



Translation of Model-based Behavioral Specifications into TLA+

Diplomarbeit

Submitted by:

Shan Huang

Informatik-Ingenieurwesen

Matriculation Number: 15277

Supervised by:

Prof. Dr. Ralf Möller (STS)
Prof. Dr. Siegfried M. Rump (TI3)
M.Sc. Miguel Garcia (STS)

Hamburg, Germany
15th April 2007

Declaration:

I declare that:

this work has been prepared by myself, all literally or content-related quotations from other sources are clearly pointed out, and no other sources or aids than the ones that are declared are used.

Hamburg, 15.04.2007

Shan Huang

Acknowledgment:

My deepest thanks to Prof. Dr. Ralf Möller and Prof. Dr. Siegfried M.Rump for giving me the opportunity to work on this thesis project under their supervision.

I would also like to express my appreciation to M. Sc. Miguel Garcia, who was very helpful in providing advice and direction on this thesis.

This project could not have been as fruitful as it was, without the support, interest and inputs from individuals and their willingness to share their experience and knowledge, and the time given to discuss related topics.

Abstract:

This paper presents an implementation approach to convert UML statechart diagram, which represents the model-based behaviour specifications, into the TLA+ language. An automatic transformation procedures is introduced which handles all of the components in the statechart diagram. The produced specifications will be checked afterwards with the Model-Checker TLC. Two examples, microwave oven system and railway gateway system will be introduced. They correspond respectively to the flat and hierarchical statechart diagram.

Table of Contents

Chapter 1. Introduction.....	7
1.1. Why do We need this Transformation?.....	7
1.2. Basic Transformation Idea.....	8
1.3. Structure of this Thesis.....	8
Chapter 2. Statecharts Metamodel.....	10
2.1. Statecharts Introduction.....	10
2.2. Statecharts Components.....	10
Chapter 3. Generate Statecharts Instance.....	16
3.1. Eclipse Modelling Framework Introduction.....	16
3.2. Emfatic Introduction.....	17
3.3. How to define the Instance of Metamodel.....	20
Chapter 4. TLA+ Specification.....	25
4.1. Microwave oven.....	25
4.2. Trailway Gateway System.....	28
Chapter 5. Transformation Algorithms.....	35
5.1. Visitor Pattern.....	35
5.2. Transformation Algorithms.....	38
5.2.1. <i>Handling State</i>	38
5.2.2. <i>Handling Attribute</i>	39
5.2.3. <i>Handling Event</i>	40
5.2.4. <i>Handling Action</i>	41
5.2.4.1. Executable UML Introduction.....	41
5.2.4.2. Handling Reference Action.....	41
5.2.5. <i>Handling Guard</i>	42
5.2.6. <i>Other Statecharts Components and TLA+ Statements</i>	43
5.3. Plugin for the Generation of Target File.....	44
Chapter 6. TLC model checker.....	46
6.1. Why model check.....	46
6.2. How TLC works?.....	47
Chapter 7. Future Work.....	52
Chapter 8. Conclusion.....	53
Appendix 1.....	55
Appendix 2.....	60
Appendix 3.....	63
Appendix 4.....	89
Reference	92

List of Figures

Fig 2.1 Class Diagram of Statecharts Metamodel.....	11
Fig 2.2 Detailed Metamodel of Action Component.....	14
Fig 2.3 Detailed Metamodel of Guard Component.....	15
Fig 3.1 Working Theory of EMF.....	16
Fig 3.2 EMF Code Generation Structure.....	20
Fig 3.3 Statecharts of TrailObserver.....	21
Fig 3.4 Instance Structure of TrailObserver.....	22
Fig 3.5 Properties View of State Component.....	23
Fig 3.6 Properties View of Guard Component.....	23
Fig 3.7 Properties View of Reference Action.....	24
Fig 4.1 Statecharts of Microwave Oven System.....	25
Fig 4.2 Trailway Gateway System.....	29
Fig 4.3 Statecharts of Trail-Light.....	30
Fig 4.4 Statecharts of Trail-Gate.....	32
Fig 4.5 Instance Structure of Complete Trailway Gateway System.....	33
Fig 4.6 Statecharts of Complete Trailway Gateway System.....	34
Fig 5.1 TLA+ Specification Generation Plugin.....	35
Fig 5.2 Working Theory of Visitor Pattern	36
Fig 5.3 State Cooking with Action.....	41
Fig 6.1 Structure of TLC Model Checker.....	47

Chapter 1. Introduction

1.1. Why do We need this Transformation?

Model-based behavioural specifications offer intuitive, accessible overviews of complex systems. Each system model can contain different states. Each state represents different context of the behaviour. Statechart diagram, originally designed for modeling reactive systems, is used by most of the current object-oriented methodologies to describe the dynamic behaviors of a system. It consists of discrete components such as states, transitions, events, actions and so on.

The user can specify a system by describing its allowed behaviours. It will lighten the future work if the user can understand a system before building it and they can write a specification of a system before implementing it, which means a written description of what a system is supposed to do. For this purpose, Leslie Lamport has invented TLA, the Temporal Logic of Actions, in the late 1980s. It is logic for specifying and reasoning about concurrent systems. TLA makes it practical to describe a system by a single formula. It provides a mathematical foundation for describing systems. Most of a TLA specification consists of ordinary, nontemporal mathematics. Temporal logic plays a significant role in describing those properties that it's good at describing. TLA also provides a nice way to formalize the style of reasoning about systems that has proved to be most effective in practice — a style known as assertional reasoning. [7]

To write the specifications, Leslie Lamport has developed a corresponding specification language. The language is called TLA+. Temporal Logic of Actions Plus (TLA+) is a formal notation for specifying the valid states of a system. It is a specification language for concurrent and reactive systems that combines the temporal logic TLA with full first-order logic and ZF set theory. In the last years, TLA+ has been refined for specifying a wide class of systems — from program interfaces (APIs) to distributed systems. It can be used to write a precise, formal description of almost any sorts of discrete system. It's especially well suited to describing asynchronous systems. [1]

The combined use of UML and TLA+ for software specification is a recent research topic. So far, the user accomplishes normally the specification of system manual, but after having UML statecharts and TLA+ language in our hands, we might need a method, with it we can automatically translate a statechart diagram to TLA+ specification. How to create a transformation between UML statechart diagram and TLA+ specifications and how to check the “correctness” of the generated TLA+ file, which is the aim of this project.

1.2. Basic Transformation Idea

The transformation is aimed to build a bridge to establish the connection between UML statecharts and TLA+ via a transformation algorithm.

1. To complete this purpose, I must first find out an appropriate metamodel of the source statechart diagram. MDA infrastructure Java codes can be generated with EMF automatically. After having the whole metamodel implementation in hand, I can define an arbitrary statecharts, i.e. an arbitrary instance of the statechart metamodel.
2. Afterwards, I must implement the transformation algorithm to determine how to generate the target specialization according to the defined instances. This algorithm must map both source metamodel and target language. Since the TLA+ statements should be generated according to the types of components defined in the metamodel and every component has its own implementation, thus visitor pattern is here a good choice to accomplish this purpose.
3. Once source metamodel, source instance and transformation algorithm are determined in this way, a TLA+ code generator could be created. The “top” state of the source statecharts must be “passed” in the generator, the generator will traverse through all of the components in the object structure with visitor and generate the corresponding specification.

1.3. Structure of this Thesis

Chapter 2

gives the reader an introduction of statecharts metamodel. All components needed for the transformation will be declared in detail.

Chapter 3

gives a basic introduction of Eclipse Modelling Framework (EMF). The keystone in this chapter is how to define the need instance of metamodel. The instances of two system models are declared as example, microwave oven system and railway gateway system, which correspond respectively the flat and hierarchical statecharts.

Chapter 4

the above defined system models have been specified with TLA+ in this chapter. It presents a detailed description of the specification.

Chapter 5

a brief description of the TLA+ generating process is declared in this chapter. The

basic working theory of used visitor pattern is shown firstly. Various TLA+ statements can be generated according to the statecharts components through this design pattern. The main transformation algorithm is presented afterwards. It is interpreted according to the types of statecharts components. A Java plugin implemented for saving and generating .tla file of the produced TLA+ specification is declared lastly.

Chapter 6

this chapter introduces briefly the working theory of the model checker TLC. The generated TLA+ specialization is inputted in the model checker as source file and the checking result is analyzed as well.

Chapter 7

gives introduction of possible developments and improvements in this researching area.

Chapter 8

Conclusions gained from this work.

Chapter 2. Statecharts Metamodel

2.1. Statecharts Introduction

A statechart diagram depicts all possible dynamic behaviours which describe an entity to respond according to the affairs and showed how that entity does a reaction according to the current state in different time. Usually we create a statechart diagram for the purpose: To describe the complex behaviours among the states, submachines, or statemachines.

UML statechart diagram is an important aid as part of the OMG UML specification in order to model the dynamic behaviours of system or subsystem. It is a visual representation of the UML specification containing various statemachines. Preferred use-area of UML statechart diagram is the modelling of discrete condition transitions in reactive system such as embedded system and process-automation-system. However statechart diagram is also suitable to model the behaviours of static elements within a static model, for example UML class or UML use-case.

UML statechart diagram makes both semi-formal and formal specification of the system performance possible, which in connection with an implementation model, enable the automatic code generation of real system. However the statecharts specification leaves also many questions of the notation, syntax and semantics of statemachines open so that the corresponding metamodel must be extended for a complete formal description of real system.

2.2. Statecharts Components

Since the UML statechart diagram is the source model in our transformation, to let the transformation works, a metamodel of it is necessary. The statecharts metamodel (i.e. the definition about the statechart diagram) can be represented by the UML class diagram. Figure 2.1 shows the basic components of statechart diagram such as statemachine, transition, event, attribute, action and a variety of different states. It defines the abstract syntax of UML statemachine. If a component describes the behaviour with another model component, it must has an appropriate association with it. In the following sections, I will shortly introduce the main components in this diagram.

call events and dispatch it at another time when needed. The detail of the implementation will be declared in the following chapter.

State is a period of time during which an object is waiting for an event to occur. State can be divided into two subclasses, simple state and composite state. Simple state contains following two various subclasses.

- **initial state** means the default state assumed upon entering a state context. The initial state looks like a transition but has a ball at the origination end in the diagram.
- **final state** is shown by a circumscribed “T” (alternative notation is a dot within a circle). This means that the object no longer accepts events and will be destroyed. It represents the completion of activity in the enclosing state and it triggers a transition on the enclosing state labelled by the implicit activity completion event (usually displayed as an unlabelled transition), if such a transition is defined.

Composite State means the states which contains the other normal states. A composite state is divided into two or more concurrent substates which called regions, or into both exclusive disjoint substates. A given state may be refined only in one of these two ways. Naturally each substate of a composite state can also be a composite state of the either types. Each region of a state has perhaps its own initial pseudostate and final state. A transition to the including state represents a transition to the initial pseudostate. A transition to a final state represents the completion of the activity in the including region.

State can be associated with its entry and exit action sequence, which denotes the actions that have to be performed as soon as the state is entered or exited. doEntry/doExit statement in TLA+ denotes the local internal activities that must be executed as long as the state is active.

Event is a one-way (asynchronous) communication from one component to another. It has following two characters. First, it is atomic (non-interruptible) and second it may cause a transition between states. As soon as a state becomes active, a doEntry event generated, which is visible only for internal transitions of the state. If a state becomes inactive, then doExit event generated, which is visible only for internal transitions likewise. If these do-Activity procedures of a state are terminated, then the event completion is generated. This event is visible within the state as well as for all transitions, which depart from this state.

Transition defines the relationship between two states. It indicates that the first state enters the second state and carries out specific actions if a specified event occurs or a specified conditions is filled. Transition is a response to an external event received by

a component. It may invoke an action and cause the component to change. It may also send an event to an external component. The trigger of a transition is the appearance of the corresponding event which is labelled on this transition. The event can have parameters which are accessible through the corresponding actions specified on the transition or the exit and entry actions associated with the source and target states.

A transition occurs, if the following conditions filled:

- The source state is active.
- The trigger event is released.
- The guard, which is defined by the state, is fulfilled.

Action Sequence is the action expressions, which may be composed of a number of distinct actions that explicitly generate events, such as sending signals or invoking operations. Action sequence can be associated with state and transition between states and it has the following various types:

- **Entry Action Sequence**
This label identifies an action sequence specified by the corresponding action expression that is performed by entering the state.
- **Exit Action Sequence**
This label identifies an action sequence specified by the corresponding action expression that is performed by exiting a state.
- **Transition Action Sequence**
This label identifies an action sequence, specified by the corresponding action expression that is performed between the states.

Figure 2.2 below shows the detailed structure of the action components in the statecharts metamodel.

Action is small atomic behaviour executed at specified points in a statemachine. They are assumed to take an insignificant amount of time to execute and are non-interruptible. Action is invoked by a transition. It is executed if and when the transition fires. Action will be associated with the corresponding action sequence. Three various actions are defined in the statecharts metamodel according to the type of object which the action deals with. [5]

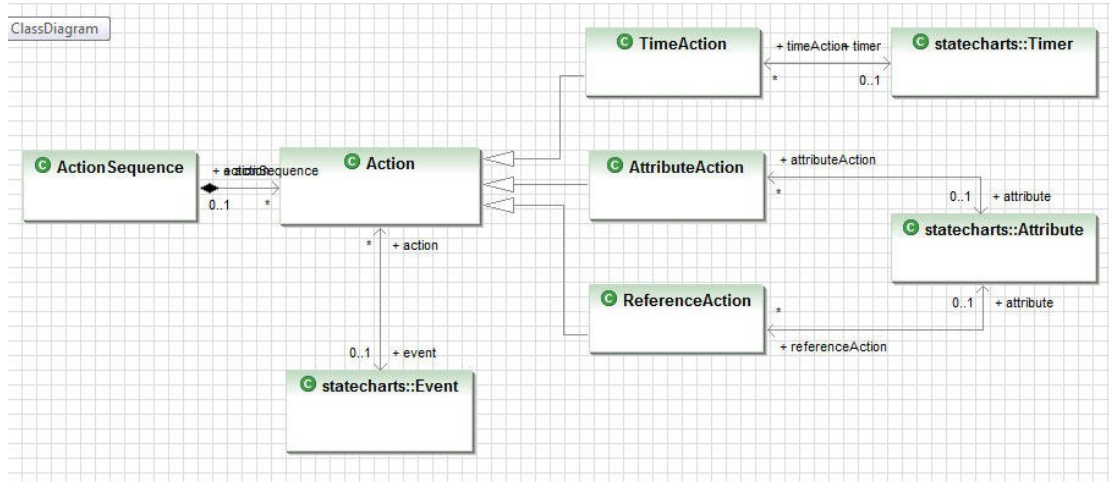


Fig 2.2 Detailed Metamodel of Action Component

- **Time Action**

In general, a time action defines the behaviors of system timer.

- **Attribute Action**

Attribute action is the activity defined for the changing of attribute. When executed, the action takes some initial values of attributes, performs processing and produces the set of output values.

- **Reference Action**

A reference action defines the behaviours occurred between the source state and the target state. It will be associated with the corresponding entry or exit action sequence defined with a state. If such a reference action occurs, any nested attributes or timer actions are forcibly performed, then the transition occurs and the new state is established.

Guard is a boolean expression written in terms of the object which the guard component guards. It returns a TRUE or FALSE value that controls whether or not a transition has taken place. Figure 2.3 below shows the detailed structure of the guard component. According to the type of the guarded object, guard component can be divided in the following four types:

- **Constant Guard**

System model needs sometimes constants as parameters to specify the initial values of variables or describe system constraints. Constant guard is used to determine whether a transition satisfies a certain constant limit before the transition sends out .

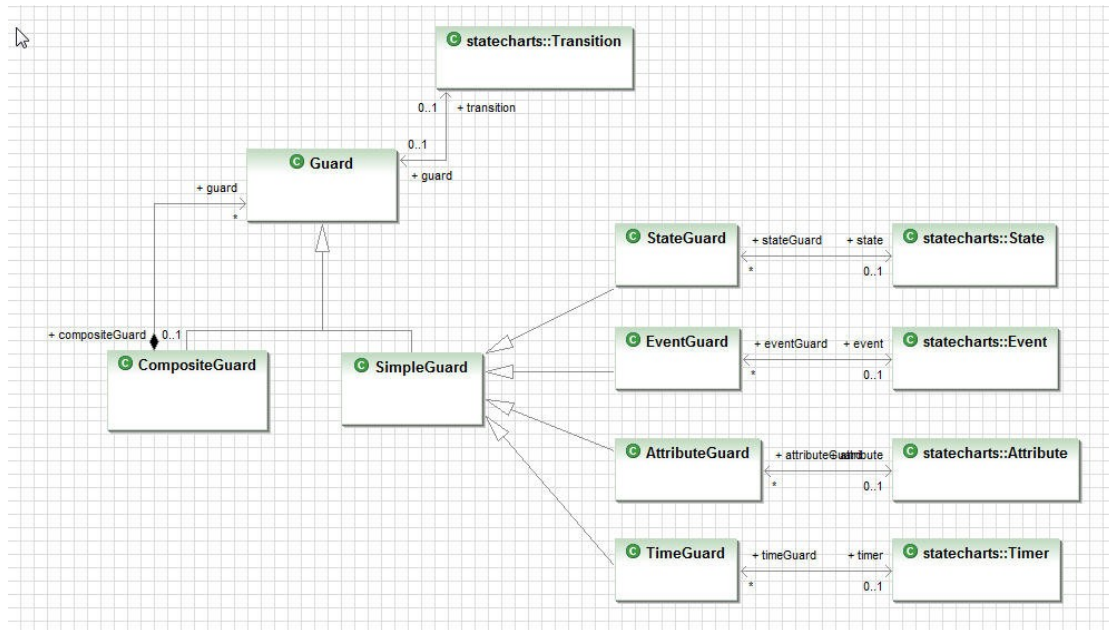


Fig 2.3 Detailed Metamodel of Guard Component

- **Attribute Guard**

Some systems need to judge before the transition sends out whether a certain system attribute is satisfied or not. Thus we need attribute guard to estimate if the value of attribute is true. Attribute guard is associated with a particular attribute of the system. It will be true when the value of attribute is true.

- **Event Guard**

Some events can be used as judgment condition for transition sending. The system needs to judge before the transition sends out whether a certain event has already been triggered or not.

- **State Guard**

Some system states can also be used as judgment condition. The system needs to judge whether the system has already been placed in a certain state before the transition sends out. State guard is associated with a particular state, i.e. if and only if the actual state equals the needed state, then the transition will be sent out.

