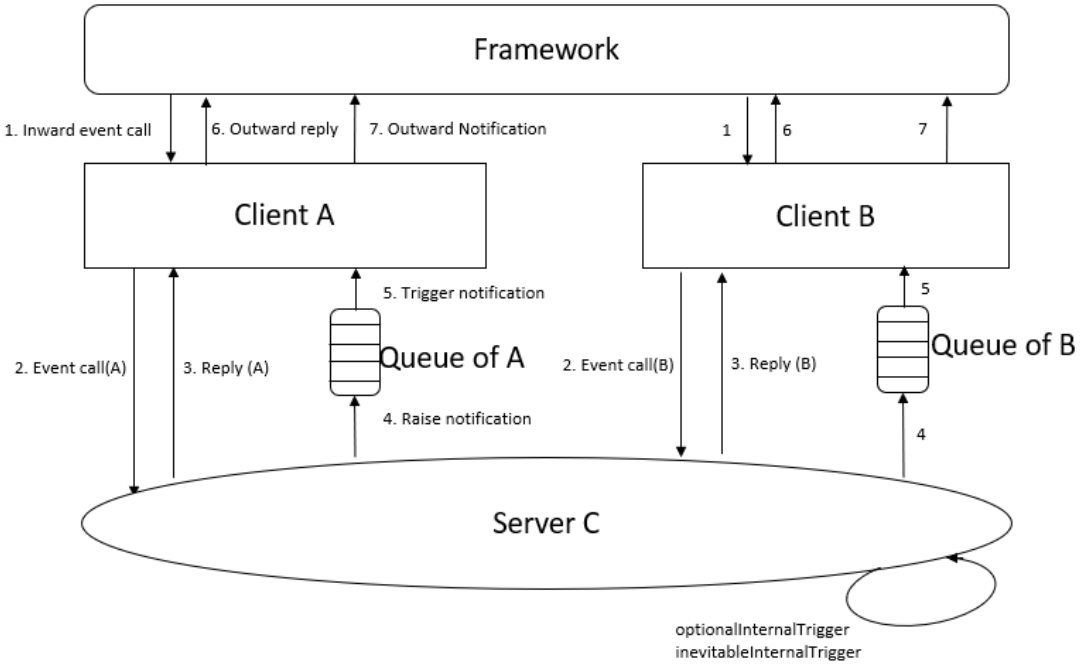

Figure 3.2: Communications in ALIAS result additional mCRL2 specifications

1. G_o
   The o stands for a control operation. This action constructs an inward call event from the framework to a component. It synchronizes with $lock\_p\_thread$, $emptyQ\_s$ and $LockQ\_s\_G_m$.

2. $invoke''G_o(G_m) \mid invoked''G_o(G_m) \rightarrow G_o(G_m)$
   These actions construct a call event from a client component to a server component.

3. $writeReply(G_r, G_m, G_{m'}) \mid readReply(G_r, G_m, G_{m'}) \rightarrow sendReply(G_r, G_m, G_{m'})$
   The r stands for a reply value which can be valued or void. The m and m' stand for control systems of a server and its client, respectively. These actions construct a reply from a server component to the client, which sent a call event to this server. Taking Figure 3.2 as an example, the m represents server C, and the m' represents the client A if the server C is invoked by client A before. However, if the server C is invoked by client B before, the m' should be client B.

4. $pushNotification\_G_m(G_n) \mid receiveNotification\_G_m(G_n) \rightarrow raiseNotification\_G_m(G_n)$
   These actions construct a notification event from a server component to the queue of a client.

5. $sendNotification\_G_m(G_n) \mid readNotification\_G_m(G_n) \rightarrow triggerNotification\_G_m(G_n)$
   These actions construct a notification event from the queue of a client to the client component. During the life cycle of the server, the queue of the client is temporarily unlocked, but the queue needs to be locked when the notification is processing. Hence, the action $readNotification\_G_m(G_n)$ synchronizes with $queue\_unlock\_p\_thread(G_m)$ and $lock\_p\_thread$.

6. $outwardRelpy$
   This action construct a reply event from a component to the framework. It synchronizes with $unlock\_p\_thread$ and $unLockQ\_s\_G_m$.

7. $outwardNotification$
   This action constructs a notification event from a component to the framework.

3.4 ALIAS to mCRL2 standard specifications

As a result of the transformation, mCRL2 consists of four parts. They are **Sort**, **Action**, **Process**, and **Initialization**. In this section, we discuss these elements one by one.

To describe these transformations more clearly, we first define some symbols. All content contained by '<' and '>' exists in the form of a string, the symbol '+' between two strings can combine them into an individual string, and everything contained in '[' and ']' is a sequence, which means order matters. Hence, for an arbitrary string $s \in String$, the form '$< s > + < s >$' becomes '$< ss >$'. In the following, we may hide '<' and '>' if we know that what they contain is a string.

- $\sum_{p \in P} f(p) \stackrel{\text{def}}{=} f(p_0) + f(p_1) + \ldots + f(p_n)$, where $P = \{p_0, p_1, \ldots, p_n\}$
- $|_{p \in P}\ f(p) \stackrel{\text{def}}{=} f(p_0)\ \ |\ \ f(p_1)\ \ |\ \ \ldots\ \ |\ \ f(p_n)$, where $P = \{p_0, p_1, \ldots, p_n\}$
- $\|_{p \in P}\ f(p) \stackrel{\text{def}}{=} f(p_0)\ \ \|\ \ f(p_1)\ \ \|\ \ \ldots\ \ \|\ \ f(p_n)$, where $P = \{p_0, p_1, \ldots, p_n\}$
- $\ddagger_{p \in P} f(p) \stackrel{\text{def}}{=} f(p_0)\ ,\ f(p_1)\ ,\ \ldots\ ,\ f(p_n)$, where $P = [p_0, p_1, \ldots, p_n]$
- $\dagger_{p \in P} f(p) \stackrel{\text{def}}{=} f(p_0)\ .\ f(p_1)\ .\ \ldots\ .\ f(p_n)$, where $P = [p_0, p_1, \ldots, p_n]$

The above function $f(p)$ stands for an arbitrary function over the variable p.

3.4.1 Sort

In mCRL2, the keyword **sort** is used to declare arbitrary data sorts. Four data sorts need to be declared in the transformation from ALIAS to mCRL2.

Given an ALIAS system $S_{complete} = (M_d, M_i, I_p, I_r, C)$. The system consists of design models $M_d = (m_{d_0}, m_{d_1}, \ldots, m_{d_n})$ and $M_i = (m_{i_0}, m_{d_1}, \ldots, m_{d_n})$. Note that all of these models belong to combined models, so a set M representing for combined models should be $M = M_d \cup M_i$. Through these combined models, it is possible to find all notifications N and replies R of this system. Furthermore, each combined model $m \in M$ corresponds to a state machine sm. We denote all state machines of this system by a set SM.

The first transformation of the **sort** part is **Component**, as the name implies, it includes all components of a combined mCRL2 model. The function $\mathsf{Comp}(M)$ transforms it, note the equal sign directly before the keyword **struct** is a content of this function:

$$\mathsf{Comp}(M) = \quad \textbf{sort} \quad \mathit{Component} = \textbf{struct} \quad |_{m \in M}\ G_m \ \ | < \mathit{NoClient} >$$

The second one is **LockingState**. Let $sm \in SM$ be a state machine, no matter it is a protocol state machine or a realization state machine. Then, the transformation is given by the function $\mathsf{LockS}(SM)$:

$$\mathsf{LockS}(SM) = \quad \textbf{sort} \quad \mathit{LockingState} = \textbf{struct} \quad |_{sm \in SM}\ (< LOCK'' > + G_{sm}) \ \ | < \mathit{NONE} >$$

Notification, the third element, consists of all notifications in the system. Let N be the set of all notifications. Then, the transformation is given by the function $\mathsf{Noti}(N)$:

$$\mathsf{Noti}(N) = \quad \textbf{sort} \quad \mathit{Notification} = \textbf{struct} \quad |_{n \in N}\ G_n$$

The last one is **Enumeration**, which is used to introduce reply values. Let R be the set of all replies. Then, the transformation is given by the function $\mathsf{Enum}(R)$:

$$\mathsf{Enum}(R) = \quad \textbf{sort} \quad \mathit{Enumeration} = \textbf{struct} \quad |_{r \in R}\ G_r \ \ | < \mathit{NoReplyValue} > \ | < \mathit{VoidReply} >$$

From the above, we conclude the whole transformation of the sort part with the function $\mathsf{SortDef}(M, SM, N, R)$:

$$\mathsf{SortDef}(M, SM, N, R) = \quad \mathsf{Comp}(M); \quad \mathsf{LockS}(SM); \quad \mathsf{Noti}(N); \quad \mathsf{Enum}(R);$$

3.4.2 Action

Actions are the basic elements of processes in mCRL2. Before using actions in processes, we need to define them first. In this section, all the actions that are needed in the transformation are defined. We list them one by one. Given the same ALIAS system $S_{complete} = (M_d, M_i, I_p, I_r, C)$, we obtain $M = M_d \cup M_i$ again. Through these combined models, it is possible to find all control operations O of this system. Similar to the **sort** part, the transformations are given by functions.

About queue lock, we have the function

$\mathsf{LockQ}(\mathsf{M}) =$

act

$\ddagger_{m \in M}(< LockQ\_s\_ > + G_m), \ddagger_{m \in M}(< LockQ\_r\_ > + G_m), \ddagger_{m \in M}(< LockQ\_ > + G_m)$

$\ddagger_{m \in M}(< unLockQ\_s\_ > + G_m), \ddagger_{m \in M}(< unLockQ\_r\_ > + G_m), \ddagger_{m \in M}(< unLockQ\_ > + G_m)$

About the empty queue check for each component, we have the function

$\mathsf{QEmpty}(\mathsf{M}) =$

act

$\ddagger_{m \in M}(< qEmpty\_s\_ > + G_m), \ddagger_{m \in M}(< qEmpty\_r\_ > + G_m), \ddagger_{m \in M}(< qEmpty\_ > + G_m)$

$\ddagger_{m \in M}(< qNonEmpty\_s\_ > + G_m), \ddagger_{m \in M}(< qNonEmpty\_r\_ > + G_m)$

$\ddagger_{m \in M}(< qNonEmpty\_ > + G_m)$

About the empty queue check for the entire system, we have the function

$\mathsf{EmptyQ}(\mathsf{M}) =$

act

$< emptyQ\_s >, \ddagger_{m \in M}(< emptyQ\_r\_ > + G_m), < emptyQ >$

About thread lock of a model, we have the function

$\mathsf{LockThread}() =$

act

$< lock\_p\_thread >, < lock\_t\_thread >, < lock\_thread >$

$< unlock\_p\_thread >, < unlock\_t\_thread >, < unlock\_thread >$

About thread lock of a queue, we have the function

$\mathsf{QueueThread}() =$

act

$< queue\_lock\_p\_thread >, < queue\_lock\_t\_thread >, < queue\_lock\_thread >: Component$

$< queue\_unlock\_p\_thread >, < queue\_unlock\_t\_thread >: Component$

$< queue\_unlock\_thread >: Component$

For the control operations which are inward call events from the framework to a component, we have the function

$\mathsf{ControlO}(\mathsf{O}) =$

act

$\ddagger_{o \in O} G_o$

And for the else control operations, we have the function

$\mathsf{Invoke(O)} =$

act
$\ddagger_{o\in O}(< invoke'' > +G_o), \ddagger_{o\in O}(< invoked'' > +G_o), \ddagger_{o\in O} G_o : Component$

About call event reply, we have the function

$\mathsf{Rep()} =$

act
$< writeReply >, < readRelpy >, < sendReply >: Enumeration\#Component\#Component$

And for other notifications, we have the function

$\mathsf{Notify(M)} =$

act
$\ddagger_{m\in M}(< pushNotification\_ > +G_m), \ddagger_{m\in M}(< receiveNotification\_ > +G_m) : Notification$
$\ddagger_{m\in M}(< sendNotification\_ > +G_m), \ddagger_{m\in M}(< readNotification\_ > +G_m) : Notification$
$\ddagger_{m\in M}(< raiseNotification\_ > +G_m), \ddagger_{m\in M}(< triggerNotification\_ > +G_m) : Notification$
$< outwardNotification >$

From the above, we conclude the whole transformation of the **sort** part with the function

$\mathsf{ActionDef}(M, O) =$

$\mathsf{LockQ}(M);\quad \mathsf{QEmpty}(M);\quad \mathsf{EmptyQ}(M);\quad \mathsf{LockThread}();$
$\mathsf{QueueThread}();\quad \mathsf{ControlO}(O);\quad \mathsf{Invoke}(O);\quad \mathsf{Rep}();\quad \mathsf{Notify}(M)$

3.4.3 Process

In mCRL2, the keyword **proc** is used to declare recursive processes. The **sorts** and **actions** introduced in the previous two sections can be combined in some forms to become ingredients of processes. With the help of the processes, behaviors of a system can be explained. In this section, we might abbreviate some actions to shorten the article. For example, $emptyQ\_s\_G_m$ will replace $(< emptyQ\_s\_ > +G_m)$. Besides, different from the previous sections, the symbol '+' means choices in conditional operators now.

Given the same ALIAS system $S_{complete} = (M_d, M_i, I_p, I_r, C)$. And the same as before, the system consists of design models $M_d = (m_{d_0}, m_{d_1}, \ldots, m_{d_n})$ and $M_i = (m_{i_0}, m_{d_1}, \ldots, m_{d_n})$. A set M representing for the combined models is obtained $M = M_d \cup M_i$.

Let $m \in M$ be a combined model. Then $m = (I_c, sm)$ has $I_c = \{i_{c_0}, i_{c_1}, \ldots, i_{c_n}\}$ and $sm = (S_{normal}, S_{eval}, V, s_{init}, v_{init})$. For each state $s \in S_{normal} \cup S_{eval}$, $s = (inv, T)$. At least one transition $t \in T$ should be there, given by $t = (tr, g, A, U, s')$. Note although the state could have an invariant inv, we do not use it during the transformation. Furthermore, we introduce a function $\mathsf{Type} : \mathcal{V} \mapsto \mathcal{D}$ that returns the data type of the state variables. Each state in ALIAS corresponds to one recursive process in mCRL2. The transformation of models to processes is given by the function

$\mathsf{ProcessDef}(S_{complete}) =$

proc

$\ddagger_{m=(I_c,(S_{normal},S_{eval},V,s_{init},v_{init}))\in M}$ (

$\ddagger_{s=(inv,T)\in S_{normal}\cup S_{eval}}$ (

$G_s(\ddagger_{v\in V}(G_v : \mathsf{Type}(v)), < rv : Enumeration, client : Component >) =$

$\sum_{t\in T} \mathsf{Transition}_m(s,t)$

))

$\ddagger_{m_d=(I_c,sm)\in M_d}$ (

$Queue\_G_{m_d}(< q : List(Notification), locked : LockingState >) =$

$\boldsymbol{sum}\ < n : Notification > .(receiveNotification\_G_{m_d}(n).Queue\_G_{m_d}(q < |n, locked))$

$+(\#(q) > 0)\ \&\&\ ((locked ==< NONE >)\ \|\ (locked == LOCK''G_{sm}))$

$\rightarrow sendNotification\_G_{m_d}(head(q)).Queue\_G_{m_d}(tail(q), LOCK''G_{sm})$

$+\boldsymbol{sum}\ < s : LockingState > .((\#(q) == 0)\ \&\&\ ((locked ==< NONE >)$

$\|\ (locked == s))) \rightarrow lockQ\_r\_G_{m_d}(s).Queue\_G_{m_d}(q, s)$

$+\boldsymbol{sum}\ < s : LockingState > .((\#(q) == 0)\ \&\&\ ((locked ==< NONE >$

$\|\ (locked == s))) \rightarrow (lockQ\_r\_G_{m_d}(s)\ |\ emptyQ\_r\_G_{m_d}).Queue\_G_{m_d}(q, s)$

$\rightarrow (lockQ\_r\_G_{m_d}(s)\ |\ emptyQ\_r\_G_{m_d}).Queue\_G_{m_d}(q, s)$

$+(locked \neq< NONE >) \rightarrow (unlockQ\_r\_G_{m_d}.Queue\_G_{m_d}(q, < NONE >))$

$+(\#(q) == 0) \rightarrow qEmpty\_r\_G_{m_d}.Queue\_G_{m_d}(q, locked)$

$+(\#(q) == 0) \rightarrow emptyQ\_r\_G_{m_d}.Queue\_G_{m_d}(q, locked)$

$+(\#(q) > 0) \rightarrow qNonEmpty\_r\_G_{m_d}.Queue\_G_{m_d}(q, locked)$

)

In the above transformation, there are two types of processes as results. The bottom one instantiates the queues for all design models (see Figure 3.2), so we do this for all $m_d \in M_d$. As this is an existing implementation, please refer to [15] to obtain more details. The top one transforms behaviors of the combined models, so we do this for all $m \in M$. Besides, it is worth mentioning that rv is a process parameter that stores the reply value. And *client* is a special parameter prepared for storing the current client of this process with the *Component* type (see section 3.4.1). And for the transformation of the function $\mathbf{Transition}_m(t)$ is given as follows:

$$\mathsf{Transition}_m(s,tr,g,A,U,s') = \begin{cases} (g) \rightarrow \boldsymbol{sum}\ < x : Component > & if\ tr \in \mathcal{O}. \\ \quad .(\mathsf{Trigger}_m(tr).\dagger_{a\in A}\mathsf{Action}_m(a) & \wedge s \in S_{normal}. \\ \quad .\mathsf{Target}_m(tr, U, s')) & \\ (g) \rightarrow \mathsf{Trigger}_m(tr).\dagger_{a\in A}\mathsf{Action}_m(a). & Otherwise. \\ \quad \mathsf{Target}_m(s, tr, U, s') & \end{cases}$$

The sum operator is introduced with a variable x in the above transformation. Before we explain the reason that the sum operator is declared, we first explain a communication problem in the transformation. Let us imagine the combined models m as a server and two clients m' and m'' at the structured level. Note that the system executes in single-thread mode. This means that when the client m' calls the server m, all components except the server m are blocked until m sends a signal back to the client m' to indicate that the job has completed. In ALIAS, there is a mechanism to ensure such behavior must execute in order. However, the situation is different in mCRL2. If we do not apply some mechanisms, it is possible that the server m sends a signal back to another client m'', causing a deadlock to occur since the components are blocked forever.

So, the occurrence of the sum operator is a mechanism to ensure the correctness of the transformation. When the trigger tr is a control operation, usually in the form of '$\boldsymbol{sum} < x : Component > .(invoked'' G_{tr}(x)...)$', this means that it is called by a client. The sum operator here basically can take all the possible components that exist in this system into account. When such a call occurs, the process stores the calling component in the variable x for later use. See below for how the x is used in the Triggers and Replies transformation.

Recall the ALIAS system is $S_{complete} = (M_d, M_i, I_p, I_r, C)$ with $C = \{C_{in}, C_{out}, C_c\}$. And for all $m = \{I_c, sm\} \in M_d \cup M_i$, we conclude a set $I_c^{'}$ that represents all the control interfaces. Referring to Definition 4 in section 3.2, for each $i_c \in I_c^{'}$, $i_c = (O, N, R)$. Note, for any triggers and actions, they are essentially one of the control operations, the notifications, and the replies. Observe that for every action for $a \in A$ or every trigger $tr \in Tr$, there is exactly one control interface i_c. It is always possible to find a connection that this interface connects with another interface $i' \in \{I_c^{'}, I_r, I_p\}$. So, through the trigger $tr \in Tr$ or $a \in A$, we obtain a connection that $i_c \mapsto i'$. We assume the existences of a function $\mathsf{TrCon} : \mathcal{T}_r \mapsto \mathcal{C}$ that returns the connection for the trigger and a function $\mathsf{AcCon} : \mathcal{A} \mapsto \mathcal{C}$ that returns the connection for the action.

$$\mathsf{Trigger}_m(tr) = \begin{cases} G_{tr} \mid emptyQ\_s \mid lockQ\_s\_G_m(LOCK'' G_m) & if\ tr \in \mathcal{O}. \\ \quad \mid lock\_p\_thread & \quad \wedge \mathsf{TrCon}(tr) \in C_{in}. \\ invoked'' G_{tr}(< x >) \mid lockQ\_s\_G_m(LOCK'' G_m) & if\ tr \in \mathcal{O}. \\ & \quad \wedge \mathsf{TrCon}(tr) \in C_c \\ readNotification\_G_m(G_{tr}) \mid lock\_p\_thread & if\ tr \in \mathcal{N}. \\ \quad \mid queue\_unlock\_p\_thread(G_m) & \\ internal \mid emptyQ\_s \mid lockQ\_s\_G_m(LOCK'' G_m) & otherwise. \\ \quad \mid lock\_p\_thread & \end{cases}$$

An action a occurs in the sequence A, and it can be found in the list that defines all actions in section 3.4.2. The transformation of actions leads to different results based on the behavior of the action. Similar to the function AcCon introduced before, given the combined model m, for every action a, there is exactly one interface, and this interface has a connection with another interface i'. In fact, i' belongs to one of the combined models, so through an action, it is always possible to find its server model. So we introduce a function $\mathsf{AcServer} : \mathcal{A} \mapsto \mathcal{M}$ that returns the server model of the action. Furthermore, given the combined model m, it is possible to find multiple clients of this model. So we define a function $\mathsf{Clients} : \mathcal{M} \mapsto \{2^{\mathcal{M}}\}$ that returns the set of clients of the combined model.

$$\mathsf{Action}_m(a) = \begin{cases} invoke'' G_a(G_m) & if\ a \in \mathcal{O} \wedge rv = void \\ \quad .readReply(rv, G_{\mathsf{AcServer}(a)}, G_m) & \\ invoke'' G_a(G_m).\boldsymbol{sum} < var : Enumeration > & if\ a \in \mathcal{O} \wedge rv \neq void \\ \quad .(readReply(var, G_{m'}, G_m)) & \\ \sum_{m' \in \mathsf{Clients}(m)} pushNotification\_G_{m'(G_a)} & if\ a \in \mathcal{N} \wedge \mathsf{AcCon}(a) \in C_c \\ outwardNotification(G_a) & if\ a \in \mathcal{N} \wedge \mathsf{AcCon}(a) \in C_{in} \\ outwardReply(rv) & if\ a \in \mathcal{R} \wedge \mathsf{AcCon}(a) \in C_{in} \end{cases}$$

For these transformations, there are two points worth noting: one is when the action belongs to the control operation ($a \in \mathcal{O}$), and the other is when the action belongs to the notification ($a \in \mathcal{N}$).

Referring to Figure 3.2, when $a \in \mathcal{O}$, the component which executes this 'invoke' action is recognized as a client model, and the component which is 'invoked' is recognized as a server model. Based on this, the three arguments in *readReply* can be explained: a reply value rv, the

globally unique name of the sever model $G_{\mathbf{AcServer}(a)}$, and the globally unique name of the client model G_m.

Something different happens when $a \in \mathcal{N}$. Referring to Figure 3.2 again, the component which executes the '*pushNotification*' action is recognized as a server model now. This action pushes the notification to one of its clients. It is a very similar situation as we discussed before: the server is unaware of which client is requesting the notification. So, a sum operator is used here to make the program choose the right client at run-time.

