VISUALIZATION ANIMATION FOR REAL-TIME REACTIVE SYSTEMS SIMULATION

Mubarak Sami Mohammad

A thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science

Concordia University

Montréal, Québec, Canada

September 2004

Your file Votre référence
ISBN: 0-612-94749-1
Our file Notre référence
ISBN: 0-612-94749-1

Abstract

Visualization Animation for Real-Time Reactive Systems Simulation

Mubarak Sami Mohammad

Simulation results form basis for the validation and verification of Real-Time Reactive systems design. The high complexity of such systems brings sophisticated simulation results that are difficult to understand. The way of presenting those results will have a great effect on the usability of the simulation tools.

This thesis addresses the design and implementation of a visualization animation tool that will improve the usability of TROMLAB Framework, a rigorous real-time reactive systems development environment being built in the Department of Computer Science, Concordia University. The new tool uses graphical representation and animation to present the simulation results produced by TROMLAB tools. Also,The thesis introduces new guidelines for improving the understandability of real-time reactive simulation results. Furthermore, the thesis presents a solution for the Steam Boiler Controller case study, a benchmark problem for Real-Time Reactive studies, for the first time using TROMLAB formalism.

To Allah.

Acknowledgments

I would like to express my profound thanks to my supervisors, D. V.S. Alagar and Dr. Olga Ormandjieva, for their help and guidance through the stages of work on this thesis. Their technical and financial support motivated me to work hard and produce high quality results.

On a personal level, I would like to thank all my family members for their endless support and prayers; specially, my brother Mohsen.

Contents

List of Figures x

List of Tables xii

1 Introduction 1

1.1 TROMLAB Related Work 3

1.2 Major Contributions 3

1.3 The Scope of the Thesis 4

2 TROMLAB Environment - a brief review 5

2.1 TROM Formalism 5

2.1.1 The First Tier: Data Abstraction Tier 6

2.1.2 The Second Tier: TROM Tier 7

2.1.3 The Third Tier: Subsystem Tier 9

2.2 TROMLAB Components 10

2.2.1 The Interpreter 10

2.2.2 The Simulator 10

3 TROMLAB Understandability Guidelines 14

3.1 Introduction 14

3.2 Understandability 15

3.3 Understandability Guidelines for TROMLAB Tools 16

3.4 Understandability Evaluation of the Simulator's User Interface 18

3.5 A New Way for Representing Simulation Results 20
3.6 General Principles for Usable Visualizations 21

4 Visualization Animation Tool's Architecture **23**
4.1 Architecture 23
4.1.1 Introduction 23
4.1.2 Purpose and Context 23
4.1.3 Tool Interface 25
4.1.4 Non-Functional Requirements 25
4.2 Architectural Overview 26
4.2.1 Pipes and Filters Architecture 26
4.2.2 Rationale Behind Selection 27
4.2.3 Architecture Diagram 28
4.3 Components 29
4.3.1 XML Producer 30
4.3.2 XML File 30
4.3.3 Parser Component 32
4.3.4 Analyzer 33
4.3.5 Visualizer Component 36
4.3.6 Animator Component 39
4.4 Development Platform 42
4.5 Dynamic Behavior : Scenario 42

5 Steam Boiler Case Study **44**
5.1 Introduction 44
5.2 Problem Description 45
5.2.1 Informal Problem Definition 45
5.2.2 Characteristics of system components 46
5.2.3 Assumptions 48
5.2.4 Safety Property 48

5.2.5 Liveness Property 49
5.3 Formal Solution 49
5.3.1 Class Diagram for Steam Boiler 49
5.3.2 Level Measuring Class 51
5.3.3 Controller Class 53
5.3.4 Pump Class 56
5.3.5 Valve Class 58
5.3.6 Timer Class 60
5.3.7 SubSystem Configuration Specification (SCS) 61
5.3.8 Sample Simulation Event List 63

6 Visualization Animation Tool **65**
6.1 VAT 65
6.1.1 The Start Page 65
6.1.2 The Simulation Details 66
6.1.3 The TROM Objects Section 67
6.1.4 The Current Status Section 68

7 Testing **72**
7.1 Testing Case Studies 72
7.2 Railroad Crossing Case Study 72
7.3 Robotic Assembly Case Study 74
7.4 Traffic Control Case Study 75

8 Related Work **77**
8.1 CD++ Real-Time Simulation 77
8.2 Simulation of Steam Boiler 78
8.3 Rational Rose RealTime 79

9 Conclusion **83**
9.1 Future Work 84

9.1.1 Simulator 84
9.1.2 VAT 84

Bibliography **85**

Appendix A **87**

List of Figures

1 The Architecture of TROMLAB 2
2 The Three Tiers of TROM Formalism 6
3 Architecture of the Interpreter 11
4 Architecture of the Simulator 12
5 Snapshot of the Simulator's simulation result 19
6 4 + 1 View Model 24
7 VAT Context Diagram 25
8 Pipe and Filter Architecture 27
9 Components Diagram for VAT Architecture 28
10 Analyzer Component class diagram 34
11 Visualizer Component class diagram 37
12 Animator Component class diagram 40
13 VAT Sequence Diagram 43
14 Steam Boiler Controller 45
15 Steam Boiler Controller class diagram 50
16 Level Measuring class diagram 51
17 Level Measuring state chart diagram 52
18 Controller class diagram 53
19 Controller state chart diagram 55
20 Pump class diagram 57
21 Pump state chart diagram 57
22 Valve class diagram 58

23 Valve state chart diagram 59
24 Timer class diagram 60
25 Timer state chart diagram 60
26 Steam Boiler collaboration diagram 62
27 Sequence diagram 64
28 VAT start page 66
29 Simulation Visualization Animation 67
30 TROM object state chart diagram 68
31 TROM object state chart diagram 69
32 TROM object state chart diagram 70
33 Railroad Crossing Visualization Animation 73
34 Robotic Assembly Visualization Animation 74
35 Traffic Control Visualization Animation 75
36 Simulation Output 78
37 Steam Boiler Simulator 79

List of Tables

1 Tool Interface 25
2 XML Producer Component Description 30
3 Parser Component Description 32
4 Analyzer Component Description 33
5 Visualizer Component Description 36
6 Animator Component Description 39

Chapter 1

Introduction

Reactive systems are the type of Real Time systems that maintain an ongoing continues interaction with their environment through stimulus and responses. This type of interaction is governed by timing constraints. Examples of such systems can be found in areas like: Transportation such as Train gate controllers, Workshop automation such as in Robotics, and Strategic defense systems such as Nuclear power plants. There are several factors that affect the complexity of such systems like size, timing constraints, criticality, and heterogeneity.

The major issue in the development of Real Time Reactive Systems is to produce a design with correct functional and temporal properties. In order to achieve success in this complex domain, the design process should be supported by a formal basis for specification, analysis, and refinement. A formal object oriented model (TROM) was invented to assist the formalization of Real-Time Reactive Systems specifications. Along with the model, a framework for practicing the language and methods of TROM and integrating formal methods with several phases of the development life cycle was created. This framework is called TROMLAB, and it consists of many components that take the system design from UML Rose model, translate it into TROM formalism, compile the formalism, analyze it, and finally produce simulation results that show the expected behavior of the system based on its design. Figure 1

shows the Architecture of TROMLAB [Liu03]. The results are presented in a tabular form and can be interpreted only by simulation experts.

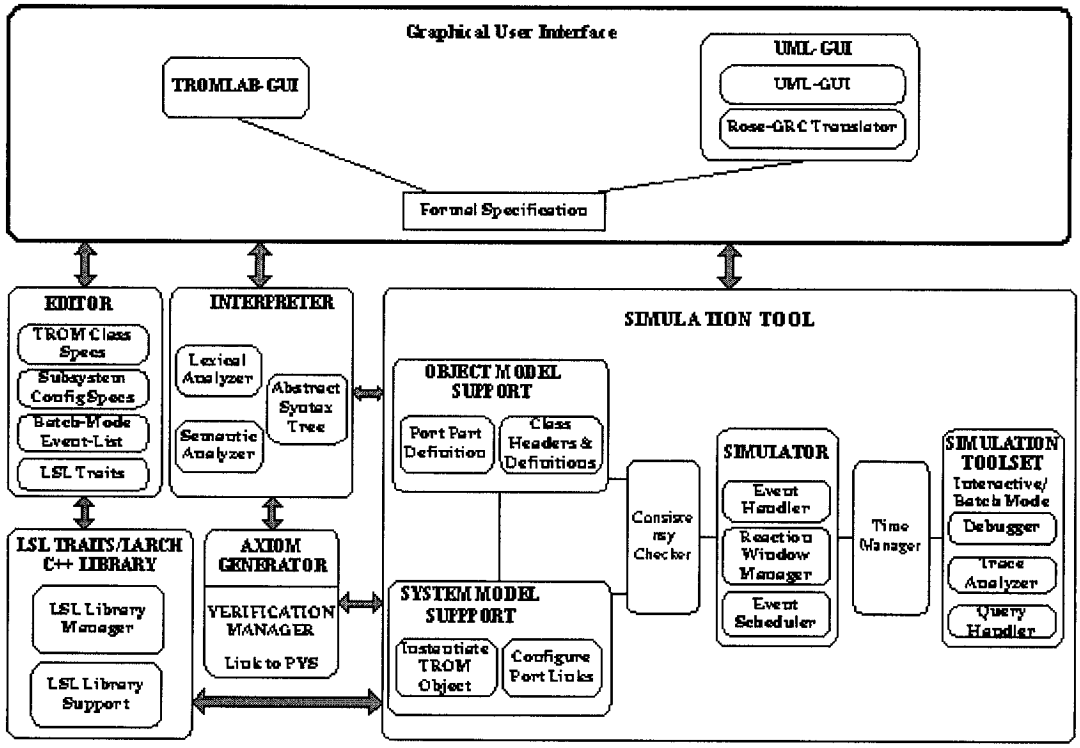

Figure 1: The Architecture of TROMLAB

No matter how excellent and accurate the simulation tools are, if they have poor usability, they will face difficulties in exposing their powerful functionalities. In the current developments of TROMLAB Tools, only those who were behind the development of the components are able to understand the results. Hence, there is a great need for improving the usability and making the results perceivable by other users from the same domain of research.

The purpose of this thesis is to improve the usability of TROMLAB tools by building a visualization animation tool that will take the simulation results as input and visualize and animate the expected behavior of the system. The user will be able to

see how the system objects will be created and how they interact with each other in an understandable easy to grasp visual model.

1.1 TROMLAB Related Work

This thesis is related to the work done by [Mut96], Simulator for executing the formal design specifications . That work had been improved by [Hai99], who added the reasoning system to the simulator. Then the work had been again improved by [Bha99], who improved the Graphical User Interface for TROMLAB environment. And finally, the latest version was updated by [Liu03], who improved the simulator and introduced parameterized events into it.

1.2 Major Contributions

This thesis offers the following contributions:

- Designs and Implements a Visualization Animation Tool for TROM simulation results.
- Solves a simplified specification of the Steam Boiler Controller case study problem using the TROM formalism and TROMLAB environment.
- Defines understandability guidelines for TROMLAB tools.
- Introduces portability capability to the simulation results by exporting simulation results to XML.

1.3 The Scope of the Thesis

Chapter 2 briefly reviews the TROMLAB environment including both TROM formalism and TROMLAB components. New understandability guidelines for improving the understandability of TROMLAB tools are introduced in Chapter 3 along with an assessment to the current usability of the simulation results. Chapter 4 introduces the architectural design of the new Visualization Animation Tool(VAT). A solution for simplified specifications of the Steam Boiler Controller case study is provided by the TROM formalism and compiled by the TROMLAB components in Chapter 5. The developed VAT is presented in Chapter 6. As proof of having achieved the goals of this thesis, three cases studies along with the Steam Boiler are visualized using VAT and the results are presented in Chapter 7. After that briefs about similar related works are presented in Chapter 8. Finally, chapter 9 offers the conclusion of the thesis and the research directions.

Chapter 2

TROMLAB Environment - a brief review

This chapter reviews the TROMLAB environment by giving a background about the work done by the previous research studies in TROMLAB.