Chapter 3. Generate Statecharts Instance

3.1. Eclipse Modelling Framework Introduction

According to the above generated statecharts metamodel, we can complete a Java infrastructure to implement all components defined in it. Afterwards we can instance the metamodel. With Eclipse Modelling Framework (EMF), all needed Java implementations can be generated automatically. Figure 3.1 below shows the basic working theory of EMF and followed with a brief introduction:

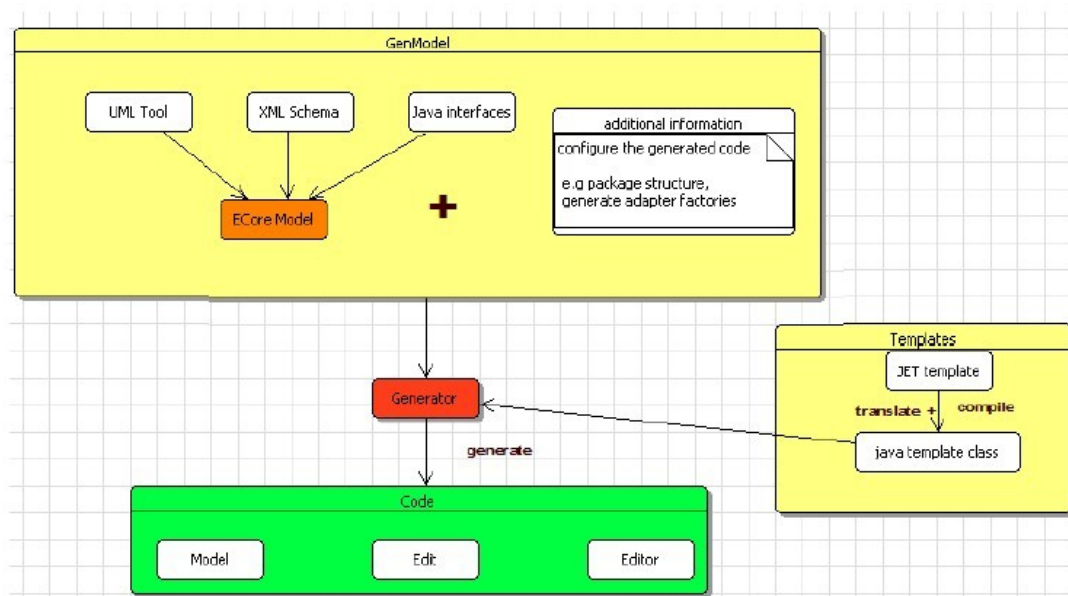


Fig 3.1 Working Theory of EMF [4]

EMF follows a model driven approach targeting software development. Transformations taken during a classical MDA (Model Driven Architecture) process are addressed by EMF. EMF consists of three main parts:

- EMF.Core comprises model description complying ECore, model persistence, change notification and reflective API.
- EMF.Edit supports the creation of EMF model editors, by including content and label provider classes and command framework.
- EMF.Codegen provides the infrastructure for generating all the needed artifacts for an EMF model editor. [4]

There are two models defined in EMF, Ecore model and Genmodel. Ecore model

represents the Platform Independent Model (PIM) and Genmodel represents the Platform Specific Model (PSM). The transformation with EMF is normally composed of two steps:

- Ecore model (PIM) must be first transformed to Genmodel (PSM). Genmodel includes additional informations about the structure and organization of future code to be generated.
- Second, the in last step generated Genmodel (PSM) could be automatically transformed to the actual code with the transformation chain support that EMF provides.

As shown in figure 3.1, the input model of EMF generator should be the Genmodel. The Genmodel is transformed from the PIM Ecore model. The user can either manual or use other language to implement Ecore model. For example, we can use UML tool, XML schema or directly Java interfaces to generate Ecore model.

3.2. Emfatic Introduction

In this project, the definition for statechart metamodel will be accomplished using Emfatic syntax. Emfatic is a language designed to represent EMF Ecore model in a textual form. The examples below introduce briefly the syntax of Emfatic and the mapping between Emfatic declarations and the corresponding Ecore constructs.

The first component that must be declared in an Emfatic file is package declaration. This necessary component is called main-package-declaration and it contains all other components of the generated Ecore file. The package declaration corresponds to the EPackage in a Ecore file.

```
package statecharts;
```

Since I have defined three packages which correspond respectively to the basic components of statechart diagram, the action components and the guard components. Thus I must import the statements defined in external Ecore models as reference. The example below demonstrates the basic syntax of import statements.

```
import "platform:/resource/statecharts/model/statechartsAction.ecore";  
import "platform:/resource/statecharts/model/statechartsGuard.ecore";
```

The following example containing class declarations demonstrates how to use keywords **ref** and **val** to define the association and aggregation respectively among statecharts components.

```

class State extends NamedElement {
    !ordered ref CompositeState #substates parent;
    !ordered ref Transition[0..*] #target incoming;
    !ordered ref StateMachine[1] statemachine;
    !ordered ref statechartsGuard.StateGuard[0..*] guards;
    !ordered val Transition[0..*] #source outgoing;
    !ordered val statechartsAction.ActionSequence[*] actionsequences;
}

```

Emfatic use keyword **attr** and **op** to define the class attributes and operations which correspond respectively the EAttribute and EOperation of Ecore file.

```

class Timer extends PredefinedClass {
    attr boolean running;
    attr int timer;
    attr int minTimerValue;
    attr int maxTimerValue;
    op void clearTime();
    op void addTime();
    !ordered val statechartsAction.TimerAction[*] timeractions;
    !ordered val statechartsGuard.TimeGuard[*] timerguards;
}

```

I give here partial Emfatic code of important components in statechart metamodel. A full version of the Emfatic model is given in Appendix 1.

```

class StateMachine extends NamedElement{
    !ordered val AttributeDef[0..*] attributes;
    !ordered val EventDef[0..*] events;
    !ordered val State[*] states;
    !ordered val PredefinedClass[0..*] predefinedclasses;
    !ordered val Submachine[*] submachines;
    !ordered val statechartsAction.Action[*] actions;
    !ordered val statechartsAction.ActionSequence[*] actionsequences;
}

```

```

class State extends NamedElement {
    !ordered ref CompositeState #substates parent;
    !ordered ref Transition[0..*] #target incoming;
    !ordered ref StateMachine[1] statemachine;
    !ordered ref statechartsGuard.StateGuard[0..*] guards;
    !ordered val Transition[0..*] #source outgoing;
    !ordered val statechartsAction.ActionSequence[*] actionsequences;
}

```

```

class Transition {
    attr String label;
    !ordered ref EventDef[0..*] triggers;
    !ordered ref State #outgoing source;
    !ordered ref State #incoming target;
    !ordered val statechartsGuard.Guard[0..1] guard;
    !ordered val statechartsAction.ActionSequence[0..1] actionsequence;
}

class EventDef extends NamedElement {
    !ordered ref Transition[0..*] transitions;
    !ordered ref StateMachine statemachine;
    !ordered ref statechartsAction.Action[0..*] actions;
    !ordered ref statechartsGuard.EventGuard[0..*] guards;
}

class AttributeDef extends NamedElement {
    attr AttributeType attrType;
    attr String attrValue;
    !ordered ref StateMachine statemachine;
    !ordered ref statechartsGuard.AttributeGuard[0..*] guards;
    !ordered val statechartsAction.AttrAction[0..*] attractions;
    !ordered val statechartsAction.RefAction[0..*] refactions;
}

class ActionSequence extends statecharts.NamedElement {
    attr ActionSequenceType type;
    !ordered ref statecharts.State[0..1] State;
    !ordered ref statecharts.Transition[0..*] transitions;
    !ordered ref statecharts.StateMachine statemachine;
    !ordered val Action[*] actions;
}

class Action extends statecharts.NamedElement {
    attr ActionType type;
    !ordered ref ActionSequence[*] actionsequences;
    !ordered ref statecharts.StateMachine statemachine;
    !ordered ref statecharts.EventDef event;
}

class Guard extends statecharts.NamedElement {
    attr String gdExpression;
    attr GuardLogicType logicRelation;
}

```

```

!ordered ref statecharts.Transition[0..1] transition;
!ordered ref CompositeGuard #subguards parentguard;
}

```

EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. According to the written Emfatic model, we can convert the Emfatic model to Ecore model with the translator provided by Emfatic plugin and create the Genmodel based on the Ecore model afterwards. With the code generator from EMF, we can generate the model, edit, editor and test plugins in a single step.

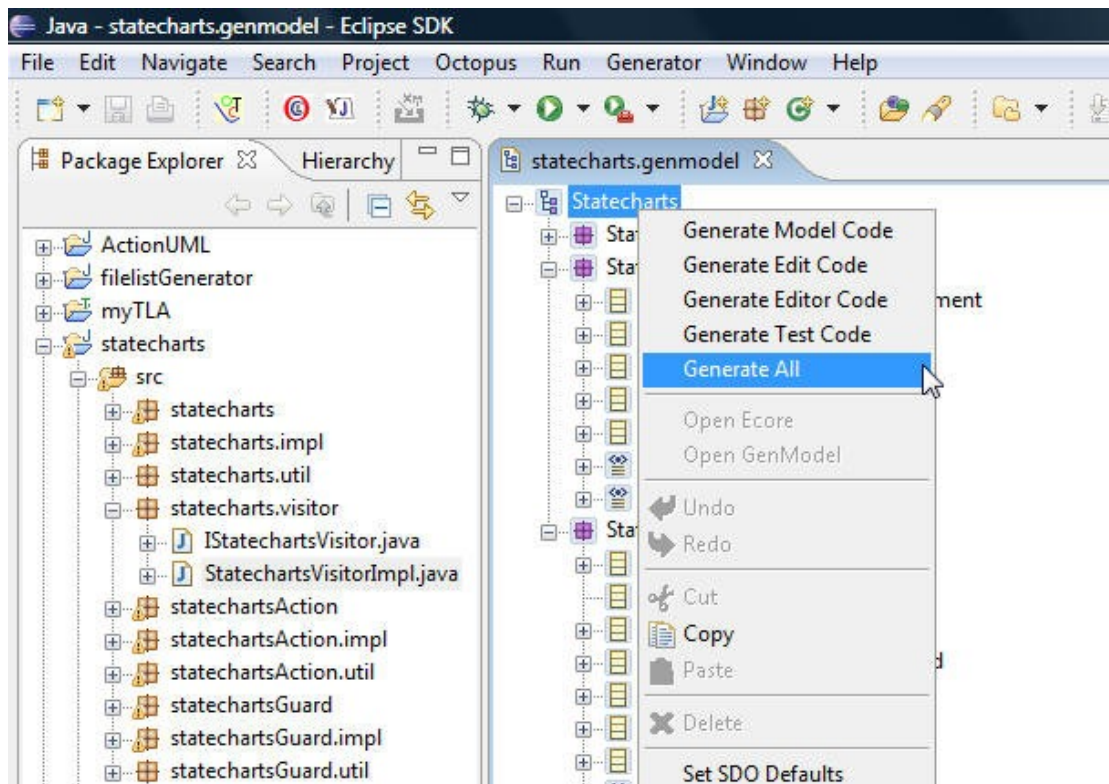


Fig 3.2 EMF Code Generation Structure

3.3. How to define the Instance of Metamodel

After we launch a second instance of Eclipse called a runtime workbench, we can create the instance of `statecharts` metamodel using the new created EMF Editor plugin.

Figure 3.3 below shows an example statechart diagram of trail-observer. Trail-observer is a component of the railway gateway system introduced by Max Göbel in his paper UML Statecharts. It is a component that observes the procedures on its

assigned track. After the activation, the track-observer is in the state Idle. If the signal trainApproaching is received by the sensor before the train passes, so it changes to the state trainPassing. But if a signal trainPassed is received at this time, then it is obviously an error (since the train is not yet here) and the observer will change to state faulty. If the signal trainPassed is received in the state trainPassing, so it will come back into the state Idle. However, if another train is coming or time exceeds the maximum duration (MAX_PASSING_DURATION) in the state trainPassing, the observer will also change to the state faulty.

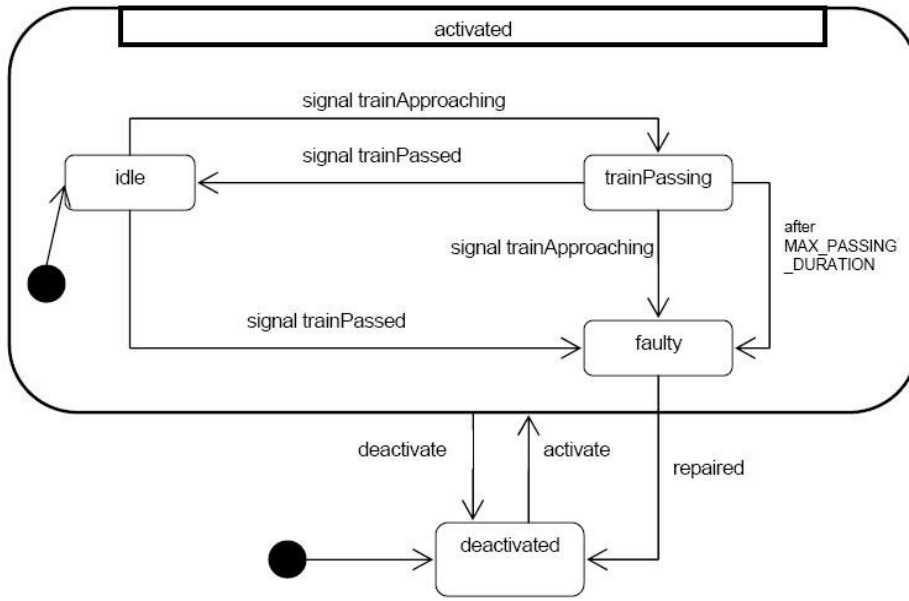


Fig 3.3 Statecharts of TrailObserver

According to the system behaviors occurred in railway gateway system, we define a statecharts instance named `TrailwayMachine.statecharts` for the complete system. The root component in this statecharts file is the statemachine `TrailwayMachine`. `TrailObserver` is a submachine of the root component. Under it we should also define some child nodes. Figure 3.4 shows the complete structure of the `TrailObserver` instance.

Some basic components have direct aggregation-associations with the submachine, such as Constants, Timers, Events and Attributes. They should be firstly instanced during the instancing process. Then all of the existing states should be instanced. The associated transitions and action sequences should be added in every state afterwards. According to the defined association in the metamodel, the target state has an aggregation association with the outgoing transition, thus the outgoing transition can be defined as the child node of state. As shown in the red part of Figure 3.4, the state `trainPassing` owns two outgoing transitions. Since the relation between source state and incoming transition is a normal association, after the defining of all outgoing

transitions, the incoming transitions will be automatically shown in the state's properties view. Figure 3.5 below shows the properties view of state.

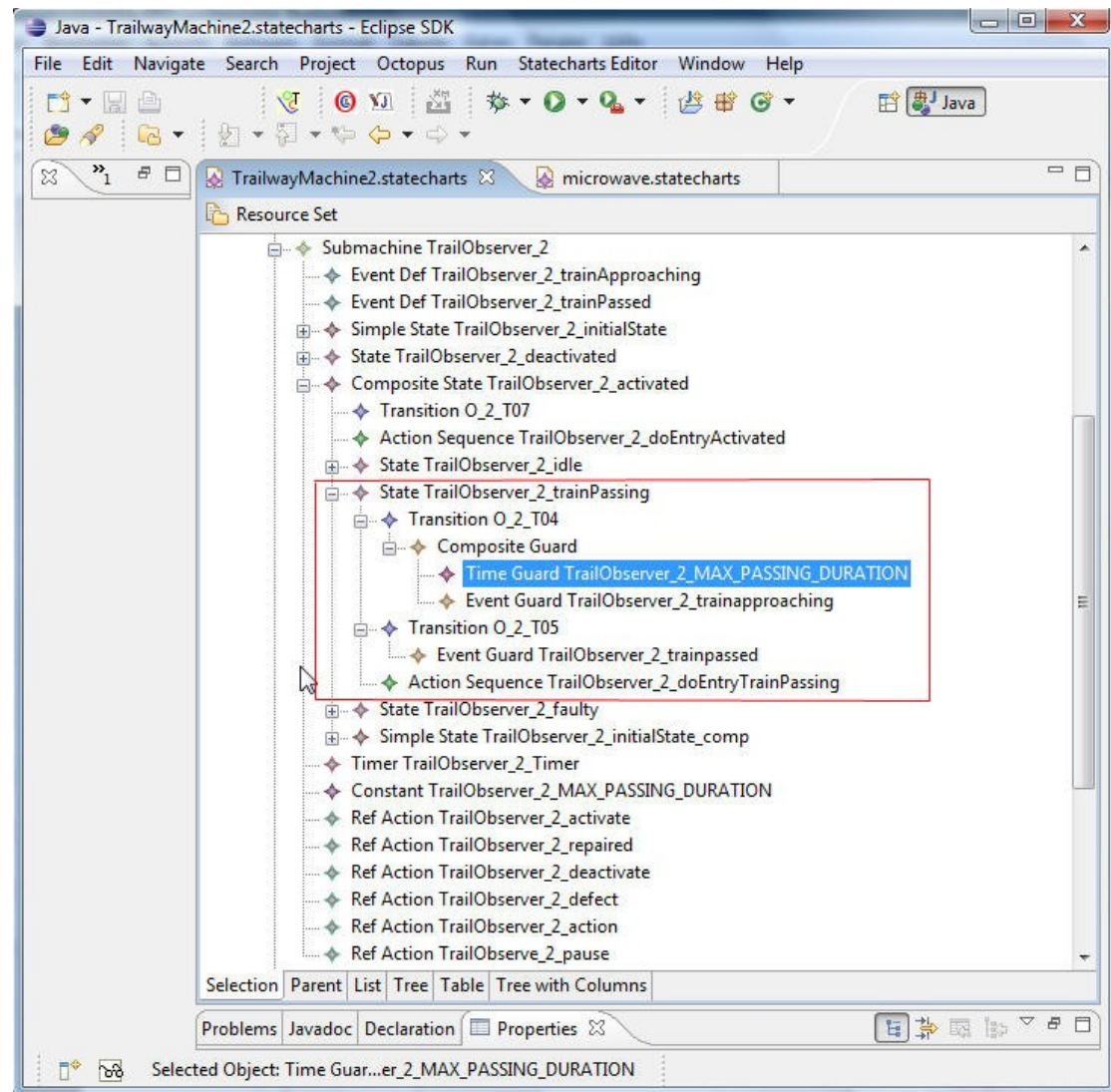


Fig 3.4 Instance Structure of TrailObserver

Each transition owns its guard. For example, the state trainPassing in above figure sends out two transitions, T04 and T05. Transition T04 owns a composite guard, which consists of a constant guard MAX_PASSING_DURATION and an event guard trainapproaching. As shown in figure 3.3 we know that these two guards are concurrent. Therefore, it is necessary to declare this relation. We can accomplish the definition of logic relation in the properties view of guard component. According to the defined logic relation, the corresponding TLA+ specification will be generated. Figure 3.6 below shows the properties view of guard component.

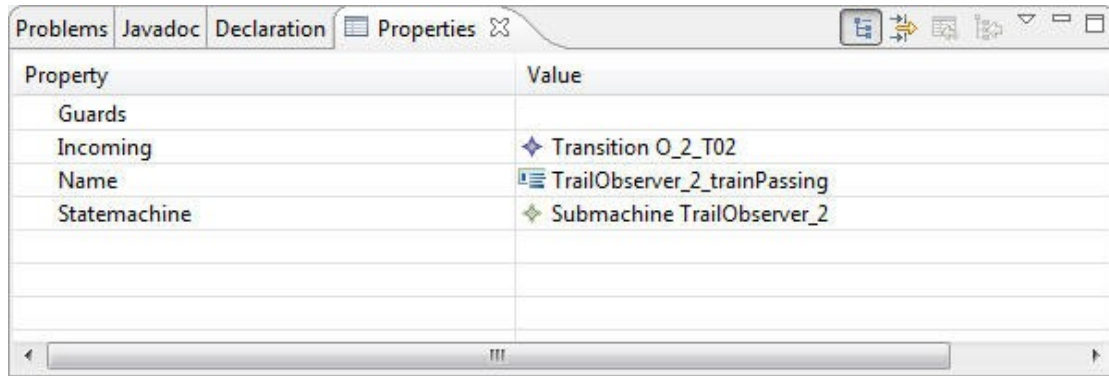


Fig 3.5 Properties View of State Component

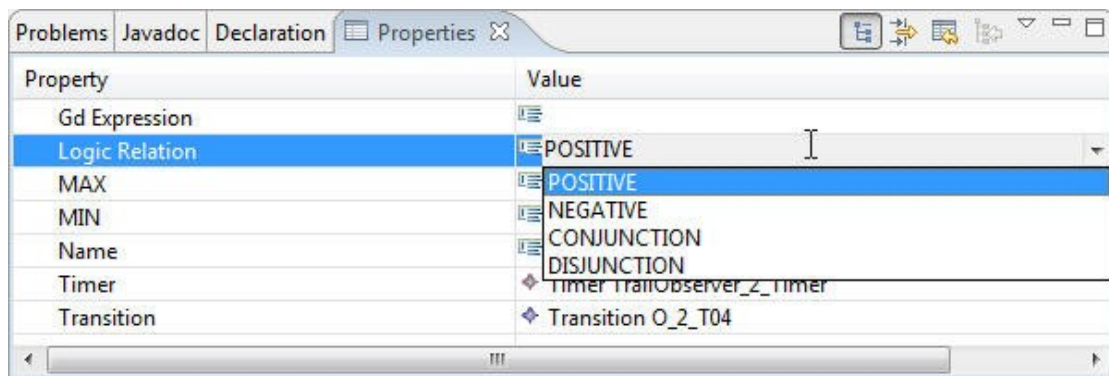


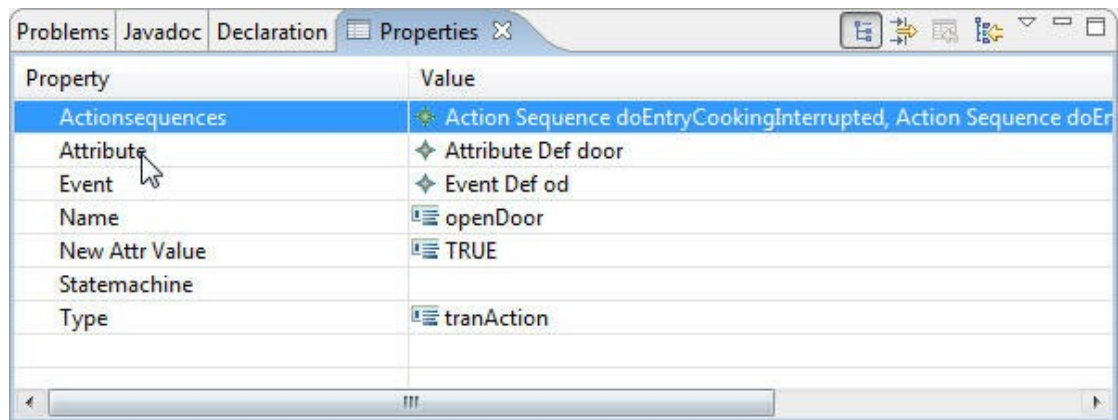
Fig 3.6 Properties View of Guard Component

According to the metamodel, each state should own its entry and exit action sequence. Therefore, action sequences have to be declared in the child nodes of state. Action sequence has mainly the following two functions:

1. Since the defined aggregation association between action sequence and action, the corresponding state changes described through attribute actions or time actions have to be defined as child nodes under action sequence.
2. Action sequence can be also used as "carrier" of reference action to connect reference action and its belonged state, since there is not direct connection between state and action. For the generation of TLA+ specification, we must build the link between action and state through action sequence. Therefore, we have to declare the corresponding action sequence in the properties view of the reference action.