The $s' \in \mathcal{S}$ is the target state of this transition t owned by the source state s. The function $\mathsf{Target}_m(s, tr, U, s')$ introduces two different transformations under different situations.

$$\mathsf{Target}_m(s,tr,U,s') = \begin{cases} qEmpty\_s\_G_m.\mathsf{Reply}_m(tr,rv) & \text{if } s \in S_{eval} \\ \quad .(G_{s'}(U, NoReplyValue, \mathsf{Update}(tr,s,s',\mathbf{true})) & \quad \vee m \in M_d \\ \quad +qNonEmpty\_s\_G_m & \\ \quad .(unlock\_p\_thread | queue\_lock\_p\_thread(G_m)) & \\ \quad .G_{s'}(U, rv, \mathsf{Update}(tr,s,s',\mathbf{false})) & \\ qEmpty\_s\_G_m.(rv! = NoReplyValue) & \text{if } \neg(s \in S_{eval} \\ \quad \rightarrow \mathsf{Reply}_m(tr,rv) & \quad \vee m \in M_d) \\ \quad \diamond unlockQ\_s\_G_m.unlock\_p\_thread. & \\ \quad .G_{s'}(U, NoReplyValue, \mathsf{Update}(tr,s,s',\mathbf{true})) & \\ \quad +qNonEmpty\_s\_G_m & \\ \quad .(unlock\_p\_thread | queue\_lock\_p\_thread(G_m)) & \\ \quad .G_{s'}(U, rv, \mathsf{Update}(tr,s,s',\mathbf{true})) & \end{cases}$$

The function $\mathsf{Target}_m(s, tr, U, s')$ includes of two sub functions. One is $\mathsf{Reply}_m(tr, rv)$, this function is used to send different replies based on the connection type between the component and its client (could be the external enviroment).

$$\mathsf{Reply}_m(tr,rv) = \begin{cases} writeReply(rv, G_m, < x >) \mid unlockQ\_s\_G_m & \text{if } \mathsf{TrCon}(tr) \in C_c \\ & \quad \wedge(s \in S_{eval} \vee m \in M_d) \\ writeReply(rv, G_m, < client >) & \text{if } \mathsf{TrCon}(tr) \in C_c \\ \quad \mid unlockQ\_s\_G_m & \quad \wedge\neg(s \in S_{eval} \vee m \in M_d) \\ (outwardReply(rv) \mid unlockQ\_s\_G_m) & \text{if } \mathsf{TrCon}(tr) \in C_{in} \\ \quad .unlock\_p\_thread & \end{cases}$$

The other one $\mathsf{Update}(tr, s, s', b)$ is used to update the parameter '*client*' of the target state. The fourth parameter b in the function is a boolean, which helps to determine if the value stored in the sum operator x is required by the target state. See the **false** in the function $\mathsf{Target}_m(s, tr, U, s')$, it happens when the reply has not been processed yet and a delayed reply is required, so the parameter x needs to be passed to the target state and then processed.

$$\mathsf{Update}(tr,s,s',b) = \begin{cases} < client > & \text{if } s = s' \\ < x > & \text{if } tr \in \mathcal{O} \wedge (s' \in S_{eval} \vee b = \mathbf{false}) \\ < NoClient > & \text{otherwise} \end{cases}$$

3.4.4 Initialization

The initialization in mCRL2 has a typical form. An allow operator applied to a communication operator which in turn is applied to parallel processes. Given the same ALIAS system $S_{complete} = (M_d, M_i, I_p, I_r, C)$, we obtain $M = M_d \cup M_i$ again. Through these combined models, it is possible to find all control operations O of this system. The following function shows the corresponding transformations
$\mathsf{InitDef}(M, O) =$

init

$\quad$ **allow**($\mathsf{Allow}(M)$, **comm**($\mathsf{Communication}(M, O)$, $\mathsf{Parallel}(M)$))

The allow operator is used to declare what multi-actions are allowed to perform. This is constructed by the function

$\mathsf{Allow}(M) =$

$\{unlock\_thread, unlock\_thread \mid queue\_lock\_thread, initialize, valuedTrigger,$
$outwardNotification, sendReply,$
$\dagger_{m\in M}(optionalInternalTrigger \mid lockQ\_G_m \mid lock\_thread \mid emptyQ),$
$\dagger_{m\in M}(inevitableInternalTrigger \mid lockQ\_G_m \mid lock\_thread \mid emptyQ),$
$\dagger_{m\in M}(lockQ\_G_m \mid lock_thread|emptyQ),$
$\dagger_{m\in M}(outwardReply \mid unlockQ\_G_m),$
$\dagger_{m\in M}(sendReply \mid unlockQ\_G_m),$
$\dagger_{m\in M}(triggerNotification\_Gm \mid lock\_thread \mid queue\_unlock\_thread),$
$\dagger_{m\in M}(raiseNotification\_G_m), \dagger_{m\in M}(unlockQ\_G_m),$
$\dagger_{m\in M}(qEmpty\_G_m), \dagger_{m\in M}(qNonEmpty\_G_m)\}$

The communication operator constructed by the function $\mathsf{Communication}(M, O)$ is used to declare what actions communicate or synchronize.

$\mathsf{Communication}(M, O) =$

$\{readReply \mid writeReply \rightarrow sendReply, lock\_t\_thread \mid lock\_p\_thread \rightarrow lock\_thread,$
$unlock\_t\_thread \mid unlock\_p\_thread \rightarrow unlock\_thread,$
$queue\_lock\_t\_thread \mid queue\_lock\_p\_thread \rightarrow queue\_lock\_thread,$
$queue\_unlock\_t\_thread \mid queue\_unlock\_p\_thread \rightarrow queue\_unlock\_thread,$
$\dagger_{m\in M}(lockQ\_s\_G_m \mid lockQ\_r\_G_m \rightarrow lockQ\_G_m),$
$\dagger_{m\in M}(unlockQ\_s\_G_m \mid unlockQ\_r\_G_m \rightarrow unlockQ\_G_m), |_{m\in M}\ emptyQ\_G_m \rightarrow emtpyQ,$
$\dagger_{m\in M}(qEmpty\_s\_G_m \mid qEmpty\_r\_G_m) \rightarrow qEmpty\_G_m,$
$\dagger_{m\in M}(qNonEmpty\_s\_G_m \mid qNonEmpty\_r\_G_m) \rightarrow qNonEmpty\_G_m,$
$\dagger_{m\in M}(sendNotification\_G_m \mid readNotification\_G_m \rightarrow triggerNotification\_G_m),$
$\dagger_{m\in M}(pushNotification\_G_m \mid receiveNotification\_G_m \rightarrow raiseNotification\_G_m),$
$\dagger_{o\in O}(invoke'' G_o \mid invoked'' G_o \rightarrow G_o)\}$

And the parallel part is shown as below:

$\mathsf{Parallel}(M) =$

$Thread \parallel queue\_thread \parallel (\|_{m\in M}\ Queue\_G_m([], < NONE >)) \parallel$
$(\|_{m\in M}\ G_{s_{init}}(v_{init}, < NoReplyValue, NoClient >))$

3.4.5 The full transformation

The transformation of an ALIAS model is divided into four pieces: Sort, Action, Process, Initialization. Each of them has been discussed. We summarize the whole transformation in this section.

Given an ALIAS system $S_{complete} = (M_d, M_i, I_p, I_r, C)$. The system consists of design models M_d and M_i. A set M representing for combined models should be $M = M_d \cup M_i$. Through these combined models, it is possible to find all control operations O, notifications N and replies R Furthermore, each combined model corresponds to a state machine. So SM represents all state machines of this system.

The function $\mathsf{Transformation}(M_d, M_i, I_p, I_r, C)$ concludes the whole transformation.

$\mathsf{Transformation}(M_d, M_i, I_p, I_r, C) =$
$\mathsf{SortDef}((M_d \cup M_i), SM, N, R); \quad \mathsf{ActionDef}((M_d \cup M_i), O);$
$\mathsf{ProcessDef}((M_d, M_i, I_p, I_r, C); \quad \mathsf{InitDef}((M_d \cup M_i), O);$

Chapter 4

Validation

In the previous chapter, the transformation from an ALIAS system to an mCRL2 model has been defined. It is crucial to ensure the transformation is correct. In this chapter, we firstly introduce the way to validate the transformation. Then several ASD models are transformed into mCRL2 models.

4.1 Validation method

To verify the result of the transformation, we choose to develop the project in Eclipse version Mars.2, an integrated development environment (IDE). Eclipse supports various tools and an interface to develop software and models.

One of the tools supported by the Eclipse IDE is Query View Transformation operational (QVTo). It is a language to define M2M transformations at a meta-model level. We show a simple example in the following listing to explain how to transform one model to another at the meta-model level. The first two lines declare the two meta-models A and B. Line 4 means that the transformation starts from the source meta-model A to the target meta-model B. In the 'main' function, given an instance model of the meta-model A, it invokes a function nameTransformation(). The function nameTransformation() uses a name element of A as an input and outputs a name element of B as a result.

```
1  modeltype A "strict" uses A ('http://A.com');
2  modeltype B "strict" uses B ('http://B.com');
3
4  transformation M2M(in source: A, out target: B);
5
6  main(){
7     source.rootObjects()[A::name] -> map nameTransformation();
8  }
9
10 mapping A::name::nameTransformation():B::name{
11     result.name := self.name;
12 }
```

The operators in QVTo are very similar to the ones from other languages. Table 4.1 gives several operators with their descriptions that are used in this work.

Description	Operator
Terminate	;
Assignment	:=
Sum	+
Minus	−
Set addition	+=
Equal to	=
Dot accessor	.
Arrow accessor	→

Table 4.1: Operators in QVTo

Furthermore, two important QVTo statements used in the project are listed below. The first one is the *if/else* statement. The *if/else* statement checks the boolean condition. If the result is **true**, the program executes statement 1. Otherwise, statement 2 is executed. Like other languages, the *else* part can be omitted.

```
1 if (condition) then
2  %-- Statement 1 --%
3 } else {
4  %-- Statement 2 --%
5 } endif;
```

Another statement is *while-loop*. While the condition is true, the *while-loop* keeps executing the statement. It is possible to use a keyword *continue* to skip the remaining part of the statement and restart the statement again. And it is also possible to exit the *while-loop* by using a keyword *break*.

```
1 while (condition) {
2  %-- Statement --%
3  %-- possible to use continue--%
4  %-- possible to use break--%
5 };
```

With the help of QVTo, we can obtain an mCRL2 model from an ALIAS system automatically. Not only does this save time, but it also ensures that our resulting model should not generate unexpected behaviors for some manual errors.

Each resulting mCRL2 model is considered as a combined mCRL2 model that can perform the behaviors of the whole system. The results of the mCRL2 models are converted to labelled transition systems (LTS) by the tools of mCRL2. The LTS can be be used to visualize state space or verify properties with the help of the modal μ-calculus. In the following, we highlight certain words. All letters using italics represent actions, notifications, and replies. And all letters using bold stand for components.

4.2 The kettle system

4.2.1 Model description

The kettle system is the first model to be validated. Its structure is shown in Figure 4.1. It is worth noting that '**IM: Kettle**' is the associated interface model of the design model '**DM: Kettle**'. So according to the results discussed in section 3.1, the interface model will be replaced by its associated design model when the transformation performs.

In the kettle system, there are two components: a **Kettle** for controlling ('**DM:Kettle**') and a **Heater** ('**IM:Heater**') . Specifications of the two components are described in Figure 4.2 and Figure 4.3. The **Kettle** can control the **Heater** to turn on and turn off, and it also can perform an *Abort* action to turn the whole system off.

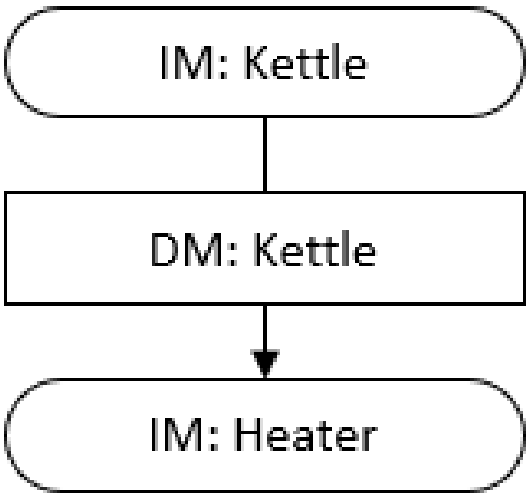

Figure 4.1: The ASD model of kettle

When the **Kettle** in the idle state, one can turn both the **Kettle** and the **Heater** on. When the **Kettle** starts on, two choices are available: turn the **Heater** off or start heating. The **Kettle** can also be turned off during the heating. If the heating is not aborted, the **Kettle** will wait for the entire heating process to end and then back to the idle state.

	Interface	Event	Guard	Actions	State Variable Updates	Target State
1	**Idle (initial state)**					
4	A	Heat		Heater:A.TurnOn; A.VoidReply		Started
8	**Started**					
12	A	Abort		Heater:A.TurnOff; A.VoidReply		Aborting
13	Heater:N	TurnedOn		NoOp		Heating
15	**Aborting**					
19	A	Abort		A.VoidReply		Aborting
20	Heater:N	TurnedOn		NoOp		Aborting
21	Heater:N	TurnedOff		N.Done		Idle
22	**Heating**					
26	A	Abort		Heater:A.TurnOff; A.VoidReply		Aborting
28	Heater:N	TurnedOff		N.Done		Idle

Figure 4.2: The specifications of the kettle model **DM: Kettle**

	Interface	Event	Guard	Actions	State Variable Updates	Target State
1	**Off (initial state)**					
3	A	TurnOn		A.VoidReply		TurningOn
4	A	TurnOff		A.VoidReply		Off
6	**TurningOn**					
8	A	TurnOn		A.VoidReply		TurningOn
9	A	TurnOff		N.TurnedOn; A.VoidReply		TurningOff
10	M	Eventually		N.TurnedOn		On
11	**TurningOff**					
13	A	TurnOn		N.TurnedOff; A.VoidReply		TurningOn
14	A	TurnOff		A.VoidReply		TurningOff
15	M	Eventually		N.TurnedOff		Off
16	**On**					
18	A	TurnOn		A.VoidReply		On
19	A	TurnOff		A.VoidReply		TurningOff
20	M	Eventually		N.TurnedOff		Off

Figure 4.3: The specifications of the kettle model **IM: Heater**

4.2.2 State space and discussion

The mCRL2 model of the kettle system can be found in Appendix C. Figure 4.4 shows the result of the state space and uninteresting actions are hidden. The green state is the initial state. It is not hard to see that after the **Kettle** is invoked by an event *Heat* from the external environment, it starts on and triggers the **Heater** to be turned on at the same time. When the **Kettle** has been started, we can see two different traces. One is going to perform the *Abort* action to turn off the **Heater** and make the **Kettle** idle eventually. Another one is that the **Kettle** is ready to perform heating after it is notified by the **Heater**. After the heating process is done or the process is aborted during the heating process, all the traces point to the state near the initial state and constitute a loop. It is interesting to see that there is a live lock in this state space. This loop is caused by the *Abort* action, line 19 in figure 4.2 implies that such a process can be executed repeatedly. Because the target state of this SBS is still its self: Aborting.

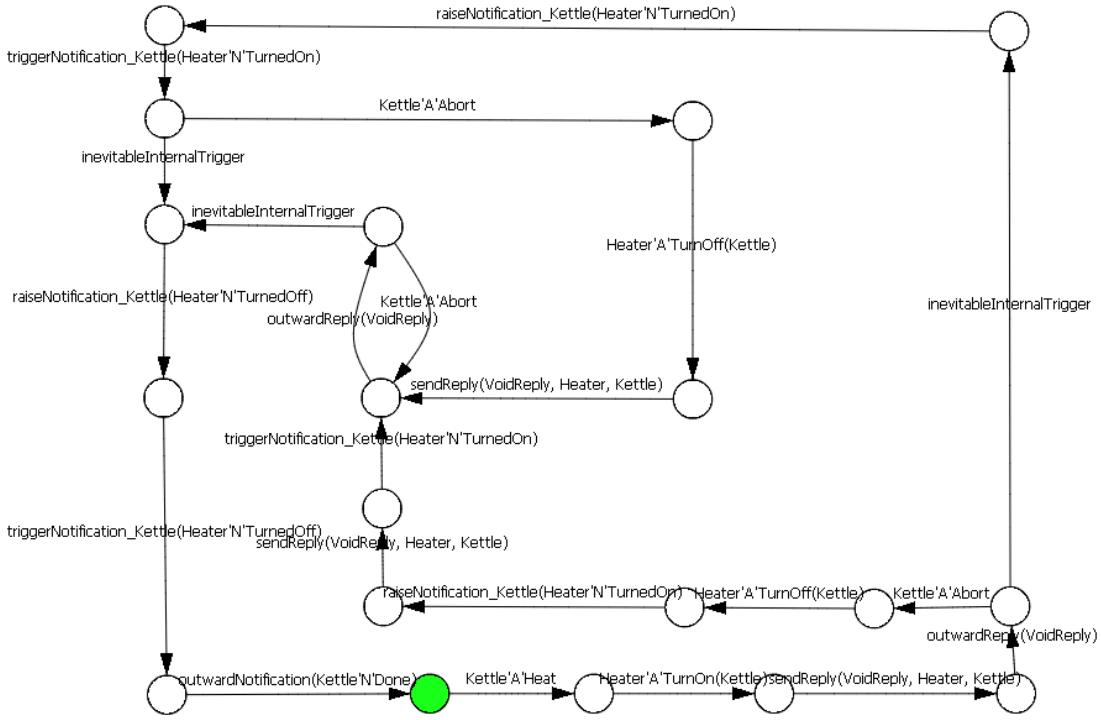

Figure 4.4: The state space of the kettle model

4.3 Lamp

4.3.1 Model description

A model that simulates a simple lamp is the second candidate for evaluation. As shown in Figure 4.5, it consists of 4 components. In the same way as for the kettle system, '**IM:LampSystem**' will be replaced by its associated design model during the transformation.

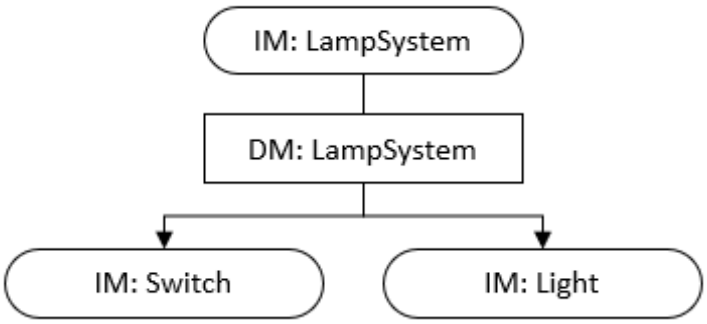

Figure 4.5: The ASD model of lamp

With the help the of specifications of the system shown in Figure 4.6, 4.7, and 4.8, it is clear to see that the **LampSystem** can be considered as a controller that controls the **Switch** and the **Light**. The **Switch** is used to send *OnReply* or *OffReply* to the **LampSystem**, and the **Light** performs actions when it receives the instructions from the **LampSystem**. So, the **Light** is turned on after the **Switch** sends an *OnReply*. The **Light** is turned off after the **Switch** sends an *OffReply* and directly after the **LampSystem** is initialized.

Init	Init (initial state)					
	Interface	Event	Guard	Actions	State Variable Updates	Target State
1	**Init (initial state)**					
4	A	Initialize		Switch:A.Initialize; Light:A.Initialize; A.VoidReply		Off
5	A	Update		A.VoidReply		Init
8	**Off**					
12	A	Update		Switch:A.Poll+		Off_Polling
15	**Off_Polling (synchronous return state)**					
20	Switch:A	OnReply		Light:A.SwitchOn; A.VoidReply		On
21	Switch:A	OffReply		A.VoidReply		Off
22	**On**					
26	A	Update		Switch:A.Poll+		On_Polling
29	**On_Polling (synchronous return state)**					
34	Switch:A	OnReply		A.VoidReply		On
35	Switch:A	OffReply		Light:A.SwitchOff; A.VoidReply		Off

Figure 4.6: The specifications of the lamp model **DM: LampSystem**

Init	Init (initial state)					
	Interface	Event	Guard	Actions	State Variable Updates	Target State
1	Init (initial state)					
3	A	Initialize		A.VoidReply		OffState
6	OffState					
9	A	SwitchOn		A.VoidReply		OnState
10	A	SwitchOff		A.VoidReply		OffState
11	OnState					
14	A	SwitchOn		A.VoidReply		OnState
15	A	SwitchOff		A.VoidReply		OffState

Figure 4.7: The specifications of the lamp model **IM: Light**

Init	Init (initial state)					
	Interface	Event	Guard	Actions	State Variable Updates	Target State
1	Init (initial state)					
3	A	Initialize		A.VoidReply		Main
5	Main					
8	A	Poll+		A.OnReply		Main
9	A	Poll+		A.OffReply		Main

Figure 4.8: The specifications of the lamp model **IM: Switch**

4.3.2 State space and discussion

The mCRL2 model of the lamp model can be found in Appendix D. Figure 4.9 shows the generated state space after the transformation and uninteresting actions are hidden. Furthermore, branching bi-simulation reduction [8] is applied to reduce the size of the state space. From the initial state,

the Loop 0 keeps executing *Update* refers to line 5 in Figure 4.6, it means before the system is initialized, the system can always update its status.

And when the system is initialized, the **Switch** has two options: it sends an *OffReply* or sends an *OnReply* to the **LampSystem**. Only the option *OnReply* can leave the Loop 1. Once the **LampSystem** receives the *OnReply*, it turns on the light. After that, the two options occur again, only after the *OffReply* can trigger the system to leave the Loop 2.

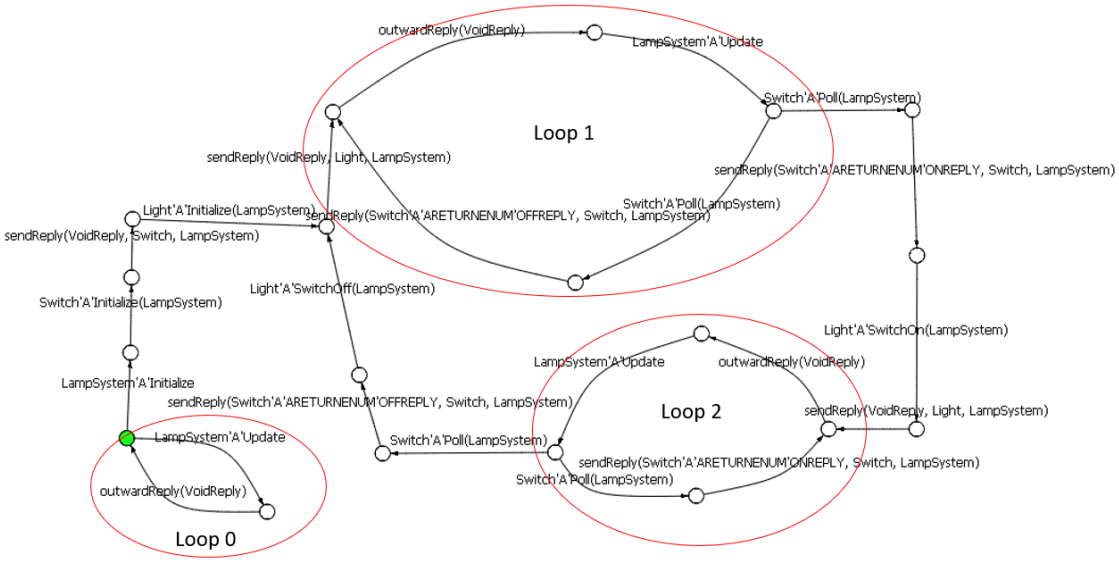

Figure 4.9: The state space of the lamp model

4.4 Assembly line

4.4.1 Model description

As we mentioned in section 3.4.3, in a complicated system that has multiple components, one problem could happen in communications. If a server has multiple clients, it could not recognize which one is its target client when it performs replies or notifications. The solution put forward in section 3.4.3 should be validated to make sure that the transformation is flawless.

A slightly more complicated model shown in Figure 4.10 helps to validate the above problem. Basically, this model simulates a simple assembly line. As the highest level component in this system, **ControlUnit** is more like a manager of the system. **UnitA** and **UnitB** act as executors to produce objects. **LoaderUnit** and **RobotArm** can take the produced objects out of the assembly line. As long as the system does not give error feedback, the assembly line keeps producing objects.

Similar to the above cases, except '**IM: RobotArm**', all interface models are replaced by their associated design models. After the combination, '**DM: UnitA**' and '**DM: UnitB**' share the same server '**DM: LoaderUnit**'.