2.1 TROM Formalism

The TROM formalism is a layered model that consists of three formal tiers shown in Figure 2 [AAM96]. In this model, each layer interacts only with its immediate upper layer. This architectural pattern creates independence between the three tiers resulting in the benefits of having modularity, reuse, encapsulation, and hierarchical decomposition. The three-tier structure describes the system configuration, reactive classes, and relative Abstract Data Types. The upper-most tier is the subsystem configuration specification. It specifies objects definitions, their collaboration, and the port links that controls the communication between objects. The middle tier is the TROM class. TROM class is a hierarchical finite state machine extended with ports, attributes, logical assertions on the attributes, and time constraints. The third tier is the Larch Shared Language (LSL) trait that represents Abstract Data Type used in the TROM classes. Figure 2 shows the three tiers [AAM96].

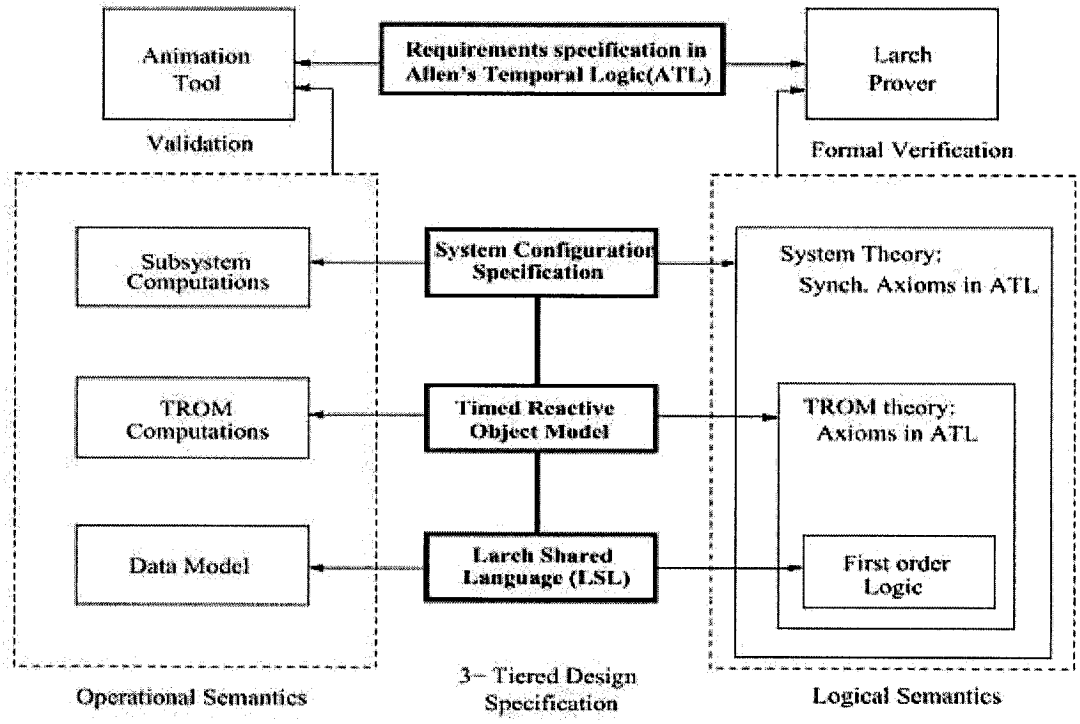

Figure 2: The Three Tiers of TROM Formalism

2.1.1 The First Tier: Data Abstraction Tier

The Data Abstract Tier encapsulates the Abstract Data Types. This tier uses the LSL trait, The Larch Shared Language [GH93], to define all data types used in the middle tier. The following example shows the Integer LSL trait :

```
Trait: Integr(I)
    Includes: Boolean
    Introduce:
      succ   : I    -> I;
      pred   : I    -> I;
```

```
    plus    : I, I -> I;
    subt    : I, I -> I;
    mult    : I, I -> I;
    quot    : I, I -> I;
    remn    : I, I -> I;
    imin    : I, I -> I;
    imax    : I, I -> I;
end
```

2.1.2 The Second Tier: TROM Tier

A TROM is a Generic Reactive Class (GRC), an extended finite state machine with port types, attributes, hierarchical states, events triggering transitions and future events constrained by strict time intervals [AAM96]. Brief summary of TROM elements as stated in [Hai99] is:

- A set of events partitioned in three sets: input, output, and internal events.
- A set of states: A state can have substates.
- A set of typed attributes. The attributes can be one of the following:
  - primitive data types,
  - abstract data types,
  - port reference type.
- An attribute function which defines the mapping of the set of attributes to the set of states.
- A set of transition specifications. Each transition specification describes the computational step associated with the occurrence of an event. The transition specification has three assertions: a pre- and post-condition, as in Hoare logic, and the port-condition specifying the port at which the event can occur.

- A set of time-constraints. Each time constraint specifies the reaction associated with a transition. A reaction can fire an output or an internal event within a defined time period, and is associated with a set of disabling states. An enabled reaction is disabled when an object enters any of the disabling states of the reaction.

The language for describing a generic reactive class, derived directly from the formal definition, is shown in the following example:

```
Class Pump [@P]
Events: OpenPump?@P, ClosePump?@P, open
States: *closed, toopen, opened
Attributes:
Traits:
Attribute-Function: closed -> {};toopen -> {};
opened -> {};
Parameter-Specifications:

Transition-Specifications:
     R1: <closed,toopen>; OpenPump[](true); true => true;
     R2: <closed,closed>; ClosePump[](true); true => true;
     R3: <toopen,opened>; open[](true); true => true;
     R4: <opened,closed>; ClosePump[](true); true => true;
     R5: <opened,opened>; OpenPump[](true); true => true;
Time-Constraints:
     TCvar1: R1, open, (0, 5), {};
end
```

2.1.3 The Third Tier: Subsystem Tier

The subsystem configuration is specified in the subsystem tier. The configuration specification uses objects instantiated from classes defined in the TROM tier. An object is instantiated from a class by defining a finite number of ports for each port type in the class specification, and by initializing the attributes included in the class. Each instantiated object will carry its own set of attributes. A port link is an abstraction of a communication medium between two objects. A port link is established between a port of one object and a compatible port in another object. Objects communicate by exchanging messages (external events) through the port links [AAM96]. An example for the syntax of a subsystem specification is :

```
SCS SteamBoiler
     Includes:
     Instantiate:
          pump1::Pump[@P:1];
          controller1::Controller[@CP:1, @CV:1, @CL:1, @CT:1];
          levelmeasuring1::LevelMeasuring[@L:1];
          timer1::Timer[@TC:1];
          valve1::Valve[@V:1];
     Configure:
          controller1.@CP1:@CP <-> pump1.@P1:@P;
          valve1.@V1:@V <-> controller1.@CV1:@CV;
          levelmeasuring1.@L1:@L <-> controller1.@CL1:@CL;
          controller1.@CT1:@CT <-> timer1.@TC1:@TC;
end
```

The Includes section lists imported subsystems. A reactive object is created in the Instantiate section, with parametric substitutions to cardinality of ports for each

port type. The Configure section defines a configuration obtained by composing objects specified in the Instantiate section and in the subsystem specifications imported through the Include section. The composition operator $\leftrightarrow$ sets up communication links between compatible ports of interacting objects. Two ports are compatible if the set of input message sequences at one port is a subset of the output message sequences at the other port.

2.2 TROMLAB Components

In this section, the functionalities of two TROMLAB components, the Interpreter and the Simulator, are briefly reviewed.

2.2.1 The Interpreter

The interpreter was the first tool to be implemented in TROMLAB by [Tao96]. It takes as input the specifications of the three TROM tiers in one source file and performs syntactic and semantic analysis and produce an internal representation of the well-formed TROM formal specification of the corresponding reactive system. The advantage of the three-tiered design was not realized in this implementation. The work had been improved by Haidar [Hai99] and Sriniva [Sri99] to include incremental and independent compilation of specifications and enhanced error reporting. Figure 3 shows the architecture of the Interpreter [Liu03].

2.2.2 The Simulator

The simulator is an animation tool that was implemented by Muthiayen [Mut96]. It worked with the first interpreter. Then the simulator was reengineered, designed and implemented, by Haidar [Hai99] by adding a reasoning capabilities. Figure 4 shows the simulator architecture [Liu03]. The simulator has two working modes:

- Debugger mode: In this mode, the user can perform debugging tasks and query

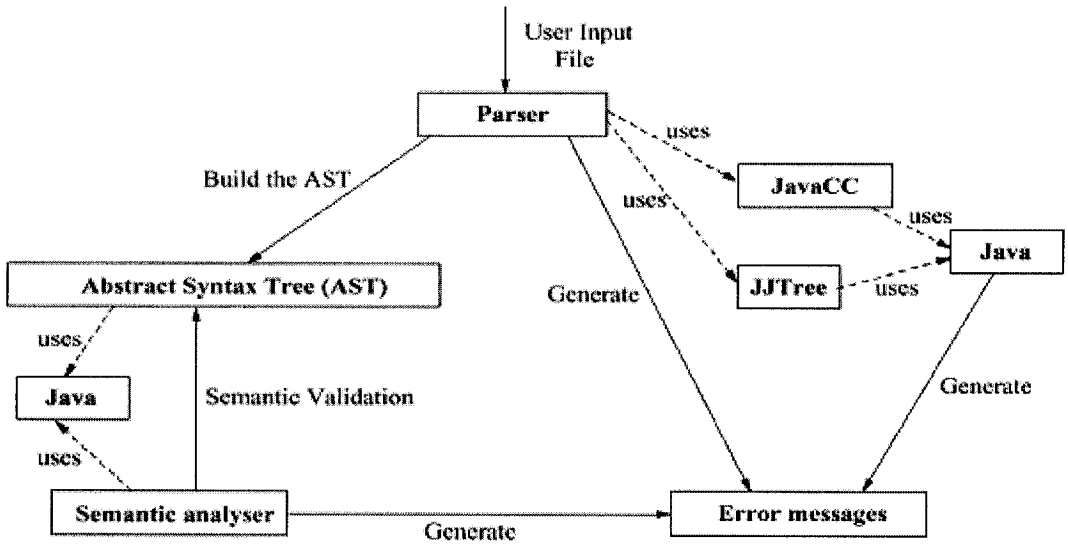

Figure 3: Architecture of the Interpreter

the system after the end of each event.

- Normal mode: In this mode the simulator will perform complete simulation according to the given scenario.

The simulation tool consists of the following components as described in [Liu03]:

- Simulator :It consists of an event handler, a reaction window manager, and an event scheduler.
  - The event handler is responsible for handling the events that are due to occur and detects the transition to be trigger by the event.
  - The reaction window manager is responsible for activating the computational step to handle the transition causing events.
  - The event scheduler causes an enabled event to occur at a random time within the corresponding reaction window. It schedules output events through the least recently used port using a round robin algorithm.
- Consistency checker: It detects deadlock configurations in order to maintain the continuous flow of interactions.

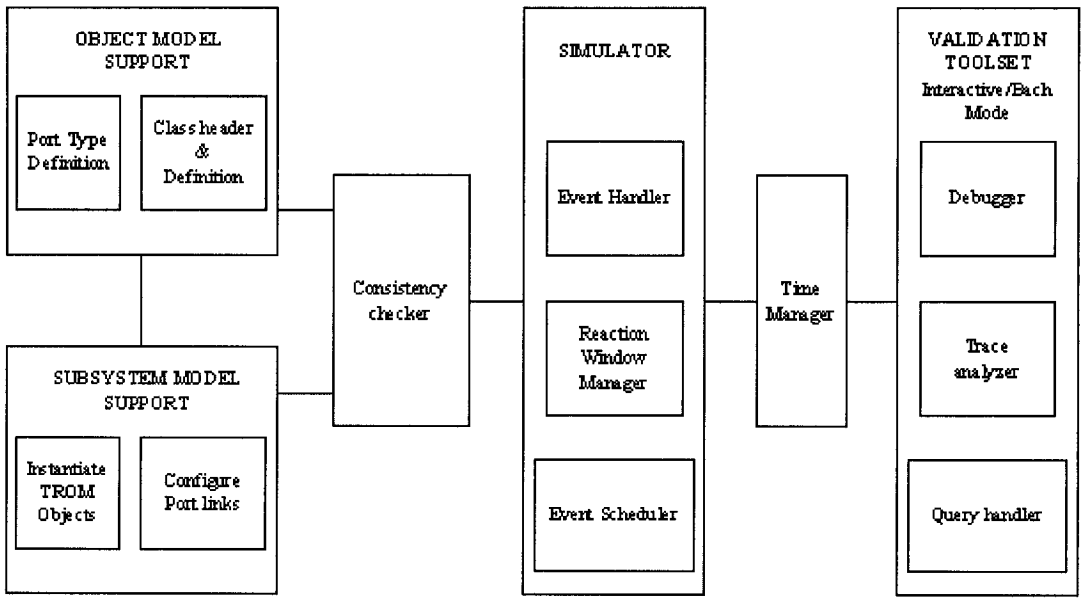

Figure 4: Architecture of the Simulator

- Validation tool: It consists of a debugger, a trace analyzer, and a query handler.
  - The debugger supports system experimentation by allowing the user to examine the evolution of the status of the system throughout the simulation process. It also supports interactive injection of simulation event, and simulation rollback to a specific point in time.
  - The trace analyzer includes facilities for the analysis of the simulation scenario. It gives feedback on the evolution of the status of the objects in the system, and the outcome of the simulation event.
  - The query handler allows examining the data in the AST for the TROM class to which the object belongs, and supporting analysis of the static components during simulation.
- Object model support: It supports the specification of the TROM classes and the evaluation of the logical assertions included in the transition specifications.
- Subsystem model support: It creates subsystems by instantiating the included

subsystems with its objects and port links.