The attribute action or time action belongs normally to an entry or exit action sequence, thus it should be declared as child nodes of action sequence. But reference action should be defined direct under the submachine, because it doesn't belong to any entry or exit action sequence of a particular state. Figure 3.7 shows the properties

view of the reference action openDoor of the microwave oven system. Noticed that the belonged action sequences have been already declared here.



Property	Value
Actionsequences	Action Sequence doEntryCookingInterrupted, Action Sequence doEn
Attribute	Attribute Def door
Event	Event Def od
Name	openDoor
New Attr Value	TRUE
Statemachine	
Type	tranAction

Fig 3.7 Properties View of Reference Action

Chapter 4. TLA+ Specification

4.1. Microwave oven

We use a microwave oven system as first example to demonstrate the TLA+ specification. Figure 4.1 shows its structure, which has one button and a light bulb inside. It defines six states to describe the behaviors of the system. If the door is closed and the button is pressed once, the oven starts cooking for 1 minute. Cooking time can be extended by pressing the button while cooking. The oven is controlled by a timer. Cooking is done using the power tube. If the door is opened, cooking is interrupted. Initially, the door is closed and the oven is not cooking.

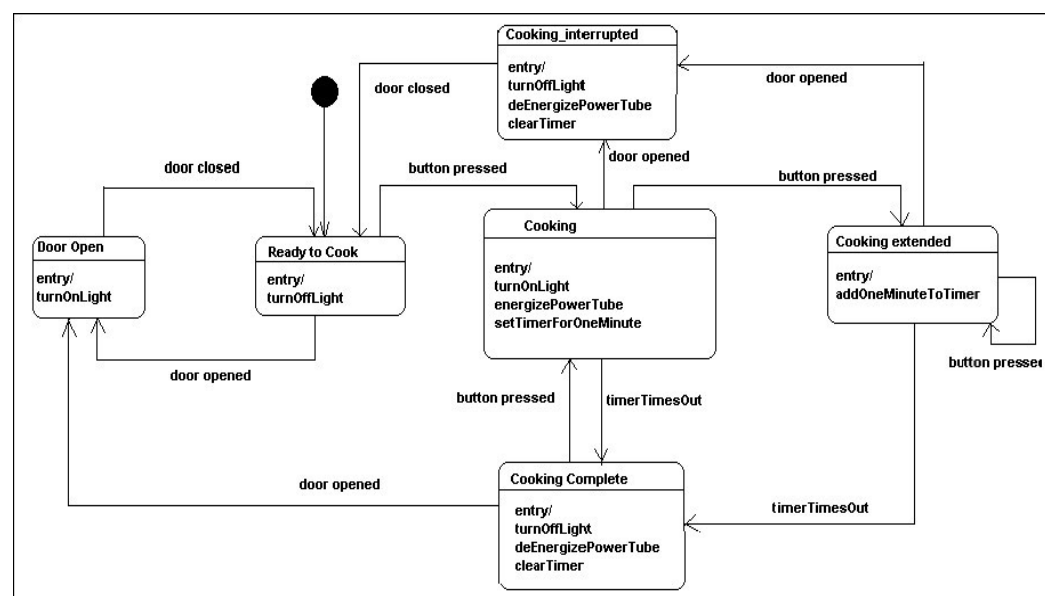


Fig 4.1 Statecharts of Microwave Oven System

The microwave can be energized or deenergized, has a door and an indicator light that shows if the microwave is currently running or not. Since the system has also some time functions, thus a timer named `microwaveOverTimer` has to be implemented, which should be also included in the variable declaration. Besides, a variable named `current` indicates the current state of the system.

This introduces in total five variables, as shown below.

VARIABLE current,
tube,
light,
door,

microwaveOvenTimer

The statement **TypeInvariant** defines the allowable initial types of these variables. For example, three of the above mentioned variables are always boolean variables. The timer has a restriction from -1 to 60 and the variable current has a type State. The TLA+ statement to express this is shown here:

```
TypeInvariant == /\tube \in BOOLEAN
                  /\current \in State
                  /\light \in BOOLEAN
                  /\door \in BOOLEAN
                  /\microwaveOvenTimer \in -1..60
```

Note that in this statement, the \wedge symbols must line up precisely underneath one another. Also, any text that follows the conjunct must appear to the right. The text can cross multiple lines if need be, but it must always appear to the right of the position of the symbol.

The **Init** statement defines the acceptable initial values of the variables. For example, in this case, current state of the system is ReadyToCook. The initial values of the microwave oven power, the indicator light and the door are off and the initial value of the microwaveOvenTimer is -1.

```
Init == /\tube = FALSE
         /\current = ReadyToCook
         /\light = FALSE
         /\door = FALSE
         /\microwaveOvenTimer = -1
```

In the metamodel, the action component have three different types, Entry-Action, Exit-Action and Transition-Action. Entry-Action/Exit-Action means the entry or exit action of the target or source state. Transition-Action is the action occurred between states during the transition takes place.

According to the object which the action deals with, the action components can be divided in various groups, Attribute-Action, Time-Action and Reference-Action. Attribute-Action is usually an entry of exit action. It is defined for the changing of attributes and associated with a particular attribute. For example the actions turnOnLight, energizePowerTube are associated with corresponding attributes light and power. The specification of a local entry or exit action in TLA+ begins normally with the definition of the target state, and it will be followed with the changing of attributes. Following code shows an example for the local entry actions of state CookingComplete and Cooking:

```

LOCAL doEntry_CookingComplete ==
    /\current' = CookingComplete
    /\light' = FALSE
    /\tube' = FALSE
    /\clearTimer

LOCAL doEntry_Cooking ==
    /\current' = Cooking
    /\light' = TRUE
    /\tube' = TRUE
    /\setTimerForOneMinute

```

Time-Action is always associated with the defined timer of the system. Through the initialization of the timer in the properties view, some basic time functions can be implemented. Some implemented time actions of microwave oven is shown below:

```

LOCAL clearTimer ==
    microwaveOvenTimer' = -1

LOCAL setTimerForOneMinute ==
    microwaveOvenTimer' = 60

LOCAL addOneMinuteToTimer ==
    microwaveOvenTimer' = microwaveOvenTimer + 60

```

Reference-Action exists normally between two through transition joined states. It has always the type Transition-Action. It defines the operations that must be occurred in order to enter the state. The possible Reference-Action include e.g. openDoor, closeDoor, and pressButton. Reference-Action in TLA+ begins normally with the validation of the current state and followed with the entry actions of the target state. Variables that are not changed during an action must be listed and indicated in UNCHANGED statement. The operation which adds the corresponding event into queue as transition trigger should be declared lastly. All same Reference-Actions should be summarized as combination of logic statements in the body of the declaration. Following code is the example code of the Reference-Action openDoor.

```

openDoor ==
    /\ /\current = Cooking
        /\doEntry_CookingInterrupted
        /\UNCHANGED << microwaveOvenTimer, tube, light >>
    /\ /\current = CookingExtended
        /\doEntry_CookingInterrupted
        /\UNCHANGED << microwaveOvenTimer, tube, light >>
    /\ /\current = ReadyToCook

```

```

/\doEntry_DoorOpen
/\UNCHANGED << microwaveOvenTimer, tube, light >>
\\/\current = CookingComplete
/\doEntry_DoorOpen
/\UNCHANGED << microwaveOvenTimer, tube, light >>
/\door' = TRUE
/\Enqueue('od')

```

The statement for Next state is defined as combination from any of the above mentioned Reference-Actions (using the disjunction symbol \vee), as shown here:

```

Next == \openDoor
        \closeDoor
        \pressButton

```

This model allows stuttering steps where all variables remain unchanged. The full specification, which includes stuttering steps, is given as the following combination of the initial state and the possible next states:

```
Spec == Init /\ [] [Next]_vars
```

A full version of the TLA+ specification is shown in Appendix 4.

4.2. Trailway Gateway System

The second example is about a trailway gateway system, which represents a hierarchical statecharts. The trailway gateway system is crossings on the railway and road. A trailway gateway system consists of following subsystems:

- **Railway platform:** the trailway gateway overpasses two railway platforms. We assume that the train comes always from the right side. A sensor will be put on the rail in a distance before the gateway depending on the maximum route-speed. It sends out a signal (trainApproaching) if a train comes. Another sensor is likewise put on the rail in a distance behind the railway platform, which will send out a signal (trainPassed) when the train left the critical range.
- **Gate:** which possesses an engine, with it the gate can be opened (startEngineUp) or closed (startEngineDown). As soon as the gate achieved the highest or lowest point, an appropriate sensor sends out a signal (gateOpened or gateClosed).
- **Traffic light,** which is posed either on red (on) or off.

If a sensor announces that a train is approaching, then the traffic lights will be switched to red and after a constant time interval (MAX_CAR_STOP_DURATION) the gate will be driven down. If the appropriate sensor behind the gateway sends out a signal, which means the train has left the critical range, the gate will be driven up and afterwards the traffic lights will be switched off (if no other train is still in the critical range).

It is noticed that all existing sensors (on the rails, at the gate) can fail or send out wrong signals. Also the gate engine or traffic light can fail. If an unexpected behaviour of a component is determined, it is perhaps due to an error of external systems (e.g. two trains come one after the other with a distance for only a few meters) or the failure of the actuator or sensor. For these cases, the railway gateway system must be equipped with a train control system. If an inconsistent state is determined, all trains within the range of gateway system must be stopped with the help of the train control system and the traffic lights will be switched to red and the gate will be driven down. Figure 4.2 below gives a graphical show of the behaviours of this system.

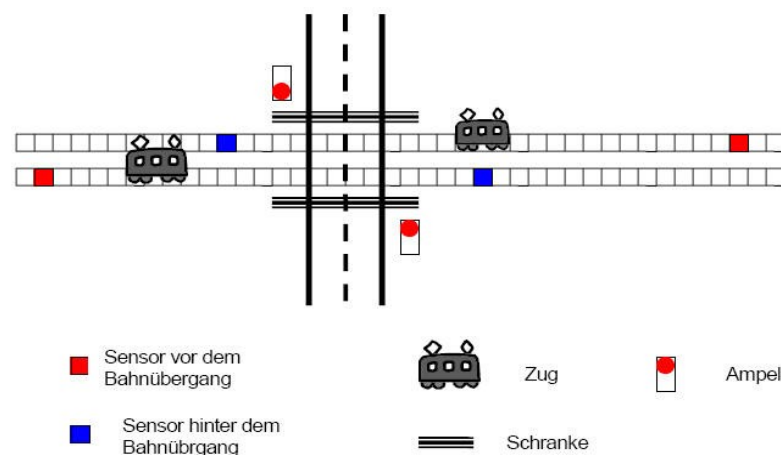


Fig 4.2 Railway Gateway System

The complete railway gateway system consists of various individual components. The individual component should be modelled same as the microwave oven system. Then the model for the complete system should be presented as a combination of the individual components.

The statechart diagram of individual components is shown below. The trail-observer has been already declared as example in the chapter 3. The only component should be declared is the state Faulty. It has altogether three entry action sequences coming from two various source states. Each transition has its guard, the specialization should be implemented as following:

```

TrailObserver_1_defect ==
  \/\current = TrailObserver_1_idle
  /\doEntry_TrailObserver_1_faulty
  /\TrailObserver_1_trainPassed
  /\UNCHANGED << TrailObserver_1_Timer >>
  \/\current = TrailObserver_1_trainPassing
  /\doEntry_TrailObserver_1_faulty
  /\TrailObserver_1_Timer > TrailObserver_1_MAX_PASSING_DURATION
  /\TrailObserver_1_trainApproaching
  /\UNCHANGED << TrailObserver_1_Timer >>

```

The traffic light shown in Figure 4.3 is simple to implement. It has a composite state Activated and a normal state Deactivated. Composite state Activated has two substates On and Off. Two reference actions switchLightOn and switchLightOff and their corresponding events switchOn and switchOff have to be implemented.

```

TrafficLight_1_switchLightOn ==
  \/\current = TrafficLight_1_OFF
  /\doEntry_TrafficLight_1_ON
  /\TrafficLight_1_light' = TRUE
  /\Enqueue('switchOn')

TrafficLight_1_switchLightOff ==
  \/\current = TrafficLight_1_ON
  /\doEntry_TrafficLight_1_OFF
  /\TrafficLight_1_light' = FALSE
  /\Enqueue('switchOff')

```

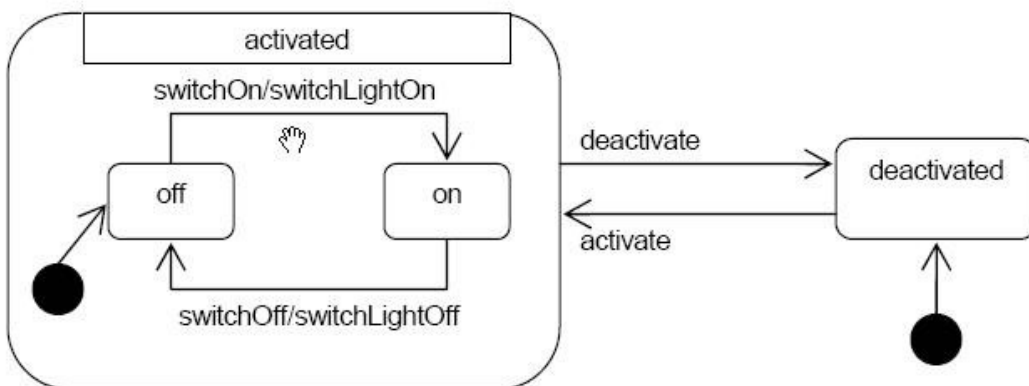


Fig 4.3 Statecharts of Trail-Light

The TLA+ specification for Trail-Gate shown in figure 4.4 is something similar. It has also a composite state Activated and a normal state Deactivated. Activated has altogether five states. The states Opening and Closing have their own entry and exit action sequence, every has only one attribute action. I have defined here two attributes named gate and running. Gate indicates whether the gate is driven up and running indicates the status of the engine. The corresponding do-Activity is shown below.

```
LOCAL doEntry_Gate_1_opening ==
  /\current' = Gate_1_opening
  /\Gate_1_gate' = TRUE
```

```
LOCAL doExit_Gate_1_opening ==
  /\current' = Gate_1_opening
  /\Gate_1_running' = FALSE
```

```
LOCAL doEntry_Gate_1_closing ==
  /\current' = Gate_1_closing
  /\Gate_1_gate' = FALSE
```

```
LOCAL doExit_Gate_1_closing ==
  /\current' = Gate_1_closing
  /\Gate_1_running' = FALSE
```

What should be noticed is the state Faulty which has two entry action sequences and one exit action sequence. Every transition where the action sequence lays has a constant guard which defines the maximal opening or closing time interval (MAX_OPENING /CLOSING_DURATION) what the gate should abide by. After the time interval, an error will arise and the gate changes into the state Faulty. I have defined a reference action named defect belonging to the above entry action sequence. The following code shows the specification of defect.

```
Gate_1_defect ==
  \/\current = Gate_1_closing
    /\doEntry_Gate_1_faulty
    /\Gate_1_Timer > Gate_1_MAX_CLOSING_DURATION
    /\UNCHANGED << Gate_1_Timer, Gate_1_gate, Gate_1_running >>
  \/\current = Gate_1_opening
    /\doEntry_Gate_1_faulty
    /\Gate_1_Timer < Gate_1_MIN_OPENING_DURATION
    /\UNCHANGED << Gate_1_Timer, Gate_1_gate, Gate_1_running >>
```

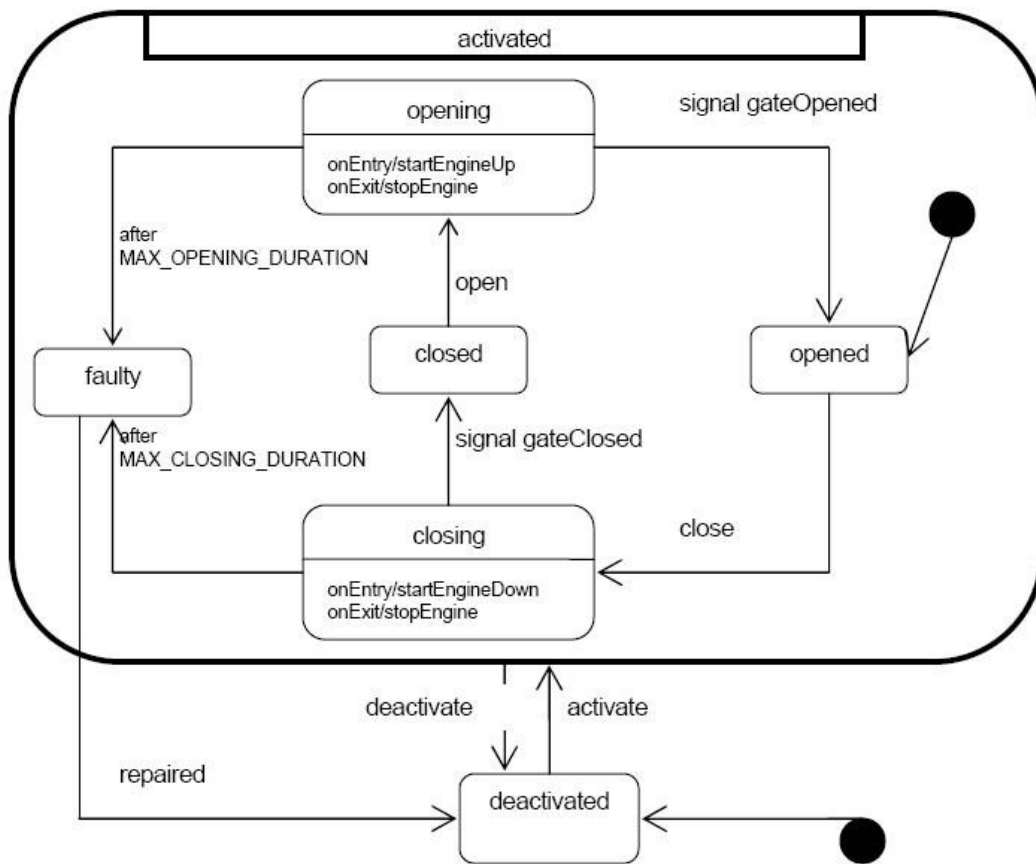


Fig 4.4 Statecharts of Trail-Gate

The instance of the complete gateway system is shown in figure 4.5, which contains two traffic lights, two trail observers, two gates, as well as a submachine for a CONTROLLER, which these components synchronizes and controls. All components are in one concurrent composite state, since they are all active processes. The CONTROLLER must be explicitly implemented here, since it must know other referenced components and synchronizes them.