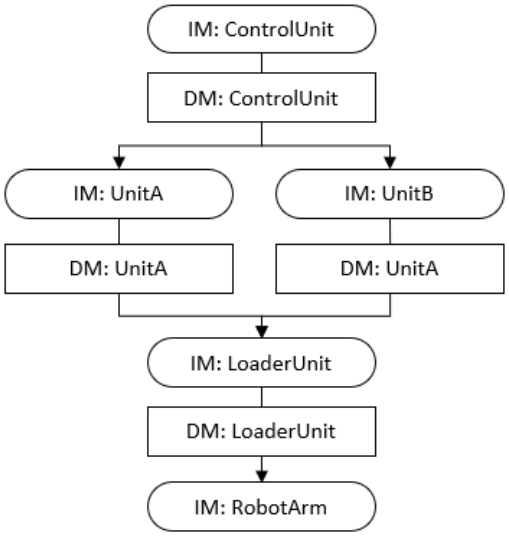

Figure 4.10: The ASD model of assemble

An interesting specification of the assembly line model is shown in Figure 4.11. After activation, the top one '**DM: ControlUnit**' will send instructions to '**DM: UnitA**' and '**DM: UnitB**' to produce objects and counts the quantities they produce. While the bottom one '**DM: LoaderUnit**' will aggregate the objects produced on '**DM: UnitA**' and '**DM: UnitB**' for use by '**IM: RobotArm**'. Furthermore, the guards in line 15 and 17 imply that there are buffers to restrict the production amount. More precisely, when there are 2 objects in the buffer of '**DM: UnitA**' that have not been loaded, the system will not keep producing. Similarly, the '**DM: UnitB**' only produces objects when its buffer is smaller than 3. For the other specifications of this model, refer to Appendix B.

Initial	Initial (initial state)					
	Interface	Event	Guard	Actions	State Variable Updates	Target State
1	**Initial (initial state)**					
4	AI	Initialize		MachineA:AI.Initialize; MachineB:AI.Initialize; AI.VoidReply		Ready
11	**Ready**					
15	AI	MakeA+	countA < 2	MachineA:AI.Make; AI.OKReply	countA++	Ready
16	AI	MakeA+	otherwise	AI.FAILReply		Ready
17	AI	MakeB+	countB < 3	MachineB:AI.Make; AI.OKReply	countB++	Ready
18	AI	MakeB+	otherwise	AI.FAILReply		Ready
19	MachineA:NI	Failed		NI.Error		Error
20	MachineA:NI	Done		NI.MadeA	countA--	Ready
21	MachineB:NI	Failed		NI.Error		Error
22	MachineB:NI	Done		NI.MadeB	countB--	Ready
23	**Error**					
27	AI	MakeA+		AI.FAILReply		Error
28	AI	MakeB+		AI.FAILReply		Error
29	MachineA:NI	Failed		NoOp		Error
30	MachineA:NI	Done		NoOp		Error
31	MachineB:NI	Failed		NoOp		Error
32	MachineB:NI	Done		NoOp		Error

Figure 4.11: The specifications of the assembly line model **DM: ControlUnit**

4.4.2 Properties and discussion

The mCRL2 model of the assembly line model can be found in Appendix E. As models become more complex, their corresponding state space becomes larger. For the assembly line model, after eliminating uninteresting actions and performing branching bi-simulation reduction, there are still more than a hundred transitions that cannot be visualized clearly. Therefore, it is particularly essential to use modal μ-calculus to verify its properties.

We formulate 7 properties and list them in Table 4.2. Except for the first property, which is about deadlock freedom, all concern end-to-end behavior. The properties are formulated as μ-calculus formulas for verification as well.

In these formulas, several common patterns in the μ-calculus are used. The first one, $[true^*]\phi$, means that ϕ holds after any sequence of actions. Based on this, $[true^*.(\overline{a}^* + \overline{b}^*).c]false$ is built to express that from any trace, if both a and b do not happen, c cannot execute. This helps to formulate property 1. A more common but complex pattern is $\mu X.[\overline{a}]X \wedge \langle true\rangle true$. This pattern expresses that the action a must eventually be done after finite number of steps. To formulate the property 7, we use data parameters. In this formula, $(A : \mathbb{N} := 0, B : \mathbb{N} := 0)$ means that two user-defined parameter A as well as B are natural numbers and are initialized to 0. Furthermore, a pattern $\neg\langle a\rangle true$ is used in formula 7 to express that the action a cannot execute.

Table 4.2: Properties of the assembly line written in μ-calculus

	Property	Formula
0	Deadlock free	$[true^*]\langle true\rangle true$
1	The **LoaderUnit** does not *Load* until both **UnitA** and **UnitB** have been *Initialized*.	$[true^*]$ $[(\overline{MachineA'AI'Initialize}^* + \overline{MachineB'AI'Initialize}^*).$ $Loader'AI'Load]false$
2	After the **ControlUnit** was *Initialized* both **UnitA** and **UnitB** must eventually must do *Make*.	$[true^*.Controller'AI'Initialize]($ $\mu X.([\overline{MachineB'AI'Make}]X \wedge \langle true\rangle true) \wedge$ $\mu X.([\overline{MachineA'AI'Make}]X \wedge \langle true\rangle true))$
3	If the **LoaderUnit** has sent a *FAILReply* or *Failure* notification, eventually the **ControlUnit** must return a *FAILReply*.	$[true^*]$ $[writeReply(Loader'AI'FAILREPLY, LoaderUnit, UnitA)+$ $raiseNotification\_UnitA(Loader'NI'Failure)+$ $[writeReply(Loader'AI'FAILREPLY, LoaderUnit, UnitB)+$ $raiseNotification\_UnitB(Loader'NI'Failure)]$ $\overline{\mu X.[\overline{outwardReply(Controller'AI'FAILREPLY)} \mid \overline{unlockQ\_ControlUnit}]X \wedge \langle true\rangle true}$
4	When the **LoaderUnit**, **UnitA** or **UnitB** raised *Failure* or *Failed* notifications, the **ControlUnit** must eventually raise an *Error* notification.	$[true^*]$ $[raiseNotification\_ControlUnit(MachineA'NI'Failed)+$ $raiseNotification\_ControlUnit(MachineB'NI'Failed)+$ $raiseNotification\_UnitA(Loader'NI'Failure)+$ $raiseNotification\_UnitB(Loader'NI'Failure)]$ $\overline{\mu X.[\overline{outwardNotification(Controller'NI'Error)}]X} \wedge$ $\langle true\rangle true$
5	The second consecutive *MakeA* request the **ControlUnit** sends must produce a *FAILReply*.	$[true^*.Controller'AI'MakeA.$ $\overline{triggerNotification\_ControlUnit(MachineA'NI'Done)}^*.$ $Controller'AI'MakeA]$ $\overline{\mu X.[\overline{outwardReply(Controller'AI'FAILREPLY)} \mid \overline{unlockQ\_ControlUnit}]X} \wedge \langle true\rangle true$
6	The third consecutive *MakeB* request the **ControlUnit** sends must produce a *FAILReply*.	$[true^*.Controller'AI'MakeB$ $\overline{triggerNotification\_ControlUnit(MachineB'NI'Done)}^*.$ $Controller'AI'MakeB.$ $\overline{triggerNotification\_ControlUnit(MachineB'NI'Done)}^*.$ $Controller'AI'MakeB]$ $\overline{\mu X.[\overline{outwardReply(Controller'AI'FAILREPLY)} \mid \overline{unlockQ\_ControlUnit}]X} \wedge \langle true\rangle true$
7	After at most two *MakeA* and three *MakeB* requests, all consecutive, the **ControlUnit** must not produce a *FAILReply*.	$\nu X(A:\mathbb{N}:=0, B:\mathbb{N}:=0).($ $\overline{[\overline{Controller'AI'MakeA} \parallel Controller'AI'MakeB \parallel \overline{triggerNotification\_ControlUnit(MachineA'NI'Done)} \parallel \overline{triggerNotification\_ControlUnit(MachineB'NI'Done)} \parallel \overline{outwardNotification(Controller'NI'Error)}]X(A,B)} \wedge$ $[Controller'AI'MakeA]X(A+1,B) \wedge$ $[triggerNotification\_ControlUnit(MachineA'NI'Done)]$ $X(A-1,B) \wedge$ $[Controller'AI'MakeB]X(A,B+1) \wedge$ $[triggerNotification\_ControlUnit(MachineB'NI'Done)]$ $X(A,B-1) \wedge ((A<2 \wedge\ B<3) \Rightarrow$ $\neg\langle outwardReply(Controller'AI'FAILREPLY) \mid$ $unlockQ\_ControlUnit\rangle true))$

Table 4.3 concludes the verification results by using mCRL2 extensive tools. Except for the properties 0 and 4, all the verification results are **true**.

Table 4.3: Properties of the assembly line written in μ-calculus

	Property	Result
0	Deadlock free	**false**
1	The **LoaderUnit** does not *Load* until both **UnitA** and **UnitB** have been *Initialized*.	**true**
2	After the **ControlUnit** was *Initialized* both **UnitA** and **UnitB** must eventually must do *Make*.	**true**
3	If the **LoaderUnit** has sent a *FAILReply* or *Failure* notification, eventually the **ControlUnit** must return a *FAILReply*.	**true**
4	When the **LoaderUnit**, **UnitA** or **UnitB** raised *Failure* or *Failed* notifications, the **ControlUnit** must eventually raise an *Error* notification.	**false**
5	The second consecutive *MakeA* request the **ControlUnit** sends must produce a *FAILReply*.	**true**
6	The third consecutive *MakeB* request the **ControlUnit** sends must produce a *FAILReply*.	**true**
7	After at most two *MakeA* and three *MakeB* requests, all consecutive, the **ControlUnit** must not produce a *FAILReply*.	**true**

Property 0 expresses whether the system is deadlock-free. The property is also checked by the ASD verification tool, which says it is deadlock-free. The difference between ASD verification and our result can be explained as follows. In fact, the mechanism to check deadlock in ASD is: each component by itself can be checked for deadlock-freedom, and the components can only be composed in ways that have been proven not to introduce deadlocks. However, this mechanism is under the assumption that one interface model that is used by one design model, which is not the case in this assembly line model. In the assembly line model, both '**DM:UnitA**' and '**DM:UnitB**' use the same interface model '**IM:LoaderUnit**', so the above assumption is invalid. mCRL2 can overcome such a weakness in verification. This is also a reason that we implement the transformation from ASD to mCRL2 to verify properties.

Regarding property 4, it describes that when the **LoaderUnit**, **UnitA** or **UnitB** raised a *Failure* or *Failed* notification, the **ControlUnit** must eventually raise an *Error* notification. The property results in **false** which can be explained. If the **UnitA** and **UnitB** send *Failure* notifications to the **ControlUnit** successively, the **ControlUnit** firstly receives a *Failure* notification from the **UnitA** (see Figure 4.11, line 19), and then enters the error state. After that, the **ControlUnit** receives a *Failure* notification from the **UnitB** (see Figure 4.11, line 31), and it does not perform any operation (*NoOP*), which proves that this property does not hold for this implementation.

Property 1 is easy to understand because the **LoaderUnit** only starts working after the components at a higher-level has been initialized. Property 2 says once the **ControlUnit** is activated, the working units must *Make* some objects. Property 3 means that once a lower-level component gives a fail reply or notification, the highest level component **ControlUnit** must send a *FAILReply* to the external environment. The buffer limitations of working units are explained in property 5 and 6. And property 7 expresses that if both the buffer of the **UnitA** and **UnitB** do not reach the limitation, then the system will not give a fail reply to the extended environment.

The above verification results of these properties not only tell us that the results reach our expectations but also reflect the limitations of the ASD verification.

Chapter 5

Conclusions

In this work, we have defined the transformation from ALIAS to mCRL2. This transformation is a part of the transformations from ASD to mCRL2. Based on our work, properties that cannot be verified on ASD verification can be verified on mCRL2.

To realize the transformation, we described the language specification of ALIAS, which are the fundamental elements that every model in ALIAS should be composed of these specifications. The transformation from ALIAS to mCRL2 was defined abstractly with the help of the specifications. Furthermore, this transformation has been dramatically improved compared to the previous works of [10] and [15]. Because it can not only transform multi-components based ALIAS models, but the transformation process is automatic.

Next, the transformation was implemented by a model to model transformation through QVTo. Three ASD models were used for the validation. The models were transformed into the models in mCRL2. With the help of some mCRL2 tools, they were converted to labeled transition systems. These labeled transition systems were visualized. By inspecting the visualization of these labeled transition systems and checking properties by the modal μ-calculus, we observed that the transformation worked as expected.

The verification results are in line with our expectations that the behaviors of the output mCRL2 models match the expected behaviors of the input ASD models. It indicates that the result of the automatic transformation for multi-component systems from ALIAS to mCRL2 realizes more possibilities for checking models on this scale. It means more interesting properties can be checked. Based on this, applying this transformation to a wider range of scenarios is also a potential possibility, such as using it to verify larger systems that currently exist.

In this work, only several models were checked. For a larger system verification, please refer to [15]. Future research should examine more complex systems to validate the transformation. Furthermore, because the tested models are relatively small, one shortcoming has not been revealed. It is precisely that as the scale of the system gets larger, the number of parallel components will increase, which will cause the verification to become extremely slow. Future research should consider the potential effects of this weakness more carefully.

Bibliography

[1] A. Annichini, A. Bouajjani, and M. Sighireanu. Trex: A tool for reachability analysis of complex systems. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Computer Aided Verification*, pages 368–372, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. 3

[2] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge University Press, New York, NY, USA, 1990. 3

[3] M. Bartholomeus, B. Luttik, and T.A.C Willemse. Modelling and analysing ertms hybrid level 3 with the mcrl2 toolset. In Falk Howar and Jiří Barnat, editors, *Formal Methods for Industrial Critical Systems*, pages 98–114, Cham, 2018. Springer International Publishing. 3

[4] G.H. Broadfoot. Asd case notes: Costs and benefits of applying formal methods to industrial control software. In J. Fitzgerald, I.J. Hayes, and A. Tarlecki, editors, *FM 2005: Formal Methods*, pages 548–551, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 5

[5] G.H. Broadfoot and J.H. Philippa. Analytical software design. *Tech. rep., Verum Consultants B.V.*, 2003. 5

[6] O. Bunte, J.F. Groote, J.J.A. Keiren, M. Laveaux, T. Neele, E.P. de Vink, W. Wesselink, A. Wijs, and T.A.C. Willemse. The mcrl2 toolset for analysing concurrent systems. In Tomáš Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 21–39, Cham, 2019. Springer International Publishing. 9

[7] J.F. Groote and M.R. Mousavi. *Modeling and analysis of communicating systems*. MIT Press, 2014. 9, 10

[8] J.F. Groote and A. Wijs. An o(m log n) algorithm for stuttering equivalence and branching bisimulation. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 607–624, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. 32

[9] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985. 3

[10] R.J.W. Jonk. The semantics of ALIAS defined in mCRL2. Master's thesis, Eindhoven University of Technology, 2016. 1, 2, 8, 13, 17, 39

[11] Object management group. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1, January 2011. 1

[12] G.H. Mealy. A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5):1045–1079, Sep. 1955. 8

[13] R. Milner. *A Calculus of Communicating Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1980. 3

[14] T. Neele, M.H. Rol, and J.F. Groote. Verifying system-wide properties of industrial component-based software. In H. Hojjat and M. Massink, editors, *Fundamentals of Software Engineering*, pages 158–175, Cham, 2019. Springer International Publishing. 8

[15] M.H. Rol. Verification of ASD multi-component systems in mCRL2. Master's thesis, Eindhoven University of Technology, 2018. 1, 2, 3, 17, 22, 39

[16] R. van Beusekom, J.F. Groote, P. Hoogendijk, R. Howe, W. Wesselink, R. Wieringa, and T.A.C. Willemse. Formalising the dezyne modelling language in mcrl2. In L. Petrucci, C. Seceleanu, and A. Cavalcanti, editors, *Critical Systems: Formal Methods and Automated Verification*, pages 217–233, Cham, 2017. Springer International Publishing. 2

Appendix A

Meta-models of mCRL2

This appendix consists of four meta-models of mCRL2 language in UML. They are Specifications, Sorts, Processes and Expressions, which are used during the transformation.

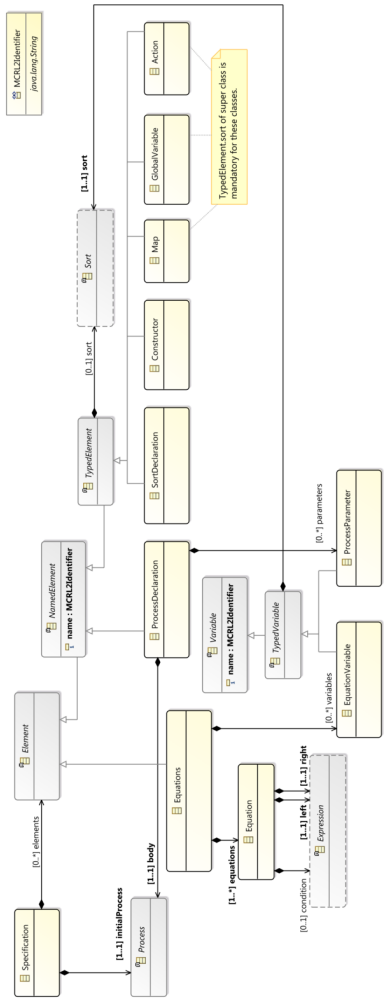

Figure A.1: mCRL2 specifications

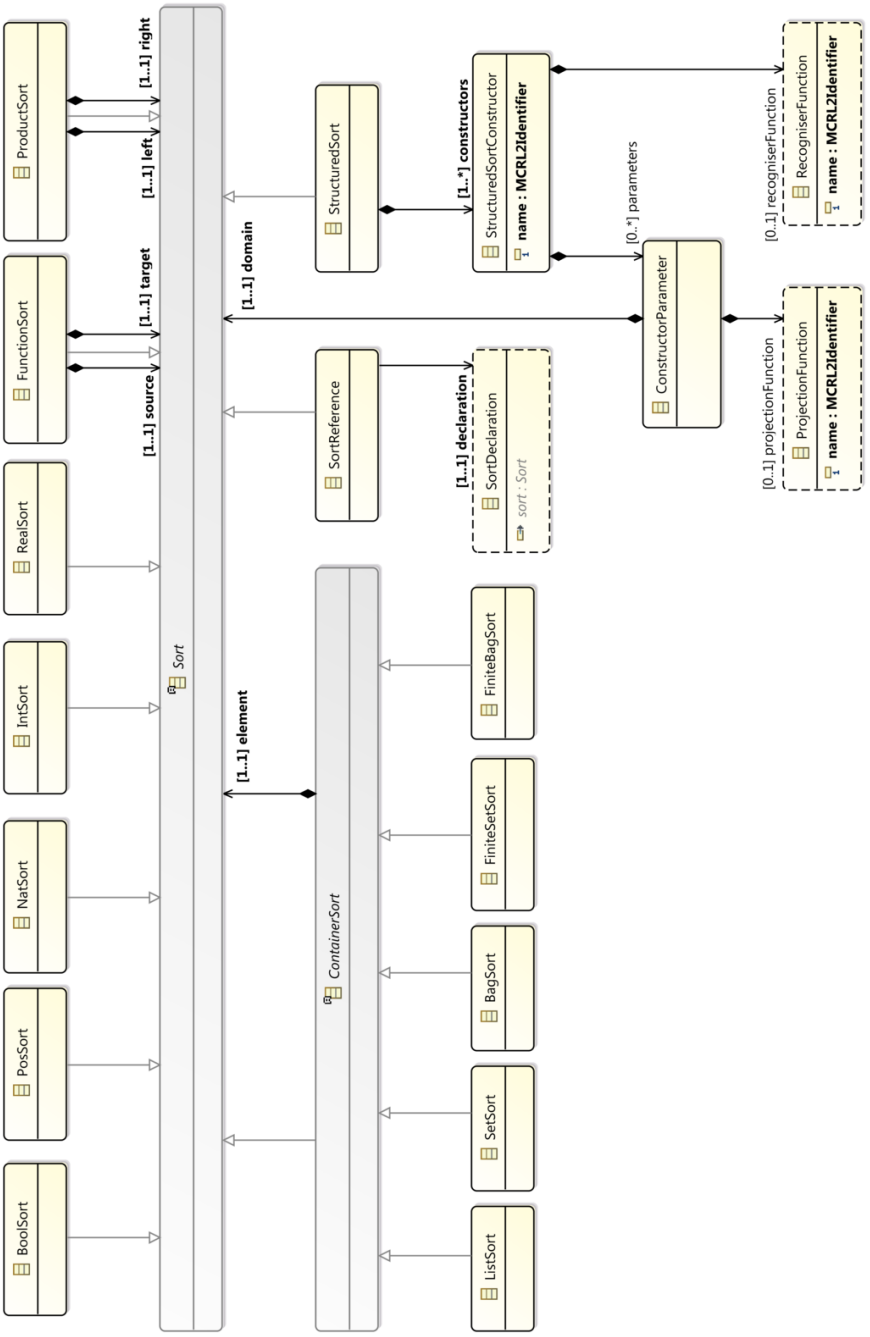

Figure A.2: mCRL2 sorts

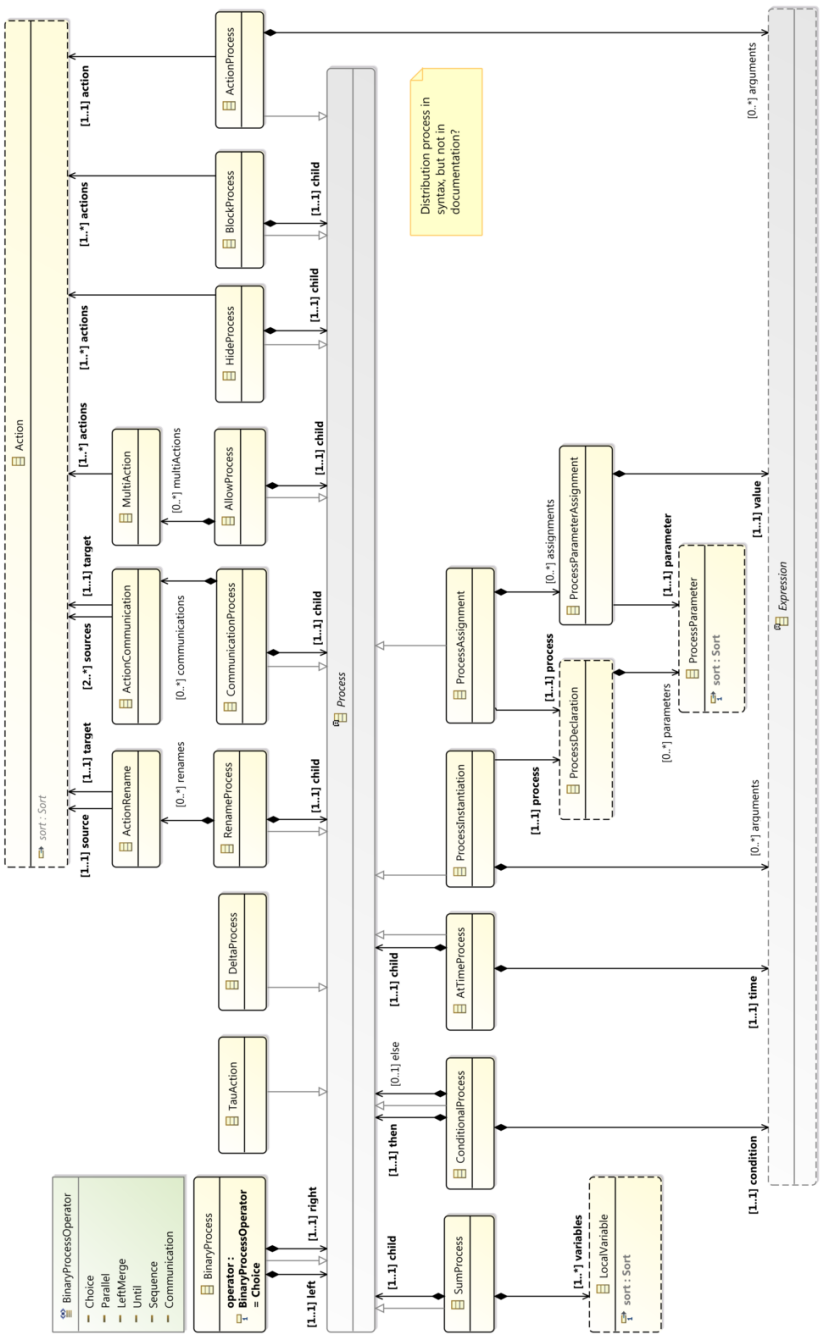

Figure A.3: mCRL2 processes

Figure A.4: mCRL2 expressions

Appendix B

Specifications of the assembly line model

This appendix gives the specifications of the assembly line model that are not given in section 4.4.1.