- Time manager: It maintains the simulation clock and updates it regularly. It allows setting the pace of the clock to suit the needs of analysis of simulation scenarios. It also allows freezing the clock while analyzing the consequences of a computation.

The brief overview that was presented in this chapter introduced the TROM Formalism and the TROMLAB tools that allow developers to practice the formalism. The tools take the formalism as input and do syntactic and semantic analysis, compilation and simulation operations, and finally produce a simulation result that shows the expected behavior of the formalized system design. Those simulation results are important for validating the correctness and completeness of the corresponding system design. The next chapter inspects the understandability of the simulation results.

Chapter 3

TROMLAB Understandability Guidelines

3.1 Introduction

The process of implementing the TROMLAB framework tools took several stages of iterative development, incremental design, validation, and formal verification of design models. It started with implementing a compiler for formal design specification [Tao96] and [Sri99]. The concentration was on having a working product that can compile and test formal specifications according to TROM formalism. At that time, functionality had higher priority than usability of tools. Gradually over time, more tools were added for simulating, debugging and querying design specifications to assist the analysis of the design and the requirements [Mut96] and [Hai99]. Because of the complexity included in such tools, graphical user interface was recognized as an important requirement for the usability of the entire system. Therefore, a graphical user interface was implemented to facilitate the interaction with the different components of TROMLAB [Sri99]. The research continued to improve the functionality and produce accurate simulation results that can verify the correctness of the formalized design. However, those simulation results remain presented in a way that is only understandable by TROMLAB experts; specifically, only the ones who implemented

the tools were able to understand the output.

Because of the increase in TROMLAB research studies and because it has become a mature project and got recognized by other research groups in the same field of study, usability and portability of simulation results have become important requirements having the same criticality as functional requirements.

This chapter introduces understandability as an important factor for satisfying usability of simulation results. It defines understandability, presents new guidelines for the TROMLAB Tools understandability, gives understandability evaluation of the current Simulator results, and introduces a new way to present simulation results based on guidelines and principles set by experts in usability. Finally, it presents general principles for usable visualizations.

3.2 Understandability

Usability research studies and standard models have recognized understandability as one of the most important factors that affect the usability and hence the quality of any software product. For example, ISO/IEC 9126-1 defines usability in terms of understandability, learnability, operability and attractiveness. According to [Cio91], one of the most important attributes of understandability is Comprehension. It is defined as the degree or ease to which the user of a particular software system grasps the information mentally.

Testing and validating the correctness of any system's TROM formalism relies mainly on the simulation results. Therefore, the correct and easy comprehension of those simulation results is very important for any TROMLAB practitioner. Consequently, improving the appeal and mental comprehension of simulation results will have a great impact on the understandability and productivity of TROMLAB users.

For software specification methods, [Wil94] details three indicators of understandability: notation, organization, and level of abstraction. Notation is the way of presenting information to the user. It should be straightforward; therefore, the information presentation should use different types of notations in order to most naturally convey different types of information. Organization is the way of structuring those notations in the user interface design. Information should be well organized so that it can be found easily in the provided interface. Abstraction is the way of presenting the necessary and important information pieces. It eliminates any irrelevant details that might distract user's focus.

Applying those understandability indicators to the simulation results will improve the usability of TROMLAB tools which will improve, consequently, the learnability and user satisfaction that will result in the increase of the utilization of those resources and, finally, the increase of research in this area.

3.3 Understandability Guidelines for TROMLAB Tools

The Object Model of TROM consists of the following important types:

- TROM object: Generic Reactive Class that forms the basic abstract structure of a reactive system [AM98].
- State: a situation during the life of an object.
- Events: messages that cause transition of objects from one state to another. Those events can be input, output, or internal. Input and output events are shared events because they cause the two objects to change their state simultaneously.

- Time: the time constrained for the objects response to a stimulus.

TROMLAB Simulation results contain combination of those types that explain the expected behavior of the system when it runs under specified simulation event list. Therefore, the way of presenting those types will have a great impact on the understandability of the simulation results.

This thesis suggests the following guidelines to achieve the goal of having an understandable simulation results:

1. Every TROM Object and State should be presented in a way so that it is clearly identified and perceived by the user. Different types of graphical notations should be used to differentiate the presentation of a TROM Objects or a State. The user should be able on the glance to tell where are the TROM Objects and the States.

2. The relationship between TROM Objects and their corresponding states should be made clear. Organization structuring can be used to group the related states together under the frame of the corresponding TROM Object.

3. States of a particular TROM Object should be presented in the same sequence of execution that happened during the simulation process. Source and destination states should be clearly grouped together in a way that make the transition clear to the user.

4. The Type of event should be identified. The user should know whether the event is internal or shared. Internal events will cause the TROM object to change state internally from one to another. Shared events, on the other hand, not only will make transition internally but also will affect another TROM Object and make it change state accordingly. Also, shared events could be input or output. In order to achieve this, internal and shared events could be presented with different notations, or labels should be added to the presented transition specification to clarify the type of event.

5. The causing event should be mentioned. For each event, if there is a causing event that synthesized the current event to occur, that causing event should be clarified in the presented transition specification.

6. Time should be always presented. During the simulation method, the user should be able to see the change of system states and firing events along with the current time and constrained time for those events and state transition.

7. System's current status and history should be always available to the user. At any time, the user should know where in the simulation process he is now. Also, he should be able to see the previous states for each object to aid in validating the sequence of execution for that object.

8. Help and support should be available to the user. The user should have help facilities to aid grasping the meaning of the different notations and grouping presented in the simulation results. Also, to help him understand the current process of simulation.

3.4 Understandability Evaluation of the Simulator's User Interface

In this section, the current simulation results of TROMLAB simulator are examined to evaluate the understandability of the presented simulation result. Figure 5 shows an example simulation result displayed in a graphical user interface.
When evaluating the simulation results that are presented in the Simulator's user interface, the following understandability problems are found:

1. Where are the TROM objects in the interface? The expert users only know that the listed names in the header of the table represent the TROM objects.

2. TROM objects that have long names are misrepresented in the header row. The names appear incomplete as some characters are missing.

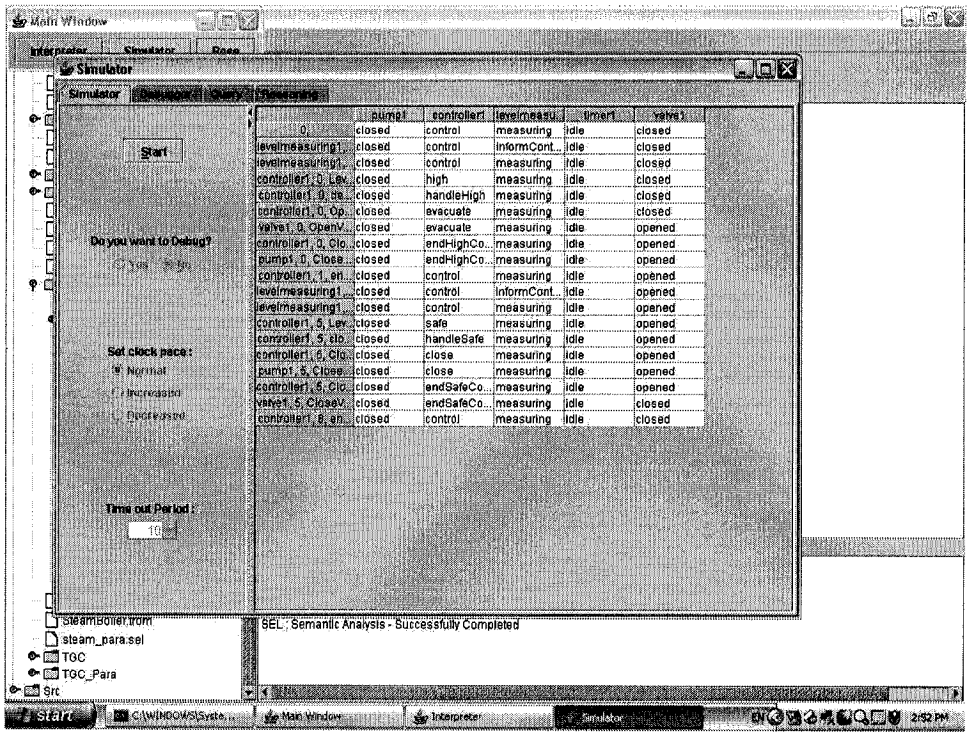

Figure 5: Snapshot of the Simulator's simulation result

3. All the data in the simulation results are presented in the same way, textual words inside table cells. There is no way to differentiate between TROM Objects, States, or events.

4. Where are the states that belong to a certain objects in the presented results? The expert users only know that the listed names inside the table represent the states for the corresponding header TROM objects.

5. There is no way of knowing the relationship between states.

6. Where are the TROM events in the presented results? The expert users only know how to locate the names of events.

7. The size of the cells that contain the TROM Object, Time, and Event name is very short. Therefore, in many cases the event name is not displayed in the interface.

8. In the event transition, the state from which the TROM object is changing is not shown in the interface.

9. The type of event is not made clear to the user.

10. The cause of the event is not displayed in the interface.

11. Because of the limited fixed size of the first column, the Time is not displayed in many cases.

12. There is no help or guidance provided to the user to describe the results or the presented information.

These problems make perceiving TROMLAB simulation results a difficult task to any one other than TROMLAB experts. Hence, they limit the usability of the powerful tools included in the TROMLAB framework. Therefore, there is a need for a better way of presenting the simulation results.

3.5 A New Way for Representing Simulation Results

For the users to get the maximum benefit from the TROM simulator, the simulation results should be easy to understand. Therefore, the way of presenting data has a key role in making data understandable and perceivable by users.

There are two major ways of presenting data: the tabular form and the graphical visualized form. The Tabular form is static; it doesn't communicate with the user in any way other than the presentation. Moreover, for tables to be understood well, the number of dimensions should be kept as minimum as possible, preferably two. Furthermore, in the tabular form, data reside inside rows and columns with limited abilities to give self-descriptions because of the limitations of interface size and to avoid complexity.

On the other hand, representing data graphically makes it easy to understand and grasp by the user. Graphics speeds up the communication of data and makes it accessible to users. It gives more dimensions to the presented data and allows it to describe itself by attributes like color, shape, size, or thickness. Hence, graphically visualizing data will improve the understandability of the presented data and make the user understand the simulation results at a glance. Therefore, this thesis presents the implementation of a visualization tool that will present the simulation results in a more appealing way.

3.6 General Principles for Usable Visualizations

This thesis is going to follow general principles of perception that appeared in [Mir98] when visualizing simulation results. The adopted principles are:

1. "Focus on the data": make the user's mental thinking and focus on the presented data by avoiding any unnecessary additional graphical items or dialogues. Therefore, our visualization avoids any distracting unnecessary information and presents information in one screen.

2. "Users perceive relationships between data in close proximity. Yet this perception of relationship due to proximity is overridden by physically distant points that have links drawn between them". Therefore, relationships between TROM objects and state transitions should conform to this principle and related objects should be visualized near each other.

3. "When data is densely displayed, users may fail to see some information due to occlusion or over plotting, causing them to misinterpret what the display is saying." In our implementation, the screen that displays the simulation results will be partitioned into three containers and each container will have a different view and does not affect the display of the other container.