Figure 4.6 shows the complete statechart diagram. Some changes have been made comparing with the simple statecharts. We use Reference-Action-Sequence replacing the simple Reference-Action. As shown in the figure 4.6, as soon as some components change into the state Faulty, a ChangeEvent will be produced which intercepts the CONTROLLER and brings the state from Activated to Deactivated. As result, the gate will be driven down and the traffic lights will be switched to red. What should be noticed here is that the guards and actions, both are composite components. The Reference-Action-Sequence deactivate is composed of various Reference-Actions from various submachines. It is guarded by a composite guard. The corresponding specialization is shown below: the guard definition has been declared in chapter 3, we use a variable named logic relation to indicate the relation between various individual guards.


```

TrailwayMachine_deactivate ==
  /\ /\ /\ TrailObserver_1_faulty
    \ /\ TrailObserver_2_faulty
      \ /\ Gate_1_faulty
        \ /\ Gate_2_faulty
  /\ TrafficLight_1_switchLightOff
  /\ TrafficLight_2_switchLightOff
  /\ Gate_1_closeGate
  /\ Gate_2_closeGate

```

Some other specifications of Reference-Action-Sequence are similar. For example, at the beginning of system's activation the CONTROLLER is in the state Idle. As soon as the sensor announces the coming of a train, a transition will be sent out from Idle to preparingToClose, as result the traffic lights will be switched to red. The corresponding specialization shows below:

```

TrailwayMachine_prepare ==
  /\ /\ TrailObserver_1_trainPassing
    \ /\ TrailObserver_2_trainPassing
  /\ TrafficLight_1_switchLightOn
  /\ TrafficLight_2_switchLightOn

```

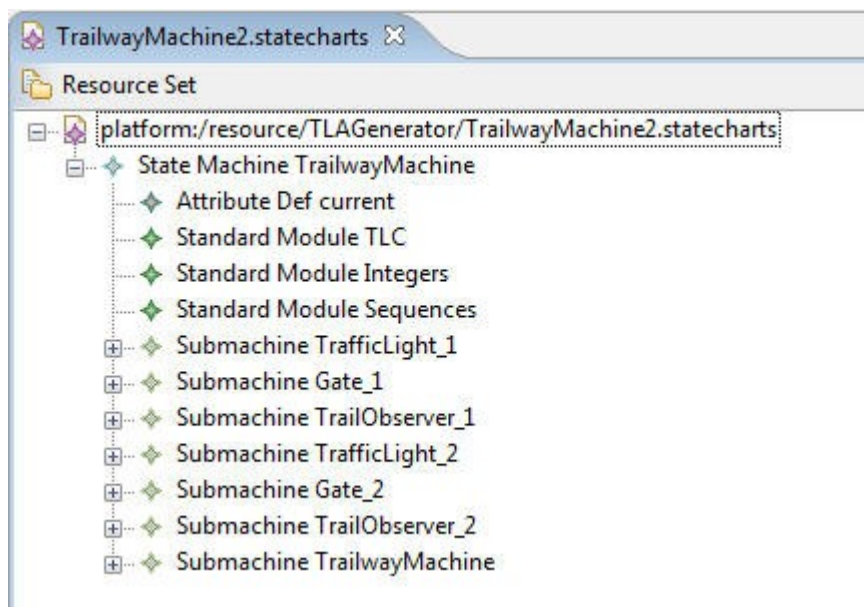


Fig 4.5 Instance Structure of Complete Trailway Gateway System

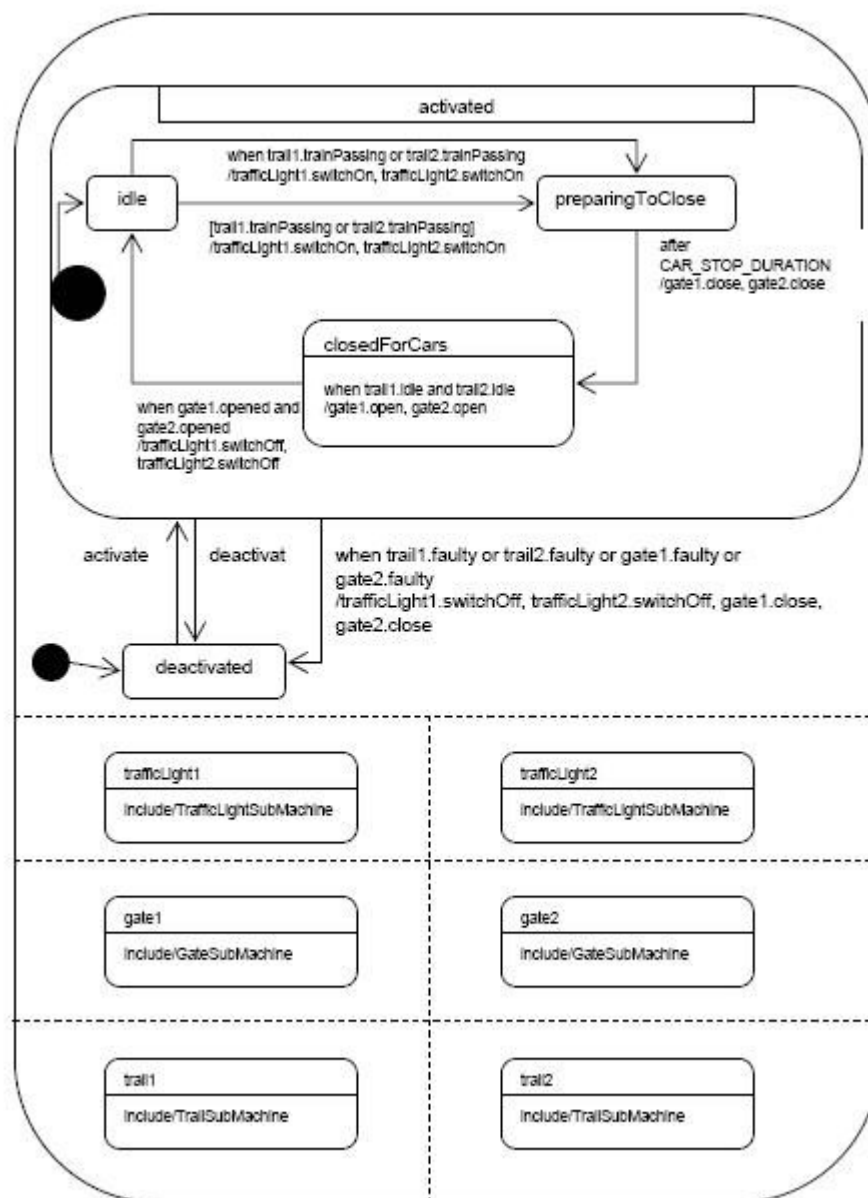


Fig 4.6 Statecharts of Complete Railway Gateway System

Chapter 5. Transformation Algorithms

After having the instance of statecharts metamodel in hand, we can launch the transformation plugin to generate the required TLA+ specification. Figure 5.1 shows the process of code generation.

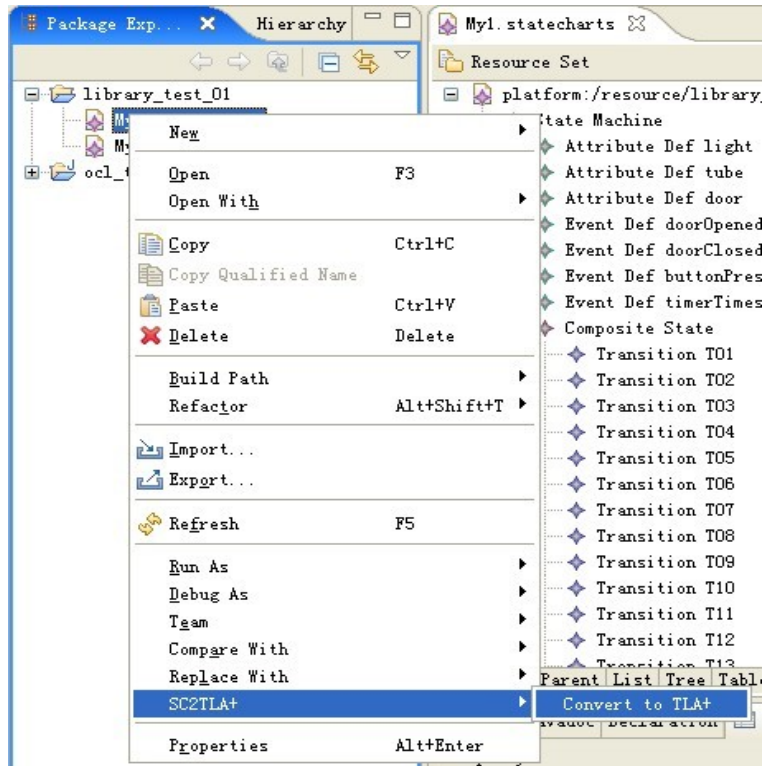


Fig 5.1 Generation of TLA+ Specification

5.1. Visitor Pattern

In the full processing life cycle, transformation is the most important part. It directly influences the correctness of final codes. I have chosen the visitor pattern to generate the corresponding TLA+ document. In this section, visitor pattern will be explained briefly.

The purpose of using visitor pattern is to encapsulate an operation that you want to perform on the elements of a data structure. In this way, you can change the operation being performed on a structure without the need of changing the classes of the elements that you are operating on. Using a visitor pattern allows you to decouple the classes for the data structure and the algorithms used upon them. Since there are various components in the UML statechart metamodel and we must generate the

corresponding statements for each of them, thus visitor pattern is here the suited choice for our purpose.

This process of visitor pattern is known as "Double Dispatching." As showing in the figure 5.2, each element in the data structure will accept a visitor, which sends a message to the visitor which includes the node's class. The visitor will then execute its visitElement method for that element. The accept method in the ConcreteElement classes realize the double dispatching call, where the visitor is passed in to the accept method, and that visitor is told to execute its visit method, and is handed the node by the node itself. Every visitor has a method for every data structure element type. The data structure elements however, only deal with the abstract Visitor, and hence only have one accept method that deals with it. That method is overridden in each concrete element, which only calls its respective method in the visitor.

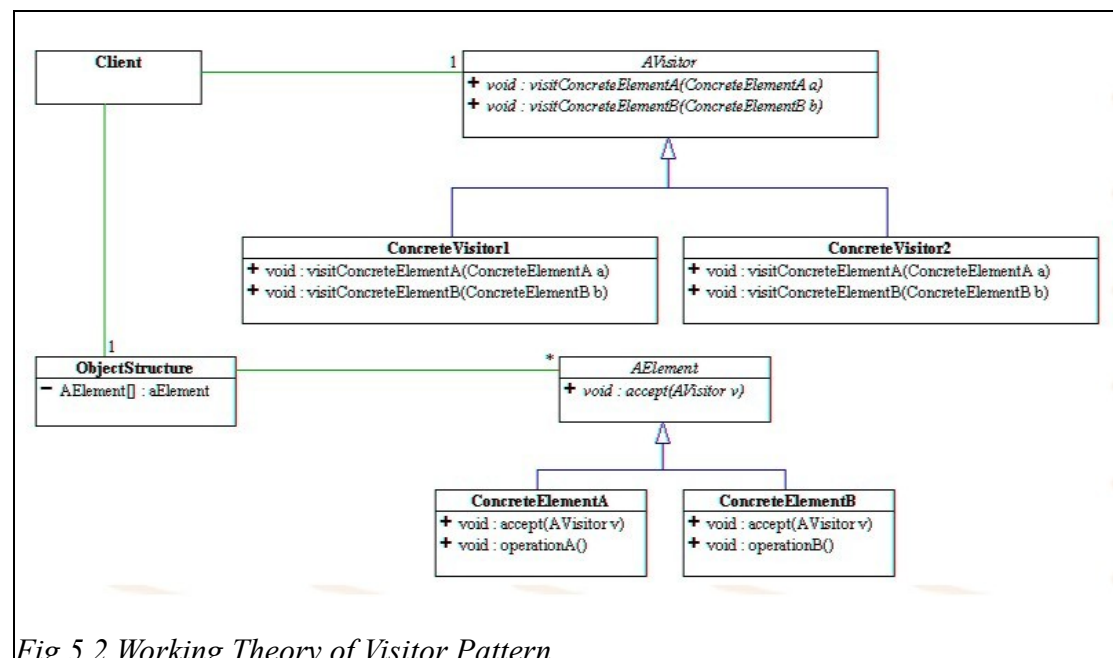


Fig 5.2 Working Theory of Visitor Pattern

Practically in this case, each component such as state, transition or event all contains an accept method that takes a visitor object as an argument. This accept method calls back the visit method for its class. As an example, the accept method of the component statemachine is shown in following code.

```

public void accept(IStatechartsVisitor visitor) {
    if ( beingVisited ) {
        return;
    }
    beingVisited = true;
    visitor.visitStateMachine(this);
    Iterator it = getStates().iterator();
    while ( it.hasNext() ) {

```

```

        State elem = (State) it.next();
        elem.accept(visitor);
    }
    it = getEvents().iterator();
    while ( it.hasNext() ) {
        EventDef elem = (EventDef) it.next();
        elem.accept(visitor);
    }
    it = getAttributes().iterator();
    while ( it.hasNext() ) {
        AttributeDef elem = (AttributeDef) it.next();
        elem.accept(visitor);
    }
    it = getPredefinedclasses().iterator();
    while ( it.hasNext() ) {
        PredefinedClass elem = (PredefinedClass) it.next();
        elem.accept(visitor);
    }
    it = getSubmachines().iterator();
    while ( it.hasNext() ) {
        SubmachineImpl elem = (SubmachineImpl) it.next();
        elem.accept(visitor);
    }
    it = getActionsequences().iterator();
    while ( it.hasNext() ) {
        ActionSequenceImpl elem = (ActionSequenceImpl) it.next();
        elem.accept(visitor);
    }
    it = getActions().iterator();
    try {
        while (it.hasNext()) {
            Action elem = (Action) it.next();
            if (elem.getClass().getName().equals("statechartsAction
            .impl.RefActionImpl")) {
                RefActionImpl elem_ref = (RefActionImpl) elem;
                elem_ref.accept(visitor);
            }
            else if (elem.getClass().getName().equals
            ("statechartsAction.impl.TimerActionImpl")) {
                TimerAction elem_time = (TimerAction) elem;
                elem_time.accept(visitor);
            }
        }
    }
}

```

```

        catch (Exception e){}
    }

```

As shown in the above code, we can define the "traverse rule" among the basic components in the accept method according to the associations defined in statecharts metamodel. The visitor can therefore know how to traverse these components and apply the correct visitElement to generate TLA+ specification. The concrete generation of TLA+ statements should be detailedly defined in the visitElement method in the implementation of the interface. A full version of the visitor implementation named `StatechartsVisitorImpl` is listed in the Appendix 3.

5.2. Transformation Algorithms

The goal of transformation is to get TLA+ specification from a statechart instance. After having a source instance and explaining the target TLA+ specification, we need to define some rules in between to let the transformation work. This section will talk about how to build the bridge between source and target language and connect them together. I must determine the corresponding relations between statechart metamodel and TLA+ statements. The TLA+ specification should be generated by handling the corresponding components. As mentioned in chapter 5.1, the visitor will execute visitElement methods for each component by implementing.

5.2.1. Handling State

I define a constant for each state occurred in the input statechart instance. They should be summarized under the TLA+ statement CONSTANTS. A TLA+ statement named STATE whose value is a set should be defined. Each state has a corresponding element in that set.

```

public Object visitState(State in) {
    ...
    constantSpaces += "" + in.getName() + ",";
    stateSpaces += "" + in.getName() + ",";
    ...
}

```

All local actions named doEntry or doExit should be implemented in each state. Since the entry and exit action sequences are always associated with the states, therefore they should also be declared by handling state.

```

...
Iterator it_actseq = in.getActionsequences().iterator();
while ( it_actseq.hasNext() ){
    ActionSequence elem_as = (ActionSequence) it_actseq.next();
}

```

```

ActionSequenceType type = elem_as.getType();
if(type.toString().equals("doEntryActionSequence")){
    if(!doEntrySpaces_tmp.contains("LOCAL doEntry_" + in.getName())){
        doEntrySpaces_tmp += "\r\n" + "LOCAL doEntry_" + in.getName() + "
        == " + "\r\n";
        doEntrySpaces_tmp += "    " + "/" + "\" + "current'" + " = " +
        in.getName() + "\r\n";
    }
    ...

```

The change of the corresponding variables should be described as combination of logic statements \wedge at the end of local action declaration.

```

...
AttrAction elem_attr = (AttrAction) elem_ac;
if (doEntrySpaces_tmp.contains(elem_attr.getAttribute().getName())) {
} else {
    doEntrySpaces_tmp += "    " + "/" + "\" + elem_attr.getAttribute().getName()
    + "'" + " = " + elem_attr.getNewAttrValue() + "\r\n";
}
...

```

5.2.2. Handling Attribute

Some attributes such as light, tube, door etc. in the microwave oven should be declared as boolean variables in the TLA+ specification. The variables should be summarized under the TLA+ statement VARIABLE. We should also define a TLA+ statement named VARS, whose value is also a set, each variable should have a corresponding element in it.

```

public Object visitAttributeDef(AttributeDef in) {
    if(varSpaces.contains(in.getName())){
    } else {
        varSpaces += "" + in.getName() + ",";
    }
    if(variableSpaces.contains(in.getName())){
    } else {
        variableSpaces += "" + in.getName() + "," + "\r\n" + "
        ";
    }
    if(initSpaces.contains(in.getName())){ else {
        initSpaces += "" + "/" + "\" + in.getName() + " = " + in.getAttrValue()
        + "\r\n";
    }
    if(typeInvariantSpaces.contains(in.getName())){ else {

```

```

        typeInvariantSpaces += "" + "/" + in.getName() + "\\in" +
        in.getAttrType() + "\\r\\n";
    }
    return null;
}

```

5.2.3. Handling Event

Event is normally used as trigger of transition. It has usually connection with the reference action. By implementing the reference action, the corresponding event should be saved and dispatched again at another time when needed.

Queue is needed for achieving the storing and dispatching functions of event. The methods enqueue and dequeue should be implemented. The method will be called at the end of the processing by every action declaration, so that the corresponding event will be added in the queue.