Initial	Initial (initial state)					
	Interface	Event	Guard	Actions	State Variable Updates	Target State
1	Initial (initial state)					
4	AI	Initialize		Loader:AI.Initialize; AI.VoidReply		Ready
6	Loader:AI	OKReply		Blocked		-
7	Loader:AI	FAILReply		Blocked		-
10	Ready					
14	AI	Make		Loader:AI.Load+	count = 1	Requesting
15	Loader:AI	OKReply		Blocked		-
16	Loader:AI	FAILReply		Blocked		-
18	Loader:NI	Failure		NI.Failed		Error
19	Requesting (synchronous return state)					
22	AI	Initialize		Blocked		-
23	AI	Make		Blocked		-
24	Loader:AI	OKReply		AI.VoidReply		Working
25	Loader:AI	FAILReply		NI.Failed; AI.VoidReply		Error
26	Loader:NI	Aligned		Blocked		-
27	Loader:NI	Failure		Blocked		-
28	Error					
32	AI	Make		AI.VoidReply; NI.Failed		Error
33	Loader:AI	OKReply		Blocked		-
34	Loader:AI	FAILReply		Blocked		-
35	Loader:NI	Aligned		NoOp		Error
36	Loader:NI	Failure		NoOp		Error
37	Working					
41	AI	Make	count < 2	Loader:AI.Load+	count++	Requesting
43	Loader:AI	OKReply		Blocked		-
44	Loader:AI	FAILReply		Blocked		-
45	Loader:NI	Aligned	count <= 1	NI.Done	count = 0	Ready
46	Loader:NI	Aligned	otherwise	NI.Done	count--	Working
47	Loader:NI	Failure		NI.Failed		Error

Figure B.1: The specifications of the assembly line model **DM: UnitA**

Initial	Initial (initial state)					
	Interface	Event	Guard	Actions	State Variable Updates	Target State
1	Initial (initial state)					
4	AI	Initialize		Loader:AI.Initialize; AI.VoidReply		Ready
6	Loader:AI	OKReply		Blocked		-
7	Loader:AI	FAILReply		Blocked		-
10	Ready					
14	AI	Make		AI.VoidReply; Loader:AI.Load+	count = 1	Requesting
15	Loader:AI	OKReply		Blocked		-
16	Loader:AI	FAILReply		Blocked		-
18	Loader:NI	Failure		NI.Failed		Error
19	Requesting (synchronous return state)					
22	AI	Initialize		Blocked		-
23	AI	Make		Blocked		-
24	Loader:AI	OKReply		NoOp		Working
25	Loader:AI	FAILReply		NI.Failed		Error
26	Loader:NI	Aligned		Blocked		-
27	Loader:NI	Failure		Blocked		-
28	Error					
32	AI	Make		AI.VoidReply; NI.Failed		Error
33	Loader:AI	OKReply		Blocked		-
34	Loader:AI	FAILReply		Blocked		-
35	Loader:NI	Aligned		NoOp		Error
36	Loader:NI	Failure		NoOp		Error
37	Working					
41	AI	Make	count < 3	AI.VoidReply; Loader:AI.Load+	count++	Requesting
43	Loader:AI	OKReply		Blocked		-
44	Loader:AI	FAILReply		Blocked		-
45	Loader:NI	Aligned	count <= 1	NI.Done	count--	Ready
46	Loader:NI	Aligned	otherwise	NI.Done	count--	Working
47	Loader:NI	Failure		NI.Failed		Error

Figure B.2: The specifications of the assembly line model **DM: UnitB**

Initial	Initial (initial state)					
	Interface	Event	Guard	Actions	State Variable Updates	Target State
1	Initial (initial state)					
4	AI	Initialize		RobotArm:AI.Initialize; AI.VoidReply		Ready
8	Ready					
11	AI	Initialize		AI.VoidReply		Ready
12	AI	Load+		RobotArm:AI.Move; AI.OKReply	queued = 1	Aligning
15	Aligning					
18	AI	Initialize		AI.VoidReply		Aligning
19	AI	Load+	queued < 4	AI.OKReply	queued++	Aligning
20	AI	Load+	otherwise	AI.FAILReply		Aligning
21	RobotArm:NI	Success	queued <= 1	NI.Aligned	queued = 0	Ready
22	RobotArm:NI	Success	otherwise	RobotArm:AI.Move; NI.Aligned	queued--	Aligning
23	RobotArm:NI	Failure		NI.Failure		Error
24	Error					
27	AI	Initialize		AI.VoidReply		Error
28	AI	Load+		AI.FAILReply		Error
29	RobotArm:NI	Success		NoOp		Error
30	RobotArm:NI	Failure		NoOp		Error

Figure B.3: The specifications of the assembly line model **DM: LoaderUnit**

Initial	Initial (initial state)					
	Interface	Event	Guard	Actions	State Variable Updates	Target State
1	Initial (initial state)					
3	AI	Initialize		AI.VoidReply		Ready
5	MI	[optionally]		Disabled		-
6	MI	eventually		Disabled		-
7	Ready					
10	AI	Move		AI.VoidReply		Moving
11	MI	[optionally]		Disabled		-
12	MI	eventually		Disabled		-
13	Moving					
17	MI	[optionally]		NI.Failure		Ready
18	MI	eventually		NI.Success		Ready

Figure B.4: The specifications of the assembly line model **DM: RobotArm**

Appendix C

The mCRL2 file of the kettle model

This appendix lists the code of the kettle model in mCRL2. This model can be converted to a labeled transition system.

```
sort Component = struct Heater | Kettle | NoClient;
sort Enumeration = struct NoReplyValue | VoidReply;
sort LockingState = struct NONE | LOCK''Kettle'sbs_Kettle | LOCK''Heater'Heater_sync'Heater;
sort Notification = struct Heater'N'TurnedOff | Heater'N'TurnedOn | Kettle'N'Done;
act Heater'A'TurnOff: Component;
act Heater'A'TurnOff;
act Heater'A'TurnOn: Component;
act Heater'A'TurnOn;
act Kettle'A'Abort: Component;
act Kettle'A'Abort;
act Kettle'A'Heat: Component;
act Kettle'A'Heat;
act emptyQ;
act emptyQ_r_Kettle;
act emptyQ_s;
act incompleteGuards;
act inevitableInternalTrigger;
act initialize;
act invalidate;
act invoke''Heater'A'TurnOff: Component;
act invoke''Heater'A'TurnOff;
act invoke''Heater'A'TurnOn: Component;
act invoke''Heater'A'TurnOn;
act invoke''Kettle'A'Abort: Component;
act invoke''Kettle'A'Abort;
act invoke''Kettle'A'Heat: Component;
act invoke''Kettle'A'Heat;
act invoked''Heater'A'TurnOff: Component;
act invoked''Heater'A'TurnOff;
act invoked''Heater'A'TurnOn: Component;
act invoked''Heater'A'TurnOn;
act invoked''Kettle'A'Abort: Component;
act invoked''Kettle'A'Abort;
act invoked''Kettle'A'Heat: Component;
act invoked''Kettle'A'Heat;
act lockQ_Kettle: LockingState;
act lockQ_r_Kettle: LockingState;
act lockQ_s_Kettle: LockingState;
act lock_p_thread;
act lock_t_thread;
act lock_thread;
act optionalInternalTrigger;
```

```
43 act outwardNotification: Notification;
44 act outwardReply: Enumeration;
45 act pushNotification_Kettle: Notification;
46 act qEmpty_Kettle;
47 act qEmpty_r_Kettle;
48 act qEmpty_s_Kettle;
49 act qNonEmpty_Kettle;
50 act qNonEmpty_r_Kettle;
51 act qNonEmpty_s_Kettle;
52 act queueSizeViolated;
53 act queue_lock_p_thread: Component;
54 act queue_lock_t_thread: Component;
55 act queue_lock_thread: Component;
56 act queue_unlock_p_thread: Component;
57 act queue_unlock_t_thread: Component;
58 act queue_unlock_thread: Component;
59 act raiseNotification_Kettle: Notification;
60 act rangeViolated;
61 act readNotification_Kettle: Notification;
62 act readReply: (Enumeration) # ((Component) # (Component));
63 act receiveNotification_Kettle: Notification;
64 act sendNotification_Kettle: Notification;
65 act sendReply: (Enumeration) # ((Component) # (Component));
66 act stateInvariantViolated;
67 act terminate;
68 act triggerNotification_Kettle: Notification;
69 act unlockQ_Kettle;
70 act unlockQ_r_Kettle;
71 act unlockQ_s_Kettle;
72 act unlock_p_thread;
73 act unlock_t_thread;
74 act unlock_thread;
75 act valuedTrigger;
76 act writeReply: (Enumeration) # ((Component) # (Component));
77 proc Heater'Heater_sync'Heater'Off(client: Component) =
78       (true) -> (sum c: Component . ((invoked''Heater'A'TurnOn(c)) . ((writeReply(VoidReply,
            Heater, c)) . (Heater'Heater_sync'Heater'TurningOn(NoClient)))))
79     + (true) -> (sum c: Component . ((invoked''Heater'A'TurnOff(c)) . ((writeReply(VoidReply,
          Heater, c)) . (Heater'Heater_sync'Heater'Off(NoClient)))))
80     + (true) -> (delta);
81 proc Heater'Heater_sync'Heater'On(client: Component) =
82       (true) -> (sum c: Component . ((invoked''Heater'A'TurnOn(c)) . ((writeReply(VoidReply,
            Heater, c)) . (Heater'Heater_sync'Heater'On(NoClient)))))
83     + (true) -> (sum c: Component . ((invoked''Heater'A'TurnOff(c)) . ((writeReply(VoidReply,
          Heater, c)) . (Heater'Heater_sync'Heater'TurningOff(NoClient)))))
84     + (true) -> (((inevitableInternalTrigger) | ((lock_p_thread) | ((emptyQ_s) |
          (lockQ_s_Kettle(LOCK''Heater'Heater_sync'Heater))))) .
          ((pushNotification_Kettle(Heater'N'TurnedOff)) . (((unlockQ_s_Kettle) . (unlock_p_thread))
          . (Heater'Heater_sync'Heater'Off(NoClient)))));
85 proc Heater'Heater_sync'Heater'TurningOff(client: Component) =
86       (true) -> (sum c: Component . ((invoked''Heater'A'TurnOn(c)) .
            ((pushNotification_Kettle(Heater'N'TurnedOff)) . ((writeReply(VoidReply, Heater, c)) .
            (Heater'Heater_sync'Heater'TurningOn(NoClient))))))
87     + (true) -> (sum c: Component . ((invoked''Heater'A'TurnOff(c)) . ((writeReply(VoidReply,
          Heater, c)) . (Heater'Heater_sync'Heater'TurningOff(NoClient)))))
88     + (true) -> (((inevitableInternalTrigger) | ((lock_p_thread) | ((emptyQ_s) |
          (lockQ_s_Kettle(LOCK''Heater'Heater_sync'Heater))))) .
          ((pushNotification_Kettle(Heater'N'TurnedOff)) . (((unlockQ_s_Kettle) . (unlock_p_thread))
          . (Heater'Heater_sync'Heater'Off(NoClient)))));
89 proc Heater'Heater_sync'Heater'TurningOn(client: Component) =
90       (true) -> (sum c: Component . ((invoked''Heater'A'TurnOn(c)) . ((writeReply(VoidReply,
            Heater, c)) . (Heater'Heater_sync'Heater'TurningOn(NoClient)))))
91     + (true) -> (sum c: Component . ((invoked''Heater'A'TurnOff(c)) .
          ((pushNotification_Kettle(Heater'N'TurnedOn)) . ((writeReply(VoidReply, Heater, c)) .
          (Heater'Heater_sync'Heater'TurningOff(NoClient))))))
92     + (true) -> (((inevitableInternalTrigger) | ((lock_p_thread) | ((emptyQ_s) |
          (lockQ_s_Kettle(LOCK''Heater'Heater_sync'Heater))))) .
```

```
        ((pushNotification_Kettle(Heater'N'TurnedOn)) . (((unlockQ_s_Kettle) . (unlock_p_thread))
        . (Heater'Heater_sync'Heater'On(NoClient)))));
93  proc Kettle'sbs_Kettle'Aborting(replyValue: Enumeration, client: Component) =
94       (true) -> (delta)
95     + (true) -> ((((lock_p_thread) | ((emptyQ_s) | (Kettle'A'Abort))) |
           (lockQ_s_Kettle(LOCK''Kettle'sbs_Kettle))) . (((qEmpty_s_Kettle) .
           ((((outwardReply(VoidReply)) | (unlockQ_s_Kettle)) . (unlock_p_thread)) .
           (Kettle'sbs_Kettle'Aborting(NoReplyValue, NoClient)))) + (((qNonEmpty_s_Kettle) .
           ((unlock_p_thread) | (queue_lock_p_thread(Kettle)))) .
           (Kettle'sbs_Kettle'Aborting(VoidReply, NoClient)))))
96     + (true) -> ((((queue_unlock_p_thread(Kettle)) | (lock_p_thread)) |
           (readNotification_Kettle(Heater'N'TurnedOn))) . (((qEmpty_s_Kettle) . ((((replyValue) !=
           (NoReplyValue)) -> (((outwardReply(replyValue)) | (unlockQ_s_Kettle)) . (unlock_p_thread))
           <> ((unlockQ_s_Kettle) . (unlock_p_thread))) . (Kettle'sbs_Kettle'Aborting(NoReplyValue,
           NoClient)))) + (((qNonEmpty_s_Kettle) . ((unlock_p_thread) |
           (queue_lock_p_thread(Kettle)))) . (Kettle'sbs_Kettle'Aborting(replyValue, NoClient)))))
97     + (true) -> ((((queue_unlock_p_thread(Kettle)) | (lock_p_thread)) |
           (readNotification_Kettle(Heater'N'TurnedOff))) . ((outwardNotification(Kettle'N'Done)) .
           (((qEmpty_s_Kettle) . ((((replyValue) != (NoReplyValue)) -> (((outwardReply(replyValue)) |
           (unlockQ_s_Kettle)) . (unlock_p_thread)) <> ((unlockQ_s_Kettle) . (unlock_p_thread))) .
           (Kettle'sbs_Kettle'Idle(NoReplyValue, NoClient)))) + (((qNonEmpty_s_Kettle) .
           ((unlock_p_thread) | (queue_lock_p_thread(Kettle)))) . (Kettle'sbs_Kettle'Idle(replyValue,
           NoClient))))));
98  proc Kettle'sbs_Kettle'Heating(replyValue: Enumeration, client: Component) =
99       (true) -> (delta)
100    + (true) -> ((((lock_p_thread) | ((emptyQ_s) | (Kettle'A'Abort))) |
           (lockQ_s_Kettle(LOCK''Kettle'sbs_Kettle))) . (((invoke''Heater'A'TurnOff(Kettle)) .
           (readReply(VoidReply, Heater, Kettle))) . (((qEmpty_s_Kettle) .
           ((((outwardReply(VoidReply)) | (unlockQ_s_Kettle)) . (unlock_p_thread)) .
           (Kettle'sbs_Kettle'Aborting(NoReplyValue, NoClient)))) + (((qNonEmpty_s_Kettle) .
           ((unlock_p_thread) | (queue_lock_p_thread(Kettle)))) .
           (Kettle'sbs_Kettle'Aborting(VoidReply, NoClient))))))
101    + (true) -> ((((queue_unlock_p_thread(Kettle)) | (lock_p_thread)) |
           (readNotification_Kettle(Heater'N'TurnedOn))) . (Terminate))
102    + (true) -> ((((queue_unlock_p_thread(Kettle)) | (lock_p_thread)) |
           (readNotification_Kettle(Heater'N'TurnedOff))) . ((outwardNotification(Kettle'N'Done)) .
           (((qEmpty_s_Kettle) . ((((replyValue) != (NoReplyValue)) -> (((outwardReply(replyValue)) |
           (unlockQ_s_Kettle)) . (unlock_p_thread)) <> ((unlockQ_s_Kettle) . (unlock_p_thread))) .
           (Kettle'sbs_Kettle'Idle(NoReplyValue, NoClient)))) + (((qNonEmpty_s_Kettle) .
           ((unlock_p_thread) | (queue_lock_p_thread(Kettle)))) . (Kettle'sbs_Kettle'Idle(replyValue,
           NoClient))))));
103 proc Kettle'sbs_Kettle'Idle(replyValue: Enumeration, client: Component) =
104      (true) -> ((((lock_p_thread) | ((emptyQ_s) | (Kettle'A'Heat))) |
             (lockQ_s_Kettle(LOCK''Kettle'sbs_Kettle))) . (((invoke''Heater'A'TurnOn(Kettle)) .
             (readReply(VoidReply, Heater, Kettle))) . (((qEmpty_s_Kettle) .
             ((((outwardReply(VoidReply)) | (unlockQ_s_Kettle)) . (unlock_p_thread)) .
             (Kettle'sbs_Kettle'Started(NoReplyValue, NoClient)))) + (((qNonEmpty_s_Kettle) .
             ((unlock_p_thread) | (queue_lock_p_thread(Kettle)))) .
             (Kettle'sbs_Kettle'Started(VoidReply, NoClient))))))
105    + (true) -> (delta)
106    + (true) -> ((((queue_unlock_p_thread(Kettle)) | (lock_p_thread)) |
           (readNotification_Kettle(Heater'N'TurnedOn))) . (Terminate))
107    + (true) -> ((((queue_unlock_p_thread(Kettle)) | (lock_p_thread)) |
           (readNotification_Kettle(Heater'N'TurnedOff))) . (Terminate));
108 proc Kettle'sbs_Kettle'Started(replyValue: Enumeration, client: Component) =
109      (true) -> (delta)
110    + (true) -> ((((lock_p_thread) | ((emptyQ_s) | (Kettle'A'Abort))) |
           (lockQ_s_Kettle(LOCK''Kettle'sbs_Kettle))) . (((invoke''Heater'A'TurnOff(Kettle)) .
           (readReply(VoidReply, Heater, Kettle))) . (((qEmpty_s_Kettle) .
           ((((outwardReply(VoidReply)) | (unlockQ_s_Kettle)) . (unlock_p_thread)) .
           (Kettle'sbs_Kettle'Aborting(NoReplyValue, NoClient)))) + (((qNonEmpty_s_Kettle) .
           ((unlock_p_thread) | (queue_lock_p_thread(Kettle)))) .
           (Kettle'sbs_Kettle'Aborting(VoidReply, NoClient))))))
111    + (true) -> ((((queue_unlock_p_thread(Kettle)) | (lock_p_thread)) |
           (readNotification_Kettle(Heater'N'TurnedOn))) . (((qEmpty_s_Kettle) . ((((replyValue) !=
           (NoReplyValue)) -> (((outwardReply(replyValue)) | (unlockQ_s_Kettle)) . (unlock_p_thread))
           <> ((unlockQ_s_Kettle) . (unlock_p_thread))) . (Kettle'sbs_Kettle'Heating(NoReplyValue,
```

```
        NoClient)))) + (((qNonEmpty_s_Kettle) . ((unlock_p_thread) |
        (queue_lock_p_thread(Kettle)))) . (Kettle'sbs_Kettle'Heating(replyValue, NoClient)))))
112     + (true) -> ((((queue_unlock_p_thread(Kettle)) | (lock_p_thread)) |
        (readNotification_Kettle(Heater'N'TurnedOff))) . (Terminate));
113 proc Queue_Kettle(q: List(Notification), locked: LockingState) =
114       sum n: Notification . ((receiveNotification_Kettle(n)) . (Queue_Kettle((q) <| (n), locked)))
115     + (((#(q)) > (0)) && (((locked) == (NONE)) || ((locked) == (LOCK''Kettle'sbs_Kettle')))) ->
        ((sendNotification_Kettle(head(q))) . (Queue_Kettle(tail(q), LOCK''Kettle'sbs_Kettle)))
116     + sum s: LockingState . ((((#(q)) == (0)) && (((locked) == (NONE)) || ((locked) == (s)))) ->
        ((lockQ_r_Kettle(s)) . (Queue_Kettle(q, s))))
117     + sum s: LockingState . ((((#(q)) == (0)) && (((locked) == (NONE)) || ((locked) == (s)))) ->
        (((lockQ_r_Kettle(s)) | (emptyQ_r_Kettle)) . (Queue_Kettle(q, s))))
118     + ((locked) != (NONE)) -> ((unlockQ_r_Kettle) . (Queue_Kettle(q, NONE)))
119     + ((#(q)) == (0)) -> ((qEmpty_r_Kettle) . (Queue_Kettle(q, locked)))
120     + ((#(q)) == (0)) -> ((emptyQ_r_Kettle) . (Queue_Kettle(q, locked)))
121     + ((#(q)) > (0)) -> ((qNonEmpty_r_Kettle) . (Queue_Kettle(q, locked)))
122     + ((#(q)) > (7)) -> ((queueSizeViolated) . (delta));
123 proc Terminate = (terminate) . (delta);
124 proc Thread = ((lock_t_thread) . (unlock_t_thread)) . (Thread);
125 proc queue_Thread = sum c: Component . ((((queue_lock_t_thread(c)) . (queue_unlock_t_thread(c))) .
    (queue_Thread)) + ((queue_unlock_t_thread(c)) . (queue_Thread)));
126 init allow({unlock_thread, unlock_thread|queue_lock_thread, initialize, invalidate, terminate,
    valuedTrigger, optionalInternalTrigger|lockQ_Kettle|lock_thread|emptyQ,
    inevitableInternalTrigger|lockQ_Kettle|lock_thread|emptyQ,
    Kettle'A'Abort|lockQ_Kettle|lock_thread|emptyQ, Kettle'A'Heat|lockQ_Kettle|lock_thread|emptyQ,
    outwardReply|unlockQ_Kettle, outwardNotification, Heater'A'TurnOff, Heater'A'TurnOn,
    sendReply, sendReply|unlockQ_Kettle,
    triggerNotification_Kettle|lock_thread|queue_unlock_thread, raiseNotification_Kettle,
    unlockQ_Kettle, queueSizeViolated, qEmpty_Kettle, qNonEmpty_Kettle},
    comm({emptyQ_r_Kettle|emptyQ_s -> emptyQ, lock_t_thread|lock_p_thread -> lock_thread,
    unlock_t_thread|unlock_p_thread -> unlock_thread, queue_lock_t_thread|queue_lock_p_thread ->
    queue_lock_thread, queue_unlock_t_thread|queue_unlock_p_thread -> queue_unlock_thread,
    readReply|writeReply -> sendReply, lockQ_s_Kettle|lockQ_r_Kettle -> lockQ_Kettle,
    unlockQ_s_Kettle|unlockQ_r_Kettle -> unlockQ_Kettle, qEmpty_s_Kettle|qEmpty_r_Kettle ->
    qEmpty_Kettle, qNonEmpty_s_Kettle|qNonEmpty_r_Kettle -> qNonEmpty_Kettle,
    sendNotification_Kettle|readNotification_Kettle -> triggerNotification_Kettle,
    pushNotification_Kettle|receiveNotification_Kettle -> raiseNotification_Kettle,
    invoke''Kettle'A'Abort|invoked''Kettle'A'Abort -> Kettle'A'Abort,
    invoke''Heater'A'TurnOn|invoked''Heater'A'TurnOn -> Heater'A'TurnOn,
    invoke''Heater'A'TurnOff|invoked''Heater'A'TurnOff -> Heater'A'TurnOff,
    invoke''Kettle'A'Heat|invoked''Kettle'A'Heat -> Kettle'A'Heat},
    ((((Heater'Heater_sync'Heater'Off(NoClient)) || (Thread)) ||
    (Kettle'sbs_Kettle'Idle(NoReplyValue, NoClient))) || (Queue_Kettle([], NONE))) ||
    (queue_Thread)));
```