4. "People remember the outline of a graphic or image more than what is inside of it." Therefore, the visualization tool implemented by this thesis will show a partial presentation of TROM Objects state chart diagrams to show the overall behavior of each TROM Object during the execution. The presentation will include only the states that exist in the simulation results.

5. "Icons facilitate interaction best when their appearance distinctively represents their function." Therefore, proper icons will be used for commands to give a good affordance to users.

6. "Users tend to invert the figure and ground when they experience excessive perceptual cueing or perceptual overload." Users have limited perceptual processing capabilities; therefore, this thesis will emphasize on simplicity of representation.

7. "The effect of users' prior knowledge and expectations on their interpretations of what they see". Users are accustomed to UML diagram notations; therefore, the same UML notations are used to represent Objects, rectangles; States, round rectangles; and Transitions, arrows.

After inspecting the understandability of the simulation results and defining understandability guidelines for the visualization of simulation results, we are ready to implement those guidelines in the visualization Animation tool.

Chapter 4

Visualization Animation Tool's Architecture

4.1 Architecture

4.1.1 Introduction

This chapter is intended to present the design of the Visualization Animation Tool (VAT). The reasoning behind the chosen architecture for the system will be detailed. The 4+1 view model of Kruchten for documenting architectures is used in this chapter [Kru95]. This model agrees with the draft recommended practice for architectural description IEEE P1471/D4.1 prepared by the Architecture Working Group of the Software Engineering Standards Committee [IEE98]. The template applied by this thesis for documenting such model view is introduced in [OCB00]. Figure 6 shows the 4+1 View Model. The Views that are presented are: the Logical View and the Scenarios. Only one scenario is provided in this chapter.

4.1.2 Purpose and Context

Real-time simulations are used to assist the validation of the correctness and completeness of the proposed real-time reactive system's design specifications. The design

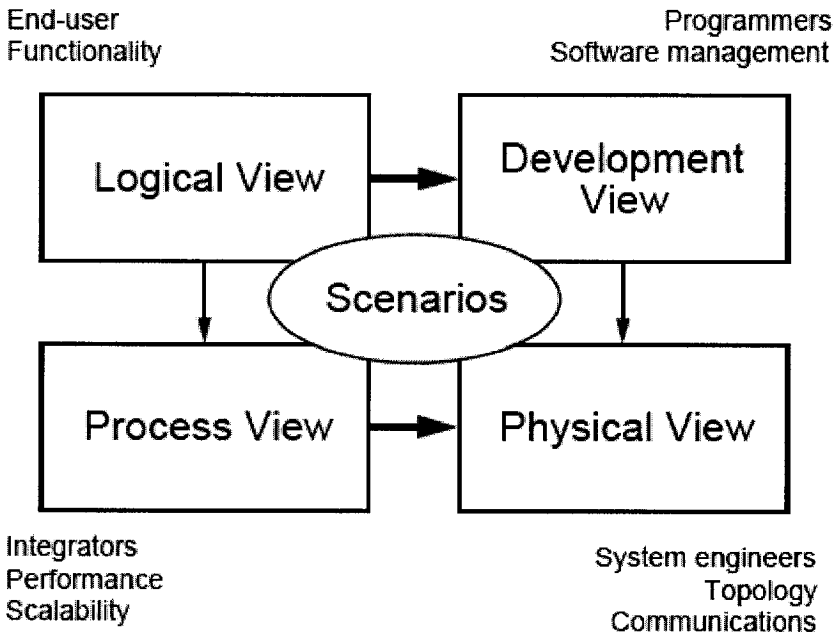

Figure 6: 4 + 1 View Model

specifications are converted to a TROM specification language and executed to inspect the predicted behavior of the system before the development stage begins. This helps improving the quality of the design. Visualizing and animating the simulated designs improves the understandability of the simulation result. Consequently, it aids in improving the testability of Real Time designs and eventually improves the quality of the design.

In TROMLAB, the Simulator tool produces a simulation event list describing the behavior of a specified TROM system. After verifying the usability of the resulting simulation output, it was decided that a tool is needed to visualize and animate it to maximize the benefit of the simulation process. The new tool will transform the simulation results into an XML format and export it to an XML file. The file will be parsed and processed in order to visualize the simulation result in the screen and animate it. The user will see a visualized and animated form of the predicted behavior of the specified TROM system on screen. Figure 7 shows the high level context

of the tool.

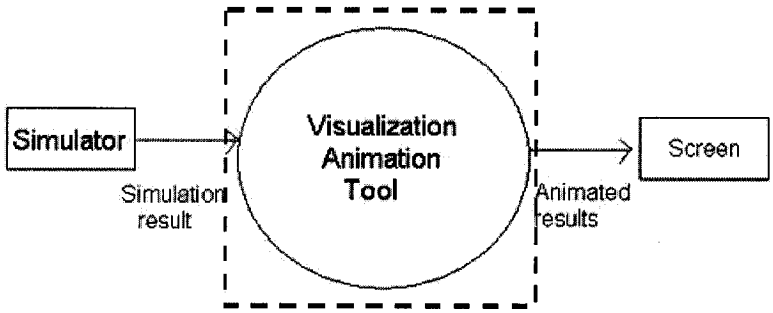

Figure 7: VAT Context Diagram

4.1.3 Tool Interface

This section introduces the services that VAT provides:

Service	Description
Export simulation results	This service extends the functionality of the simulator tool and allows it to export the simulation results as an XML file that can be used by any other tool.
Parse Simulation results	This service starts the collaboration of the VAT with other TROMLAB tools. VAT shall be able to parse any simulation result based on a specified XML schema
Analyze the parsed results	This service shall analyze the parsed simulation results and restructure it in memory so that it will be ready for use by any other tool.
Visualize simulation results	This service shall visualize the information analyzed from the simulation results and display it in the user screen
Animate visual information	This service shall animate the visualized simulation results.

Table 1: Tool Interface

4.1.4 Non-Functional Requirements

- Usability: the tool should provide an understandable representation of the simulation result. The objects, events, and states should be clear to the user. The

tool should inform the user about the current state of execution.

- Portability: the resulted animation should be able to be used by both web and windows application for different operating systems.
- Extensibility: the design should allow additional functionalities and components to be added to the tool.

4.2 Architectural Overview

The following section describes the logical view of the tool. It details the static structure of the proposed design.

Software Architecture is defined as a collection of computational building blocks (Components) along with the description of the relationships and interaction between these blocks (the connectors) [GS94]. In Addition, Software Architecture contains the rationale behind the specified architecture. The rationale explains the motivation and reasons for the selected Architecture style, the choice of components and connectors and the constraints of the system. Another definition by [AAG93] is : "At the software architectural level of abstraction, a system is typically described as a collection of interacting components. Components perform the primary computations of the system. Interactions between components include high level communication abstractions such as pipes, procedure calls, message passing, and event broadcast."

4.2.1 Pipes and Filters Architecture

The selected architecture type for the design of the Visualization Animation tool is the Pipes and Filters Architecture. In this architecture style, the system consists of several components connected in a series of pipes. The first component produces the input and the last component renders the output in the required format for the user. Each component in between receives input from the component preceding it and filters

this input by performing the required data transformation so that it will be ready for consumption by the following component. Each component in this architecture style is called a filter, and each connector is called a pipe. The pipes serve to deliver input from one component to another.

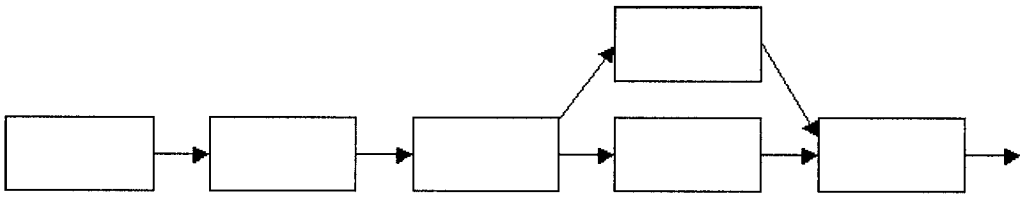

Figure 8: Pipe and Filter Architecture

In Figure 8, the Filters (Components) are represented as rectangles and the Pipes (Connectors) are represented as arrows.

4.2.1.1 Advantages

Some of the advantages of this architecture type are listed in [GS94]:

- It helps understanding the overall input/output behavior of the system.
- It allows reusability of filters.
- It eases maintainability of the system by treating each filter separately.
- It facilitates some types of specialized analysis.
- It supports concurrent execution.

4.2.2 Rationale Behind Selection

The structural organization (architectural style) of the components and their connections that is used in the design is the Pipes and Filters. This style provides increased flexibility, maintainability, reusability, and scalability. First, flexibility provided in this design will result in benefits for the development phase as Filters can be built

separately with different programming languages. Second, maintainability will increase productivity in work because each Filter can be treated alone, tested, and maintained without affecting the work of other Filters. Third, the Pipes and Filters architecture provided in our design will make it easy to reuse the components in other designs. Finally, this style provides scalability as more filters can be added without affecting the existing ones. Hence, this style was selected to structure our design.

4.2.3 Architecture Diagram

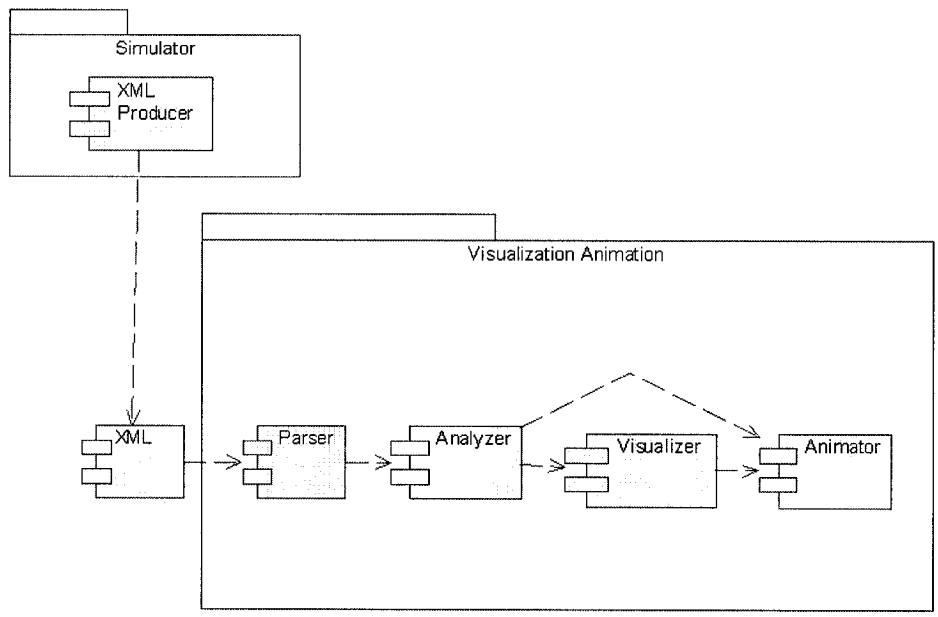

Figure 9: Components Diagram for VAT Architecture

Figure 9 shows the packages and components forming the VAT. It shows two packages, the Simulator package and the Visualization Animation package. The two packages communicate throw pipes that deliver the XML format of the Simulation result from the first package to the second package. The Simulator package consists of

many components that work together to produce at the end the simulation event list. In this thesis, another component was added to the Simulator, which is the XML producer component. The output of this component is received by the Parser component in the Visualization Animation package. After filtering the information, the output is passed through the pipe to the Analyzer component. The Analyzer component does the necessary transformations and passes the output to the Visualization component through the connecting pipe. The Visualization component transforms the output to a format ready for the Animator component to execute and produce the final result. At the same time the Animator component receives the analyzed output from the Analyzer to aid in performing its functionality. The final result after applying all the filters is displayed in the user interface.

4.3 Components

This section describes each component in the architecture diagram. Each component is described in terms of its responsibilities, the interfaces that it provides for interaction with it and the rationale for the component. As well, the other components that the component requests services from in order to fulfill its purpose are listed as collaborators. There are also notes that provide information on constraints or other useful information about the component, such as the main classes in the components and the patterns that may have been used within the component. Finally, any issues that are unresolved may be listed.