Java 1.5 adds a new Queue interface to the `java.util` package. The Queue interface gives you the new offer and poll methods. The poll method will return a null (i.e., not throw an exception) if there are no elements in the queue. In some cases, you might not want to extract the element at the head of the queue, but rather just take a peek at it. The Queue interface provides us with a peek method to do just that. If you're dealing with an empty queue, peek returns a null. As in the add-offer and remove-poll relationship, there is a peek-element relationship. In the case of an empty queue, the element method throws an unchecked exception. But if there are elements in the queue, the peek method allows you to take a gander at the first element without actually pulling it from the queue. Usage of the peek method is demonstrated in the `SimpleQueueUsageExamplePreferred` class and the code below shows the implementation of enqueue() and dequeue() methods.

```

public String enqueue(EventDef event) {
    eventQueue.offer(event.getName());
    return null;
}

public String dequeue(RefAction refaction) {
    String enqueued = "";
    Object pull = eventQueue.peek();
    if (pull!=null) enqueued = eventQueue.poll();
    return enqueued;
}

```


5.2.4. Handling Action

5.2.4.1. Executable UML Introduction

According to the description in the book "Executable UML: A Foundation for Model-Driven Architecture", Executable UML is a single language in the UML family, designed for a single purpose: to define the semantics of subject matters precisely. Executable UML is a particular usage, or profile, the formal manner in which we specify a set of rules for how particular elements in UML fit together for a particular purpose. It is designed to produce a comprehensive and understandable model of a solution independent of the organization of the software implementation. It is a highly abstract thinking tool that aids in the formalization of knowledge and is also a way of describing the concepts that make up abstract solutions to software development problems. Physically, an executable UML specification comprises a set of models represented as diagrams that describe and define the conceptualization and behavior of the real or hypothetical world under study. [6]

In statechart diagram, each state has local action sequence that is executed upon entry or exit of that state. The procedure is composed of actions, which are specified in an action language. The action is performed when the state is occupied and the event occurs. An entry action is performed whenever the state is entered. It is specified inside the state by the entry keyword, followed by a slash (/), followed by the action name. An exit action is performed whenever the state is exited. It is specified inside the state as the exit keyword, followed by a slash (/), followed by the action name. An Example of microwave oven is given below:

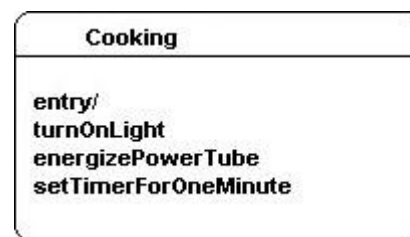


Fig. 5.3 State Cooking with Action

5.2.4.2. Handling Reference Action

Transition-Action such as openDoor, closeDoor or pressButton etc., which are occurred between the states, should be declared as action statements in the TLA+ specification. The preconditions for the Transition-Action should be checked firstly before the processing. For example, the current state/event must accord with the declared state/event. The attributes changed in this transition-action should be listed at

the end of the declaration. It is possible that a same transition-action is happened on different transitions. In this case, different transitions should be summarized as combination of logic statements \wedge or \vee in the action declaration.

```
public Object visitRefAction(RefAction in){
    ...
    actionSpaces += "\r\n" + in.getName() + " == " + "\r\n";
    ...
    Iterator noattr_it_actseq = in.getActionsequences().iterator();
    if(noattr_it_actseq.hasNext()){
        ActionSequence noattr_as1 = (ActionSequence)noattr_it_actseq
            .next();
        if(noattr_it_actseq.hasNext()){
            Iterator noattr_it_tr1 = noattr_as1.getTransitions()
                .iterator();
            if(noattr_it_tr1.hasNext()){
                Transition noattr_transition1 = (Transition)noattr_it_tr1
                    .next();
                actionSpaces += "    " + "\\\" + "/\" + "current = " +
                    noattr_transition1.getSource().getName() + "\r\n";
                actionSpaces += "    " + "    " + "/\" + "doEntry_ " +
                    noattr_transition1.getTarget().getName() + "\r\n";
                ...
            }
        }
    }
}
```

Lastly a TLA+ statement named UNCHANGED is used for all variables, which are not changed after the processing. Therefore, all of the unchanged variables should be summarized an the end of the action declaration.

```
unchangedSpace_tmp = printUnchangedAttr(in, unchangedSpace_current);
unchangedSpace_current = "";
if(unchangedSpace_tmp == null || unchangedSpace_tmp.equals("")){
} else {
    actionSpaces += "    " + "    " + "/\" + "UNCHANGED" + " << " +
        unchangedSpace_tmp + " >>" + "\r\n";
}
...
```

5.2.5. Handling Guard

Guards or conditions which are required in the input statecharts could be declared as single assumptions in TLA+. What should be noticed is that Guard must be lastly announced in the TLA+ statement THEOREM.

A guard condition is a boolean expression that may be attached to a transition in order

to determine whether that transition is enabled or not. The transition is fired only when the guard is true at the time the trigger event occurs. The boolean expression is written in terms of parameters of triggering event, timer, attributes of the components or occurrence of the state. There are various methods to implement a transition with guard condition. A simple transition can be extended to a tree structure of branches. Each branch has its own guard condition. When the trigger event occurs, if a branch has its guard condition true, it is fired. If no branch has a true condition, the event is ignored. [2]

To implement a transition with guard condition, we must put the guard in the TLA+ statement. The `visitRefAction` method in the visitor will be called whenever the corresponding reference action occurs while the source state is active and actual transitions will be called afterwards. A method name `generateGuard(transition)` will be used to generate guard statements in the generated TLA+ file and the result must be outputted line by line (following code shows the implementation). It must deal with the complex guards as described in last section. The concrete Java implementation of this method is shown in Appendix 3:

```
...
guardSpaces_tmp = generateGuard(noattr_transition2);
if(guardSpaces_tmp == null || guardSpaces_tmp.equals("")){
} else {
try {
    String guardSpaces_line = "";
    StringReader sr= new StringReader(guardSpaces_tmp); // wrap String
    BufferedReader br= new BufferedReader(sr); // wrap StringReader
    actionSpaces += "    " + "    " + "/\\" + br.readLine() + "\r\n";
    while((guardSpaces_line = br.readLine()) != null){
        actionSpaces += "    " + "    " + "    " + guardSpaces_line + "\r\n";
    }
} catch (Exception e) {}
}
...
```

5.2.6. Other Statecharts Components and TLA+ Statements

A state named `currentState` will be defined for the current state in the statechart diagram. Each state declared in the TLA+ constant `ALL_STATES` can be chosen as the value of `currentState`. An event named `currentEvent` will be defined for the current event. Similar to variable `currentState`, it can be assigned from any events that we have defined.

All Transition-Actions, which can change in any steps, should be summarized as combination of logic statement \vee in the statement **NEXT**.

The statement **TypeInvariant** in TLA+ will be used to explicitly describe the types of variables. Therefore, we should summarize all of the types of variables in it.

A statement named **Spec** will be declared at the end of TLA+ document for correctness checking. It is comprised normally of the statements INIT, NEXT and liveness condition.

Lastly, statement **THEOREM** should be declared. It asserts that Spec can be proved from the TypeInvariant and other assumptions, which we have already declared.

5.3. Plugin for the Generation of Target File

After having the above mentioned transformation algorithm, actual question is how to save the generated TLA+ specification? Therefore, we need also define a new project besides the Java project implementing the statecharts metamodel to accomplish the saving work. The Java package named `s2t.action` contains three files named `TLAPlusGenerator`, `GenerateTLAPlus` and `tlaFileWriter` which separately implements the following functions.:

- A TLA+ code generator should be created. The “top” state of the statecharts must be “passed” in the generator, the generator will traverse through all of the models in the object structure with the visitor from this point and generate the corresponding specification. A Java file named `TLAPlusGenerator.java` implements this function, which contains two methods: The method `getResource` returns the defined statecharts instance as the operation object of the method `generate`. The method `generate` gets first the top state of the statecharts and instances a new `StatechartsVisitor`, then traverses through all components in the statecharts and generate TLA+ specification.

```
String scFilePath = scFile.getFullPath().toString();
Resource ecoreResource = getResource(_resourceSet, scFilePath);
statemachine = (StateMachineImpl)ecoreResource.getContents().get(0);
StatechartsVisitorImpl v = new StatechartsVisitorImpl();
statemachine.accept(v);
```

- A second class named `GenerateTLAPlus` is used to handle the change of startcharts instance such as, add new model, delete new model or change the selection area. It extends `org.eclipse.ui.IObjectActionDelegate`, an Interface for an object action that is contributed into a popup menu for a view or editor. The following three methods must be implemented.

```
run(IAction)
```

This method is called by the proxy action when the action has been triggered. Implement this method to do the actual work.

`selectionChanged(IAction, ISelection)`

Notifies this action delegate that the selection in the workbench has changed.

`setActivePart(IAction, IWorkbenchPart)`

Sets the active part for the delegate.

A class named `GenerateTLAPlusJob` extending the `org.eclipse.core.runtime.jobs.Job` has to be created within this class. It is used to instance the TLA+ generator class and call its `generate` method through implementation of the `run` method.

```
private class GenerateTLAPlusJob extends Job {  
    protected IStatus run(IProgressMonitor monitor) {  
        TLAPlusGenerator generator = new TLAPlusGenerator();  
        generator.generate(_file, monitor);  
        ...  
    }  
    ...  
}
```

- Lastly, a class named `tlaFileWriter` has to be implemented to save the generated TLA+ code into a tla file. The key of this class is the implementation of the `writeFile` method.

For the full version of the package `s2t.action` please check the corresponding method in Appendix 2.

Chapter 6. TLC model checker

I describe in this section TLC model checker, how TLC works, and the experiences to use it.

6.1. Why model check

With TLA+, we can specify and verify any properties of a system. But the systems are probably too large and complicated to be completely verified by model checking. In spite of TLA+ consists of formal notations, but they may contain errors which can not be found only by formal reasoning. Therefore we need an external model checker to help us debug / verify our formal models. The model checker should be applied normally in a system with finite-state. It can both catch simple design errors and create an error report. Experiences tell us that using a model checker to debug formal models can speed the proof process. Checking the actual specification of a system with model checker can also improve the entire design process of a complicated system. It can eliminate the effort of debugging during the designing.

TLC is a new model checker for debugging a TLA+ specification by checking invariance properties of a finite-state model of the specification. It accepts a subclass of TLA+ specifications that should include most descriptions of real system designs. It has been used by engineers to find errors in the cache coherence protocol for a new Compaq multiprocessor. [3] Experiences tell us that the most effective way to find errors in a specification is by trying to verify that it satisfies properties that it should. TLC can check that the specification satisfies (implies) a large class of TLA formulas. Normally, there are two kinds of error that can be checked by TLC.

“Silliness” errors. a silly expression is one like $3 + < 1, 2 >$, whose meaning is not determined by the semantics of TLA+. A specification is incorrect if whether or not some particular behaviour satisfies it depends on the meaning of a silly expression.

Deadlock. The absence of deadlock is a particular property that we often want a specification to satisfy; it is expressed by the invariance property $[](\text{enabled Next})$. A counterexample to this property is a behaviour exhibiting deadlock - that is, reaching a state in which Next is not enabled, so no further step is possible. TLC normally checks for deadlock, but this checking can be disabled since, for some systems, deadlock may just indicate successful termination. [9]

Most TLA system specifications have the form $\text{Init} \wedge [][\text{Next}]_v \wedge \text{Liveness}$, where **Init** means the initial conditions, **Next** specifies the next-state relation, **v** means all of the variables in the specification and **Liveness** is a liveness property written as the conjunction of fairness conditions on actions. TLC does not support liveness

properties checking now, so Init and Next assertion checking are all that we concern.

6.2. How TLC works?

The input of TLC is specification of the system and a configuration file. The configuration file gives values to the constants and states the properties to be checked. The configuration file tells for example the names of the initial condition (Init), the next-state relation (Next), and some other constraint. For example, the configuration file for microwave oven system contains firstly the declaration

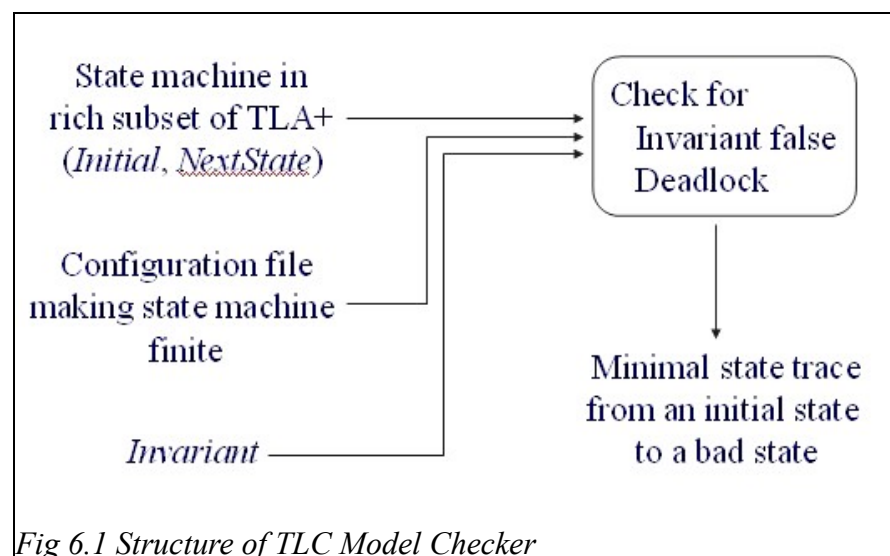
```
SPECIFICATION Spec
```

This statement tells TLC that Spec is the specification to be checked. The INVARIANT statement tells TLC to check that TypeInvariant is an invariant of the specification.

```
INVARIANT TypeInvariant
```

The configuration file also declares some values for the constants. TLC requires also that every declared constant in the specification be assigned a value by a CONSTANT statement in the configuration. Besides this, it should also contain the names of one or more invariants.

All above mentioned declarations define the model that TLC should check. Following figure 6.1 shows the structure of TLC model checker.



As mentioned above, TLC explores reachable states, that is all states that can occur in behaviors satisfying the formula $\text{Init} \wedge \llbracket \text{Next} \rrbracket \text{vars}$, and looks for two kinds of

errors.

1. an invariant is not satisfied
2. deadlock

The error report includes a minimal length trace that leads from an initial state to the bad state. TLC stops when it has examined all states reachable by traces that contain only states satisfying the constraint. (TLC may never terminate if this set of reachable states is not finite. In practice, it is easy to choose the constraint to ensure that the set is finite.)

TLC maintains two data structures: a set *seen* of all states known to be reachable and a FIFO queue *sq* containing elements of *seen* whose successor states have not been examined. The elements of *sq* are actual states, while *seen* contains only the fingerprints of its states. TLC's fingerprints are 64-bit, probabilistically unique checksums. For error reporting, an entry in *seen* also has a pointer to a predecessor state in *seen*. (The pointer is null for an initial state.) TLC begins by generating and checking all possible states satisfying the initial predicate and setting *seen* and *sq* to contain exactly those states. TLC next rewrites the next-state relation as a disjunction of as many simple subactions as possible.

TLC then launches a set of worker threads, each of which repeatedly does the following. It removes the state *s* from the front of *sq*. For each subaction *A*, the worker generates every possible next state *t* such that the pair of states *s*, *t* satisfies *A*. If there is no possible next state *t* for any subaction, a deadlock is reported. For each next state *t* that it generates, the worker does the following:

- Check if *t* is in *seen*.
- If it isn't, check if *t* satisfies the invariant.
- If it does, add *t* to *seen* (with a pointer to *s*).
- If *t* satisfies the constraint, add it to the end of *sq*.

An error is reported if a next state *t* is found that does not satisfy the invariant, or if *s* has no next state. In this case, TLC generates a trace ending in *t*, or in *s* if there is no next state *t*. Using the pointers in *seen*, TLC can generate a sequence of fingerprints of states. To generate the actual trace, TLC reruns the algorithm in the obvious goal-directed way. [8]

In our microwave oven system, TLC checks firstly all possible states according to the defined initial predicate:

```
Init = /\ light = false
      /\ tube = false
      /\ current = ReadyToCook
```



```
/\ open = false
```

and then it sets seen and sq to contain exactly those states satisfying the predicate. And then it decomposes the next-state relation

```
Next ==  /\  \/ openDoor
         \/ closeDoor
         \/ pressButton
```

as a disjunction of simple subactions. Here, the next state relation will be divided in three independent subactions. Each working thread launched from TLC will take a state from the front of sq, for example ReadyToCook. For each subaction, the worker thread looks for every possible next state that the pair satisfies it. For example, the states DoorOpen and ReadyToCook satisfying the action openDoor and the states Cooking and ReadyToCook satisfying the action pressButton. After the thoroughly checking through all subactions and states, a deadlock will be reported if there is no possible next state for any subaction. After that, the working thread will analyse each next state t according to the following mentioned method. First, it checks whether t is in seen. If it isn't, it will check whether t satisfies the invariant. If it does, t will be added to set seen (with a pointer to s). If t satisfies any constraint, it will be also added to the end of sq.

After the brief introduce of TLC model checking method, I would like to test the TLC with some given errors for the performance. I made some errors myself and let TLC test it. TLC eventually detects and reports all errors made from me.

First, I have defined some syntactic error in it. I write `vars == < light, tube, door, current >>` instead of the correct syntax of TLA+ code. The syntactic analyser detects immediately the error and gives out the following report. In this case, TLC will be not called.

```
Parsing file E:\Java\workspace_3.2.0\myTLA\src\micro.tla
***Parse Error***
    Encountered infix op < in block line 14, col 9 to line 14, col 9 of
    module micro on empty stack.
Residual stack trace follows:
Definition starting at line 14, column 1.
Module body starting at line 5, column 1.
Module definition starting at line 1, column 1.
Fatal errors while parsing TLA+ spec in file micro.tla
*** Errors: 1
Unknown location
Could not parse module micro from file E:/Java/workspace_3.2.0/myTLA/src/micro.tla
tlasany.parser.ParseException
```

```

at org.zambrovski.tla.tlasany.TLASyntaxParser.parse (TLASyntaxParser.java:184)
at org.zambrovski.tla.tlasany.TLASyntaxParser.parse (TLASyntaxParser.java:64)
at de.techjava.tla.core.parser.TLCParserRuntime.parse (TLCParserRuntime.java:47)
at de.techjava.tla.ui.builders.TLABuildVisitor.visit (TLABuildVisitor.java:149)
at org.eclipse.core.internal.resources.Resource$2.visit (Resource.java:105)
at org.eclipse.core.internal.resources.Resource$1.visitElement (Resource.java:57)
at
org.eclipse.core.internal.watson.ElementTreeIterator.doIteration (ElementTreeIterator.java:81)
at
org.eclipse.core.internal.watson.ElementTreeIterator.iterate (ElementTreeIterator.java:126)
at org.eclipse.core.internal.resources.Resource.accept (Resource.java:67)
at org.eclipse.core.internal.resources.Resource.accept (Resource.java:103)
at org.eclipse.core.internal.resources.Resource.accept (Resource.java:87)
at de.techjava.tla.ui.builders.TLABuilder.runBuild (TLABuilder.java:57)
at de.techjava.tla.ui.builders.TLABuilder.build (TLABuilder.java:33)
at org.eclipse.core.internal.events.BuildManager$2.run (BuildManager.java:603)
at org.eclipse.core.runtime.SafeRunner.run (SafeRunner.java:37)
at org.eclipse.core.internal.events.BuildManager.basicBuild (BuildManager.java:167)
at org.eclipse.core.internal.events.BuildManager.basicBuild (BuildManager.java:201)
at org.eclipse.core.internal.events.BuildManager$1.run (BuildManager.java:230)
at org.eclipse.core.runtime.SafeRunner.run (SafeRunner.java:37)
at org.eclipse.core.internal.events.BuildManager.basicBuild (BuildManager.java:233)
at org.eclipse.core.internal.events.BuildManager.basicBuildLoop (BuildManager.java:252)
at org.eclipse.core.internal.events.BuildManager.build (BuildManager.java:285)
at org.eclipse.core.internal.events.AutoBuildJob.doBuild (AutoBuildJob.java:145)
at org.eclipse.core.internal.events.AutoBuildJob.run (AutoBuildJob.java:208)
at org.eclipse.core.internal.jobs.Worker.run (Worker.java:58)

```

And then I corrected the syntactic error and set some relational errors in the specification. I deleted some statements for the unchanged variables in Transition-Action.

```

pressButton ==
  \/\ current = ReadyToCook
    /\ doEntryCooking
  \/\ current = CookingComplete
    /\ doEntryCooking
  \/\ current = Cooking
    /\ doEntryCookingExtended
  \/\ current = CookingExtended
    /\ doEntryCookingExtended

```

TLC detected and gave out the following report after a very short temporal interval

```

Starting...
-----
Main file      : micro
Config file   : micro
Dump file     : no set
-----

Root Dir      : E:\Java\workspace_3.2.0\myTLA\src\
Config Dir    : E:\Java\workspace_3.2.0\myTLA\config\
Work Dir      : E:\Java\workspace_3.2.0\myTLA\work\
-----

Parsing file E:\Java\workspace_3.2.0\myTLA\src\micro.tla
Parsing file E:\eclipse_3.2.0\plugins\de.techjava.tla.modules_0.0.1\modules\TLC.tla
Parsing file E:\eclipse_3.2.0\plugins\de.techjava.tla.modules_0.0.1\modules\Integers.tla
Parsing file E:\eclipse_3.2.0\plugins\de.techjava.tla.modules_0.0.1\modules\Sequences.tla
Parsing file E:\eclipse_3.2.0\plugins\de.techjava.tla.modules_0.0.1\modules\Naturals.tla
Semantic processing of module Naturals
Semantic processing of module Sequences
Semantic processing of module TLC
Semantic processing of module Integers
Semantic processing of module micro
Finished computing initial states: 1 distinct state generated.
Error: Successor state is not completely specified by the next-state action.
The behavior up to this point is:
STATE 1: <Initial predicate>
/\ tube = FALSE
/\ light = FALSE
/\ door = FALSE
/\ current = ReadyToCook

STATE 2: <Action line 69, col 6 to line 70, col 29 of module micro>
/\ tube = TRUE
/\ light = TRUE
/\ door = null
/\ current = Cooking

2 states generated, 1 distinct states found, 0 states left on queue.
The depth of the complete state graph search is 1.