Appendix D

The mCRL2 file of the lamp model

This appendix lists the code of the lamp model in mCRL2. This model can be converted to a labeled transition system.

```
1  sort Component = struct LampSystem | Light | Switch | NoClient;
2  sort Enumeration = struct NoReplyValue | Switch'A'ARETURNENUM'OFFREPLY |
       Switch'A'ARETURNENUM'ONREPLY | VoidReply;
3  sort LockingState = struct NONE | LOCK''LampSystem'sbs_LampSystem | LOCK''Light'A'Light |
       LOCK''Switch'A'Switch;
4  sort Notification = struct dummyNotification;
5  act LampSystem'A'Initialize: Component;
6  act LampSystem'A'Initialize;
7  act LampSystem'A'Update: Component;
8  act LampSystem'A'Update;
9  act Light'A'Initialize: Component;
10 act Light'A'Initialize;
11 act Light'A'SwitchOff: Component;
12 act Light'A'SwitchOff;
13 act Light'A'SwitchOn: Component;
14 act Light'A'SwitchOn;
15 act Switch'A'Initialize: Component;
16 act Switch'A'Initialize;
17 act Switch'A'Poll: Component;
18 act Switch'A'Poll;
19 act emptyQ;
20 act emptyQ_r_LampSystem;
21 act emptyQ_s;
22 act incompleteGuards;
23 act inevitableInternalTrigger;
24 act initialize;
25 act invalidate;
26 act invoke''LampSystem'A'Initialize: Component;
27 act invoke''LampSystem'A'Initialize;
28 act invoke''LampSystem'A'Update: Component;
29 act invoke''LampSystem'A'Update;
30 act invoke''Light'A'Initialize: Component;
31 act invoke''Light'A'Initialize;
32 act invoke''Light'A'SwitchOff: Component;
33 act invoke''Light'A'SwitchOff;
34 act invoke''Light'A'SwitchOn: Component;
35 act invoke''Light'A'SwitchOn;
36 act invoke''Switch'A'Initialize: Component;
37 act invoke''Switch'A'Initialize;
38 act invoke''Switch'A'Poll: Component;
39 act invoke''Switch'A'Poll;
40 act invoked''LampSystem'A'Initialize: Component;
```

```
41 act invoked''LampSystem'A'Initialize;
42 act invoked''LampSystem'A'Update: Component;
43 act invoked''LampSystem'A'Update;
44 act invoked''Light'A'Initialize: Component;
45 act invoked''Light'A'Initialize;
46 act invoked''Light'A'SwitchOff: Component;
47 act invoked''Light'A'SwitchOff;
48 act invoked''Light'A'SwitchOn: Component;
49 act invoked''Light'A'SwitchOn;
50 act invoked''Switch'A'Initialize: Component;
51 act invoked''Switch'A'Initialize;
52 act invoked''Switch'A'Poll: Component;
53 act invoked''Switch'A'Poll;
54 act lockQ_LampSystem: LockingState;
55 act lockQ_r_LampSystem: LockingState;
56 act lockQ_s_LampSystem: LockingState;
57 act lock_p_thread;
58 act lock_t_thread;
59 act lock_thread;
60 act optionalInternalTrigger;
61 act outwardNotification: Notification;
62 act outwardReply: Enumeration;
63 act pushNotification_LampSystem: Notification;
64 act qEmpty_LampSystem;
65 act qEmpty_r_LampSystem;
66 act qEmpty_s_LampSystem;
67 act qNonEmpty_LampSystem;
68 act qNonEmpty_r_LampSystem;
69 act qNonEmpty_s_LampSystem;
70 act queueSizeViolated;
71 act queue_lock_p_thread: Component;
72 act queue_lock_t_thread: Component;
73 act queue_lock_thread: Component;
74 act queue_unlock_p_thread: Component;
75 act queue_unlock_t_thread: Component;
76 act queue_unlock_thread: Component;
77 act raiseNotification_LampSystem: Notification;
78 act rangeViolated;
79 act readNotification_LampSystem: Notification;
80 act readReply: (Enumeration) # ((Component) # (Component));
81 act receiveNotification_LampSystem: Notification;
82 act sendNotification_LampSystem: Notification;
83 act sendReply: (Enumeration) # ((Component) # (Component));
84 act stateInvariantViolated;
85 act terminate;
86 act triggerNotification_LampSystem: Notification;
87 act unlockQ_LampSystem;
88 act unlockQ_r_LampSystem;
89 act unlockQ_s_LampSystem;
90 act unlock_p_thread;
91 act unlock_t_thread;
92 act unlock_thread;
93 act valuedTrigger;
94 act writeReply: (Enumeration) # ((Component) # (Component));
95 proc LampSystem'sbs_LampSystem'Init(replyValue: Enumeration, client: Component) =
96       (true) -> ((((lock_p_thread) | ((emptyQ_s) | (LampSystem'A'Initialize))) |
            (lockQ_s_LampSystem(LOCK''LampSystem'sbs_LampSystem))) .
            (((invoke''Switch'A'Initialize(LampSystem)) . (readReply(VoidReply, Switch,
            LampSystem))) . (((invoke''Light'A'Initialize(LampSystem)) . (readReply(VoidReply,
            Light, LampSystem))) . (((qEmpty_s_LampSystem) . ((((outwardReply(VoidReply)) |
            (unlockQ_s_LampSystem)) . (unlock_p_thread)) .
            (LampSystem'sbs_LampSystem'Off(NoReplyValue, NoClient)))) + (((qNonEmpty_s_LampSystem) .
            ((unlock_p_thread) | (queue_lock_p_thread(LampSystem)))) .
            (LampSystem'sbs_LampSystem'Off(VoidReply, NoClient)))))))
97     + (true) -> ((((lock_p_thread) | ((emptyQ_s) | (LampSystem'A'Update))) |
          (lockQ_s_LampSystem(LOCK''LampSystem'sbs_LampSystem))) . (((qEmpty_s_LampSystem) .
          ((((outwardReply(VoidReply)) | (unlockQ_s_LampSystem)) . (unlock_p_thread)) .
```

```
        (LampSystem'sbs_LampSystem'Init(NoReplyValue, client)))) + (((qNonEmpty_s_LampSystem) .
        ((unlock_p_thread) | (queue_lock_p_thread(LampSystem)))) .
        (LampSystem'sbs_LampSystem'Init(VoidReply, client)))));
98  proc LampSystem'sbs_LampSystem'Off(replyValue: Enumeration, client: Component) =
99       (true) -> (delta)
100    + (true) -> ((((lock_p_thread) | ((emptyQ_s) | (LampSystem'A'Update))) |
         (lockQ_s_LampSystem(LOCK''LampSystem'sbs_LampSystem))) .
         ((invoke''Switch'A'Poll(LampSystem)) . (sum LampSystem'sbs_LampSystem'Off'Poll_EvalVar12:
         Enumeration . ((readReply(LampSystem'sbs_LampSystem'Off'Poll_EvalVar12, Switch,
         LampSystem)) .
         (LampSystem'sbs_LampSystem'Off_Polling(LampSystem'sbs_LampSystem'Off'Poll_EvalVar12,
         replyValue, NoClient))))));
101 proc
      LampSystem'sbs_LampSystem'Off_Polling(LampSystem'sbs_LampSystem'Off_Polling'Off_PollingAReturnEnumEP:
      Enumeration, replyValue: Enumeration, client: Component) =
102       ((LampSystem'sbs_LampSystem'Off_Polling'Off_PollingAReturnEnumEP) ==
            (Switch'A'ARETURNENUM'ONREPLY)) -> ((valuedTrigger) .
            (((invoke''Light'A'SwitchOn(LampSystem)) . (readReply(VoidReply, Light, LampSystem))) .
            (((qEmpty_s_LampSystem) . ((((outwardReply(VoidReply)) | (unlockQ_s_LampSystem)) .
            (unlock_p_thread)) . (LampSystem'sbs_LampSystem'On(NoReplyValue, NoClient)))) +
            (((qNonEmpty_s_LampSystem) . ((unlock_p_thread) | (queue_lock_p_thread(LampSystem)))) .
            (LampSystem'sbs_LampSystem'On(VoidReply, NoClient))))))
103    + ((LampSystem'sbs_LampSystem'Off_Polling'Off_PollingAReturnEnumEP) ==
         (Switch'A'ARETURNENUM'OFFREPLY)) -> ((valuedTrigger) . (((qEmpty_s_LampSystem) .
         ((((outwardReply(VoidReply)) | (unlockQ_s_LampSystem)) . (unlock_p_thread)) .
         (LampSystem'sbs_LampSystem'Off(NoReplyValue, NoClient)))) + (((qNonEmpty_s_LampSystem) .
         ((unlock_p_thread) | (queue_lock_p_thread(LampSystem)))) .
         (LampSystem'sbs_LampSystem'Off(VoidReply, NoClient)))));
104 proc LampSystem'sbs_LampSystem'On(replyValue: Enumeration, client: Component) =
105      (true) -> (delta)
106    + (true) -> ((((lock_p_thread) | ((emptyQ_s) | (LampSystem'A'Update))) |
         (lockQ_s_LampSystem(LOCK''LampSystem'sbs_LampSystem))) .
         ((invoke''Switch'A'Poll(LampSystem)) . (sum LampSystem'sbs_LampSystem'On'Poll_EvalVar26:
         Enumeration . ((readReply(LampSystem'sbs_LampSystem'On'Poll_EvalVar26, Switch,
         LampSystem)) .
         (LampSystem'sbs_LampSystem'On_Polling(LampSystem'sbs_LampSystem'On'Poll_EvalVar26,
         replyValue, NoClient))))));
107 proc
      LampSystem'sbs_LampSystem'On_Polling(LampSystem'sbs_LampSystem'On_Polling'On_PollingAReturnEnumEP:
      Enumeration, replyValue: Enumeration, client: Component) =
108       ((LampSystem'sbs_LampSystem'On_Polling'On_PollingAReturnEnumEP) ==
            (Switch'A'ARETURNENUM'ONREPLY)) -> ((valuedTrigger) . (((qEmpty_s_LampSystem) .
            ((((outwardReply(VoidReply)) | (unlockQ_s_LampSystem)) . (unlock_p_thread)) .
            (LampSystem'sbs_LampSystem'On(NoReplyValue, NoClient)))) + (((qNonEmpty_s_LampSystem) .
            ((unlock_p_thread) | (queue_lock_p_thread(LampSystem)))) .
            (LampSystem'sbs_LampSystem'On(VoidReply, NoClient)))))
109    + ((LampSystem'sbs_LampSystem'On_Polling'On_PollingAReturnEnumEP) ==
         (Switch'A'ARETURNENUM'OFFREPLY)) -> ((valuedTrigger) .
         (((invoke''Light'A'SwitchOff(LampSystem)) . (readReply(VoidReply, Light, LampSystem))) .
         (((qEmpty_s_LampSystem) . ((((outwardReply(VoidReply)) | (unlockQ_s_LampSystem)) .
         (unlock_p_thread)) . (LampSystem'sbs_LampSystem'Off(NoReplyValue, NoClient)))) +
         (((qNonEmpty_s_LampSystem) . ((unlock_p_thread) | (queue_lock_p_thread(LampSystem)))) .
         (LampSystem'sbs_LampSystem'Off(VoidReply, NoClient))))));
110 proc Light'A'Light'Init(client: Component) =
111      (true) -> (sum c: Component . ((invoked''Light'A'Initialize(c)) . ((writeReply(VoidReply,
           Light, c)) . (Light'A'Light'OffState(NoClient)))))
112    + (true) -> (sum c: Component . ((invoked''Light'A'SwitchOn(c)) . (Terminate)))
113    + (true) -> (sum c: Component . ((invoked''Light'A'SwitchOff(c)) . (Terminate)));
114 proc Light'A'Light'OffState(client: Component) =
115      (true) -> (sum c: Component . ((invoked''Light'A'Initialize(c)) . (Terminate)))
116    + (true) -> (sum c: Component . ((invoked''Light'A'SwitchOn(c)) . ((writeReply(VoidReply,
         Light, c)) . (Light'A'Light'OnState(NoClient)))))
117    + (true) -> (sum c: Component . ((invoked''Light'A'SwitchOff(c)) . ((writeReply(VoidReply,
         Light, c)) . (Light'A'Light'OffState(client)))));
118 proc Light'A'Light'OnState(client: Component) =
119      (true) -> (sum c: Component . ((invoked''Light'A'Initialize(c)) . (Terminate)))
120    + (true) -> (sum c: Component . ((invoked''Light'A'SwitchOn(c)) . ((writeReply(VoidReply,
```

```
        Light, c)) . (Light'A'Light'OnState(client)))))
121     + (true) -> (sum c: Component . ((invoked''Light'A'SwitchOff(c)) . ((writeReply(VoidReply,
        Light, c)) . (Light'A'Light'OffState(NoClient)))));
122 proc Queue_LampSystem(q: List(Notification), locked: LockingState) =
123      sum n: Notification . ((receiveNotification_LampSystem(n)) . (Queue_LampSystem((q) <| (n),
          locked)))
124     + (((#(q)) > (0)) && (((locked) == (NONE)) || ((locked) == (LOCK''LampSystem'sbs_LampSystem))))
        -> ((sendNotification_LampSystem(head(q))) . (Queue_LampSystem(tail(q),
        LOCK''LampSystem'sbs_LampSystem)))
125     + sum s: LockingState . ((((#(q)) == (0)) && (((locked) == (NONE)) || ((locked) == (s)))) ->
        ((lockQ_r_LampSystem(s)) . (Queue_LampSystem(q, s))))
126     + sum s: LockingState . ((((#(q)) == (0)) && (((locked) == (NONE)) || ((locked) == (s)))) ->
        (((lockQ_r_LampSystem(s)) | (emptyQ_r_LampSystem)) . (Queue_LampSystem(q, s))))
127     + ((locked) != (NONE)) -> ((unlockQ_r_LampSystem) . (Queue_LampSystem(q, NONE)))
128     + ((#(q)) == (0)) -> ((qEmpty_r_LampSystem) . (Queue_LampSystem(q, locked)))
129     + ((#(q)) == (0)) -> ((emptyQ_r_LampSystem) . (Queue_LampSystem(q, locked)))
130     + ((#(q)) > (0)) -> ((qNonEmpty_r_LampSystem) . (Queue_LampSystem(q, locked)))
131     + ((#(q)) > (7)) -> ((queueSizeViolated) . (delta));
132 proc Switch'A'Switch'Init(client: Component) =
133      (true) -> (sum c: Component . ((invoked''Switch'A'Initialize(c)) . ((writeReply(VoidReply,
           Switch, c)) . (Switch'A'Switch'Main(NoClient)))))
134     + (true) -> (sum c: Component . ((invoked''Switch'A'Poll(c)) . (Terminate)));
135 proc Switch'A'Switch'Main(client: Component) =
136      (true) -> (sum c: Component . ((invoked''Switch'A'Initialize(c)) . (Terminate)))
137     + (true) -> (sum c: Component . ((invoked''Switch'A'Poll(c)) .
        ((writeReply(Switch'A'ARETURNENUM'ONREPLY, Switch, c)) . (Switch'A'Switch'Main(client)))))
138     + (true) -> (sum c: Component . ((invoked''Switch'A'Poll(c)) .
        ((writeReply(Switch'A'ARETURNENUM'OFFREPLY, Switch, c)) .
        (Switch'A'Switch'Main(client)))));
139 proc Terminate = (terminate) . (delta);
140 proc Thread = ((lock_t_thread) . (unlock_t_thread)) . (Thread);
141 proc queue_Thread = sum c: Component . ((((queue_lock_t_thread(c)) . (queue_unlock_t_thread(c))) .
    (queue_Thread)) + ((queue_unlock_t_thread(c)) . (queue_Thread)));
142 init allow({unlock_thread, unlock_thread|queue_lock_thread, initialize, invalidate, terminate,
    valuedTrigger, optionalInternalTrigger|lockQ_LampSystem|lock_thread|emptyQ,
    inevitableInternalTrigger|lockQ_LampSystem|lock_thread|emptyQ,
    LampSystem'A'Initialize|lockQ_LampSystem|lock_thread|emptyQ,
    LampSystem'A'Update|lockQ_LampSystem|lock_thread|emptyQ, outwardReply|unlockQ_LampSystem,
    outwardNotification, Light'A'Initialize, Light'A'SwitchOff, Light'A'SwitchOn,
    Switch'A'Initialize, Switch'A'Poll, sendReply, sendReply|unlockQ_LampSystem,
    triggerNotification_LampSystem|lock_thread|queue_unlock_thread, raiseNotification_LampSystem,
    unlockQ_LampSystem, queueSizeViolated, qEmpty_LampSystem, qNonEmpty_LampSystem},
    comm({emptyQ_r_LampSystem|emptyQ_s -> emptyQ, lock_t_thread|lock_p_thread -> lock_thread,
    unlock_t_thread|unlock_p_thread -> unlock_thread, queue_lock_t_thread|queue_lock_p_thread ->
    queue_lock_thread, queue_unlock_t_thread|queue_unlock_p_thread -> queue_unlock_thread,
    readReply|writeReply -> sendReply, lockQ_s_LampSystem|lockQ_r_LampSystem -> lockQ_LampSystem,
    unlockQ_s_LampSystem|unlockQ_r_LampSystem -> unlockQ_LampSystem,
    qEmpty_s_LampSystem|qEmpty_r_LampSystem -> qEmpty_LampSystem,
    qNonEmpty_s_LampSystem|qNonEmpty_r_LampSystem -> qNonEmpty_LampSystem,
    sendNotification_LampSystem|readNotification_LampSystem -> triggerNotification_LampSystem,
    pushNotification_LampSystem|receiveNotification_LampSystem -> raiseNotification_LampSystem,
    invoke''Light'A'Initialize|invoked''Light'A'Initialize -> Light'A'Initialize,
    invoke''LampSystem'A'Initialize|invoked''LampSystem'A'Initialize -> LampSystem'A'Initialize,
    invoke''LampSystem'A'Update|invoked''LampSystem'A'Update -> LampSystem'A'Update,
    invoke''Light'A'SwitchOn|invoked''Light'A'SwitchOn -> Light'A'SwitchOn,
    invoke''Switch'A'Poll|invoked''Switch'A'Poll -> Switch'A'Poll,
    invoke''Switch'A'Initialize|invoked''Switch'A'Initialize -> Switch'A'Initialize,
    invoke''Light'A'SwitchOff|invoked''Light'A'SwitchOff -> Light'A'SwitchOff}, (((((Thread) ||
    (Queue_LampSystem([], NONE))) || (LampSystem'sbs_LampSystem'Init(NoReplyValue, NoClient))) ||
    (Switch'A'Switch'Init(NoClient))) || (queue_Thread)) || (Light'A'Light'Init(NoClient))));
```