4.3.1 XML Producer

Component	XML Producer
Responsibilities	This component is part of the Simulator package, which is the package responsible for building the simulation event list. The simulation result contains information about Trom Objects, Event specifications, States, and Transition information.There are two types of simulation results: Normal Mode Simulation Debug Mode Simulation-The Normal Mode results in having the whole simulation process as one shot, whereas, the Debug Mode compiles the simulation event by event.The purpose of this component is to expose the Simulation result in XML format and store it in an XML file. This is done during the simulation process while the simulator is compiling the simulation event list along with the TROM and System Configuration Specification of a specific Timed Reactive system.
Collaborators	Collaborates with Simulator Package components.
Notes	This Component will run each time the Simulator simulates any TROM specifications system.The produced XML file should conform to a specific schema that will be discussed later in the next section.
Issues	

Table 2: XML Producer Component Description

4.3.2 XML File

The simulation result will be transformed to an XML file. The reason why XML was used is because it has become a standard for data representation and communication. Its extensibility and platform independence are among the most favourable characteristics that make it a good solution for any textual representation need. Extensibility will make communication between Simulator and other tools that will use its simulation results in future very flexible for any additions or updates as more details can be exported with time. Platform independence, also, will make the simulation results portable to different applications running in different operating systems or using different types of technologies. It has been agreed by the TROMLAB members to use