```

Chapter 7. Future Work

The specification of TLA+ contains some complicated statements or properties, which is not implemented in this project. I finish this project by choosing some parts of them, which I considered necessary. It makes my statecharts metamodel not too complex to understand and implement. Similarly, the statecharts metamodel I defined contains just the most important components. One can not instance arbitrarily complex behavioural system with it. Further extension or improvements can be done in these area. Some possible development such as:

1. the current work implements just part of the existing statements of the language TLA+. Additional features should be incorporated. For example, the current timer can only specify some simplest functions such as the minimum and maximum elapsed time, initial value of timer and some regular changing. This can be extended to more accurately specified real-time systems, pointing to the interesting topic of modeling real time systems with UML and TLA+.
2. Another possible direction is to extend the scalability of the UML statecharts. Some more complicated components should be concerned such as the more complex tree transition, which brings more particularly guard conditions and more precise delineation of actions. For these larger models, the communication and synchronization among these components is more complicated than the current version. Therefore, a more complicate and efficient transformation algorithms should be developed to deal with the above mentioned instance.
3. The transformation algorithm should be developed to generate more precise specification for the action components defined with the Executable UML, which has a much more complex syntax structure.
4. For the implementation of guard components exist also other solution. For example, it can be implemented with OCL expressions, which can provide implicitly more concrete describing about the constraint.
5. Another interesting topic is to investigate the mappings from other UML diagram to TLA+ specification. For example, UML Class Diagram, which contains more complicated components as statechart diagram. Future work with more complicated diagram will generate more precise and efficient specialization, which builds a relationship between these two modeling system.

Chapter 8. Conclusion

This paper describes an implementation of transformation from UML statechart diagram to language TLA+. Two examples have been declared, which represent respectively the flat and hierarchical statecharts.

During the development of this project, i must first find out an appropriate metamodel to represent the different behaviours which may occur in the statecharts. I must continuously adjust the structure of the metamodel which guarantees that it can be suited with various presentations of statecharts and at the same time corresponds to the basic syntax structure of the language TLA+.

Eclipse Modelling Framework has been used to generate the MDA infrastructure Java codes and instance the given statecharts according to the defined metamodel. Since various TLA+ statements must be generated according to the components defined in metamodel, the use of visitor pattern can greatly simplify the difficulty of transformation of multi-components statecharts.

By generating the TLA+ specification, I must also develop a transformation algorithm to deal with different components of the statecharts. Flat or hierarchical statechart diagram should be transformed into appropriate TLA+ specification based on it. The developed transformation algorithm can successfully deal with almost all basic statecharts components, which include such as states, transitions, action events and so on.

Lastly, the generated TLA+ specification would be checked with TLC, which is a new model checker for debugging a TLA+ specification by checking invariance properties of a finite-state model of the specification.

Until here, I have finished the complete project. Since the time limit of this project, I have only transformed the main components of the statecharts and accordingly only the most important statements of TLA+ has been implemented. Further developments and improvements are possible such as

1. modeling more complicated UML statecharts.
2. incorporating more TLA+ notations.
3. handling the dynamic action components (Executable UML) with more complicated syntax.
4. handling the guard components with other implementation solution with more precision, such as OCL expressions.

5. transforming other UML diagram to TLA+

This project is an initial research in the area which transforms UML behavioral diagram to the language TLA+ and checks the correctness afterwards. I hope that this project can be used as a basis for automatic code generation from a model-based behavioral diagram to TLA+ and some different transforming solutions for different UML behavioral diagrams can be achieved along this idea in the future.

Appendix 1

Statechart.emf

```
@namespace(uri="http://example/statecharts",prefix="statecharts")
package statecharts;

import "platform:/resource/statecharts/model/statechartsAction.ecore";
import "platform:/resource/statecharts/model/statechartsGuard.ecore";

class NamedElement {
    attr String name;
}

class StateMachine extends NamedElement{
    !ordered val AttributeDef[0..*] attributes;
    !ordered val EventDef[0..*] events;
    !ordered val State[*] states;
    !ordered val PredefinedClass[0..*] predefinedclasses;
    !ordered val Submachine[*] submachines;
    !ordered val statechartsAction.Action[*] actions;
    !ordered val statechartsAction.ActionSequence[*] actionsequences;
}

class Submachine extends StateMachine{
    !ordered ref StateMachine[1] statemachine;
}

class State extends NamedElement {
    !ordered ref CompositeState #substates parent;
    !ordered ref Transition[0..*] #target incoming;
    !ordered ref StateMachine[1] statemachine;
    !ordered ref statechartsGuard.StateGuard[0..*] guards;
    !ordered val Transition[0..*] #source outgoing;
    !ordered val statechartsAction.ActionSequence[*] actionsequences;
}

class SimpleState extends State {
    attr SimpleStateType type;
}

class CompositeState extends State {
```

```

        !ordered val State[0..*] #parent substates;
    }

class Transition {
    attr String label;
    !ordered ref EventDef[0..*] triggers;
    !ordered ref State #outgoing source;
    !ordered ref State #incoming target;
    !ordered val statechartsGuard.Guard[0..1] guard;
    !ordered val statechartsAction.ActionSequence[0..1] actionsequence;
}

class EventDef extends NamedElement {
    !ordered ref Transition[0..*] transitions;
    !ordered ref StateMachine statemachine;
    !ordered ref statechartsAction.Action[0..*] actions;
    !ordered ref statechartsGuard.EventGuard[0..*] guards;
}

class AttributeDef extends NamedElement {
    attr AttributeType attrType;
    attr String attrValue;
    !ordered ref StateMachine statemachine;
    !ordered ref statechartsGuard.AttributeGuard[0..*] guards;
    !ordered val statechartsAction.AttrAction[0..*] attractions;
    !ordered val statechartsAction.RefAction[0..*] refactions;
}

class PredefinedClass extends NamedElement{
    !ordered ref StateMachine statemachine;
}

class Timer extends PredefinedClass {
    attr boolean running;
    attr int timer;
    attr int minTimerValue;
    attr int maxTimerValue;
    op void clearTime();
    op void addTime();
    !ordered val statechartsAction.TimerAction[*] timeractions;
    !ordered val statechartsGuard.TimeGuard[*] timerguards;
}

class QueueEmulate extends PredefinedClass {

```



```

    op String enqueue(EventDef event);
    op String dequeue(statechartsAction.RefAction refaction);
    op boolean containsItem(String ItemName);
}

```

```

class StandardModule extends PredefinedClass {

}

```

```

class Constant extends PredefinedClass {
    attr String constantValue;
}

```

```

enum SimpleStateType {
    initialState = 0;
    endState = 1;
    normalState = 2;
}

```

```

enum AttributeType {
    STRING = 0;
    BOOLEAN = 1;
    State = 2;
}

```

statechartsAction.emf

```

@namespace(uri="http://example/statechartsAction",prefix="statechartsAction")

```

```

package statechartsAction;

```

```

import "platform:/resource/statecharts/model/statecharts.ecore";
import "platform:/resource/statecharts/model/statechartsGuard.ecore";

```

```

class ActionSequence extends statecharts.NamedElement {
    attr ActionSequenceType type;
    !ordered ref statecharts.State[0..1] State;
    !ordered ref statecharts.Transition[0..*] transitions;
    !ordered ref statecharts.StateMachine statemachine;
    !ordered val Action[*] actions;
}

```

```

class Action extends statecharts.NamedElement {
    attr ActionType type;
}

```

```

    !ordered ref ActionSequence[*] actionsequences;
    !ordered ref statecharts.StateMachine statemachine;
    !ordered ref statecharts.EventDef event;
}

```

```

class AttrAction extends Action {
    attr String newAttrValue;
    !ordered ref statecharts.AttributeDef[1] attribute;
}

```

```

class RefAction extends Action {
    attr String newAttrValue;
    !ordered ref statecharts.AttributeDef[1] attribute;
}

```

```

class TimerAction extends Action {
    attr String newTime;
    !ordered ref statecharts.Timer[1] timer;
}

```

```

enum ActionSequenceType {
    doEntryActionSequence = 0;
    doExitActionSequence = 1;
    tranActionSequence = 2;
}

```

```

enum ActionType {
    doEntryAction = 0;
    doExitAction = 1;
    tranAction = 2;
}

```

statechartsGuard.emf

```

@namespace(uri="http://example/statechartsGuard",prefix="statechartsGuard")
package statechartsGuard;

```

```

import "platform:/resource/statecharts/model/statecharts.ecore";
import "platform:/resource/statecharts/model/statechartsAction.ecore";

```

```

class Guard extends statecharts.NamedElement {
    attr String gdExpression;
    attr GuardLogicType logicRelation;
    !ordered ref statecharts.Transition[0..1] transition;
}

```

```

    !ordered ref CompositeGuard #subguards parentguard;
}

class SimpleGuard extends Guard {

}

class TimeGuard extends SimpleGuard {
    attr String MAX;
    attr String MIN;
    !ordered ref statecharts.Timer[1] timer;
}

class AttributeGuard extends SimpleGuard {
    attr String MAX;
    attr String MIN;
    !ordered ref statecharts.AttributeDef[0..1] attribute;
}

class EventGuard extends SimpleGuard {
    !ordered ref statecharts.EventDef[0..1] event;
}

class StateGuard extends SimpleGuard {
    !ordered ref statecharts.State[0..1] state;
}

class CompositeGuard extends Guard {
    !ordered val Guard[0..*] #parentguard subguards;
}

enum GuardLogicType {
    POSITIVE = 0;
    NEGATIVE = 1;
    CONJUNCTION = 2;
    DISJUNCTION = 3;
}

```

Appendix 2

GenerateTLAPlus.java

```
public class GenerateTLAPlus implements IObjectActionDelegate {
    private IFile _file;
    private GenerateTLAPlusJob _job;

    public GenerateTLAPlus() {}

    private class GenerateTLAPlusJob extends Job {
        protected IStatus run(IPrimaryProgressMonitor monitor) {
            TLAPlusGenerator generator = new TLAPlusGenerator();
            generator.generate(_file, monitor);
            _job = null;
            return Status.OK_STATUS;
        }
        GenerateTLAPlusJob(){
            super("Generating TLA+ Source for " + _file.getName());
        }
    }

    public void run(IAction action) {
        if(_file != null && _job == null) {
            _job = new GenerateTLAPlusJob();
            _job.schedule();
        }
    }

    public void selectionChanged(IAction action, ISelection selection) {
        _file = null;
        if(selection instanceof IStructuredSelection) {
            IStructuredSelection sel = (IStructuredSelection)selection;
            Object selElem = sel.getFirstElement();
            if(selElem instanceof IFile)
                _file = (IFile)selElem;
        }
    }

    public void setActivePart(IAction iaction, IWorkbenchPart iworkbenchpart)
    {}
}
```

TLAPlusGenerator.java

```
public class TLAPlusGenerator {
    private ResourceSet _resourceSet;
```

```

private static EClassImpl eclass = null;

public TLAPlusGenerator() {
    _resourceSet = new ResourceSetImpl();
}

public void generate(IFile scFile, IProgressMonitor monitor) {
    try {
        String scFilePath = scFile.getFullPath().toString();
        Resource ecoreResource = getResource(_resourceSet, scFilePath);
        eclass = (EClassImpl)ecoreResource.getContents().get(0);
        StatechartsVisitorImpl v = new StatechartsVisitorImpl();
        eclass.accept(v);
        StringBuffer sb = StatechartsVisitorImpl.generateTLA();
        tlaFileWriter efw = new tlaFileWriter();
        efw.write(scFile.getProject(), sb);
    }
    catch(Exception ex) {
        ex.printStackTrace();
    }
}

private Resource getResource(ResourceSet resourceSet, String filePath) {
    URI uri = URI.createPlatformResourceURI(filePath);
    Resource resource = resourceSet.getResource(uri, true);
    return resource;
}
}

```

tlaFileWriter.java

```

public void write(IProject project, StringBuffer sb) {
    this.project = project;
    String fileName = project.getName() ;
    writeFile(sb, fileName);
}

private void writeFile(StringBuffer myResult, String name) {
    IPath filePath = null;
    try {
        filePath = buildFileNameFor(name, "tla");
        IFile file = OctopusPlugin.getWorkspace().getRoot().getFile(
            filePath);
        if (file != null) {
            if (file.exists()) {
                if (file.isDerived()) {
                    createFile(file, myResult, null, true);
                }
            }
        }
    }
    catch (Exception e) {
        // ...
    }
}

```

```

        }
    } else {
        createFile(file, myResult, null, true);
    }
}

} catch (CoreException e) {
    OctopusPlugin.getDefault().logError(this.getClass().getName(), e);
} catch (IOException e) {
    OctopusPlugin.getDefault().logError(this.getClass().getName(), e);
}
}
}

```

Appendix 3

```
package statecharts.visitor;

import java.io.BufferedReader;
import java.io.StringReader;
import java.util.Iterator;
import statecharts.AttributeDef;
import statecharts.CompositeState;
import statecharts.Constant;
import statecharts.EventDef;
import statecharts.NamedElement;
import statecharts.PredefinedClass;
import statecharts.QueueEmulate;
import statecharts.SimpleState;
import statecharts.StandardModule;
import statecharts.StateMachine;
import statecharts.Submachine;
import statecharts.State;
import statecharts.Timer;
import statecharts.Transition;
import statecharts.impl.QueueEmulateImpl;
import statechartsAction.ActionSequenceType;
import statechartsAction.ActionSequence;
import statechartsAction.Action;
import statechartsAction.AttrAction;
import statechartsAction.RefAction;
import statechartsAction.TimerAction;
import statechartsAction.impl.RefActionImpl;
import statechartsGuard.AttributeGuard;
import statechartsGuard.CompositeGuard;
import statechartsGuard.EventGuard;
import statechartsGuard.Guard;
import statechartsGuard.SimpleGuard;
import statechartsGuard.StateGuard;
import statechartsGuard.TimeGuard;
import statechartsGuard.impl.AttributeGuardImpl;
import statechartsGuard.impl.CompositeGuardImpl;
import statechartsGuard.impl.EventGuardImpl;
import statechartsGuard.impl.StateGuardImpl;
import statechartsGuard.impl.TimeGuardImpl;

public class StatechartsVisitorImpl implements IStatechartsVisitor {
```

```

public static String constantSpaces = "";
public static String stateSpaces = "";
public static String doEntrySpaces = "";
public static String doExitSpaces = "";
public static String nextSpaces = "";
public static String varSpaces = "";
public static String variableSpaces = "";
public static String initSpaces = "";
public static String unchangedSpace_tmp;
public static String unchangedSpace_current;
public static String typeInvariantSpaces = "";
public static String actionSpaces = "";
public static String timeactionSpaces = "";
public static String refseqSpaces = "";
public static String moduleName = "";
public static String neededModule = "";
public static String guardSpaces_tmp = "";

QueueEmulate evQueue = new QueueEmulateImpl();
public Submachine submachine = null;

public Object visitNamedElement(NamedElement in) {
    return null;
}

public Object visitStateMachine(StateMachine in) {
    if(in.getClass().getName().equals("statecharts.impl.StateMachineImpl")){
        moduleName = in.getName();
    }
    return null;
}

public Object visitSubmachine(Submachine in){
    submachine = in;
    return null;
}

public Object visitState(State in) {
    String doEntrySpaces_tmp = "";
    String doExitSpaces_tmp = "";
    if(in.getName() == null){
        return null;
    }
}

```



```

constantSpaces += "" + in.getName() + ",";
stateSpaces += "" + in.getName() + ",";

// attribute added here
Iterator it_actseq = in.getActionsequences().iterator();
while ( it_actseq.hasNext() ){
    ActionSequence elem_as = (ActionSequence) it_actseq.next();
    ActionSequenceType type = elem_as.getType();
    if(type.toString().equals("doEntryActionSequence")){
        // it ensures that the doEntry statement appears only one time.
        if(!doEntrySpaces_tmp.contains("LOCAL doEntry_" + in.getName())){
            doEntrySpaces_tmp += "\r\n" + "LOCAL doEntry_" + in.getName() + " == " +
            "\r\n";
            doEntrySpaces_tmp += "    " + "/" + in.getName() + " = " + in.getName() +
            "\r\n";
        }
        Iterator it_ac = elem_as.getActions().iterator();
        try {
            while (it_ac.hasNext()) {
                Action elem_ac = (Action) it_ac.next();

                if (elem_ac.getClass().getName().equals("statechartsAction.impl.
                TimerActionImpl")) {
                    TimerAction elem_time = (TimerAction) elem_ac;
                    if (doEntrySpaces_tmp.contains(elem_time.getName())) {
                        //do nothing
                    } else {
                        doEntrySpaces_tmp += "    " + "/" + elem_time.getName() +
                        "\r\n";
                    }

                    if (timeactionSpaces.contains(elem_time.getName())) {
                        //do nothing
                    } else {
                        timeactionSpaces += "LOCAL " + elem_time.getName() + " == " +
                        "\r\n";
                        timeactionSpaces += "    " + elem_time.getTimer().getName() +
                        "" + " = " + elem_time.getNewTime() + "\r\n\r\n";
                    }
                }
            }
        } else if (elem_ac.getClass().getName().equals("statechartsAction
        .impl.AttrActionImpl")){
            AttrAction elem_attr = (AttrAction) elem_ac;
            if (doEntrySpaces_tmp.contains(elem_attr.getAttribute())

```

```

        .getName())) {
            //do nothing
        } else {
            doEntrySpaces_tmp += "    " + "/" + elem_attr.getAttribute()
            .getName() + "'" + " = " + elem_attr.getNewAttrValue() + "\r\n";
        }
    }
}

}

}

catch (Exception e){}

}

else {
    if(!doExitSpaces_tmp.contains("LOCAL doExit_" + in.getName())){
        doExitSpaces_tmp += "\r\n" + "LOCAL doExit_" + in.getName() + " == " +
        "\r\n";
        doExitSpaces_tmp += "    " + "/" + "current'" + " = " + in.getName() +
        "\r\n";
    }

    Iterator it_ac = elem_as.getActions().iterator();
    try {
        while (it_ac.hasNext()) {
            Action elem_ac = (Action) it_ac.next();

            if (elem_ac.getClass().getName().equals("statechartsAction
            .impl.TimerActionImpl")) {
                TimerAction elem_time = (TimerAction) elem_ac;
                if (doExitSpaces_tmp.contains(elem_time.getName())) {
                    //do nothing
                } else {
                    doExitSpaces_tmp += "    " + "/" + elem_time.getName() +
                    "\r\n";
                }

                if (timeactionSpaces.contains(elem_time.getName())) {
                    //do nothing
                } else {
                    timeactionSpaces += "LOCAL " + elem_time.getName() + " == " +
                    "\r\n";
                    timeactionSpaces += "    " + elem_time.getTimer().getName() +
                    "'" + " = " + elem_time.getNewTime() + "\r\n\r\n";
                }
            }
        }
    }
    else if (elem_ac.getClass().getName().equals("statechartsAction
    .impl.AttrActionImpl")){