Appendix E

The mCRL2 file of the assembly line model

This appendix lists the code of the assembly line model in mCRL2. This model can be converted to a labeled transition system.

```
1  sort Component = struct ControlUnit | LoaderUnit | RobotArm | UnitA | UnitB | NoClient;
2  sort Enumeration = struct Controller'AI'AIRETURNENUM'FAILREPLY |
       Controller'AI'AIRETURNENUM'OKREPLY | Loader'AI'AIRETURNENUM'FAILREPLY |
       Loader'AI'AIRETURNENUM'OKREPLY | NoReplyValue | VoidReply;
3  sort LockingState = struct NONE | LOCK''UnitA'sbs_UnitA | LOCK''UnitB'sbs_UnitB |
       LOCK''LoaderUnit'sbs_LoaderUnit | LOCK''ControlUnit'sbs_ControlUnit |
       LOCK''RobotArm'RobotArm_sync'RobotArm;
4  sort Notification = struct Controller'NI'Error | Controller'NI'MadeA | Controller'NI'MadeB |
       Loader'NI'Aligned | Loader'NI'Failure | MachineA'NI'Done | MachineA'NI'Failed |
       MachineB'NI'Done | MachineB'NI'Failed | RobotArm'NI'Failure | RobotArm'NI'Success;
5  act Controller'AI'Initialize: Component;
6  act Controller'AI'Initialize;
7  act Controller'AI'MakeA: Component;
8  act Controller'AI'MakeA;
9  act Controller'AI'MakeB: Component;
10 act Controller'AI'MakeB;
11 act Loader'AI'Initialize: Component;
12 act Loader'AI'Initialize;
13 act Loader'AI'Load: Component;
14 act Loader'AI'Load;
15 act MachineA'AI'Initialize: Component;
16 act MachineA'AI'Initialize;
17 act MachineA'AI'Make: Component;
18 act MachineA'AI'Make;
19 act MachineB'AI'Initialize: Component;
20 act MachineB'AI'Initialize;
21 act MachineB'AI'Make: Component;
22 act MachineB'AI'Make;
23 act RobotArm'AI'Initialize: Component;
24 act RobotArm'AI'Initialize;
25 act RobotArm'AI'Move: Component;
26 act RobotArm'AI'Move;
27 act emptyQ;
28 act emptyQ_r_ControlUnit;
29 act emptyQ_r_LoaderUnit;
30 act emptyQ_r_UnitA;
31 act emptyQ_r_UnitB;
32 act emptyQ_s;
33 act incompleteGuards;
34 act inevitableInternalTrigger;
35 act initialize;
36 act invalidate;
```

```
act invoke''Controller'AI'Initialize: Component;
act invoke''Controller'AI'Initialize;
act invoke''Controller'AI'MakeA: Component;
act invoke''Controller'AI'MakeA;
act invoke''Controller'AI'MakeB: Component;
act invoke''Controller'AI'MakeB;
act invoke''Loader'AI'Initialize: Component;
act invoke''Loader'AI'Initialize;
act invoke''Loader'AI'Load: Component;
act invoke''Loader'AI'Load;
act invoke''MachineA'AI'Initialize: Component;
act invoke''MachineA'AI'Initialize;
act invoke''MachineA'AI'Make: Component;
act invoke''MachineA'AI'Make;
act invoke''MachineB'AI'Initialize: Component;
act invoke''MachineB'AI'Initialize;
act invoke''MachineB'AI'Make: Component;
act invoke''MachineB'AI'Make;
act invoke''RobotArm'AI'Initialize: Component;
act invoke''RobotArm'AI'Initialize;
act invoke''RobotArm'AI'Move: Component;
act invoke''RobotArm'AI'Move;
act invoked''Controller'AI'Initialize: Component;
act invoked''Controller'AI'Initialize;
act invoked''Controller'AI'MakeA: Component;
act invoked''Controller'AI'MakeA;
act invoked''Controller'AI'MakeB: Component;
act invoked''Controller'AI'MakeB;
act invoked''Loader'AI'Initialize: Component;
act invoked''Loader'AI'Initialize;
act invoked''Loader'AI'Load: Component;
act invoked''Loader'AI'Load;
act invoked''MachineA'AI'Initialize: Component;
act invoked''MachineA'AI'Initialize;
act invoked''MachineA'AI'Make: Component;
act invoked''MachineA'AI'Make;
act invoked''MachineB'AI'Initialize: Component;
act invoked''MachineB'AI'Initialize;
act invoked''MachineB'AI'Make: Component;
act invoked''MachineB'AI'Make;
act invoked''RobotArm'AI'Initialize: Component;
act invoked''RobotArm'AI'Initialize;
act invoked''RobotArm'AI'Move: Component;
act invoked''RobotArm'AI'Move;
act lockQ_ControlUnit: LockingState;
act lockQ_LoaderUnit: LockingState;
act lockQ_UnitA: LockingState;
act lockQ_UnitB: LockingState;
act lockQ_r_ControlUnit: LockingState;
act lockQ_r_LoaderUnit: LockingState;
act lockQ_r_UnitA: LockingState;
act lockQ_r_UnitB: LockingState;
act lockQ_s_ControlUnit: LockingState;
act lockQ_s_LoaderUnit: LockingState;
act lockQ_s_UnitA: LockingState;
act lockQ_s_UnitB: LockingState;
act lock_p_thread;
act lock_t_thread;
act lock_thread;
act optionalInternalTrigger;
act outwardNotification: Notification;
act outwardReply: Enumeration;
act pushNotification_ControlUnit: Notification;
act pushNotification_LoaderUnit: Notification;
act pushNotification_UnitA: Notification;
act pushNotification_UnitB: Notification;
act qEmpty_ControlUnit;
```

```
104 act qEmpty_LoaderUnit;
105 act qEmpty_UnitA;
106 act qEmpty_UnitB;
107 act qEmpty_r_ControlUnit;
108 act qEmpty_r_LoaderUnit;
109 act qEmpty_r_UnitA;
110 act qEmpty_r_UnitB;
111 act qEmpty_s_ControlUnit;
112 act qEmpty_s_LoaderUnit;
113 act qEmpty_s_UnitA;
114 act qEmpty_s_UnitB;
115 act qNonEmpty_ControlUnit;
116 act qNonEmpty_LoaderUnit;
117 act qNonEmpty_UnitA;
118 act qNonEmpty_UnitB;
119 act qNonEmpty_r_ControlUnit;
120 act qNonEmpty_r_LoaderUnit;
121 act qNonEmpty_r_UnitA;
122 act qNonEmpty_r_UnitB;
123 act qNonEmpty_s_ControlUnit;
124 act qNonEmpty_s_LoaderUnit;
125 act qNonEmpty_s_UnitA;
126 act qNonEmpty_s_UnitB;
127 act queueSizeViolated;
128 act queue_lock_p_thread: Component;
129 act queue_lock_t_thread: Component;
130 act queue_lock_thread: Component;
131 act queue_unlock_p_thread: Component;
132 act queue_unlock_t_thread: Component;
133 act queue_unlock_thread: Component;
134 act raiseNotification_ControlUnit: Notification;
135 act raiseNotification_LoaderUnit: Notification;
136 act raiseNotification_UnitA: Notification;
137 act raiseNotification_UnitB: Notification;
138 act rangeViolated;
139 act readNotification_ControlUnit: Notification;
140 act readNotification_LoaderUnit: Notification;
141 act readNotification_UnitA: Notification;
142 act readNotification_UnitB: Notification;
143 act readReply: (Enumeration) # ((Component) # (Component));
144 act receiveNotification_ControlUnit: Notification;
145 act receiveNotification_LoaderUnit: Notification;
146 act receiveNotification_UnitA: Notification;
147 act receiveNotification_UnitB: Notification;
148 act sendNotification_ControlUnit: Notification;
149 act sendNotification_LoaderUnit: Notification;
150 act sendNotification_UnitA: Notification;
151 act sendNotification_UnitB: Notification;
152 act sendReply: (Enumeration) # ((Component) # (Component));
153 act stateInvariantViolated;
154 act terminate;
155 act triggerNotification_ControlUnit: Notification;
156 act triggerNotification_LoaderUnit: Notification;
157 act triggerNotification_UnitA: Notification;
158 act triggerNotification_UnitB: Notification;
159 act unlockQ_ControlUnit;
160 act unlockQ_LoaderUnit;
161 act unlockQ_UnitA;
162 act unlockQ_UnitB;
163 act unlockQ_r_ControlUnit;
164 act unlockQ_r_LoaderUnit;
165 act unlockQ_r_UnitA;
166 act unlockQ_r_UnitB;
167 act unlockQ_s_ControlUnit;
168 act unlockQ_s_LoaderUnit;
169 act unlockQ_s_UnitA;
170 act unlockQ_s_UnitB;
```

```
171 act unlock_p_thread;
172 act unlock_t_thread;
173 act unlock_thread;
174 act valuedTrigger;
175 act writeReply: (Enumeration) # ((Component) # (Component));
176 proc ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA: Int,
        ControlUnit'sbs_ControlUnit'countB: Int, replyValue: Enumeration, client: Component) =
177       (true) -> (delta)
178     + (true) -> ((((lock_p_thread) | ((emptyQ_s) | (Controller'AI'MakeA))) |
            (lockQ_s_ControlUnit(LOCK''ControlUnit'sbs_ControlUnit))) . (((qEmpty_s_ControlUnit) .
            ((((outwardReply(Controller'AI'AIRETURNENUM'FAILREPLY)) | (unlockQ_s_ControlUnit)) .
            (unlock_p_thread)) .
            (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
            ControlUnit'sbs_ControlUnit'countB, NoReplyValue, client)))) + (((qNonEmpty_s_ControlUnit)
            . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit)))) .
            (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
            ControlUnit'sbs_ControlUnit'countB, Controller'AI'AIRETURNENUM'FAILREPLY, client)))))
179     + (true) -> ((((lock_p_thread) | ((emptyQ_s) | (Controller'AI'MakeB))) |
            (lockQ_s_ControlUnit(LOCK''ControlUnit'sbs_ControlUnit))) . (((qEmpty_s_ControlUnit) .
            ((((outwardReply(Controller'AI'AIRETURNENUM'FAILREPLY)) | (unlockQ_s_ControlUnit)) .
            (unlock_p_thread)) .
            (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
            ControlUnit'sbs_ControlUnit'countB, NoReplyValue, client)))) + (((qNonEmpty_s_ControlUnit)
            . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit)))) .
            (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
            ControlUnit'sbs_ControlUnit'countB, Controller'AI'AIRETURNENUM'FAILREPLY, client)))))
180     + (true) -> ((((queue_unlock_p_thread(ControlUnit)) | (lock_p_thread)) |
            (readNotification_ControlUnit(MachineA'NI'Failed))) . (((qEmpty_s_ControlUnit) .
            ((((replyValue) != (NoReplyValue)) -> (((outwardReply(replyValue)) |
            (unlockQ_s_ControlUnit)) . (unlock_p_thread)) <> ((unlockQ_s_ControlUnit) .
            (unlock_p_thread))) .
            (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
            ControlUnit'sbs_ControlUnit'countB, NoReplyValue, client)))) + (((qNonEmpty_s_ControlUnit)
            . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit)))) .
            (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
            ControlUnit'sbs_ControlUnit'countB, replyValue, client)))))
181     + (true) -> ((((queue_unlock_p_thread(ControlUnit)) | (lock_p_thread)) |
            (readNotification_ControlUnit(MachineA'NI'Done))) . (((qEmpty_s_ControlUnit) .
            ((((replyValue) != (NoReplyValue)) -> (((outwardReply(replyValue)) |
            (unlockQ_s_ControlUnit)) . (unlock_p_thread)) <> ((unlockQ_s_ControlUnit) .
            (unlock_p_thread))) .
            (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
            ControlUnit'sbs_ControlUnit'countB, NoReplyValue, client)))) + (((qNonEmpty_s_ControlUnit)
            . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit)))) .
            (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
            ControlUnit'sbs_ControlUnit'countB, replyValue, client)))))
182     + (true) -> ((((queue_unlock_p_thread(ControlUnit)) | (lock_p_thread)) |
            (readNotification_ControlUnit(MachineB'NI'Failed))) . (((qEmpty_s_ControlUnit) .
            ((((replyValue) != (NoReplyValue)) -> (((outwardReply(replyValue)) |
            (unlockQ_s_ControlUnit)) . (unlock_p_thread)) <> ((unlockQ_s_ControlUnit) .
            (unlock_p_thread))) .
            (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
            ControlUnit'sbs_ControlUnit'countB, NoReplyValue, client)))) + (((qNonEmpty_s_ControlUnit)
            . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit)))) .
            (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
            ControlUnit'sbs_ControlUnit'countB, replyValue, client)))))
183     + (true) -> ((((queue_unlock_p_thread(ControlUnit)) | (lock_p_thread)) |
            (readNotification_ControlUnit(MachineB'NI'Done))) . (((qEmpty_s_ControlUnit) .
            ((((replyValue) != (NoReplyValue)) -> (((outwardReply(replyValue)) |
            (unlockQ_s_ControlUnit)) . (unlock_p_thread)) <> ((unlockQ_s_ControlUnit) .
            (unlock_p_thread))) .
            (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
            ControlUnit'sbs_ControlUnit'countB, NoReplyValue, client)))) + (((qNonEmpty_s_ControlUnit)
            . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit)))) .
            (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
            ControlUnit'sbs_ControlUnit'countB, replyValue, client)))));
184 proc ControlUnit'sbs_ControlUnit'Initial(ControlUnit'sbs_ControlUnit'countA: Int,
```

```
     ControlUnit'sbs_ControlUnit'countB: Int, replyValue: Enumeration, client: Component) =
185      (true) -> ((((lock_p_thread) | ((emptyQ_s) | (Controller'AI'Initialize))) |
             (lockQ_s_ControlUnit(LOCK''ControlUnit'sbs_ControlUnit))) .
             (((invoke''MachineA'AI'Initialize(ControlUnit)) . (readReply(VoidReply, UnitA,
             ControlUnit))) . (((invoke''MachineB'AI'Initialize(ControlUnit)) . (readReply(VoidReply,
             UnitB, ControlUnit))) . (((qEmpty_s_ControlUnit) . ((((outwardReply(VoidReply)) |
             (unlockQ_s_ControlUnit)) . (unlock_p_thread)) .
             (ControlUnit'sbs_ControlUnit'Ready(ControlUnit'sbs_ControlUnit'countA,
             ControlUnit'sbs_ControlUnit'countB, NoReplyValue, NoClient)))) +
             (((qNonEmpty_s_ControlUnit) . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit))))
             . (ControlUnit'sbs_ControlUnit'Ready(ControlUnit'sbs_ControlUnit'countA,
             ControlUnit'sbs_ControlUnit'countB, VoidReply, NoClient)))))))
186    + (true) -> (delta)
187    + (true) -> (delta)
188    + (true) -> ((((queue_unlock_p_thread(ControlUnit)) | (lock_p_thread)) |
           (readNotification_ControlUnit(MachineA'NI'Failed))) . (Terminate))
189    + (true) -> ((((queue_unlock_p_thread(ControlUnit)) | (lock_p_thread)) |
           (readNotification_ControlUnit(MachineA'NI'Done))) . (Terminate))
190    + (true) -> ((((queue_unlock_p_thread(ControlUnit)) | (lock_p_thread)) |
           (readNotification_ControlUnit(MachineB'NI'Failed))) . (Terminate))
191    + (true) -> ((((queue_unlock_p_thread(ControlUnit)) | (lock_p_thread)) |
           (readNotification_ControlUnit(MachineB'NI'Done))) . (Terminate));
192  proc ControlUnit'sbs_ControlUnit'Ready(ControlUnit'sbs_ControlUnit'countA: Int,
     ControlUnit'sbs_ControlUnit'countB: Int, replyValue: Enumeration, client: Component) =
193      (true) -> (delta)
194    + ((ControlUnit'sbs_ControlUnit'countA) < (2)) -> ((((lock_p_thread) | ((emptyQ_s) |
           (Controller'AI'MakeA))) | (lockQ_s_ControlUnit(LOCK''ControlUnit'sbs_ControlUnit))) .
           (((invoke''MachineA'AI'Make(ControlUnit)) . (readReply(VoidReply, UnitA, ControlUnit))) .
           (((qEmpty_s_ControlUnit) . ((((outwardReply(Controller'AI'AIRETURNENUM'OKREPLY)) |
           (unlockQ_s_ControlUnit)) . (unlock_p_thread)) .
           (ControlUnit'sbs_ControlUnit'Ready((ControlUnit'sbs_ControlUnit'countA) + (1),
           ControlUnit'sbs_ControlUnit'countB, NoReplyValue, client)))) + (((qNonEmpty_s_ControlUnit)
           . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit)))) .
           (ControlUnit'sbs_ControlUnit'Ready((ControlUnit'sbs_ControlUnit'countA) + (1),
           ControlUnit'sbs_ControlUnit'countB, Controller'AI'AIRETURNENUM'OKREPLY, client))))))
195    + (!((ControlUnit'sbs_ControlUnit'countA) < (2))) -> ((((lock_p_thread) | ((emptyQ_s) |
           (Controller'AI'MakeA))) | (lockQ_s_ControlUnit(LOCK''ControlUnit'sbs_ControlUnit))) .
           (((qEmpty_s_ControlUnit) . ((((outwardReply(Controller'AI'AIRETURNENUM'FAILREPLY)) |
           (unlockQ_s_ControlUnit)) . (unlock_p_thread)) .
           (ControlUnit'sbs_ControlUnit'Ready(ControlUnit'sbs_ControlUnit'countA,
           ControlUnit'sbs_ControlUnit'countB, NoReplyValue, client)))) + (((qNonEmpty_s_ControlUnit)
           . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit)))) .
           (ControlUnit'sbs_ControlUnit'Ready(ControlUnit'sbs_ControlUnit'countA,
           ControlUnit'sbs_ControlUnit'countB, Controller'AI'AIRETURNENUM'FAILREPLY, client)))))
196    + ((ControlUnit'sbs_ControlUnit'countB) < (3)) -> ((((lock_p_thread) | ((emptyQ_s) |
           (Controller'AI'MakeB))) | (lockQ_s_ControlUnit(LOCK''ControlUnit'sbs_ControlUnit))) .
           (((invoke''MachineB'AI'Make(ControlUnit)) . (readReply(VoidReply, UnitB, ControlUnit))) .
           (((qEmpty_s_ControlUnit) . ((((outwardReply(Controller'AI'AIRETURNENUM'OKREPLY)) |
           (unlockQ_s_ControlUnit)) . (unlock_p_thread)) .
           (ControlUnit'sbs_ControlUnit'Ready(ControlUnit'sbs_ControlUnit'countA,
           (ControlUnit'sbs_ControlUnit'countB) + (1), NoReplyValue, client)))) +
           (((qNonEmpty_s_ControlUnit) . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit)))) .
           (ControlUnit'sbs_ControlUnit'Ready(ControlUnit'sbs_ControlUnit'countA,
           (ControlUnit'sbs_ControlUnit'countB) + (1), Controller'AI'AIRETURNENUM'OKREPLY,
           client))))))
197    + (!((ControlUnit'sbs_ControlUnit'countB) < (3))) -> ((((lock_p_thread) | ((emptyQ_s) |
           (Controller'AI'MakeB))) | (lockQ_s_ControlUnit(LOCK''ControlUnit'sbs_ControlUnit))) .
           (((qEmpty_s_ControlUnit) . ((((outwardReply(Controller'AI'AIRETURNENUM'FAILREPLY)) |
           (unlockQ_s_ControlUnit)) . (unlock_p_thread)) .
           (ControlUnit'sbs_ControlUnit'Ready(ControlUnit'sbs_ControlUnit'countA,
           ControlUnit'sbs_ControlUnit'countB, NoReplyValue, client)))) + (((qNonEmpty_s_ControlUnit)
           . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit)))) .
           (ControlUnit'sbs_ControlUnit'Ready(ControlUnit'sbs_ControlUnit'countA,
           ControlUnit'sbs_ControlUnit'countB, Controller'AI'AIRETURNENUM'FAILREPLY, client)))))
198    + (true) -> ((((queue_unlock_p_thread(ControlUnit)) | (lock_p_thread)) |
           (readNotification_ControlUnit(MachineA'NI'Failed))) .
           ((outwardNotification(Controller'NI'Error)) . (((qEmpty_s_ControlUnit) . ((((replyValue)
```

```
        != (NoReplyValue)) -> (((outwardReply(replyValue)) | (unlockQ_s_ControlUnit)) .
        (unlock_p_thread)) <> ((unlockQ_s_ControlUnit) . (unlock_p_thread))) .
        (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
        ControlUnit'sbs_ControlUnit'countB, NoReplyValue, NoClient)))) +
        (((qNonEmpty_s_ControlUnit) . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit)))) .
        (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
        ControlUnit'sbs_ControlUnit'countB, replyValue, NoClient))))))
199     + (true) -> ((((queue_unlock_p_thread(ControlUnit)) | (lock_p_thread)) |
        (readNotification_ControlUnit(MachineA'NI'Done))) .
        ((outwardNotification(Controller'NI'MadeA)) . (((qEmpty_s_ControlUnit) . ((((replyValue)
        != (NoReplyValue)) -> (((outwardReply(replyValue)) | (unlockQ_s_ControlUnit)) .
        (unlock_p_thread)) <> ((unlockQ_s_ControlUnit) . (unlock_p_thread))) .
        (ControlUnit'sbs_ControlUnit'Ready((ControlUnit'sbs_ControlUnit'countA) - (1),
        ControlUnit'sbs_ControlUnit'countB, NoReplyValue, client)))) + (((qNonEmpty_s_ControlUnit)
        . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit)))) .
        (ControlUnit'sbs_ControlUnit'Ready((ControlUnit'sbs_ControlUnit'countA) - (1),
        ControlUnit'sbs_ControlUnit'countB, replyValue, client))))))
200     + (true) -> ((((queue_unlock_p_thread(ControlUnit)) | (lock_p_thread)) |
        (readNotification_ControlUnit(MachineB'NI'Failed))) .
        ((outwardNotification(Controller'NI'Error)) . (((qEmpty_s_ControlUnit) . ((((replyValue)
        != (NoReplyValue)) -> (((outwardReply(replyValue)) | (unlockQ_s_ControlUnit)) .
        (unlock_p_thread)) <> ((unlockQ_s_ControlUnit) . (unlock_p_thread))) .
        (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
        ControlUnit'sbs_ControlUnit'countB, NoReplyValue, NoClient)))) +
        (((qNonEmpty_s_ControlUnit) . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit)))) .
        (ControlUnit'sbs_ControlUnit'Error(ControlUnit'sbs_ControlUnit'countA,
        ControlUnit'sbs_ControlUnit'countB, replyValue, NoClient))))))