the following XML schema for exchanging the simulation result:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT cause (#PCDATA)>
 <!ATTLIST cause
    trom (#PCDATA) #REQUIRED
    time (#PCDATA) #REQUIRED
    causing (#PCDATA) #REQUIRED>
<!ELEMENT event (name, state-set, cause)>
 <!ATTLIST event
    trom (#PCDATA) #REQUIRED
    time (#PCDATA) #REQUIRED>
<!ELEMENT event-set (event+)>
<!ELEMENT initial (#PCDATA)>
 <!ATTLIST initial
    trom (#PCDATA) #REQUIRED
    state (#PCDATA) #REQUIRED>
<!ELEMENT initial-states (initial+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT simulation (initial-states, tromevent)>
 <!ELEMENT state (#PCDATA)>
 <!ATTLIST state
    trom (#PCDATA) #REQUIRED>
<!ELEMENT state-set (state+)>
<!ELEMENT tromevent (event-set)>
```

4.3.3 Parser Component

Component	Parser
Responsibilities	This component is part of the Visualization Animation Tool. The main purpose of this component is to parse the XML file that contains the simulation result, which is produced by the XML Producer component during the simulation process. The Parser will extract all the information and passes it to the Analyzer component.
Collaborators	Collaborates with the Analyzer Component
Notes	The Parser will be able to parse the simulation result that is produced from the Simulator directly at the simulation process. Also, it can parse any old simulation result by allowing the user to browse and select any available simulation result and then it start parsing it and sending it to the Analyzer component.
Issues	

Table 3: Parser Component Description

4.3.4 Analyzer

Component	Analyzer
Responsibilities	This component is part of the Visualization Animation Tool. The main purpose of this component is to analyze the parsed simulation result and build the data structures necessary for the visualization and animation of the simulation result. The Analyzer will build a list of events containing the details of each event, a list of TROM Objects containing details about each TROM Object, and a list of states for each TROM Object.
Collaborators	Collaborates with: Parser, Visualizer,and Animator Components
Notes	The built data structures will be stored in the Main Memory so that the other components can read them and utilize them in order to do the visualization for each analyzed object or animation for each analyzed event.
Issues	

Table 4: Analyzer Component Description

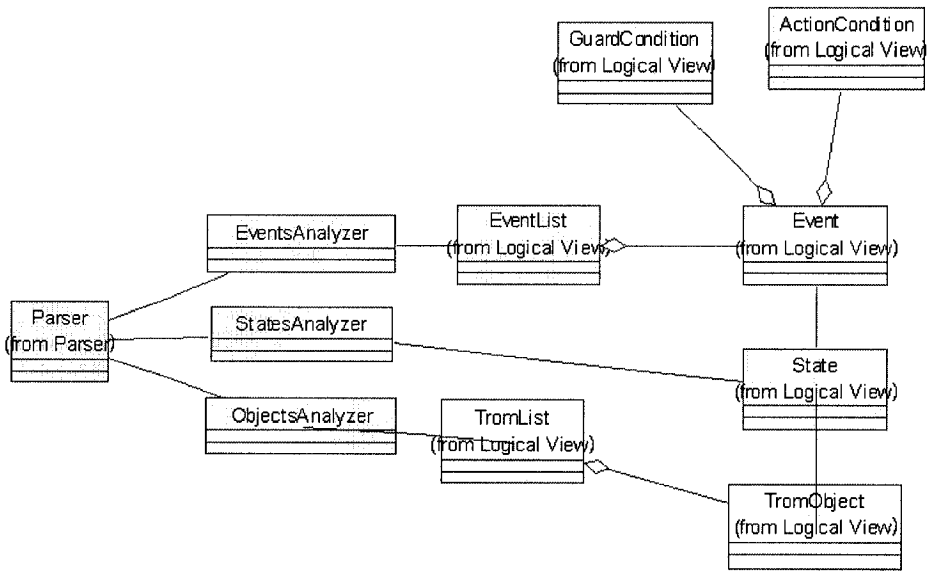

Figure 10: Analyzer Component class diagram

Figure 10 shows the class diagram of the Analyzer component. There are three analyzer classes each of which is responsible for a different TROM concept.

- Events Analyzer: this class is responsible for building the Events list data structure. It will read events from the parser and build a node inside the event list for each event.
- Eventlist:this data structure contains list of simulation result events.
- Event: this class maintains details about each event such as:
  - Source TROM Object issuing the event.
  - Destination TROM Object receiving the event.
  - Source State issuing the event.
  - Destination State that the system will go to after issuing the event.
  - Event Time: the time at which the source object raised the event.
  - Causing event: the event that caused this event to happen.

  - Shared event: whether this event is shared or not

- State Analyzer: this class is responsible for building a list of states and associate each one with the corresponding TROM object. Also, it is a base for the visualization of states in the Visualizer Component.

- Object Analyzer: this class is responsible for building the list of TROM objects. It will extract the object details from the parser and maintain the list by adding new nodes of TROM objects or adding details for each object.

- Trom Object: class representing each TROM object. It contains the details about each object such as States, Transitions , and Attributes.

4.3.5 Visualizer Component

Component	Visualizer
Responsibilities	This component is part of the Visualization Animation Tool. The main purpose of this component is to visualize the simulation result elements. This component will partition the user interface screen into several parts and distribute the simulation result elements in an understandable way. It will load all the TROM objects and construct a partial representation of the state chart diagram for each object with only the active states that participated in the simulation result.
Collaborators	Collaborates with the Analyzer and Animator Components.
Notes	The visualized objects will be loaded into the interface screen and will be ready for play and animation that will be controlled by the Animator.
Issues	Visualized objects will be in a two dimensional view.

Table 5: Visualizer Component Description

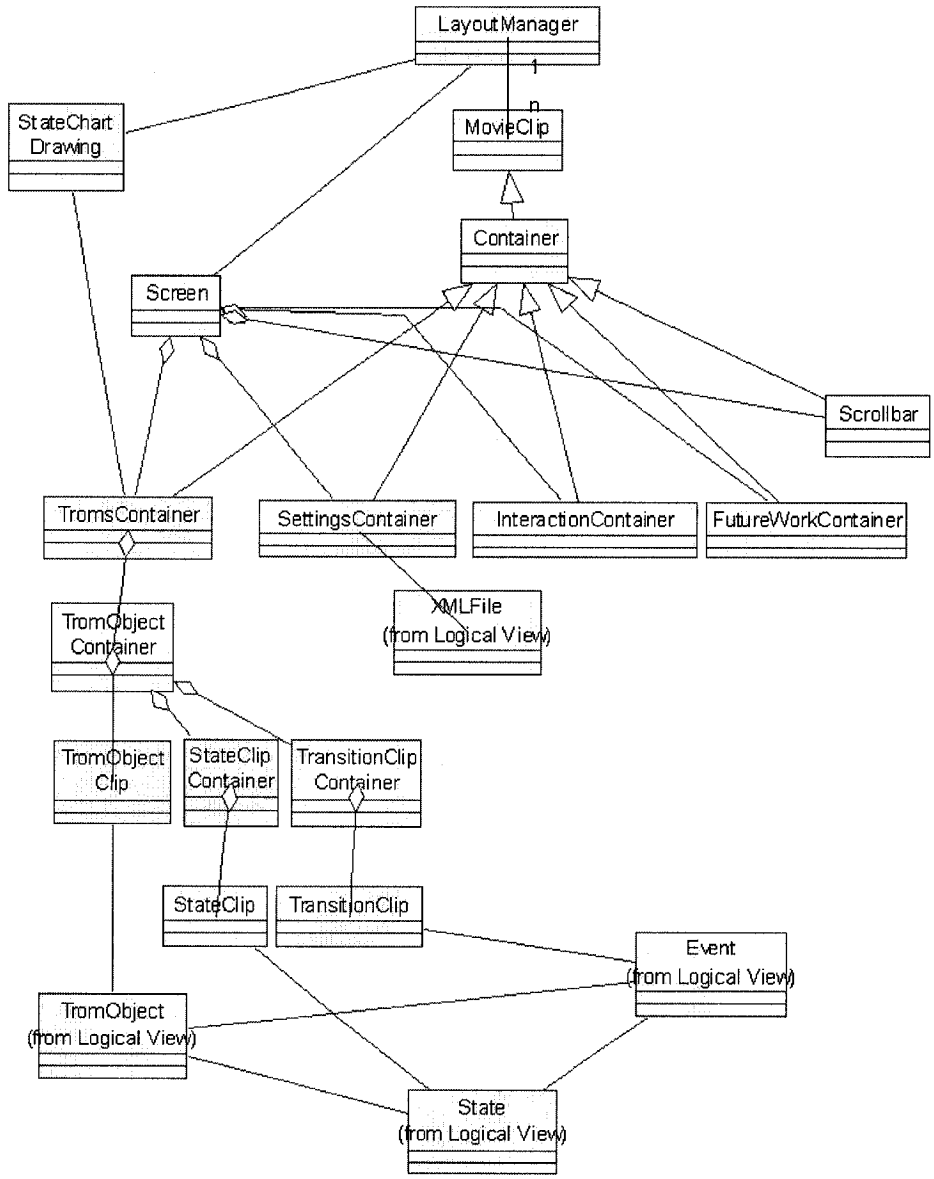

Figure 11: Visualizer Component class diagram

Figure 11 shows the class diagram that contains details about the components inside the Visualizer component. The classes are: Layout Manager: this class is responsible for managing the whole interface screen. It partitions the screen into 4 main containers: Troms Container, Settings Container, Interaction Container, and Future work Container. Furthermore, the layout manager will load the contents of each container.

- MovieClip: this is the system base class for any visualized element. It has the attributes and functions necessary to locate and interact with the visualized element.

- Container: this class inherits from MovieClip and has the common layout for any container that will hold any simulation result element.

- State Chart Drawing: this class is responsible for constructing the state chart diagram for all the TROM objects. It will display only the active transition routes that happened during the simulation process.

- TROM Container: this class manages the screen part that will host the list of TROM object containers. Each TROM object Container will contain the state chart diagrams for that specific TROM object. It will include the states and transitions the correspond to the events that happened to this TROM object during the simulation process.

- Settings Container: this class manages the screen part that will contain the settings for playing the animation, selecting the source XML file, and displaying the Timer.

- Interaction Container: this class manages the animation of the simulation result. More details are provided in the next section.

4.3.6 Animator Component

Component	Animator
Responsibilities	This component is part of the Visualization Animation Tool. The main purpose of this component is to animate and play the visualized simulation result elements. It shows the interaction between TROM objects and event transitions between states. The animation starts from the initial state where all the TROM objects are ready and continues event by event in the same order that happened during the simulation process until the system reaches the ready state at the end of the simulation. Therefore, this Animator takes control over the visualized elements.
Collaborators	Collaborates with the Analyzer and Visualizer Components.
Notes	
Issues	

Table 6: Animator Component Description

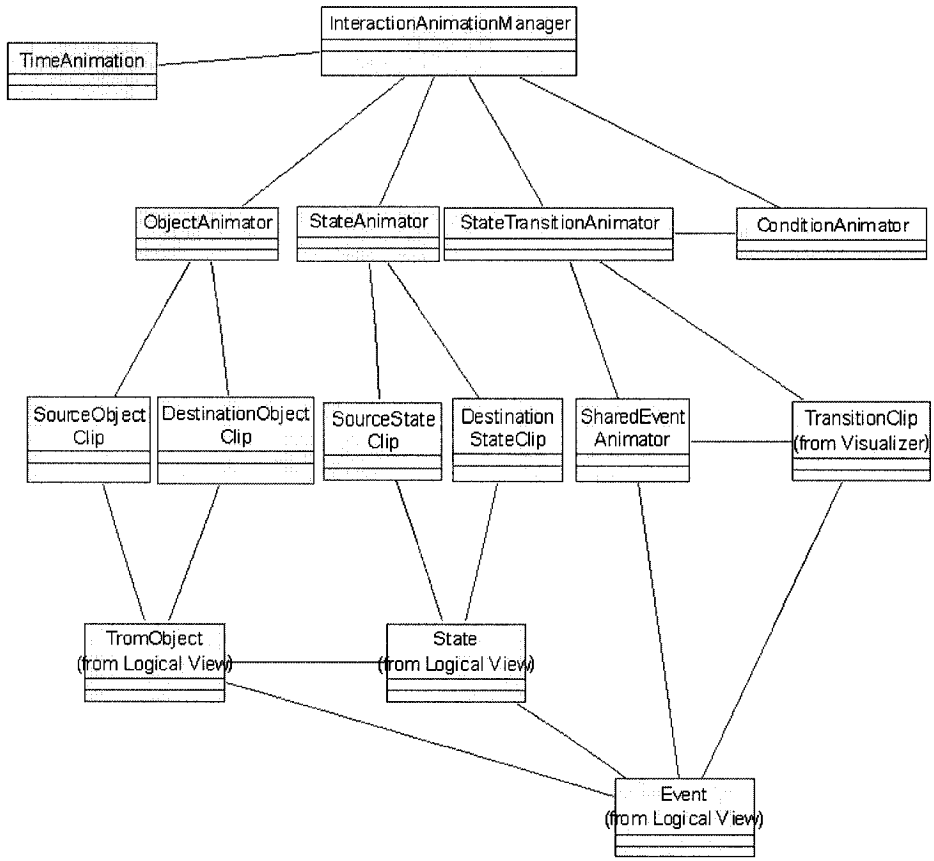

Figure 12: Animator Component class diagram

Figure 12 shows the class diagram of the Animator Component. It contains the classes that manage the animation process of the simulation result.

- Interaction Animation Manager: it manages the animation by controlling the stop and play, and the load and unload of visualized elements.
- Object Animator: animates the TROM object interactions and shared events.
- State Animator: animates states load/unload and user interaction.
- State Transition Animator: animates the state transition when an event happens at a specific state and changes the TROM object to another state.

4.4 Development Platform

The development environment used to develop the VAT is Macromedia Flash MX Professional 2004. It is a powerful development platform that has the strength of being an authoring tool for building graphics and interactive animation to web applications. In addition, it has an Object Oriented scripting language called ActionScript that has all the powerful features of OO languages. ActionScript enables developers to visualize, animate, process data, create dynamic content, and manipulate visual components.

The advantages of Flash MX Professional 2004 that motivated the selection of it are:

1. Browser independent.
2. Platform independent.
3. Screen Resolution independent.
4. Loads fast on the internet.

4.5 Dynamic Behavior : Scenario

The dynamic behavior of the VAT is described in terms of scenarios that are derived from the use cases defined in the tool interface.
Figure 13 shows one possible high level scenario for the execution of the VAT tool. The simulator triggers the XML Producer to start exporting the simulation result into an XML file. The XML Producer will translate the results and at the same time, if the running mode is Debug, or after finishing, when the running mode is Normal, it will trigger the Parser to start parsing the simulation results. During parsing, the Analyzer will restructure the information to build data structures that are ready for being visualized by the Visualizer. After finishing the analysis, the Visualizer will

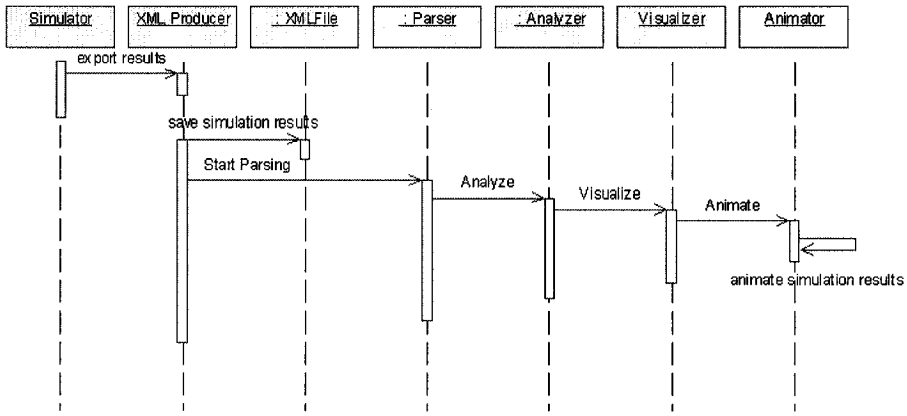

Figure 13: VAT Sequence Diagram

visualize the data and trigger the Animator to start animating it.

Another possible scenario is that the user can load any XML file that was produced in a previous simulation process or by any other Real Time Reactive Framework, provided that it conforms to the defined schema, and starts visualizing and animating the simulation results.

The introduced VAT architecture should be able to visualize the simulation results of any real-time reactive system provided that it conforms to the specified XML schema. In the next chapter, a solution for a simplified specification of the Steam Boiler Controller case study is introduced using TROM Formalism. The simulation result of this case study will be used to test the VAT.

Chapter 5

Steam Boiler Case Study

5.1 Introduction

Many case studies have been proposed for testing and evaluating semantic methods in computer science with respect to their abilities to solve real time embedded control system design problems. One of those case studies is the Steam Boiler Control Specification Problem of J.-R. Abrial and E. Brger [Abr91]. This specification problem has been derived from an original text that has been written by LtCol. J.C. Bauer for the Institute for Risk Research of the University of Waterloo, Ontario, Canada. The original text has been submitted as a competition problem to be solved by the participants of the International Software Safety Symposium organized by the Institute for Risk Research.

This chapter introduces a formal specification solution for a simplified model of the problem. The purpose is to provide specification design that assures safe operation of a steam boiler by maintaining the ratio of the water level in the boiler and the amount of steam emanating from it with the help of the corresponding measuring devices. The requirements have been reduced to include only the normal mode of working where the boiler has been initialized successfully and all the physical units of it are functioning properly.

5.2 Problem Description

5.2.1 Informal Problem Definition

The Steam Boiler System consists of the following physical units:

- Steam Boiler: the container that keeps water inside it.
- Pump: to pour water inside the steam boiler.
- Valve: to evacuate water from the steam boiler.
- Water Level Measuring: a sensor to measure the quantity of water q (in liters) and inform the system whenever it risks the minimum or maximum allowed amounts.

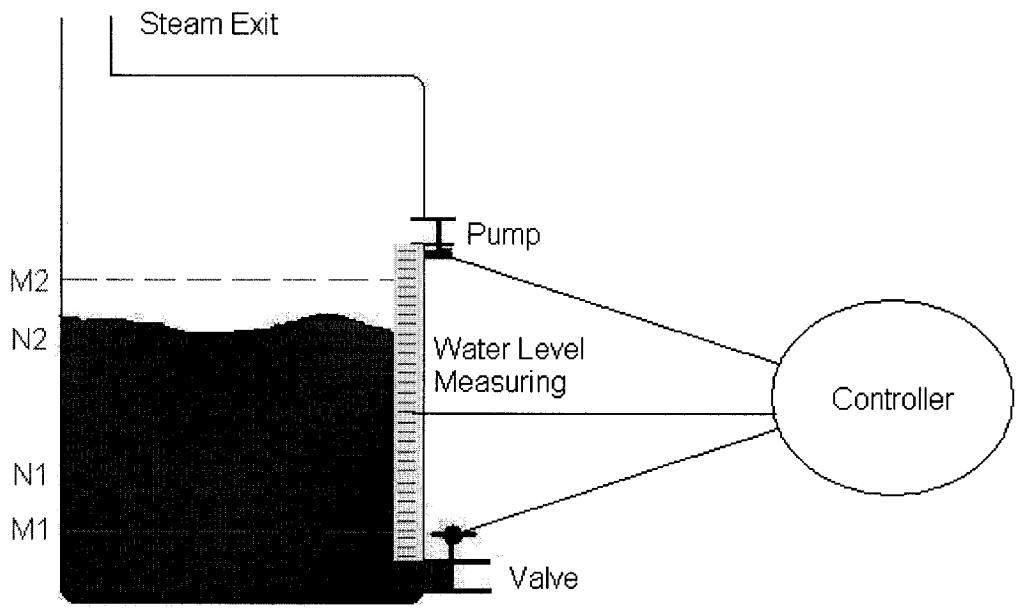

Figure 14: Steam Boiler Controller

Figure 14 shows the Steam Boiler and the relations between its components. The Steam Boiler is assumed to start up with a safe amount of water and the Controller

runs a controlling cycle each 5 time units to check the current amount of water. It triggers the Water Level Measuring device to measure the level and send it to the Controller. Then the Controller receives the current level and checks if it is in the normal level, above, or below: if the water is in the normal level, it will not do any thing; if the water level is risking to the minimum safe level, it will trigger the Pump to pour water; and if the water is risking above the normal limits, it will trigger the Valve to evacuate water.

5.2.2 Characteristics of system components

Steam Boiler is characterized by the following:

- The total Capacity C (in liters)
- The minimal limit quantity M1 (in liters), below which the steam boiler would be in danger after 5 time units.
- The maximal limit quantity M2 (in liters), above which the steam boiler would be in danger after 5 time units.
- The minimal normal quantity N1 (in liters) to be maintained by the program during normal operations (N1 is greater than M1).
- The maximal normal quantity N2 (in liters) to be maintained by the program during normal operations (N2 is less than M2).

The Pump is characterized by the following:

- The main program will control the pump.
- The Controller sends an Open Pump message to trigger the pump to pour water inside the boiler.
- The Pump needs five time units to start pouring water into the boiler, which is the amount of time needed to balance the pressure of the steam inside the steam boiler.

- As soon as the level reaches N2 the pump should be closed.
- The Controller sends Close Pump message to trigger the pump to close instantaneously.

Valve is characterized by the following:

- The Controller sends an Open Valve message to trigger the valve to open and evacuate water.
- The Controller sends Close Valve message to trigger the valve to close.

The Water Level Measuring sensor is characterized by the following:

- Initially the water level measuring will send the current water level; assuming that every thing is ok, the 5-second cycle is starting, and it got the request from the Controller.
- This device will continue to measure the quantity of water whenever it gets a request from the Controller.
- It will send a message that contains the current measured quantity of water (in liters) in less than one time unit.

5.2.2.1 Controller Operations

The program should follow a cycle that takes place each 5 time units and consists of the following functionality:

- The Controller is going to communicate with the physical units through dedicated lines where the time of message transmission is ignored.
- After each 5 time units, the Controller will request the current level of water from the Water level measuring unit.

- When the Controller receives the current level, it should maintain a safe level of water between N1 and N2, assuming that all the physical units are operating without any deficiency.
- if water level is more than N2 and the pump is opened, it should command the pump to switch off.
- If the water level is more than N2 and the pump is closed, it should open the valve.
- if the water level is lower than N1, it should command the pump to switch on and pour water.
- If the water level is between N1 and N2, nothing will happen.

5.2.3 Assumptions

- The system starts with its water quantity q within safe ranges [M1, M2] and the water measuring device work well.

  $\uparrow R \Longrightarrow M1 < q > M2 \bigwedge S$

where R: means that the boiler system is running and S stands for the water-measuring device.

- There is enough space between N1 and N2 so that it takes at least 10 time units to fill up such space.

  (fill(N2-N1) $\geq 10 time units$)

5.2.4 Safety Property

The water quantity q in the steam boiler can never exceed the safe bounds for more than 5 time units continuously.

$((\ q < \text{M1} \bigwedge R) \Longrightarrow t \leq 5) \bigwedge ((q > M2 \bigwedge R) \Longrightarrow t \leq 5)$

where:

- M1 refers to Minimal limit of water quantity.
- M2 refers to Maximal limit of water quantity.
- t refers to operational time.
- R means the boiler system.

5.2.5 Liveness Property

Liveness property ensures that if the water level goes beyond the limits, it would eventually come back to normal level. In terms of formal language, it can be written as:

(WaterLevel (q < M1 , t1) $\bigvee WaterLevel(q > M2, t1)) \Longrightarrow \exists t2.(WaterLevel(q > M1, t2) \bigwedge WaterLevel(q < M2, t2) \bigwedge (t2 > t1))$

where: WaterLevel(P, t) is true when a condition P is true at time t.

5.3 Formal Solution

The following subsections introduce the formal specification and design of the Steam Boiler problem.

5.3.1 Class Diagram for Steam Boiler

Figure 15 shows the GRC classes and PortTypes of the solution:
The Controller class is an aggregation of port types @CP, @CV, @CT, and @CL. Also, it contains one parameterized attribute, which is quantity. This attribute contains the current quantity of water inside the Steam Boiler, the Pump class is an aggregation of port type @P, the Valve class is an aggregation of port type @V, the Level Measuring class is an aggregation of port type @L. Also, it contains one parameterized attribute, which is quantity. This attribute contains the current quantity of water inside the Steam Boiler, and the Timer class is an aggregation of port type @T.

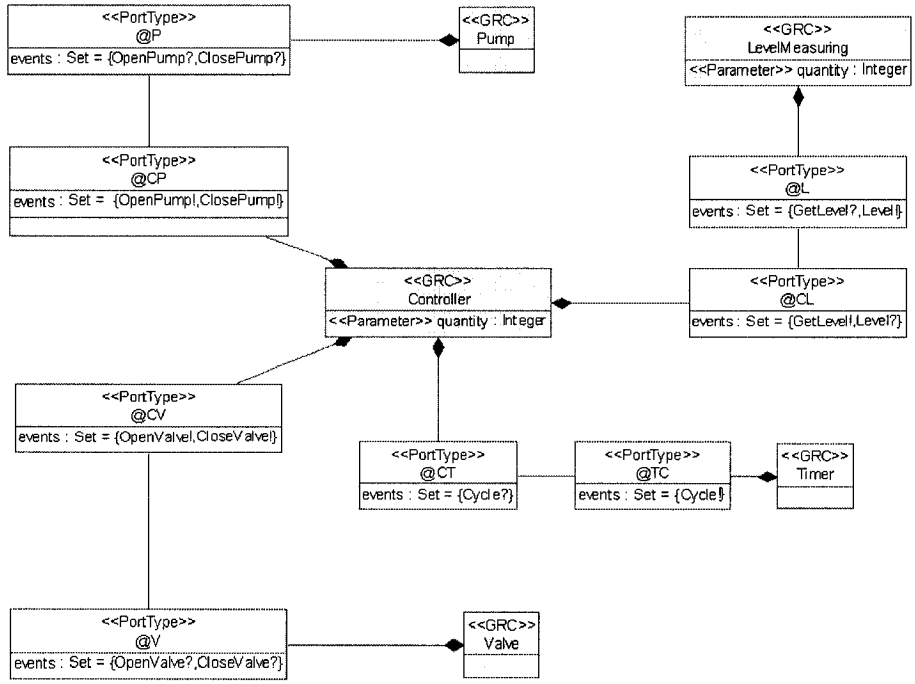

Figure 15: Steam Boiler Controller class diagram

The link between the port type @CL of the Controller and the port type @L of the Level Measuring means that the Controller uses port @CL to communicate with the Level Measuring through its port @L.

The link between the port type @CP of the Controller and the port type @P of the Pump means that the Controller uses port @CP to communicate with the Pump through its port @P.

The link between the port type @CV of the Controller and the port type @V of the Valve means that the Controller uses port @CV to communicate with the Valve through its port @V.

The link between the port type @CT of the Controller and the port type @T of the Timer means that the Controller uses port @CT to communicate with the Timer through its port @T.

5.3.2 Level Measuring Class

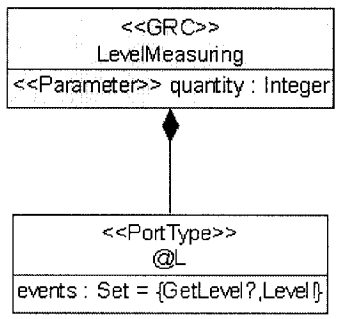

Figure 16: Level Measuring class diagram

The Level Measuring class is an environmental class in the system, and all its output events cannot be constrained. Each 5 time units, it receives a GetLevel event message from the Controller through the port @L. after that it will transit to state Send or informController and issue the shared event message Level that has the parameter quantity, which is the current water level. This message will go to the Controller through the port @CL. Finally, the Level Measuring will return back to its measuring state. Figure 17 shows the state chart diagram for the Level Measuring Class.

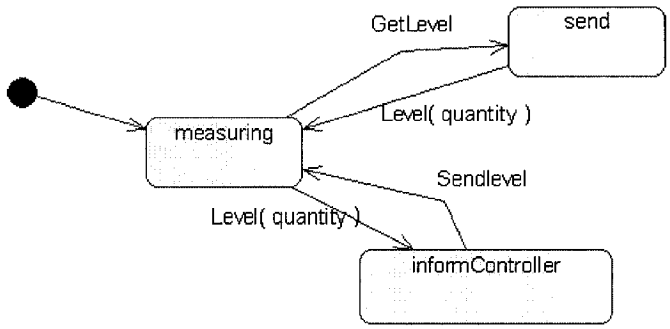

Figure 17: Level Measuring state chart diagram

The TROM formal specification description for the Level Measuring class is:

```
Class LevelMeasuring [@L]
Events: Level!@L, GetLevel?@L, Sendlevel
States: *measuring, informController, send
Attributes: quantity:Int
Traits:
Attribute-Function: measuring -> {quantity};
informController-> {};send -> {quantity};
Parameter-Specifications:
     Level: quantity;
Transition-Specifications:
     R1: <measuring,informController>; Level[quantity](true);
     true => true;
     R2: <measuring,send>; GetLevel[](true); true => true;
     R3: <informController,measuring>; Sendlevel[](true);
     true => true;
     R4: <send,measuring>; Level[quantity](true); true => true;
Time-Constraints:

end
```

5.3.3 Controller Class

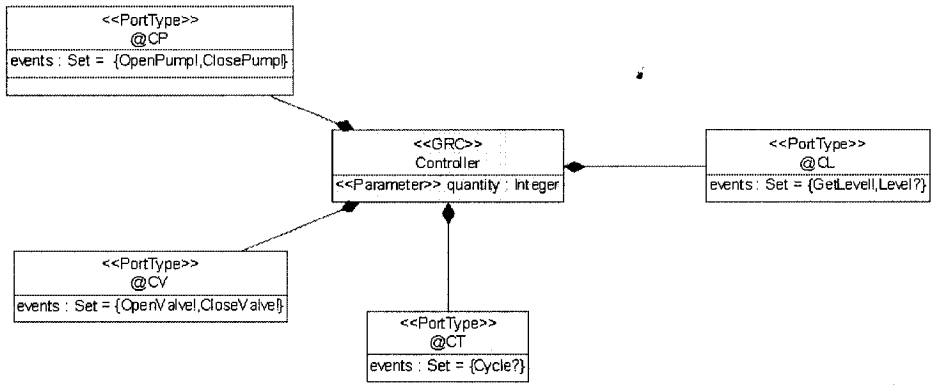

Figure 18: Controller class diagram

The controller class communicates with the environmental classes in order to maintain safe water level. Each 5 time units, it sends the shared event message GetLevel and receives the shared event message Level from the Level Measuring unit through the port @CL. Depending on the quantity attribute, the Controller will change state to high, low, or safe.

If the state is high, which means the current quantity of water is above the allowed normal limit, the controller will issue the internal event decrease and establish a timing constraint to decrease the amount of water by opening the Valve and closing the Pump in a time period between [1,4] time units. During this period, the Controller changes state to handleHigh, at which it issues OpenValve shared event message. This message will go to the Valve through the port type @CV. After this the Controller will go to state evacuate, at which the valve will be opened and the class will issue a shared event message ClosePump. This message will go to the Pump through the port type @CP. Then the Controller will change state to endHighControl and issue the internal event endHigh. Finally, the Controller will get back to its normal state control and get ready to another controlling cycle. The whole cycle takes between [1,4] time units.

If the state is low, which means the current quantity of water is below the allowed normal limit, the controller will issue the internal event increase and establish timing constraint to increase the amount of water by opening the Pump and closing the Valve in a time period between [1,4] time units. During this period, the Controller changes state to handleLow, at which it issues OpenPump shared event message. This message will go to the Pump through the port type @CP. After this the Controller will go to state pourwater, at which the Pump will be opened and the class will issue a shared event message CloseValve. This message will go to the Valve through the port type @CV. Then the Controller will change state to endLowControl and issue the internal event endLow. Finally, the Controller will get back to its normal state control and get ready for another controlling cycle. The whole cycle takes between [1,4] time units.

The last possibility is that the state is safe, which means the current quantity of water is within the allowed normal limit, the controller will issue the internal event closeAll and establish timing constraint to ensure the amount of water is safe by closing the Pump and the Valve in a time period between [1,4] time units. During this period, the Controller changes state to handleSafe, at which it issues ClosePump shared event message. This message will go to the Pump through the port type @CP. After this the Controller will go to state close, at which the Pump will be closed and the class will issue a shared event message CloseValve. This message will go to the Valve through the port type @CV. Then the Controller will change state to endSafe-Control and issue the internal event endSafe. Finally, the Controller will get back to its normal state control and get ready for another controlling cycle. The whole cycle takes between [1,4] time units. Figure 19 shows the state chart diagram of the Controller class.

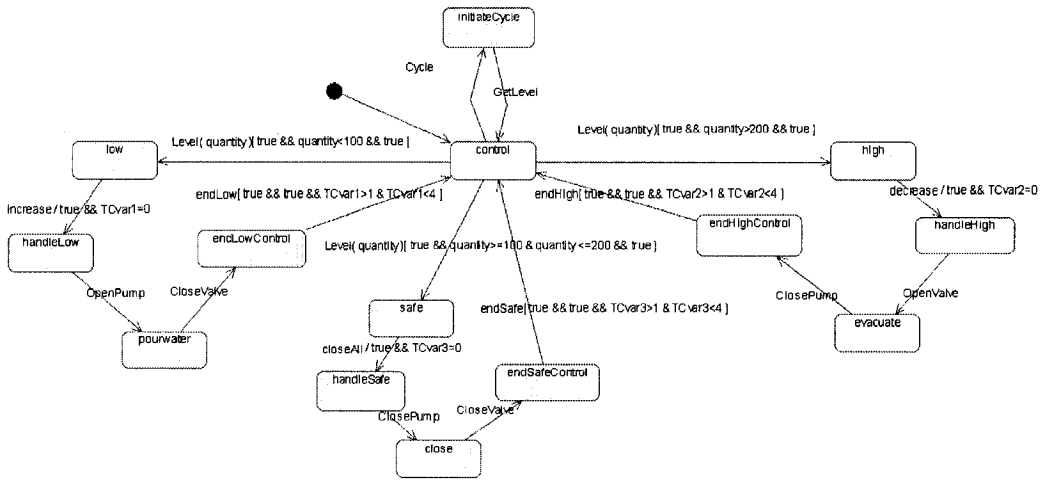

Figure 19: Controller state chart diagram

The TROM formal specification description for the Controller class is:

```
Class Controller [@CP, @CV, @CL, @CT] Events:
Cycle?@CT,Level?@CL,GetLevel!@CL, decrease, increase, ClosePump!@CP,
CloseValve!@CV, closeAll, OpenPump!@CP, endLow, OpenValve!@CV,
endHigh, endSafe States: *control, initiateCycle, high, low,
evacuate, pourwater, safe, close, handleLow, endLowControl,
handleHigh, endHighControl, handleSafe, endSafeControl

Attributes: quantity:Int
Traits:
Attribute-Function:
control ->{quantity};initiateCycle -> {};high -> {};low -> {};
evacuate ->{};pourwater -> {};safe -> {};close -> {};
handleLow ->{};endLowControl -> {};handleHigh -> {};
endHighControl ->{};handleSafe -> {};endSafeControl -> {};
Parameter-Specifications:
     Level: quantity;
```