```

```

        AttrAction elem_attr = (AttrAction) elem_ac;
        if(doExitSpaces_tmp.contains(elem_attr.getAttribute().getName())){
            //do nothing
        } else {
            doExitSpaces_tmp += "    " + "/" + elem_attr.getAttribute()
                .getName() + "'"+ " = " + elem_attr.getNewAttrValue() + "\r\n";
        }
    }
}

    }
    }
    }
    catch (Exception e){}
}

doEntrySpaces += doEntrySpaces_tmp + "\r\n";
doExitSpaces += doExitSpaces_tmp + "\r\n";
return null;
}

public Object visitSimpleState(SimpleState in) {
    return null;
}

public Object visitCompositeState(CompositeState in) {

    return null;
}

public Object visitTransition(Transition in) {
    return null;
}

public Object visitEventDef(EventDef in) {
    return null;
}

public Object visitAttributeDef(AttributeDef in) {
    if(varSpaces.contains(in.getName())){
        /* do nothing */
    } else {
        varSpaces += "" + in.getName() + ",";
    }
    if(variableSpaces.contains(in.getName())){
        /* do nothing */
    } else {

```

```

        variableSpaces += "\"" + in.getName() + "," + "\r\n" + "        ";
    }
    if(initSpaces.contains(in.getName())){
        /* do nothing */
    } else {
        initSpaces += "\"" + "/" + in.getName() + " = " + in.getAttrValue() + "\r\n" + " ";
    }
    if(typeInvariantSpaces.contains(in.getName())){
        /* do nothing */
    } else {
        typeInvariantSpaces += "\"" + "/" + in.getName() + " \\in " + in.getAttrType() +
            "\r\n" + "        ";
    }
    return null;
}

// visit PredefinedClass
public Object visitPredefinedClass(PredefinedClass in) {
    return null;
}

public Object visitTimer(Timer in) {
    varSpaces += "\"" + in.getName() + ",";
    variableSpaces += "\"" + in.getName() + "," + "\r\n" + "        ";
    initSpaces += "\"" + "/" + in.getName() + " = " + "-1" + "\r\n" + "        ";
    typeInvariantSpaces += "\"" + "/" + in.getName() + " \\in " + in.getMinTimerValue() +
        ".." + in.getMaxTimerValue() + "\r\n" + "        ";
    return null;
}

public Object visitQueue(QueueEmulate in) {
    return null;
}

public Object visitStandardModule(StandardModule in) {
    neededModule += in.getName() + ",";
    return null;
}

public Object visitConstant(Constant in) {
    constantSpaces += in.getName() + ",";
    return null;
}

```

```

/* statechartsAction */

public Object visitActionSequence(ActionSequence in){
    String refseqSpaces_tmp = "";
    refseqSpaces += "\r\n\r\n" + in.getName() + " == " + "\r\n";
    if (in.getType().toString().equals("tranActionSequence")) {
        Iterator it = in.getTransitions().iterator();
        while (it.hasNext()) {
            Transition elem_tr = (Transition) it.next();
            refseqSpaces_tmp = generateGuard(elem_tr);
            if (refseqSpaces_tmp == null || refseqSpaces_tmp.equals("")){
                /* do nothing */
            } else {
                try {
                    String refactionseqSpaces_line = "";
                    StringReader sr= new StringReader(refseqSpaces_tmp); // wrap String
                    BufferedReader br= new BufferedReader(sr); // wrap StringReader
                    refseqSpaces += "    " + "/\\" + br.readLine() + "\r\n";
                    while((refactionseqSpaces_line = br.readLine()) != null){
                        refseqSpaces += "    " + "    " + refactionseqSpaces_line + "\r\n";
                    }
                } catch (Exception e) {}
            }
        }
        it = in.getActions().iterator();
        while (it.hasNext()) {
            Action elem = (Action) it.next();
            if (elem.getClass().getName().equals("statechartsAction.impl.RefActionImpl")){
                RefActionImpl elem_ra = (RefActionImpl) elem;
                refseqSpaces += "    " + "/\\" + elem_ra.getName() + "\r\n";
            }
        }
    }
    return null;
}

public Object visitAction(Action in){
    return null;
}

public Object visitRefAction(RefAction in){

    unchangedSpace_current = "";
    nextSpaces += "\\/" + in.getName() + "\r\n" + "    ";
}

```

```

// get attribute
AttributeDef attr = null;
try {
    attr = in.getAttribute();
} catch (Exception e) {}

// test if the transition has a trigger
boolean existTrigger = false;
if(in.getEvent() != null){
    existTrigger = true;
    // add the trigger to queue
    addToQueue(in);
}

actionSpaces += "\r\n" + in.getName() + " == " + "\r\n";

// no attribute, no trigger
if (attr == null && existTrigger == false){
    Iterator noattr_it_actseq = in.getActionsequences().iterator();
    if(noattr_it_actseq.hasNext()){
        ActionSequence noattr_as1 = (ActionSequence) noattr_it_actseq.next();
        // exist many actionsequence
        if(noattr_it_actseq.hasNext()){
            // first actionsequence
            Iterator noattr_it_tr1 = noattr_as1.getTransitions().iterator();
            if(noattr_it_tr1.hasNext()){
                Transition noattr_transition1 = (Transition) noattr_it_tr1.next();
                // first transition
                actionSpaces += "    " + "\\/" + "/" + "current = " +
                    noattr_transition1.getSource().getName() + "\r\n";
                actionSpaces += "    " + " " + "/" + "doEntry_" +
                    noattr_transition1.getTarget().getName() + "\r\n";
                guardSpaces_tmp = generateGuard(noattr_transition1);
                if(guardSpaces_tmp == null || guardSpaces_tmp.equals("")){
                    /* do nothing */
                } else {
                    // output the guard row by row
                    try {
                        String guardSpaces_line = "";
                        StringReader sr= new StringReader(guardSpaces_tmp);
                        BufferedReader br= new BufferedReader(sr);
                        actionSpaces += "    " + " " + "/" + br.readLine() + "\r\n";
                        while((guardSpaces_line = br.readLine()) != null){
                            actionSpaces += "    " + " " + " " + guardSpaces_line +

```

```

        "\r\n";
    }
    } catch (Exception e) {}
}

// decide if current state exists
if (noattr_transition1.getSource().getName().equals
(noattr_transition1.getTarget().getName())){
    unchangedSpace_current = "current, ";
}

unchangedSpace_tmp = printUnchangedAttr(in, unchangedSpace_current);
unchangedSpace_current = "";
if(unchangedSpace_tmp == null || unchangedSpace_tmp.equals("")){
    /* do nothing */
} else {
    actionSpaces += "    " + " " + "/" + "\" + "UNCHANGED" + " << " +
    unchangedSpace_tmp + " >>" + "\r\n";
}

// other transitions
while(noattr_it_tr1.hasNext()){
    Transition noattr_transition2 = (Transition) noattr_it_tr1.next();
    actionSpaces += "    " + "\" + "/" + "\" + "current = " +
    noattr_transition2.getSource().getName() + "\r\n";
    actionSpaces += "    " + " " + "/" + "\" + "doEntry_" +
    noattr_transition2.getTarget().getName() + "\r\n";
    guardSpaces_tmp = generateGuard(noattr_transition2);
    if(guardSpaces_tmp == null || guardSpaces_tmp.equals("")){
        /* do nothing */
    } else {
        // output the guard row by row
        try {
            String guardSpaces_line = "";
            StringReader sr= new StringReader(guardSpaces_tmp);
            BufferedReader br= new BufferedReader(sr);
            actionSpaces += "    " + " " + "/" + "\" + br.readLine() +
            "\r\n";
            while((guardSpaces_line = br.readLine()) != null){
                actionSpaces += "    " + " " + " " + guardSpaces_line
                + "\r\n";
            }
        } catch (Exception e) {}
    }

    if (noattr_transition2.getSource().getName().equals
    (noattr_transition2.getTarget().getName())){
        unchangedSpace_current = "current, ";
    }
}

```

```

    }
    unchangedSpace_tmp = printUnchangedAttr(in,
    unchangedSpace_current);
    unchangedSpace_current = "";
    if(unchangedSpace_tmp == null || unchangedSpace_tmp.equals("")){
        /* do nothing */
    } else {
        actionSpaces += "    " + " " + "/" + "\" + "UNCHANGED" + " << " +
        unchangedSpace_tmp + " >>" + "\r\n";
    }
}
}

// exist only one actionsequence
else{
    Iterator noattr_it_tr2 = noattr_as1.getTransitions().iterator();
    if(noattr_it_tr2.hasNext()){
        Transition noattr_transition3 = (Transition) noattr_it_tr2.next();
        // exist many transitions, first transition
        if(noattr_it_tr2.hasNext()){
            actionSpaces += "    " + "/" + "\" + "current = " +
            noattr_transition3.getSource().getName() + "\r\n";
            actionSpaces += "    " + " " + "/" + "\" + "doEntry_" +
            noattr_transition3.getTarget().getName() + "\r\n";
            guardSpaces_tmp = generateGuard(noattr_transition3);
            if(guardSpaces_tmp == null || guardSpaces_tmp.equals("")){
                /* do nothing */
            } else {
                // output the guard row by row
                try {
                    String guardSpaces_line = "";
                    StringReader sr= new StringReader(guardSpaces_tmp);
                    BufferedReader br= new BufferedReader(sr);
                    actionSpaces += "    " + " " + "/" + "\" + br.readLine() +
                    "\r\n";
                    while((guardSpaces_line = br.readLine()) != null){
                        actionSpaces += "    " + " " + " " + guardSpaces_line
                        + "\r\n";
                    }
                } catch (Exception e) {}
            }
        }
        if (noattr_transition3.getSource().getName().equals
        (noattr_transition3.getTarget().getName())){

```



```

        unchangedSpace_current = "current, ";
    }
    unchangedSpace_tmp = printUnchangedAttr(in,
    unchangedSpace_current);
    unchangedSpace_current = "";
    if(unchangedSpace_tmp == null || unchangedSpace_tmp.equals("")){
        /* do nothing */
    } else {
        actionSpaces += "    " + " " + "/" + "\" + "UNCHANGED" + " " << " " +
        unchangedSpace_tmp + " >>" + "\r\n";
    }
}
// exist only one transition, one actionsequence
else{
    actionSpaces += "    " + "/" + "\" + "current = " +
    noattr_transition3.getSource().getName() + "\r\n";
    actionSpaces += "    " + "/" + "\" + "doEntry_" +
    noattr_transition3.getTarget().getName() + "\r\n";
    guardSpaces_tmp = generateGuard(noattr_transition3);
    if(guardSpaces_tmp == null || guardSpaces_tmp.equals("")){
        /* do nothing */
    } else {
        // output the guard row by row
        try {
            String guardSpaces_line = "";
            StringReader sr= new StringReader(guardSpaces_tmp);
            BufferedReader br= new BufferedReader(sr);
            actionSpaces += "    " + "/" + "\" + br.readLine() + "\r\n";
            while((guardSpaces_line = br.readLine()) != null){
                actionSpaces += "    " + " " + guardSpaces_line +
                "\r\n";
            }
        } catch (Exception e) {}
    }
    if (noattr_transition3.getSource().getName().equals
    (noattr_transition3.getTarget().getName())){
        unchangedSpace_current = "current, ";
    }
    unchangedSpace_tmp = printUnchangedAttr(in,
    unchangedSpace_current);
    unchangedSpace_current = "";
    if(unchangedSpace_tmp == null || unchangedSpace_tmp.equals("")){
        /* do nothing */
    } else {

```



```

    }//else
    // other actionsequences, if there are many actionsequence
    while(noattr_it_actseq.hasNext()){
        ActionSequence noattr_as2 = (ActionSequence) noattr_it_actseq.next();
        Iterator noattr_it_tr3 = noattr_as2.getTransitions().iterator();
        while(noattr_it_tr3.hasNext()){
            Transition noattr_transition5 = (Transition) noattr_it_tr3.next();
            actionSpaces += "    " + "\\/" + "/\" + "current = " +
            noattr_transition5.getSource().getName() + "\r\n";
            actionSpaces += "    " + "    " + "/\" + "doEntry_ " +
            noattr_transition5.getTarget().getName() + "\r\n";
            guardSpaces_tmp = generateGuard(noattr_transition5);
            if(guardSpaces_tmp == null || guardSpaces_tmp.equals("")){
                /* do nothing */
            } else {
                // output the guard row by row
                try {
                    String guardSpaces_line = "";
                    StringReader sr= new StringReader(guardSpaces_tmp);
                    BufferedReader br= new BufferedReader(sr);
                    actionSpaces += "    " + "    " + "/\" + br.readLine() + "\r\n";
                    while((guardSpaces_line = br.readLine()) != null){
                        actionSpaces += "    " + "    " + "    " + guardSpaces_line +
                        "\r\n";
                    }
                } catch (Exception e) {}
            }
            if (noattr_transition5.getSource().getName().equals
            (noattr_transition5.getTarget().getName())){
                unchangedSpace_current = "current, ";
            }
            unchangedSpace_tmp = printUnchangedAttr(in, unchangedSpace_current);
            unchangedSpace_current = "";
            if(unchangedSpace_tmp == null || unchangedSpace_tmp.equals("")){
                /* do nothing */
            } else {
                actionSpaces += "    " + "    " + "/\" + "UNCHANGED" + " << " +
                unchangedSpace_tmp + " >>" + "\r\n";
            }
        }
    }//while
} //if(noattr_it_actseq.hasNext())
}
// with attribute or queue

```

```

else {
    Iterator it_actseq = in.getActionsequences().iterator();
    if(it_actseq.hasNext()){
        ActionSequence as1 = (ActionSequence) it_actseq.next();
        // exist many actionsequence
        if(it_actseq.hasNext()){
            // first actionsequence
            Iterator it_tr1 = as1.getTransitions().iterator();
            if(it_tr1.hasNext()){
                Transition transition1 = (Transition) it_tr1.next();
                // first transition
                actionSpaces += "    " + "/" + "\" + "\" + "/" + "\" + "current = " +
                transition1.getSource().getName() + "\r\n";
                actionSpaces += "    " + "    " + "    " + "/" + "\" + "doEntry_" +
                transition1.getTarget().getName() + "\r\n";
                guardSpaces_tmp = generateGuard(transition1);
                if(guardSpaces_tmp == null || guardSpaces_tmp.equals("")){
                    /* do nothing */
                } else {
                    // output the guard row by row
                    try {
                        String guardSpaces_line = "";
                        StringReader sr= new StringReader(guardSpaces_tmp);
                        BufferedReader br= new BufferedReader(sr);
                        actionSpaces += "    " + "    " + "    " + "/" + "\" + br.readLine() +
                        "\r\n";
                        while((guardSpaces_line = br.readLine()) != null){
                            actionSpaces += "    " + "    " + "    " + "    " +
                            guardSpaces_line + "\r\n";
                        }
                    } catch (Exception e) {}
                }
            }
            if (transition1.getSource().getName().equals
            (transition1.getTarget().getName())){
                unchangedSpace_current = "current, ";
            }
            unchangedSpace_tmp = printUnchangedAttr(in, unchangedSpace_current);
            unchangedSpace_current = "";
            if(unchangedSpace_tmp == null || unchangedSpace_tmp.equals("")){
                /* do nothing */
            } else {
                actionSpaces += "    " + "    " + "    " + "/" + "\" + "UNCHANGED" + " << "
                + unchangedSpace_tmp + " >>" + "\r\n";
            }
        }
    }
}

```

```

// other transitions
while(it_tr1.hasNext()){
    Transition transition2 = (Transition) it_tr1.next();
    actionSpaces += "    " + " " + "\\/" + "/\" + "current = " +
    transition2.getSource().getName() + "\r\n";
    actionSpaces += "    " + " " + " " + "/\" + "doEntry_" +
    transition2.getTarget().getName() + "\r\n";
    guardSpaces_tmp = generateGuard(transition2);
    if(guardSpaces_tmp == null || guardSpaces_tmp.equals("")){
        /* do nothing */
    } else {
        // output the guard row by row
        try {
            String guardSpaces_line = "";
            StringReader sr= new StringReader(guardSpaces_tmp);
            BufferedReader br= new BufferedReader(sr);
            actionSpaces += "    " + " " + " " + "/\" +
            br.readLine() + "\r\n";
            while((guardSpaces_line = br.readLine()) != null){
                actionSpaces += "    " + " " + " " + " " + " " +
                guardSpaces_line + "\r\n";
            }
        } catch (Exception e) {}
    }
    if (transition2.getSource().getName().equals
    (transition2.getTarget().getName())){
        unchangedSpace_current = "current, ";
    }
    unchangedSpace_tmp = printUnchangedAttr(in,
    unchangedSpace_current);
    unchangedSpace_current = "";
    if(unchangedSpace_tmp == null || unchangedSpace_tmp.equals("")){
        /* do nothing */
    } else {
        actionSpaces += "    " + " " + " " + " " + "/\" + "UNCHANGED" + " "
        << " + unchangedSpace_tmp + " >>" + "\r\n";
    }
}
}

// exist only one actionsequence
else{
    Iterator it_tr2 = as1.getTransitions().iterator();

```

```

if(it_tr2.hasNext()){
    Transition transition3 = (Transition) it_tr2.next();
    // exist many transitions, first transition
    if(it_tr2.hasNext()){
        actionSpaces += "    " + "/" + "\" + "\" + "/" + "\" + "current = " +
        transition3.getSource().getName() + "\r\n";
        actionSpaces += "    " + "    " + "    " + "/" + "\" + "doEntry_" +
        transition3.getTarget().getName() + "\r\n";
        guardSpaces_tmp = generateGuard(transition3);
        if(guardSpaces_tmp == null || guardSpaces_tmp.equals("")){
            /* do nothing */
        } else {
            // output the guard row by row
            try {
                String guardSpaces_line = "";
                StringReader sr= new StringReader(guardSpaces_tmp);
                BufferedReader br= new BufferedReader(sr);
                actionSpaces += "    " + "    " + "    " + "/" + "\" +
                br.readLine() + "\r\n";
                while((guardSpaces_line = br.readLine()) != null){
                    actionSpaces += "    " + "    " + "    " + "    " +
                    guardSpaces_line + "\r\n";
                }
            } catch (Exception e) {}
        }
        if (transition3.getSource().getName().equals
        (transition3.getTarget().getName())){
            unchangedSpace_current = "current, ";
        }
        unchangedSpace_tmp = printUnchangedAttr(in,
        unchangedSpace_current);
        unchangedSpace_current = "";
        if(unchangedSpace_tmp == null || unchangedSpace_tmp.equals("")){
            /* do nothing */
        } else {
            actionSpaces += "    " + "    " + "    " + "/" + "\" + "UNCHANGED" + "
            << " + unchangedSpace_tmp + " >>" + "\r\n";
        }
    }
    // exist only one transition, one actionsequence
    else{
        actionSpaces += "    " + "/" + "\" + "/" + "\" + "current = " +
        transition3.getSource().getName() + "\r\n";
        actionSpaces += "    " + "    " + "    " + "/" + "\" + "doEntry_" +

```

```

transition3.getTarget().getName() + "\r\n";
guardSpaces_tmp = generateGuard(transition3);
if(guardSpaces_tmp == null || guardSpaces_tmp.equals("")){
    /* do nothing */
} else {
    // output the guard row by row
    try {
        String guardSpaces_line = "";
        StringReader sr= new StringReader(guardSpaces_tmp);
        BufferedReader br= new BufferedReader(sr);
        actionSpaces += "    " + "    " + "/\\" + br.readLine() +
            "\r\n";
        while((guardSpaces_line = br.readLine()) != null){
            actionSpaces += "    " + "    " + "    " + guardSpaces_line
                + "\r\n";
        }
    } catch (Exception e) {}
}
if (transition3.getSource().getName().equals
(transition3.getTarget().getName())){
    unchangedSpace_current = "current, ";
}
unchangedSpace_tmp = printUnchangedAttr(in,
unchangedSpace_current);
unchangedSpace_current = "";
if(unchangedSpace_tmp == null || unchangedSpace_tmp.equals("")){
    /* do nothing */
} else {
    actionSpaces += "    " + "    " + "/\\" + "UNCHANGED" + " << " +
        unchangedSpace_tmp + " >>" + "\r\n";
}
}

// other transitions, if there are many transitions
while(it_tr2.hasNext()){
    Transition transition4 = (Transition) it_tr2.next();
    actionSpaces += "    " + "    " + "\\/" + "/\\" + "current = " +
        transition4.getSource().getName() + "\r\n";
    actionSpaces += "    " + "    " + "    " + "/\\" + "doEntry_" +
        transition4.getTarget().getName() + "\r\n";
    guardSpaces_tmp = generateGuard(transition4);
    if(guardSpaces_tmp == null || guardSpaces_tmp.equals("")){
        /* do nothing */
    } else {