201     + (true) -> ((((queue_unlock_p_thread(ControlUnit)) | (lock_p_thread)) |
        (readNotification_ControlUnit(MachineB'NI'Done))) .
        ((outwardNotification(Controller'NI'MadeB)) . (((qEmpty_s_ControlUnit) . ((((replyValue)
        != (NoReplyValue)) -> (((outwardReply(replyValue)) | (unlockQ_s_ControlUnit)) .
        (unlock_p_thread)) <> ((unlockQ_s_ControlUnit) . (unlock_p_thread))) .
        (ControlUnit'sbs_ControlUnit'Ready(ControlUnit'sbs_ControlUnit'countA,
        (ControlUnit'sbs_ControlUnit'countB) - (1), NoReplyValue, client)))) +
        (((qNonEmpty_s_ControlUnit) . ((unlock_p_thread) | (queue_lock_p_thread(ControlUnit)))) .
        (ControlUnit'sbs_ControlUnit'Ready(ControlUnit'sbs_ControlUnit'countA,
        (ControlUnit'sbs_ControlUnit'countB) - (1), replyValue, client))))));
202 proc LoaderUnit'sbs_LoaderUnit'Aligning(LoaderUnit'sbs_LoaderUnit'queued: Int, replyValue:
     Enumeration, client: Component) =
203      (true) -> (sum c: Component . (((invoked''Loader'AI'Initialize(c)) |
           (lockQ_s_LoaderUnit(LOCK''LoaderUnit'sbs_LoaderUnit))) . (((qEmpty_s_LoaderUnit) .
           (((writeReply(VoidReply, LoaderUnit, c)) | (unlockQ_s_LoaderUnit)) .
           (LoaderUnit'sbs_LoaderUnit'Aligning(LoaderUnit'sbs_LoaderUnit'queued, NoReplyValue,
           client)))) + (((qNonEmpty_s_LoaderUnit) . ((unlock_p_thread) |
           (queue_lock_p_thread(LoaderUnit)))) .
           (LoaderUnit'sbs_LoaderUnit'Aligning(LoaderUnit'sbs_LoaderUnit'queued, VoidReply,
           client))))))
204     + ((LoaderUnit'sbs_LoaderUnit'queued) < (4)) -> (sum c: Component .
        (((invoked''Loader'AI'Load(c)) | (lockQ_s_LoaderUnit(LOCK''LoaderUnit'sbs_LoaderUnit))) .
        (((qEmpty_s_LoaderUnit) . (((writeReply(Loader'AI'AIRETURNENUM'OKREPLY, LoaderUnit, c)) |
        (unlockQ_s_LoaderUnit)) .
        (LoaderUnit'sbs_LoaderUnit'Aligning((LoaderUnit'sbs_LoaderUnit'queued) + (1),
        NoReplyValue, client)))) + (((qNonEmpty_s_LoaderUnit) . ((unlock_p_thread) |
        (queue_lock_p_thread(LoaderUnit)))) .
        (LoaderUnit'sbs_LoaderUnit'Aligning((LoaderUnit'sbs_LoaderUnit'queued) + (1),
        Loader'AI'AIRETURNENUM'OKREPLY, client))))))
205     + (!((LoaderUnit'sbs_LoaderUnit'queued) < (4))) -> (sum c: Component .
        (((invoked''Loader'AI'Load(c)) | (lockQ_s_LoaderUnit(LOCK''LoaderUnit'sbs_LoaderUnit))) .
        (((qEmpty_s_LoaderUnit) . (((writeReply(Loader'AI'AIRETURNENUM'FAILREPLY, LoaderUnit, c))
        | (unlockQ_s_LoaderUnit)) .
        (LoaderUnit'sbs_LoaderUnit'Aligning(LoaderUnit'sbs_LoaderUnit'queued, NoReplyValue,
        client)))) + (((qNonEmpty_s_LoaderUnit) . ((unlock_p_thread) |
        (queue_lock_p_thread(LoaderUnit)))) .
        (LoaderUnit'sbs_LoaderUnit'Aligning(LoaderUnit'sbs_LoaderUnit'queued,
        Loader'AI'AIRETURNENUM'FAILREPLY, client))))))
206     + ((LoaderUnit'sbs_LoaderUnit'queued) <= (1)) -> ((((queue_unlock_p_thread(LoaderUnit)) |
        (lock_p_thread)) | (readNotification_LoaderUnit(RobotArm'NI'Success))) .
```

```
        (((pushNotification_UnitB(Loader'NI'Aligned)) +
        (pushNotification_UnitA(Loader'NI'Aligned))) . (((qEmpty_s_LoaderUnit) . ((((replyValue)
        != (NoReplyValue)) -> ((writeReply(replyValue, LoaderUnit, client)) |
        (unlockQ_s_LoaderUnit)) <> ((unlockQ_s_LoaderUnit) . (unlock_p_thread))) .
        (LoaderUnit'sbs_LoaderUnit'Ready(0, NoReplyValue, NoClient)))) +
        (((qNonEmpty_s_LoaderUnit) . ((unlock_p_thread) | (queue_lock_p_thread(LoaderUnit)))) .
        (LoaderUnit'sbs_LoaderUnit'Ready(0, replyValue, NoClient))))))
207     + (!((LoaderUnit'sbs_LoaderUnit'queued) <= (1))) -> ((((queue_unlock_p_thread(LoaderUnit)) |
        (lock_p_thread)) | (readNotification_LoaderUnit(RobotArm'NI'Success))) .
        (((invoke''RobotArm'AI'Move(LoaderUnit)) . (readReply(VoidReply, RobotArm, LoaderUnit))) .
        (((pushNotification_UnitB(Loader'NI'Aligned)) +
        (pushNotification_UnitA(Loader'NI'Aligned))) . (((qEmpty_s_LoaderUnit) . ((((replyValue)
        != (NoReplyValue)) -> ((writeReply(replyValue, LoaderUnit, client)) |
        (unlockQ_s_LoaderUnit)) <> ((unlockQ_s_LoaderUnit) . (unlock_p_thread))) .
        (LoaderUnit'sbs_LoaderUnit'Aligning((LoaderUnit'sbs_LoaderUnit'queued) - (1),
        NoReplyValue, client)))) + (((qNonEmpty_s_LoaderUnit) . ((unlock_p_thread) |
        (queue_lock_p_thread(LoaderUnit)))) .
        (LoaderUnit'sbs_LoaderUnit'Aligning((LoaderUnit'sbs_LoaderUnit'queued) - (1), replyValue,
        client)))))))
208     + (true) -> ((((queue_unlock_p_thread(LoaderUnit)) | (lock_p_thread)) |
        (readNotification_LoaderUnit(RobotArm'NI'Failure))) .
        (((pushNotification_UnitB(Loader'NI'Failure)) +
        (pushNotification_UnitA(Loader'NI'Failure))) . (((qEmpty_s_LoaderUnit) . ((((replyValue)
        != (NoReplyValue)) -> ((writeReply(replyValue, LoaderUnit, client)) |
        (unlockQ_s_LoaderUnit)) <> ((unlockQ_s_LoaderUnit) . (unlock_p_thread))) .
        (LoaderUnit'sbs_LoaderUnit'Error(LoaderUnit'sbs_LoaderUnit'queued, NoReplyValue,
        NoClient)))) + (((qNonEmpty_s_LoaderUnit) . ((unlock_p_thread) |
        (queue_lock_p_thread(LoaderUnit)))) .
        (LoaderUnit'sbs_LoaderUnit'Error(LoaderUnit'sbs_LoaderUnit'queued, replyValue,
        NoClient))))));
209 proc LoaderUnit'sbs_LoaderUnit'Error(LoaderUnit'sbs_LoaderUnit'queued: Int, replyValue:
        Enumeration, client: Component) =
210      (true) -> (sum c: Component . (((invoked''Loader'AI'Initialize(c)) |
            (lockQ_s_LoaderUnit(LOCK''LoaderUnit'sbs_LoaderUnit))) . (((qEmpty_s_LoaderUnit) .
            (((writeReply(VoidReply, LoaderUnit, c)) | (unlockQ_s_LoaderUnit)) .
            (LoaderUnit'sbs_LoaderUnit'Error(LoaderUnit'sbs_LoaderUnit'queued, NoReplyValue,
            client)))) + (((qNonEmpty_s_LoaderUnit) . ((unlock_p_thread) |
            (queue_lock_p_thread(LoaderUnit)))) .
            (LoaderUnit'sbs_LoaderUnit'Error(LoaderUnit'sbs_LoaderUnit'queued, VoidReply,
            client))))))
211     + (true) -> (sum c: Component . (((invoked''Loader'AI'Load(c)) |
           (lockQ_s_LoaderUnit(LOCK''LoaderUnit'sbs_LoaderUnit))) . (((qEmpty_s_LoaderUnit) .
           (((writeReply(Loader'AI'AIRETURNENUM'FAILREPLY, LoaderUnit, c)) | (unlockQ_s_LoaderUnit))
           . (LoaderUnit'sbs_LoaderUnit'Error(LoaderUnit'sbs_LoaderUnit'queued, NoReplyValue,
           client)))) + (((qNonEmpty_s_LoaderUnit) . ((unlock_p_thread) |
           (queue_lock_p_thread(LoaderUnit)))) .
           (LoaderUnit'sbs_LoaderUnit'Error(LoaderUnit'sbs_LoaderUnit'queued,
           Loader'AI'AIRETURNENUM'FAILREPLY, client))))))
212     + (true) -> ((((queue_unlock_p_thread(LoaderUnit)) | (lock_p_thread)) |
           (readNotification_LoaderUnit(RobotArm'NI'Success))) . (((qEmpty_s_LoaderUnit) .
           ((((replyValue) != (NoReplyValue)) -> ((writeReply(replyValue, LoaderUnit, client)) |
           (unlockQ_s_LoaderUnit)) <> ((unlockQ_s_LoaderUnit) . (unlock_p_thread))) .
           (LoaderUnit'sbs_LoaderUnit'Error(LoaderUnit'sbs_LoaderUnit'queued, NoReplyValue,
           client)))) + (((qNonEmpty_s_LoaderUnit) . ((unlock_p_thread) |
           (queue_lock_p_thread(LoaderUnit)))) .
           (LoaderUnit'sbs_LoaderUnit'Error(LoaderUnit'sbs_LoaderUnit'queued, replyValue, client)))))
213     + (true) -> ((((queue_unlock_p_thread(LoaderUnit)) | (lock_p_thread)) |
           (readNotification_LoaderUnit(RobotArm'NI'Failure))) . (((qEmpty_s_LoaderUnit) .
           ((((replyValue) != (NoReplyValue)) -> ((writeReply(replyValue, LoaderUnit, client)) |
           (unlockQ_s_LoaderUnit)) <> ((unlockQ_s_LoaderUnit) . (unlock_p_thread))) .
           (LoaderUnit'sbs_LoaderUnit'Error(LoaderUnit'sbs_LoaderUnit'queued, NoReplyValue,
           client)))) + (((qNonEmpty_s_LoaderUnit) . ((unlock_p_thread) |
           (queue_lock_p_thread(LoaderUnit)))) .
           (LoaderUnit'sbs_LoaderUnit'Error(LoaderUnit'sbs_LoaderUnit'queued, replyValue, client)))));
214 proc LoaderUnit'sbs_LoaderUnit'Initial(LoaderUnit'sbs_LoaderUnit'queued: Int, replyValue:
        Enumeration, client: Component) =
215      (true) -> (sum c: Component . (((invoked''Loader'AI'Initialize(c)) |
```

```
          (lockQ_s_LoaderUnit(LOCK''LoaderUnit'sbs_LoaderUnit))) .
          (((invoke''RobotArm'AI'Initialize(LoaderUnit)) . (readReply(VoidReply, RobotArm,
          LoaderUnit))) . (((qEmpty_s_LoaderUnit) . (((writeReply(VoidReply, LoaderUnit, c)) |
          (unlockQ_s_LoaderUnit)) .
          (LoaderUnit'sbs_LoaderUnit'Ready(LoaderUnit'sbs_LoaderUnit'queued, NoReplyValue,
          NoClient)))) + (((qNonEmpty_s_LoaderUnit) . ((unlock_p_thread) |
          (queue_lock_p_thread(LoaderUnit)))) .
          (LoaderUnit'sbs_LoaderUnit'Ready(LoaderUnit'sbs_LoaderUnit'queued, VoidReply, c)))))))
    + (true) -> (delta)
    + (true) -> ((((queue_unlock_p_thread(LoaderUnit)) | (lock_p_thread)) |
       (readNotification_LoaderUnit(RobotArm'NI'Success))) . (Terminate))
    + (true) -> ((((queue_unlock_p_thread(LoaderUnit)) | (lock_p_thread)) |
       (readNotification_LoaderUnit(RobotArm'NI'Failure))) . (Terminate));
proc LoaderUnit'sbs_LoaderUnit'Ready(LoaderUnit'sbs_LoaderUnit'queued: Int, replyValue:
     Enumeration, client: Component) =
      (true) -> (sum c: Component . (((invoked''Loader'AI'Initialize(c)) |
          (lockQ_s_LoaderUnit(LOCK''LoaderUnit'sbs_LoaderUnit))) . (((qEmpty_s_LoaderUnit) .
          (((writeReply(VoidReply, LoaderUnit, c)) | (unlockQ_s_LoaderUnit)) .
          (LoaderUnit'sbs_LoaderUnit'Ready(LoaderUnit'sbs_LoaderUnit'queued, NoReplyValue,
          client)))) + (((qNonEmpty_s_LoaderUnit) . ((unlock_p_thread) |
          (queue_lock_p_thread(LoaderUnit)))) .
          (LoaderUnit'sbs_LoaderUnit'Ready(LoaderUnit'sbs_LoaderUnit'queued, VoidReply,
          client))))))
    + (true) -> (sum c: Component . (((invoked''Loader'AI'Load(c)) |
       (lockQ_s_LoaderUnit(LOCK''LoaderUnit'sbs_LoaderUnit))) .
       (((invoke''RobotArm'AI'Move(LoaderUnit)) . (readReply(VoidReply, RobotArm, LoaderUnit))) .
       (((qEmpty_s_LoaderUnit) . (((writeReply(Loader'AI'AIRETURNENUM'OKREPLY, LoaderUnit, c)) |
       (unlockQ_s_LoaderUnit)) . (LoaderUnit'sbs_LoaderUnit'Aligning(1, NoReplyValue,
       NoClient)))) + (((qNonEmpty_s_LoaderUnit) . ((unlock_p_thread) |
       (queue_lock_p_thread(LoaderUnit)))) . (LoaderUnit'sbs_LoaderUnit'Aligning(1,
       Loader'AI'AIRETURNENUM'OKREPLY, c)))))))
    + (true) -> ((((queue_unlock_p_thread(LoaderUnit)) | (lock_p_thread)) |
       (readNotification_LoaderUnit(RobotArm'NI'Success))) . (Terminate))
    + (true) -> ((((queue_unlock_p_thread(LoaderUnit)) | (lock_p_thread)) |
       (readNotification_LoaderUnit(RobotArm'NI'Failure))) . (Terminate));
proc Queue_ControlUnit(q: List(Notification), locked: LockingState) =
     sum n: Notification . ((receiveNotification_ControlUnit(n)) . (Queue_ControlUnit((q) <| (n),
          locked)))
    + (((#(q)) > (0)) && (((locked) == (NONE)) || ((locked) ==
       (LOCK''ControlUnit'sbs_ControlUnit)))) -> ((sendNotification_ControlUnit(head(q))) .
       (Queue_ControlUnit(tail(q), LOCK''ControlUnit'sbs_ControlUnit)))
    + sum s: LockingState . ((((#(q)) == (0)) && (((locked) == (NONE)) || ((locked) == (s)))) ->
       ((lockQ_r_ControlUnit(s)) . (Queue_ControlUnit(q, s))))
    + sum s: LockingState . ((((#(q)) == (0)) && (((locked) == (NONE)) || ((locked) == (s)))) ->
       (((lockQ_r_ControlUnit(s)) | (emptyQ_r_ControlUnit)) . (Queue_ControlUnit(q, s))))
    + ((locked) != (NONE)) -> ((unlockQ_r_ControlUnit) . (Queue_ControlUnit(q, NONE)))
    + ((#(q)) == (0)) -> ((qEmpty_r_ControlUnit) . (Queue_ControlUnit(q, locked)))
    + ((#(q)) == (0)) -> ((emptyQ_r_ControlUnit) . (Queue_ControlUnit(q, locked)))
    + ((#(q)) > (0)) -> ((qNonEmpty_r_ControlUnit) . (Queue_ControlUnit(q, locked)))
    + ((#(q)) > (7)) -> ((queueSizeViolated) . (delta));
proc Queue_LoaderUnit(q: List(Notification), locked: LockingState) =
     sum n: Notification . ((receiveNotification_LoaderUnit(n)) . (Queue_LoaderUnit((q) <| (n),
          locked)))
    + (((#(q)) > (0)) && (((locked) == (NONE)) || ((locked) == (LOCK''LoaderUnit'sbs_LoaderUnit))))
       -> ((sendNotification_LoaderUnit(head(q))) . (Queue_LoaderUnit(tail(q),
       LOCK''LoaderUnit'sbs_LoaderUnit)))
    + sum s: LockingState . ((((#(q)) == (0)) && (((locked) == (NONE)) || ((locked) == (s)))) ->
       ((lockQ_r_LoaderUnit(s)) . (Queue_LoaderUnit(q, s))))
    + sum s: LockingState . ((((#(q)) == (0)) && (((locked) == (NONE)) || ((locked) == (s)))) ->
       (((lockQ_r_LoaderUnit(s)) | (emptyQ_r_LoaderUnit)) . (Queue_LoaderUnit(q, s))))
    + ((locked) != (NONE)) -> ((unlockQ_r_LoaderUnit) . (Queue_LoaderUnit(q, NONE)))
    + ((#(q)) == (0)) -> ((qEmpty_r_LoaderUnit) . (Queue_LoaderUnit(q, locked)))
    + ((#(q)) == (0)) -> ((emptyQ_r_LoaderUnit) . (Queue_LoaderUnit(q, locked)))
    + ((#(q)) > (0)) -> ((qNonEmpty_r_LoaderUnit) . (Queue_LoaderUnit(q, locked)))
    + ((#(q)) > (7)) -> ((queueSizeViolated) . (delta));
proc Queue_UnitA(q: List(Notification), locked: LockingState) =
     sum n: Notification . ((receiveNotification_UnitA(n)) . (Queue_UnitA((q) <| (n), locked)))
```

```
246     + (((#(q)) > (0)) && (((locked) == (NONE)) || ((locked) == (LOCK''UnitA'sbs_UnitA)))) ->
            ((sendNotification_UnitA(head(q))) . (Queue_UnitA(tail(q), LOCK''UnitA'sbs_UnitA)))
247     + sum s: LockingState . ((((#(q)) == (0)) && (((locked) == (NONE)) || ((locked) == (s)))) ->
            ((lockQ_r_UnitA(s)) . (Queue_UnitA(q, s))))
248     + sum s: LockingState . ((((#(q)) == (0)) && (((locked) == (NONE)) || ((locked) == (s)))) ->
            (((lockQ_r_UnitA(s)) | (emptyQ_r_UnitA)) . (Queue_UnitA(q, s))))
249     + ((locked) != (NONE)) -> ((unlockQ_r_UnitA) . (Queue_UnitA(q, NONE)))
250     + ((#(q)) == (0)) -> ((qEmpty_r_UnitA) . (Queue_UnitA(q, locked)))
251     + ((#(q)) == (0)) -> ((emptyQ_r_UnitA) . (Queue_UnitA(q, locked)))
252     + ((#(q)) > (0)) -> ((qNonEmpty_r_UnitA) . (Queue_UnitA(q, locked)))
253     + ((#(q)) > (7)) -> ((queueSizeViolated) . (delta));
254 proc Queue_UnitB(q: List(Notification), locked: LockingState) =
255       sum n: Notification . ((receiveNotification_UnitB(n)) . (Queue_UnitB((q) <| (n), locked)))
256     + (((#(q)) > (0)) && (((locked) == (NONE)) || ((locked) == (LOCK''UnitB'sbs_UnitB)))) ->
            ((sendNotification_UnitB(head(q))) . (Queue_UnitB(tail(q), LOCK''UnitB'sbs_UnitB)))
257     + sum s: LockingState . ((((#(q)) == (0)) && (((locked) == (NONE)) || ((locked) == (s)))) ->
            ((lockQ_r_UnitB(s)) . (Queue_UnitB(q, s))))
258     + sum s: LockingState . ((((#(q)) == (0)) && (((locked) == (NONE)) || ((locked) == (s)))) ->
            (((lockQ_r_UnitB(s)) | (emptyQ_r_UnitB)) . (Queue_UnitB(q, s))))
259     + ((locked) != (NONE)) -> ((unlockQ_r_UnitB) . (Queue_UnitB(q, NONE)))
260     + ((#(q)) == (0)) -> ((qEmpty_r_UnitB) . (Queue_UnitB(q, locked)))
261     + ((#(q)) == (0)) -> ((emptyQ_r_UnitB) . (Queue_UnitB(q, locked)))
262     + ((#(q)) > (0)) -> ((qNonEmpty_r_UnitB) . (Queue_UnitB(q, locked)))
263     + ((#(q)) > (7)) -> ((queueSizeViolated) . (delta));
264 proc RobotArm'RobotArm_sync'RobotArm'Initial(client: Component) =
265       (true) -> (sum c: Component . ((invoked''RobotArm'AI'Initialize(c)) . ((writeReply(VoidReply,
              RobotArm, c)) . (RobotArm'RobotArm_sync'RobotArm'Ready(NoClient)))))
266     + (true) -> (sum c: Component . ((invoked''RobotArm'AI'Move(c)) . (Terminate)))
267     + (true) -> (delta)
268     + (true) -> (delta);
269 proc RobotArm'RobotArm_sync'RobotArm'Moving(client: Component) =
270       (true) -> (sum c: Component . ((invoked''RobotArm'AI'Initialize(c)) . (Terminate)))
271     + (true) -> (sum c: Component . ((invoked''RobotArm'AI'Move(c)) . (Terminate)))
272     + (true) -> (((optionalInternalTrigger) | ((lock_p_thread) | ((emptyQ_s) |
            (lockQ_s_LoaderUnit(LOCK''RobotArm'RobotArm_sync'RobotArm))))) .
            ((pushNotification_LoaderUnit(RobotArm'NI'Failure)) . (((unlockQ_s_LoaderUnit) .
            (unlock_p_thread)) . (RobotArm'RobotArm_sync'RobotArm'Ready(NoClient)))))
273     + (true) -> (((inevitableInternalTrigger) | ((lock_p_thread) | ((emptyQ_s) |
            (lockQ_s_LoaderUnit(LOCK''RobotArm'RobotArm_sync'RobotArm))))) .
            ((pushNotification_LoaderUnit(RobotArm'NI'Success)) . (((unlockQ_s_LoaderUnit) .
            (unlock_p_thread)) . (RobotArm'RobotArm_sync'RobotArm'Ready(NoClient)))));
274 proc RobotArm'RobotArm_sync'RobotArm'Ready(client: Component) =
275       (true) -> (sum c: Component . ((invoked''RobotArm'AI'Initialize(c)) . (Terminate)))
276     + (true) -> (sum c: Component . ((invoked''RobotArm'AI'Move(c)) . ((writeReply(VoidReply,
            RobotArm, c)) . (RobotArm'RobotArm_sync'RobotArm'Moving(NoClient)))))
277     + (true) -> (delta)
278     + (true) -> (delta);
279 proc Terminate = (terminate) . (delta);
280 proc Thread = ((lock_t_thread) . (unlock_t_thread)) . (Thread);
281 proc UnitA'sbs_UnitA'Error(UnitA'sbs_UnitA'count: Int, replyValue: Enumeration, client: Component)
        =
282       (true) -> (delta)
283     + (true) -> (sum c: Component . (((invoked''MachineA'AI'Make(c)) |
            (lockQ_s_UnitA(LOCK''UnitA'sbs_UnitA))) .
            ((pushNotification_ControlUnit(MachineA'NI'Failed)) . (((qEmpty_s_UnitA) .
            (((writeReply(VoidReply, UnitA, c)) | (unlockQ_s_UnitA)) .
            (UnitA'sbs_UnitA'Error(UnitA'sbs_UnitA'count, NoReplyValue, client)))) +
            (((qNonEmpty_s_UnitA) . ((unlock_p_thread) | (queue_lock_p_thread(UnitA)))) .
            (UnitA'sbs_UnitA'Error(UnitA'sbs_UnitA'count, VoidReply, client)))))))
284     + (true) -> ((((queue_unlock_p_thread(UnitA)) | (lock_p_thread)) |
            (readNotification_UnitA(Loader'NI'Aligned))) . (((qEmpty_s_UnitA) . ((((replyValue) !=
            (NoReplyValue)) -> ((writeReply(replyValue, UnitA, client)) | (unlockQ_s_UnitA)) <>
            ((unlockQ_s_UnitA) . (unlock_p_thread))) . (UnitA'sbs_UnitA'Error(UnitA'sbs_UnitA'count,
            NoReplyValue, client)))) + (((qNonEmpty_s_UnitA) . ((unlock_p_thread) |
            (queue_lock_p_thread(UnitA)))) . (UnitA'sbs_UnitA'Error(UnitA'sbs_UnitA'count, replyValue,
            client)))))
285     + (true) -> ((((queue_unlock_p_thread(UnitA)) | (lock_p_thread)) |
```

```
        (readNotification_UnitA(Loader'NI'Failure))) . (((qEmpty_s_UnitA) . ((((replyValue) !=
        (NoReplyValue)) -> ((writeReply(replyValue, UnitA, client)) | (unlockQ_s_UnitA)) <>
        ((unlockQ_s_UnitA) . (unlock_p_thread))) . (UnitA'sbs_UnitA'Error(UnitA'sbs_UnitA'count,
        NoReplyValue, client)))) + (((qNonEmpty_s_UnitA) . ((unlock_p_thread) |
        (queue_lock_p_thread(UnitA)))) . (UnitA'sbs_UnitA'Error(UnitA'sbs_UnitA'count, replyValue,
        client)))));
286 proc UnitA'sbs_UnitA'Initial(UnitA'sbs_UnitA'count: Int, replyValue: Enumeration, client:
    Component) =
287      (true) -> (sum c: Component . (((invoked''MachineA'AI'Initialize(c)) |
           (lockQ_s_UnitA(LOCK''UnitA'sbs_UnitA))) . (((invoke''Loader'AI'Initialize(UnitA)) .
           (readReply(VoidReply, LoaderUnit, UnitA))) . (((qEmpty_s_UnitA) .
           (((writeReply(VoidReply, UnitA, c)) | (unlockQ_s_UnitA)) .
           (UnitA'sbs_UnitA'Ready(UnitA'sbs_UnitA'count, NoReplyValue, NoClient)))) +
           (((qNonEmpty_s_UnitA) . ((unlock_p_thread) | (queue_lock_p_thread(UnitA)))) .
           (UnitA'sbs_UnitA'Ready(UnitA'sbs_UnitA'count, VoidReply, c)))))))
288    + (true) -> (delta)
289    + (true) -> ((((queue_unlock_p_thread(UnitA)) | (lock_p_thread)) |
           (readNotification_UnitA(Loader'NI'Aligned))) . (Terminate))
290    + (true) -> ((((queue_unlock_p_thread(UnitA)) | (lock_p_thread)) |
           (readNotification_UnitA(Loader'NI'Failure))) . (Terminate));
291 proc UnitA'sbs_UnitA'Ready(UnitA'sbs_UnitA'count: Int, replyValue: Enumeration, client: Component)
    =
292      (true) -> (delta)
293    + (true) -> (sum c: Component . (((invoked''MachineA'AI'Make(c)) |
           (lockQ_s_UnitA(LOCK''UnitA'sbs_UnitA))) . ((invoke''Loader'AI'Load(UnitA)) . (sum
           UnitA'sbs_UnitA'Ready'Load_EvalVar14: Enumeration .
           ((readReply(UnitA'sbs_UnitA'Ready'Load_EvalVar14, LoaderUnit, UnitA)) .
           (UnitA'sbs_UnitA'Requesting(1, UnitA'sbs_UnitA'Ready'Load_EvalVar14, replyValue, c)))))))
294    + (true) -> ((((queue_unlock_p_thread(UnitA)) | (lock_p_thread)) |
           (readNotification_UnitA(Loader'NI'Aligned))) . (Terminate))
295    + (true) -> ((((queue_unlock_p_thread(UnitA)) | (lock_p_thread)) |
           (readNotification_UnitA(Loader'NI'Failure))) .
           ((pushNotification_ControlUnit(MachineA'NI'Failed)) . (((qEmpty_s_UnitA) . ((((replyValue)
           != (NoReplyValue)) -> ((writeReply(replyValue, UnitA, client)) | (unlockQ_s_UnitA)) <>
           ((unlockQ_s_UnitA) . (unlock_p_thread))) . (UnitA'sbs_UnitA'Error(UnitA'sbs_UnitA'count,
           NoReplyValue, NoClient)))) + (((qNonEmpty_s_UnitA) . ((unlock_p_thread) |
           (queue_lock_p_thread(UnitA)))) . (UnitA'sbs_UnitA'Error(UnitA'sbs_UnitA'count, replyValue,
           NoClient))))));
296 proc UnitA'sbs_UnitA'Requesting(UnitA'sbs_UnitA'count: Int,
    UnitA'sbs_UnitA'Requesting'RequestingAIReturnEnumEP: Enumeration, replyValue: Enumeration,
    client: Component) =
297      ((UnitA'sbs_UnitA'Requesting'RequestingAIReturnEnumEP) == (Loader'AI'AIRETURNENUM'OKREPLY))
             -> ((valuedTrigger) . (((qEmpty_s_UnitA) . (((writeReply(VoidReply, UnitA, client)) |
           (unlockQ_s_UnitA)) . (UnitA'sbs_UnitA'Working(UnitA'sbs_UnitA'count, NoReplyValue,
           NoClient)))) + (((qNonEmpty_s_UnitA) . ((unlock_p_thread) |
           (queue_lock_p_thread(UnitA)))) . (UnitA'sbs_UnitA'Working(UnitA'sbs_UnitA'count,
           VoidReply, NoClient)))))
298    + ((UnitA'sbs_UnitA'Requesting'RequestingAIReturnEnumEP) == (Loader'AI'AIRETURNENUM'FAILREPLY))
           -> ((valuedTrigger) . ((pushNotification_ControlUnit(MachineA'NI'Failed)) .
           (((qEmpty_s_UnitA) . (((writeReply(VoidReply, UnitA, client)) | (unlockQ_s_UnitA)) .
           (UnitA'sbs_UnitA'Error(UnitA'sbs_UnitA'count, NoReplyValue, NoClient)))) +
           (((qNonEmpty_s_UnitA) . ((unlock_p_thread) | (queue_lock_p_thread(UnitA)))) .
           (UnitA'sbs_UnitA'Error(UnitA'sbs_UnitA'count, VoidReply, NoClient))))));
299 proc UnitA'sbs_UnitA'Working(UnitA'sbs_UnitA'count: Int, replyValue: Enumeration, client:
    Component) =
300      (true) -> (delta)
301    + ((UnitA'sbs_UnitA'count) < (2)) -> (sum c: Component . (((invoked''MachineA'AI'Make(c)) |
           (lockQ_s_UnitA(LOCK''UnitA'sbs_UnitA'))) . ((invoke''Loader'AI'Load(UnitA)) . (sum
           UnitA'sbs_UnitA'Working'Load_EvalVar41: Enumeration .
           ((readReply(UnitA'sbs_UnitA'Working'Load_EvalVar41, LoaderUnit, UnitA)) .
           (UnitA'sbs_UnitA'Requesting((UnitA'sbs_UnitA'count) + (1),
           UnitA'sbs_UnitA'Working'Load_EvalVar41, replyValue, c)))))))
302    + (!((UnitA'sbs_UnitA'count) < (2))) -> (delta)
303    + ((UnitA'sbs_UnitA'count) <= (1)) -> ((((queue_unlock_p_thread(UnitA)) | (lock_p_thread)) |
           (readNotification_UnitA(Loader'NI'Aligned))) .
           ((pushNotification_ControlUnit(MachineA'NI'Done)) . (((qEmpty_s_UnitA) . ((((replyValue)
           != (NoReplyValue)) -> ((writeReply(replyValue, UnitA, client)) | (unlockQ_s_UnitA)) <>
```

```
        ((unlockQ_s_UnitA) . (unlock_p_thread))) . (UnitA'sbs_UnitA'Ready(0, NoReplyValue,
        NoClient)))) + (((qNonEmpty_s_UnitA) . ((unlock_p_thread) | (queue_lock_p_thread(UnitA))))
        . (UnitA'sbs_UnitA'Ready(0, replyValue, NoClient))))))
304     + (!((UnitA'sbs_UnitA'count) <= (1))) -> ((((queue_unlock_p_thread(UnitA)) | (lock_p_thread)) |
        (readNotification_UnitA(Loader'NI'Aligned))) .
        ((pushNotification_ControlUnit(MachineA'NI'Done)) . (((qEmpty_s_UnitA) . ((((replyValue)
        != (NoReplyValue)) -> ((writeReply(replyValue, UnitA, client)) | (unlockQ_s_UnitA)) <>
        ((unlockQ_s_UnitA) . (unlock_p_thread))) .
        (UnitA'sbs_UnitA'Working((UnitA'sbs_UnitA'count) - (1), NoReplyValue, client)))) +
        (((qNonEmpty_s_UnitA) . ((unlock_p_thread) | (queue_lock_p_thread(UnitA)))) .
        (UnitA'sbs_UnitA'Working((UnitA'sbs_UnitA'count) - (1), replyValue, client))))))
305     + (true) -> ((((queue_unlock_p_thread(UnitA)) | (lock_p_thread)) |
        (readNotification_UnitA(Loader'NI'Failure))) .
        ((pushNotification_ControlUnit(MachineA'NI'Failed)) . (((qEmpty_s_UnitA) . ((((replyValue)
        != (NoReplyValue)) -> ((writeReply(replyValue, UnitA, client)) | (unlockQ_s_UnitA)) <>
        ((unlockQ_s_UnitA) . (unlock_p_thread))) . (UnitA'sbs_UnitA'Error(UnitA'sbs_UnitA'count,
        NoReplyValue, NoClient)))) + (((qNonEmpty_s_UnitA) . ((unlock_p_thread) |
        (queue_lock_p_thread(UnitA)))) . (UnitA'sbs_UnitA'Error(UnitA'sbs_UnitA'count, replyValue,
        NoClient))))));
306 proc UnitB'sbs_UnitB'Error(UnitB'sbs_UnitB'count: Int, replyValue: Enumeration, client: Component)
        =
307       (true) -> (delta)
308     + (true) -> (sum c: Component . (((invoked''MachineB'AI'Make(c)) |
        (lockQ_s_UnitB(LOCK''UnitB'sbs_UnitB))) .
        ((pushNotification_ControlUnit(MachineB'NI'Failed)) . (((qEmpty_s_UnitB) .
        (((writeReply(VoidReply, UnitB, c)) | (unlockQ_s_UnitB)) .
        (UnitB'sbs_UnitB'Error(UnitB'sbs_UnitB'count, NoReplyValue, client)))) +
        (((qNonEmpty_s_UnitB) . ((unlock_p_thread) | (queue_lock_p_thread(UnitB)))) .
        (UnitB'sbs_UnitB'Error(UnitB'sbs_UnitB'count, VoidReply, client)))))))
309     + (true) -> ((((queue_unlock_p_thread(UnitB)) | (lock_p_thread)) |
        (readNotification_UnitB(Loader'NI'Aligned))) . (((qEmpty_s_UnitB) . ((((replyValue) !=
        (NoReplyValue)) -> ((writeReply(replyValue, UnitB, client)) | (unlockQ_s_UnitB)) <>
        ((unlockQ_s_UnitB) . (unlock_p_thread))) . (UnitB'sbs_UnitB'Error(UnitB'sbs_UnitB'count,
        NoReplyValue, client)))) + (((qNonEmpty_s_UnitB) . ((unlock_p_thread) |
        (queue_lock_p_thread(UnitB)))) . (UnitB'sbs_UnitB'Error(UnitB'sbs_UnitB'count, replyValue,
        client)))))
310     + (true) -> ((((queue_unlock_p_thread(UnitB)) | (lock_p_thread)) |
        (readNotification_UnitB(Loader'NI'Failure))) . (((qEmpty_s_UnitB) . ((((replyValue) !=
        (NoReplyValue)) -> ((writeReply(replyValue, UnitB, client)) | (unlockQ_s_UnitB)) <>
        ((unlockQ_s_UnitB) . (unlock_p_thread))) . (UnitB'sbs_UnitB'Error(UnitB'sbs_UnitB'count,
        NoReplyValue, client)))) + (((qNonEmpty_s_UnitB) . ((unlock_p_thread) |
        (queue_lock_p_thread(UnitB)))) . (UnitB'sbs_UnitB'Error(UnitB'sbs_UnitB'count, replyValue,
        client)))));
311 proc UnitB'sbs_UnitB'Initial(UnitB'sbs_UnitB'count: Int, replyValue: Enumeration, client:
      Component) =
312       (true) -> (sum c: Component . (((invoked''MachineB'AI'Initialize(c)) |
            (lockQ_s_UnitB(LOCK''UnitB'sbs_UnitB))) . (((invoke''Loader'AI'Initialize(UnitB)) .
            (readReply(VoidReply, LoaderUnit, UnitB))) . (((qEmpty_s_UnitB) .
            (((writeReply(VoidReply, UnitB, c)) | (unlockQ_s_UnitB)) .
            (UnitB'sbs_UnitB'Ready(UnitB'sbs_UnitB'count, NoReplyValue, NoClient)))) +
            (((qNonEmpty_s_UnitB) . ((unlock_p_thread) | (queue_lock_p_thread(UnitB)))) .
            (UnitB'sbs_UnitB'Ready(UnitB'sbs_UnitB'count, VoidReply, c)))))))
313     + (true) -> (delta)
314     + (true) -> ((((queue_unlock_p_thread(UnitB)) | (lock_p_thread)) |
        (readNotification_UnitB(Loader'NI'Aligned))) . (Terminate))
315     + (true) -> ((((queue_unlock_p_thread(UnitB)) | (lock_p_thread)) |
        (readNotification_UnitB(Loader'NI'Failure))) . (Terminate));
316 proc UnitB'sbs_UnitB'Ready(UnitB'sbs_UnitB'count: Int, replyValue: Enumeration, client: Component)
      =
317       (true) -> (delta)
318     + (true) -> (sum c: Component . (((invoked''MachineB'AI'Make(c)) |
        (lockQ_s_UnitB(LOCK''UnitB'sbs_UnitB))) . ((invoke''Loader'AI'Load(UnitB)) . (sum
        UnitB'sbs_UnitB'Ready'Load_EvalVar14: Enumeration .
        ((readReply(UnitB'sbs_UnitB'Ready'Load_EvalVar14, LoaderUnit, UnitB)) .
        (UnitB'sbs_UnitB'Requesting(1, UnitB'sbs_UnitB'Ready'Load_EvalVar14, VoidReply, c)))))))
319     + (true) -> ((((queue_unlock_p_thread(UnitB)) | (lock_p_thread)) |
        (readNotification_UnitB(Loader'NI'Aligned))) . (Terminate))
```

```
320     + (true) -> ((((queue_unlock_p_thread(UnitB)) | (lock_p_thread)) |
            (readNotification_UnitB(Loader'NI'Failure))) .
            ((pushNotification_ControlUnit(MachineB'NI'Failed)) . (((qEmpty_s_UnitB) . ((((replyValue)
            != (NoReplyValue)) -> ((writeReply(replyValue, UnitB, client)) | (unlockQ_s_UnitB)) <>
            ((unlockQ_s_UnitB) . (unlock_p_thread))) . (UnitB'sbs_UnitB'Error(UnitB'sbs_UnitB'count,
            NoReplyValue, NoClient)))) + (((qNonEmpty_s_UnitB) . ((unlock_p_thread) |
            (queue_lock_p_thread(UnitB)))) . (UnitB'sbs_UnitB'Error(UnitB'sbs_UnitB'count, replyValue,
            NoClient))))));
321 proc UnitB'sbs_UnitB'Requesting(UnitB'sbs_UnitB'count: Int,
        UnitB'sbs_UnitB'Requesting'RequestingAIReturnEnumEP: Enumeration, replyValue: Enumeration,
        client: Component) =
322         ((UnitB'sbs_UnitB'Requesting'RequestingAIReturnEnumEP) == (Loader'AI'AIRETURNENUM'OKREPLY))
                -> ((valuedTrigger) . (((qEmpty_s_UnitB) . ((((replyValue) != (NoReplyValue)) ->
                ((writeReply(replyValue, UnitB, client)) | (unlockQ_s_UnitB)) <> ((unlockQ_s_UnitB) .
                (unlock_p_thread))) . (UnitB'sbs_UnitB'Working(UnitB'sbs_UnitB'count, NoReplyValue,
                NoClient)))) + (((qNonEmpty_s_UnitB) . ((unlock_p_thread) |
                (queue_lock_p_thread(UnitB)))) . (UnitB'sbs_UnitB'Working(UnitB'sbs_UnitB'count,
                replyValue, NoClient)))))
323     + ((UnitB'sbs_UnitB'Requesting'RequestingAIReturnEnumEP) == (Loader'AI'AIRETURNENUM'FAILREPLY))
            -> ((valuedTrigger) . ((pushNotification_ControlUnit(MachineB'NI'Failed)) .
            (((qEmpty_s_UnitB) . ((((replyValue) != (NoReplyValue)) -> ((writeReply(replyValue, UnitB,
            client)) | (unlockQ_s_UnitB)) <> ((unlockQ_s_UnitB) . (unlock_p_thread))) .
            (UnitB'sbs_UnitB'Error(UnitB'sbs_UnitB'count, NoReplyValue, NoClient)))) +
            (((qNonEmpty_s_UnitB) . ((unlock_p_thread) | (queue_lock_p_thread(UnitB)))) .
            (UnitB'sbs_UnitB'Error(UnitB'sbs_UnitB'count, replyValue, NoClient))))));
324 proc UnitB'sbs_UnitB'Working(UnitB'sbs_UnitB'count: Int, replyValue: Enumeration, client:
        Component) =
325         (true) -> (delta)
326     + ((UnitB'sbs_UnitB'count) < (3)) -> (sum c: Component . (((invoked''MachineB'AI'Make(c)) |
            (lockQ_s_UnitB(LOCK''UnitB'sbs_UnitB))) . ((invoke''Loader'AI'Load(UnitB)) . (sum
            UnitB'sbs_UnitB'Working'Load_EvalVar41: Enumeration .
            ((readReply(UnitB'sbs_UnitB'Working'Load_EvalVar41, LoaderUnit, UnitB)) .
            (UnitB'sbs_UnitB'Requesting((UnitB'sbs_UnitB'count) + (1),
            UnitB'sbs_UnitB'Working'Load_EvalVar41, VoidReply, c)))))))
327     + (!((UnitB'sbs_UnitB'count) < (3))) -> (delta)
328     + ((UnitB'sbs_UnitB'count) <= (1)) -> ((((queue_unlock_p_thread(UnitB)) | (lock_p_thread)) |
            (readNotification_UnitB(Loader'NI'Aligned))) .
            ((pushNotification_ControlUnit(MachineB'NI'Done)) . (((qEmpty_s_UnitB) . ((((replyValue)
            != (NoReplyValue)) -> ((writeReply(replyValue, UnitB, client)) | (unlockQ_s_UnitB)) <>
            ((unlockQ_s_UnitB) . (unlock_p_thread))) . (UnitB'sbs_UnitB'Ready((UnitB'sbs_UnitB'count)
            - (1), NoReplyValue, NoClient)))) + (((qNonEmpty_s_UnitB) . ((unlock_p_thread) |
            (queue_lock_p_thread(UnitB)))) . (UnitB'sbs_UnitB'Ready((UnitB'sbs_UnitB'count) - (1),
            replyValue, NoClient))))))
329     + (!((UnitB'sbs_UnitB'count) <= (1))) -> ((((queue_unlock_p_thread(UnitB)) | (lock_p_thread)) |
            (readNotification_UnitB(Loader'NI'Aligned))) .
            ((pushNotification_ControlUnit(MachineB'NI'Done)) . (((qEmpty_s_UnitB) . ((((replyValue)
            != (NoReplyValue)) -> ((writeReply(replyValue, UnitB, client)) | (unlockQ_s_UnitB)) <>
            ((unlockQ_s_UnitB) . (unlock_p_thread))) .
            (UnitB'sbs_UnitB'Working((UnitB'sbs_UnitB'count) - (1), NoReplyValue, client)))) +
            (((qNonEmpty_s_UnitB) . ((unlock_p_thread) | (queue_lock_p_thread(UnitB)))) .
            (UnitB'sbs_UnitB'Working((UnitB'sbs_UnitB'count) - (1), replyValue, client))))))
330     + (true) -> ((((queue_unlock_p_thread(UnitB)) | (lock_p_thread)) |
            (readNotification_UnitB(Loader'NI'Failure))) .
            ((pushNotification_ControlUnit(MachineB'NI'Failed)) . (((qEmpty_s_UnitB) . ((((replyValue)
            != (NoReplyValue)) -> ((writeReply(replyValue, UnitB, client)) | (unlockQ_s_UnitB)) <>
            ((unlockQ_s_UnitB) . (unlock_p_thread))) . (UnitB'sbs_UnitB'Error(UnitB'sbs_UnitB'count,
            NoReplyValue, NoClient)))) + (((qNonEmpty_s_UnitB) . ((unlock_p_thread) |
            (queue_lock_p_thread(UnitB)))) . (UnitB'sbs_UnitB'Error(UnitB'sbs_UnitB'count, replyValue,
            NoClient))))));
331 proc queue_Thread = sum c: Component . ((((queue_lock_t_thread(c)) . (queue_unlock_t_thread(c))) .
        (queue_Thread)) + ((queue_unlock_t_thread(c)) . (queue_Thread)));
332 init allow({unlock_thread, unlock_thread|queue_lock_thread, initialize, invalidate, terminate,
        valuedTrigger, optionalInternalTrigger|lockQ_ControlUnit|lock_thread|emptyQ,
        inevitableInternalTrigger|lockQ_ControlUnit|lock_thread|emptyQ,
        optionalInternalTrigger|lockQ_LoaderUnit|lock_thread|emptyQ,
        inevitableInternalTrigger|lockQ_LoaderUnit|lock_thread|emptyQ,
        optionalInternalTrigger|lockQ_UnitA|lock_thread|emptyQ,
```

```
    inevitableInternalTrigger|lockQ_UnitA|lock_thread|emptyQ,
    optionalInternalTrigger|lockQ_UnitB|lock_thread|emptyQ,
    inevitableInternalTrigger|lockQ_UnitB|lock_thread|emptyQ,
    Controller'AI'Initialize|lockQ_ControlUnit|lock_thread|emptyQ,
    Controller'AI'MakeA|lockQ_ControlUnit|lock_thread|emptyQ,
    Controller'AI'MakeB|lockQ_ControlUnit|lock_thread|emptyQ,
    Loader'AI'Initialize|lockQ_LoaderUnit, Loader'AI'Load|lockQ_LoaderUnit,
    MachineA'AI'Initialize|lockQ_UnitA, MachineA'AI'Make|lockQ_UnitA,
    MachineB'AI'Initialize|lockQ_UnitB, MachineB'AI'Make|lockQ_UnitB,
    outwardReply|unlockQ_ControlUnit, outwardReply|unlockQ_LoaderUnit, outwardReply|unlockQ_UnitA,
    outwardReply|unlockQ_UnitB, outwardNotification, Loader'AI'Initialize, Loader'AI'Load,
    MachineA'AI'Initialize, MachineA'AI'Make, MachineB'AI'Initialize, MachineB'AI'Make,
    RobotArm'AI'Initialize, RobotArm'AI'Move, sendReply, sendReply|unlockQ_ControlUnit,
    sendReply|unlockQ_LoaderUnit, sendReply|unlockQ_UnitA, sendReply|unlockQ_UnitB,
    triggerNotification_ControlUnit|lock_thread|queue_unlock_thread,
    triggerNotification_LoaderUnit|lock_thread|queue_unlock_thread,
    triggerNotification_UnitA|lock_thread|queue_unlock_thread,
    triggerNotification_UnitB|lock_thread|queue_unlock_thread, raiseNotification_ControlUnit,
    raiseNotification_LoaderUnit, raiseNotification_UnitA, raiseNotification_UnitB,
    unlockQ_ControlUnit, unlockQ_LoaderUnit, unlockQ_UnitA, unlockQ_UnitB, queueSizeViolated,
    qEmpty_ControlUnit, qNonEmpty_ControlUnit, qEmpty_LoaderUnit, qNonEmpty_LoaderUnit,
    qEmpty_UnitA, qNonEmpty_UnitA, qEmpty_UnitB, qNonEmpty_UnitB},
333 comm({emptyQ_r_ControlUnit|emptyQ_r_LoaderUnit|emptyQ_r_UnitA|emptyQ_r_UnitB|emptyQ_s -> emptyQ,
    lock_t_thread|lock_p_thread -> lock_thread, unlock_t_thread|unlock_p_thread -> unlock_thread,
    queue_lock_t_thread|queue_lock_p_thread -> queue_lock_thread,
    queue_unlock_t_thread|queue_unlock_p_thread -> queue_unlock_thread, readReply|writeReply ->
    sendReply, lockQ_s_ControlUnit|lockQ_r_ControlUnit -> lockQ_ControlUnit,
    unlockQ_s_ControlUnit|unlockQ_r_ControlUnit -> unlockQ_ControlUnit,
    qEmpty_s_ControlUnit|qEmpty_r_ControlUnit -> qEmpty_ControlUnit,
    qNonEmpty_s_ControlUnit|qNonEmpty_r_ControlUnit -> qNonEmpty_ControlUnit,
    sendNotification_ControlUnit|readNotification_ControlUnit -> triggerNotification_ControlUnit,
    pushNotification_ControlUnit|receiveNotification_ControlUnit -> raiseNotification_ControlUnit,
    lockQ_s_LoaderUnit|lockQ_r_LoaderUnit -> lockQ_LoaderUnit,
    unlockQ_s_LoaderUnit|unlockQ_r_LoaderUnit -> unlockQ_LoaderUnit,
    qEmpty_s_LoaderUnit|qEmpty_r_LoaderUnit -> qEmpty_LoaderUnit,
    qNonEmpty_s_LoaderUnit|qNonEmpty_r_LoaderUnit -> qNonEmpty_LoaderUnit,
    sendNotification_LoaderUnit|readNotification_LoaderUnit -> triggerNotification_LoaderUnit,
    pushNotification_LoaderUnit|receiveNotification_LoaderUnit -> raiseNotification_LoaderUnit,
    lockQ_s_UnitA|lockQ_r_UnitA -> lockQ_UnitA, unlockQ_s_UnitA|unlockQ_r_UnitA -> unlockQ_UnitA,
    qEmpty_s_UnitA|qEmpty_r_UnitA -> qEmpty_UnitA, qNonEmpty_s_UnitA|qNonEmpty_r_UnitA ->
    qNonEmpty_UnitA, sendNotification_UnitA|readNotification_UnitA -> triggerNotification_UnitA,
    pushNotification_UnitA|receiveNotification_UnitA -> raiseNotification_UnitA,
    lockQ_s_UnitB|lockQ_r_UnitB -> lockQ_UnitB, unlockQ_s_UnitB|unlockQ_r_UnitB -> unlockQ_UnitB,
    qEmpty_s_UnitB|qEmpty_r_UnitB -> qEmpty_UnitB, qNonEmpty_s_UnitB|qNonEmpty_r_UnitB ->
    qNonEmpty_UnitB, sendNotification_UnitB|readNotification_UnitB -> triggerNotification_UnitB,
    pushNotification_UnitB|receiveNotification_UnitB -> raiseNotification_UnitB,
    invoke''MachineB'AI'Make|invoked''MachineB'AI'Make -> MachineB'AI'Make,
    invoke''Loader'AI'Load|invoked''Loader'AI'Load -> Loader'AI'Load,
    invoke''Loader'AI'Initialize|invoked''Loader'AI'Initialize -> Loader'AI'Initialize,
    invoke''MachineA'AI'Initialize|invoked''MachineA'AI'Initialize -> MachineA'AI'Initialize,
    invoke''Controller'AI'MakeA|invoked''Controller'AI'MakeA -> Controller'AI'MakeA,
    invoke''MachineA'AI'Make|invoked''MachineA'AI'Make -> MachineA'AI'Make,
    invoke''Controller'AI'MakeB|invoked''Controller'AI'MakeB -> Controller'AI'MakeB,
    invoke''MachineB'AI'Initialize|invoked''MachineB'AI'Initialize -> MachineB'AI'Initialize,
    invoke''Controller'AI'Initialize|invoked''Controller'AI'Initialize ->
    Controller'AI'Initialize, invoke''RobotArm'AI'Initialize|invoked''RobotArm'AI'Initialize ->
    RobotArm'AI'Initialize, invoke''RobotArm'AI'Move|invoked''RobotArm'AI'Move ->
    RobotArm'AI'Move},
334 ((((((((((Thread) || (ControlUnit'sbs_ControlUnit'Initial(0, 0, NoReplyValue, NoClient))) ||
    (Queue_UnitB([], NONE))) || (queue_Thread)) || (Queue_ControlUnit([], NONE))) ||
    (LoaderUnit'sbs_LoaderUnit'Initial(0, NoReplyValue, NoClient))) || (UnitA'sbs_UnitA'Initial(0,
    NoReplyValue, NoClient))) || (Queue_LoaderUnit([], NONE))) || (Queue_UnitA([], NONE))) ||
    (RobotArm'RobotArm_sync'RobotArm'Initial(NoClient))) || (UnitB'sbs_UnitB'Initial(0,
    NoReplyValue, NoClient))));
```