```

```

        // output the guard row by row
        try {
            String guardSpaces_line = "";
            StringReader sr= new StringReader(guardSpaces_tmp);
            BufferedReader br= new BufferedReader(sr);
            actionSpaces += "    " + "    " + "    " + "\\" +
            br.readLine() + "\r\n";
            while((guardSpaces_line = br.readLine()) != null){
                actionSpaces += "    " + "    " + "    " + "    " +
                guardSpaces_line + "\r\n";
            }
        } catch (Exception e) {}
    }
    if (transition4.getSource().getName().equals
    (transition4.getTarget().getName())){
        unchangedSpace_current = "current, ";
    }
    unchangedSpace_tmp = printUnchangedAttr(in,
    unchangedSpace_current);
    unchangedSpace_current = "";
    if(unchangedSpace_tmp == null || unchangedSpace_tmp.equals("")){
        /* do nothing */
    } else {
        actionSpaces += "    " + "    " + "    " + "\\" + "UNCHANGED" + "
        << " + unchangedSpace_tmp + " >>" + "\r\n";
    }
}
}

// other actionsequences, if there are many actionsequence
while(it_actseq.hasNext()){
    ActionSequence as2 = (ActionSequence) it_actseq.next();
    Iterator it_tr3 = as2.getTransitions().iterator();
    while(it_tr3.hasNext()){
        Transition transition5 = (Transition) it_tr3.next();
        actionSpaces += "    " + "    " + "\\" + "\\" + "current = " +
        transition5.getSource().getName() + "\r\n";
        actionSpaces += "    " + "    " + "    " + "\\" + "doEntry_" +
        transition5.getTarget().getName() + "\r\n";
        guardSpaces_tmp = generateGuard(transition5);
        if(guardSpaces_tmp == null || guardSpaces_tmp.equals("")){
            /* do nothing */
        } else {
            // output the guard row by row

```



```

        try {
            String guardSpaces_line = "";
            StringReader sr= new StringReader(guardSpaces_tmp);
            BufferedReader br= new BufferedReader(sr);
            actionSpaces += "    " + " " + " " + " " + "/" + br.readLine() +
                "\r\n";
            while((guardSpaces_line = br.readLine()) != null){
                actionSpaces += "    " + " " + " " + " " + " " + " " +
                    guardSpaces_line + "\r\n";
            }
        } catch (Exception e) {}
    }
    if (transition5.getSource().getName().equals
        (transition5.getTarget().getName())){
        unchangedSpace_current = "current, ";
    }
    unchangedSpace_tmp = printUnchangedAttr(in, unchangedSpace_current);
    unchangedSpace_current = "";
    if(unchangedSpace_tmp == null || unchangedSpace_tmp.equals("")){
        /* do nothing */
    } else {
        actionSpaces += "    " + " " + " " + " " + "/" + "UNCHANGED" + " "
            << " + unchangedSpace_tmp + " >>" + "\r\n";
    }
}

}

/* list attribute */
if (attr != null) {
    actionSpaces += "    " + "/" + in.getAttribute().getName() + "'" + " = "
        + in.getNewAttrValue() + "\r\n";
}

//add the statement enqueue.
String evName = evQueue.dequeue(in);
if(evName.equals("")){
} else {
    actionSpaces += "    " + "/" + "Enqueue(' + evName + ')" + "\r\n";
}
}

}

return null;
}

public Object visitAttrAction(AttrAction in){
    return null;
}

```

```

}

public Object visitTimerAction(TimerAction in){
    return null;
}

// Guard
public Object visitGuard(Guard in) {
    return null;
}

public Object visitSimpleGuard(SimpleGuard in) {
    return null;
}

public Object visitTimeGuard(TimeGuard in) {
    return null;
}

public Object visitAttributeGuard(AttributeGuard in) {
    return null;
}

public Object visitEventGuard(EventGuard in) {
    return null;
}

public Object visitStateGuard(StateGuard in) {
    return null;
}

public Object visitCompositeGuard(CompositeGuard in) {
    return null;
}

public static StringBuffer generateTLA(){

    StringBuffer sb = new StringBuffer();
    sb.append("----- MODULE " + moduleName + " -----");
    sb.append("\r\n\r\n");

    neededModule = neededModule.substring(0, neededModule.length()-1);
    sb.append("EXTENDS " + neededModule);
    sb.append("\r\n\r\n");
}

```

```

/* delete the last character */
if(stateSpaces.endsWith(",")){
    stateSpaces = stateSpaces.substring(0, stateSpaces.length()-1);
}
if(constantSpaces.endsWith(",")){
    constantSpaces = constantSpaces.substring(0, constantSpaces.length()-1);
}
if(varSpaces.endsWith(",")){
    varSpaces = varSpaces.substring(0, varSpaces.length()-1);
}
variableSpaces = variableSpaces.substring(0, variableSpaces.length()-12);

sb.append("CONSTANTS " + constantSpaces);
sb.append("\r\n\r\n");
sb.append("State == " + "{" + stateSpaces + "}");
sb.append("\r\n\r\n");
sb.append("VARIABLE " + variableSpaces );
sb.append("\r\n\r\n");
sb.append("vars == " + "<< " + varSpaces + " >>");
sb.append("\r\n\r\n");
sb.append("Init == " + initSpaces);
sb.append("\r\n\r\n");
sb.append(timeactionSpaces);
sb.append(doEntrySpaces);
sb.append("\r\n\r\n");
sb.append(doExitSpaces);
sb.append("\r\n\r\n");
sb.append(actionSpaces);
sb.append("\r\n\r\n");
sb.append(refseqSpaces);
sb.append("\r\n\r\n");
sb.append("Next == " + nextSpaces);
sb.append("\r\n\r\n");
sb.append("TypeInvariant == " + typeInvariantSpaces);
sb.append("\r\n\r\n");
sb.append("Spec == Init /\ \ [] [Next]_vars");
sb.append("\r\n\r\n");
sb.append("-----");
sb.append("\r\n\r\n");
sb.append("THEOREM Spec => [] (TypeInvariant)");
sb.append("\r\n\r\n");
sb.append("=====");
sb.append("\r\n\r\n");

```

```

        return sb;
    }

    /* find the unchanged attributes */
    public String printUnchangedAttr(RefAction in, String unchangedState){
        String unchangedSpace = "";
        Iterator it = submachine.getPredefinedclasses().iterator();
        try {
            while (it.hasNext()) {
                PredefinedClass elem = (PredefinedClass) it.next();
                if (elem.getClass().getName().equals("statecharts.impl.TimerImpl")){
                    unchangedSpace = elem.getName() + ", ";
                    break;
                }
                unchangedSpace = "";
            }
        }
        catch (Exception e){}

        // add unchanged attributes
        String attrname = "";
        try {
            attrname = in.getAttribute().getName();
        } catch (java.lang.NullPointerException e) {}

        Iterator it_attr = submachine.getAttributes().iterator();
        while(it_attr.hasNext()){
            AttributeDef elem = (AttributeDef) it_attr.next();
            String name = elem.getName();
            if (name.equals("current")){
                // if current, add the unchangedState to unchangedSpace
                unchangedSpace += unchangedState;
            } else if (name.equals(attrname)){
                // do nothing, if a same attribute exists
            } else {
                // add the other attributes to unchangedSpace
                unchangedSpace += name + ", ";
            }
        }

        if(unchangedSpace.endsWith(", ")){
            unchangedSpace = unchangedSpace.substring(0, unchangedSpace.length() - 2);
        }
        return unchangedSpace;
    }

```

```

}

public void addToQueue(Action action){
    try{
        if (evQueue.containsItem(action.getEvent().getName())){
            // do nothing
        } else {
            evQueue.enqueue(action.getEvent());
        }
    } catch(Exception e){}
}

public String generateGuard(Transition tran){
    String guard_tmp = "";
    try {
        if (tran.getGuard().getClass().getName().equals("statechartsGuard.impl
        .TimeGuardImpl")){
            TimeGuardImpl elem_tg = (TimeGuardImpl) tran.getGuard();
            if (elem_tg.getMAX() != null) {
                guard_tmp += elem_tg.getTimer().getName() + " < " + elem_tg.getMAX();
            } else if (elem_tg.getMIN() != null) {
                guard_tmp += elem_tg.getTimer().getName() + " > " + elem_tg.getMIN();
            } else if (elem_tg.getGdExpression() != null) {
                guard_tmp += elem_tg.getTimer().getName() + " " + elem_tg
                .getGdExpression();
            }
        }
        else if (tran.getGuard().getClass().getName().equals("statechartsGuard.impl
        .AttributeGuardImpl")){
            AttributeGuardImpl elem_ag = (AttributeGuardImpl) tran.getGuard();
            guard_tmp += elem_ag.getAttribute().getName();
        } else if (tran.getGuard().getClass().getName().equals("statechartsGuard.impl
        .EventGuardImpl")){
            EventGuardImpl elem_eg = (EventGuardImpl) tran.getGuard();
            guard_tmp += elem_eg.getEvent().getName();
        } else if (tran.getGuard().getClass().getName().equals("statechartsGuard.impl
        .StateGuardImpl")){
            StateGuardImpl elem_sg = (StateGuardImpl) tran.getGuard();
            guard_tmp += elem_sg.getState().getName();
        } else if (tran.getGuard().getClass().getName().equals("statechartsGuard.impl
        .CompositeGuardImpl")){
            CompositeGuardImpl cg = (CompositeGuardImpl) tran.getGuard();
            guard_tmp += generateCompositeGuard(cg);
        }
    }
    catch (Exception e) {}
}

```

```

        return guard_tmp;
    }

    public String generateCompositeGuard(CompositeGuardImpl guard){
        String guard_comp = "";
        if(guard.getLogicRelation().toString().equals("DISJUNCTION")){
            Iterator it = guard.getSubguards().iterator();
            while (it.hasNext()) {
                Guard elem = (Guard) it.next();
                if (elem.getClass().getName().equals("statechartsGuard.impl.TimeGuardImpl")){
                    TimeGuardImpl elem_tg = (TimeGuardImpl) elem;
                    if (!elem_tg.getMAX().equals("")) {
                        guard_comp += "\\/" + elem_tg.getTimer().getName() + " < " +
                            elem_tg.getMAX() + "\\r\\n";
                    } else if (!elem_tg.getMIN().equals("")) {
                        guard_comp += "\\/" + elem_tg.getTimer().getName() + " > " +
                            elem_tg.getMIN() + "\\r\\n";
                    } else if (!elem_tg.getGdExpression().equals("")) {
                        guard_comp += "\\/" + elem_tg.getTimer().getName() + " " +
                            elem_tg.getGdExpression() + "\\r\\n";
                    }
                } else if (elem.getClass().getName().equals("statechartsGuard.impl
                .AttributeGuardImpl")){
                    AttributeGuardImpl elem_ag = (AttributeGuardImpl) elem;
                    guard_comp += "\\/" + elem_ag.getAttribute().getName() + "\\r\\n";
                } else if (elem.getClass().getName().equals("statechartsGuard.impl
                .EventGuardImpl")){
                    EventGuardImpl elem_eg = (EventGuardImpl) elem;
                    guard_comp += "\\/" + elem_eg.getEvent().getName() + "\\r\\n";
                } else if (elem.getClass().getName().equals("statechartsGuard.impl
                .StateGuardImpl")){
                    StateGuardImpl elem_sg = (StateGuardImpl) elem;
                    guard_comp += "\\/" + elem_sg.getState().getName() + "\\r\\n";
                } else if (elem.getClass().getName().equals("statechartsGuard.impl
                .CompositeGuardImpl")){
                    CompositeGuardImpl cg = (CompositeGuardImpl)elem;
                    String guard_cg = generateCompositeGuard(cg);
                    try {
                        String guardSpaces_line = "";
                        StringReader sr= new StringReader(guard_cg); // wrap String
                        BufferedReader br= new BufferedReader(sr); // wrap StringReader
                        guard_comp += "\\/" + br.readLine() + "\\r\\n";
                        while((guardSpaces_line = br.readLine()) != null){
                            guard_comp += " " + guardSpaces_line + "\\r\\n";
                        }
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
    }
}

```

```

        }
    } catch (Exception e) {}
}

}

} else if (guard.getLogicRelation().toString().equals("CONJUNCTION")) {
    Iterator it = guard.getSubguards().iterator();
    while (it.hasNext()) {
        Guard elem = (Guard) it.next();
        if (elem.getClass().getName().equals("statechartsGuard.impl.TimeGuardImpl")) {
            TimeGuardImpl elem_tg = (TimeGuardImpl) elem;
            if (!elem_tg.getMAX().equals("")) {
                guard_comp += "/" + elem_tg.getTimer().getName() + " < " +
                    elem_tg.getMAX() + "\r\n";
            } else if (!elem_tg.getMIN().equals("")) {
                guard_comp += "/" + elem_tg.getTimer().getName() + " > " +
                    elem_tg.getMIN() + "\r\n";
            } else if (!elem_tg.getGdExpression().equals("")) {
                guard_comp += "/" + elem_tg.getTimer().getName() + " + " +
                    elem_tg.getGdExpression() + "\r\n";
            }
        } else if (elem.getClass().getName().equals("statechartsGuard.impl.
        AttributeGuardImpl")) {
            AttributeGuardImpl elem_ag = (AttributeGuardImpl) elem;
            guard_comp += "/" + elem_ag.getAttribute().getName() + "\r\n";
        } else if (elem.getClass().getName().equals("statechartsGuard.impl.
        EventGuardImpl")) {
            EventGuardImpl elem_eg = (EventGuardImpl) elem;
            guard_comp += "/" + elem_eg.getEvent().getName() + "\r\n";
        } else if (elem.getClass().getName().equals("statechartsGuard.impl.
        StateGuardImpl")) {
            StateGuardImpl elem_sg = (StateGuardImpl) elem;
            guard_comp += "/" + elem_sg.getState().getName() + "\r\n";
        } else if (elem.getClass().getName().equals("statechartsGuard.impl.
        CompositeGuardImpl")) {
            CompositeGuardImpl cg = (CompositeGuardImpl) elem;
            String guard_cg = generateCompositeGuard(cg);
            try {
                String guardSpaces_line = "";
                StringReader sr= new StringReader(guard_cg); // wrap String
                BufferedReader br= new BufferedReader(sr); // wrap StringReader
                guard_comp += "/" + br.readLine() + "\r\n";
                while((guardSpaces_line = br.readLine()) != null) {
                    guard_comp += " " + guardSpaces_line + "\r\n";
                }
            }
        }
    }
}

```

```
        } catch (Exception e) {}  
    }  
}  
}  
return guard_comp;  
}  
  
}
```


Appendix 4

```
----- MODULE micro -----

EXTENDS TLC, Integers, Sequences

CONSTANTS ReadyToCook, DoorOpen, CookingComplete, Cooking, CookingInterrupted, CookingExtended

State == {ReadyToCook, DoorOpen, CookingComplete, Cooking, CookingInterrupted, CookingExtended}

VARIABLE tube,
           current,
           light,
           door,
           microwaveOvenTimer

vars == << tube, current, light, door, microwaveOvenTimer >>

Init == /\tube = FALSE
        /\current = ReadyToCook
        /\light = FALSE
        /\door = FALSE
        /\microwaveOvenTimer = -1

LOCAL clearTimer ==
    microwaveOvenTimer' = -1

LOCAL setTimerForOneMinute ==
    microwaveOvenTimer' = 60

LOCAL addOneMinuteToTimer ==
    microwaveOvenTimer' = microwaveOvenTimer + 60

LOCAL doEntry_ReadyToCook ==
    /\current' = ReadyToCook
    /\light' = FALSE

LOCAL doEntry_DoorOpen ==
    /\current' = DoorOpen
    /\light' = TRUE

LOCAL doEntry_CookingComplete ==
    /\current' = CookingComplete
```

```

/\light' = FALSE
/\tube' = FALSE
/\clearTimer

LOCAL doEntry_Cooking ==
    /\current' = Cooking
    /\light' = TRUE
    /\tube' = TRUE
    /\setTimerForOneMinute

LOCAL doEntry_CookingInterrupted ==
    /\current' = CookingInterrupted
    /\light' = FALSE
    /\tube' = FALSE
    /\clearTimer

LOCAL doEntry_CookingExtended ==
    /\current' = CookingExtended
    /\addOneMinuteToTimer

openDoor ==
    /\//\current = Cooking
        /\doEntry_CookingInterrupted
        /\UNCHANGED << microwaveOvenTimer, tube, light >>
    \//\current = CookingExtended
        /\doEntry_CookingInterrupted
        /\UNCHANGED << microwaveOvenTimer, tube, light >>
    \//\current = ReadyToCook
        /\doEntry_DoorOpen
        /\UNCHANGED << microwaveOvenTimer, tube, light >>
    \//\current = CookingComplete
        /\doEntry_DoorOpen
        /\UNCHANGED << microwaveOvenTimer, tube, light >>
    /\door' = TRUE
    /\Enqueue('od')

closeDoor ==
    /\//\current = CookingInterrupted
        /\doEntry_ReadyToCook
        /\UNCHANGED << microwaveOvenTimer, tube, light >>
    \//\current = DoorOpen
        /\doEntry_ReadyToCook
        /\UNCHANGED << microwaveOvenTimer, tube, light >>
    /\door' = FALSE

```

```

/\Enqueue('cd')

pressButton ==
  /\ \/\current = ReadyToCook
    /\doEntry_Cooking
      /\UNCHANGED << microwaveOvenTimer, tube, light, door >>
  /\ \current = CookingComplete
    /\doEntry_Cooking
      /\UNCHANGED << microwaveOvenTimer, tube, light, door >>
  /\ \current = Cooking
    /\doEntry_CookingExtended
      /\UNCHANGED << microwaveOvenTimer, tube, light, door >>
  /\ \current = CookingExtended
    /\doEntry_CookingExtended
      /\UNCHANGED << microwaveOvenTimer, tube, current, light, door >>
  /\Enqueue('pb')

Next == \openDoor
        \closeDoor
        \pressButton

TypeInvariant == /\tube \in BOOLEAN
                  /\current \in State
                  /\light \in BOOLEAN
                  /\door \in BOOLEAN
                  /\microwaveOvenTimer \in -1..60

Spec == Init /\ [] [Next]_vars
-----
THEOREM Spec => [] (TypeInvariant)
=====

```

Reference

- [1] The TLA Web Page:
<http://research.microsoft.com/users/lamport/tla/tla.html>
- [2] Lamport, L., Specifying Systems: *The TLA+ Language and Tools for Hardware and Software Engineers*, Addison-Wesley, 2003.
- [3] Yuan Yu, Panagiotis Manolios, Leslie Lamport: *Model Checking TLA+ Specifications*, Compaq Systems Research Center,
- [4] Frank B., David S., Ed M., Raymond E., Timothy J. G.: *Eclipse Modeling Framework: A Developer's Guide*, Addison Wesley, 2003
- [5] Thomas Santen Dirk Seifert: *Executing UML State Machines*, TU-Berlin, 2003
- [6] Stephen J. Mellor, Marc J. Balcer: *Executable UML: A Foundation for Model-Driven Architecture*, Addison Wesley, 2002
- [7] Carol Lisa Freinkel: *An Approach to Combining UML and TLA+ in Software Specification*, 2003
- [8] Leslie Lamport: *Specifying Systems*, 2002
- [9] Zs. Pap, I. Majzik¹, A. Pataricza and A. Szegi: *Completeness and Consistency Analysis of UML Statechart Specifications*