```
Transition-Specifications:
     R1: <control,initiateCycle>; Cycle[](true); true => true;
     R2: <control,high>; Level[quantity](true); quantity>200 => true;
     R3: <control,low>; Level[quantity](true); quantity<100 => true;
     R4: <control,safe>; Level[quantity](true);
     quantity>=100&quantity<=200 => true;
     R5: <initiateCycle,control>; GetLevel[](true); true => true;
     R6: <high,handleHigh>; decrease[](true); true => true;
     R7: <low,handleLow>; increase[](true); true => true;
     R8: <evacuate,endHighControl>; ClosePump[](true); true => true;
     R9: <pourwater,endLowControl>; CloseValve[](true); true => true;
     R10: <safe,handleSafe>; closeAll[](true); true => true;
     R11: <close,endSafeControl>; CloseValve[](true); true => true;
     R12: <handleLow,pourwater>; OpenPump[](true); true => true;
     R13: <endLowControl,control>; endLow[](true); true => true;
     R14: <handleHigh,evacuate>; OpenValve[](true); true => true;
     R15: <endHighControl,control>; endHigh[](true); true => true;
     R16: <handleSafe,close>; ClosePump[](true); true => true;
     R17: <endSafeControl,control>; endSafe[](true); true => true;
Time-Constraints:
     TCvar1: R7, endLow, (1, 4), {};
     TCvar2: R6, endHigh, (1, 4), {};
     TCvar3: R10, endSafe, (1, 4), {};
end
```

5.3.4 Pump Class

The Pump class is an environmental class in the system. In each cycle of execution, this class receives controlling message from the Controller class according to the current level of water. If the level is low, Pump will receive a shared event message

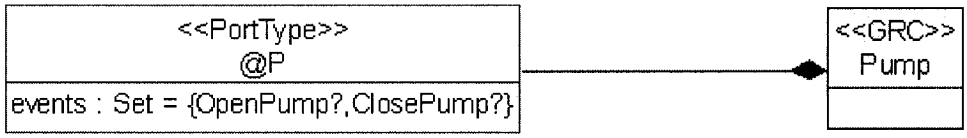

Figure 20: Pump class diagram

OpenPump from its port type @P. Then Pump will change state to toOpen and issue the internal event open. This event is time constrained, which means that the Pump should be opened in a time period between [0,5]. The Pump will remain in this state until it receives the shared event message ClosePump. Then it will return back to its close state. Figure 21 shows the state chart diagram for the Pump class.

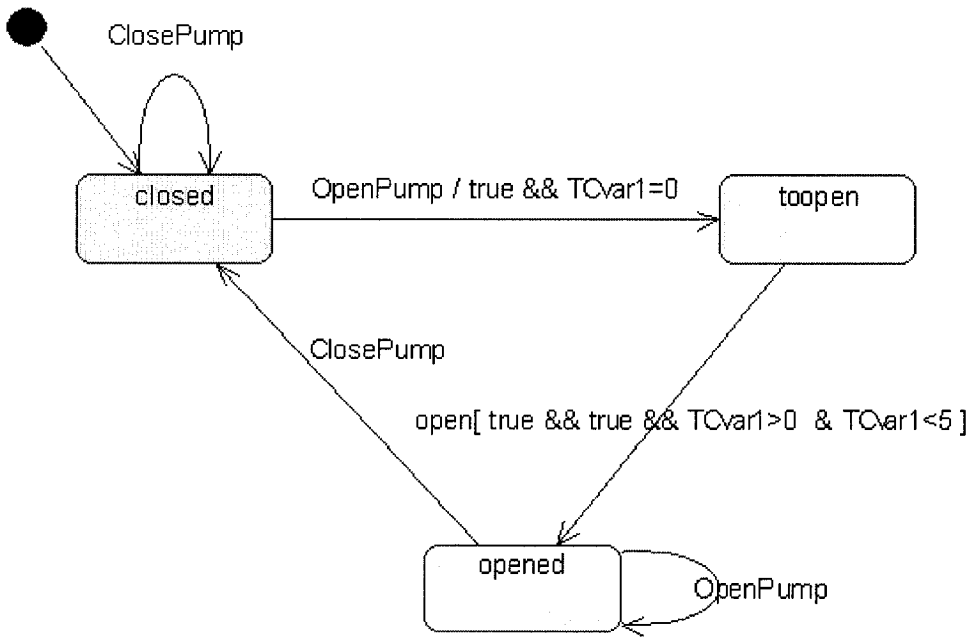

Figure 21: Pump state chart diagram

The TROM formal specification description for the Pump class is:

```
Class Pump [@P]
Events: OpenPump?@P, ClosePump?@P, open
```

```
States:*closed, toopen, opened
Attributes:
Traits:
Attribute-Function:closed -> {};toopen -> {};opened -> {};
Parameter-Specifications:

Transition-Specifications:
     R1: <closed,toopen>; OpenPump[](true); true => true;
     R2: <closed,closed>; ClosePump[](true); true => true;
     R3: <toopen,opened>; open[](true); true => true;
     R4: <opened,closed>; ClosePump[](true); true => true;
     R5: <opened,opened>; OpenPump[](true); true => true;
Time-Constraints:
     TCvar1: R1, open, (0, 5), {};
End
```

5.3.5 Valve Class

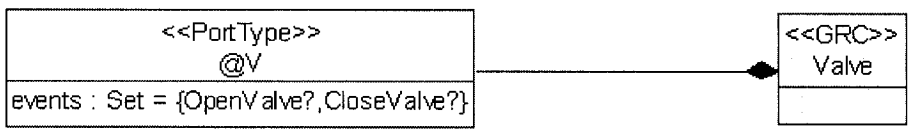

Figure 22: Valve class diagram

Another environmental class in the Steam Boiler system is the Valve class. During the control cycles of execution, this class receives controlling shared message from the Controller class based on the current measured level of water inside the steam boiler. If the level is high, the Valve will receive a shared event message OpenValve through its port type @V. Then the Valve will change state to opened. The transition will happen instantly at the receive of the shared event. The Valve will remain in this

state until it receives the shared event message CloseValve. Then it will return back to its closed state. Figure 23 shows the state chart diagram for the Valve class.

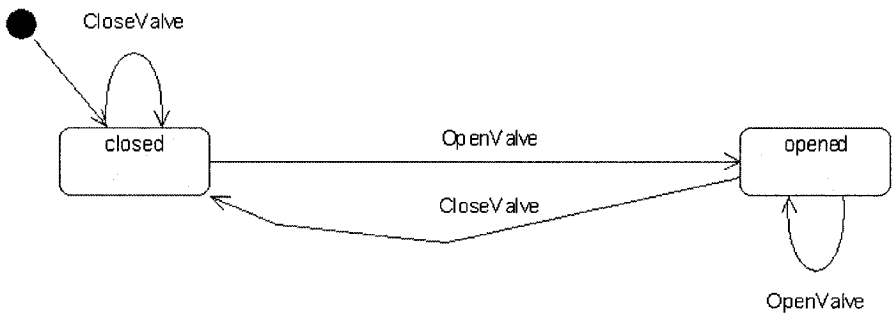

Figure 23: Valve state chart diagram

The TROM formal specification description for the Valve class is:

```
Class Valve [@V]
Events: OpenValve?@V, CloseValve?@V
States:*closed, opened
Attributes:
Traits:
Attribute-Function: closed ->{};opened -> {};
Parameter-Specifications:

Transition-Specifications:
     R1: <closed,opened>; OpenValve[](true); true => true;
     R2: <closed,closed>; CloseValve[](true); true => true;
     R3: <opened,closed>; CloseValve[](true); true => true;
     R4: <opened,opened>; OpenValve[](true); true => true;
Time-Constraints:
```

End

5.3.6 Timer Class

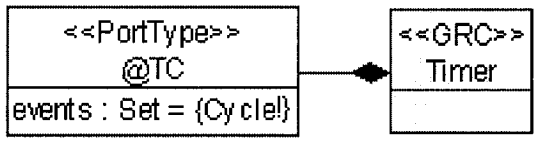

Figure 24: Timer class diagram

This class is an external component attached to the Controller class. Its main purpose is to initiate the controlling cycle after each 5 time units by triggering the Controller to start a new cycle. It sends a shared event message through its port type @T. then the Timer returns back to its idle state and wait for the second cycle time. Figure 25 shows the state chart diagram for the Timer.

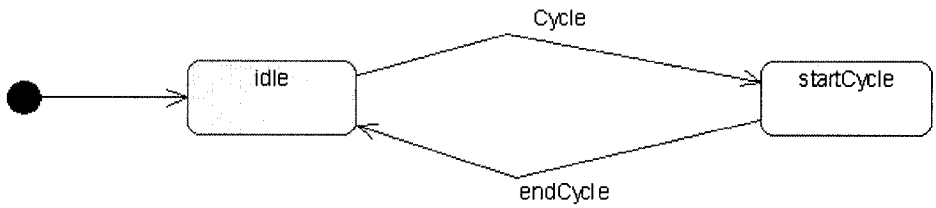

Figure 25: Timer state chart diagram

The TROM formal specification description for the Valve class is:

```
Class Timer [@TC] Events: Cycle!@TC, endCycle
States:*idle,startCycle
Attributes:
Traits:
Attribute-Function:idle->{};startCycle -> {};
```

```
Parameter-Specifications:

Transition-Specifications:
     R1: <idle,startCycle>; Cycle[](true); true => true;
     R2: <startCycle,idle>; endCycle[](true); true => true;
Time-Constraints:

End
```

5.3.7 SubSystem Configuration Specification (SCS)

A System Configuration Specification provides the specification for a system or a subsystem by composing reactive classes. A subsystem specification consists of three sections: Includes, Instantiate and Configure. The Includes section imports other systems. The Instantiate section defines objects by parametric substitutions to the cardinality of ports for each port type. The Configure section defines the configuration of the systems architecture by composing the specified objects. The composition operator $\leftrightarrow$ sets up a communication link between compatible ports of interacting objects.

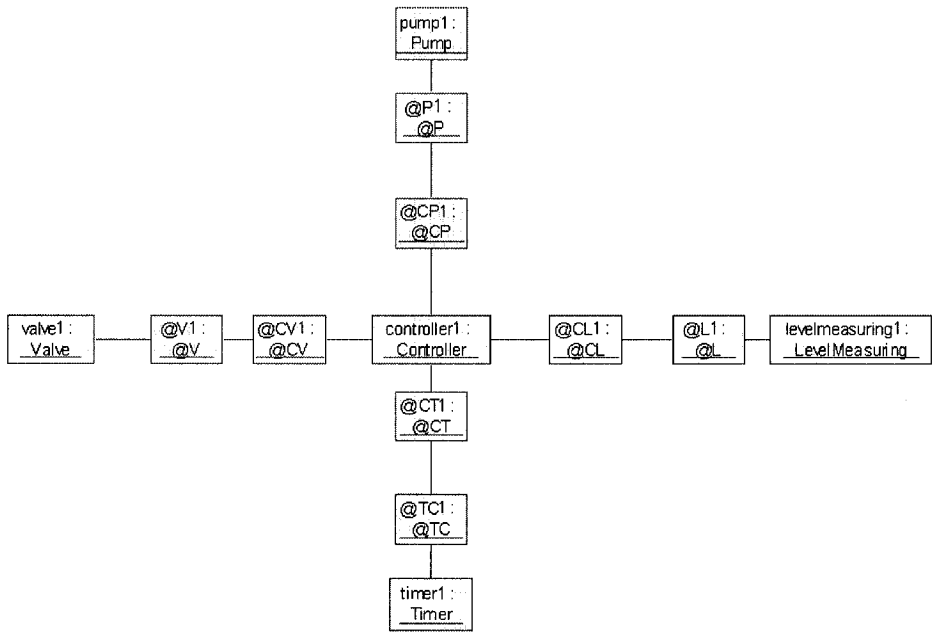

Figure 26: Steam Boiler collaboration diagram

Figure 26 shows the Collaboration diagram of the Steam Boiler system. There is one instance of Controller called controller1, Pump called pump1, Valve called valve1, Level Measuring called levelmeasuring1, and Timer called timer1. Also, there is one instance of each port type that functions as a communication media between any two system objects.

The subsystem TROM configuration specification of the Steam boiler is:

```
SCS SteamBoiler
     Includes:
     Instantiate:
          pump1::Pump[@P:1];
          controller1::Controller[@CP:1, @CV:1, @CL:1, @CT:1];
          levelmeasuring1::LevelMeasuring[@L:1];
          timer1::Timer[@TC:1];
```

```
        valve1::Valve[@V:1];
    Configure:
        controller1.@CP1:@CP <-> pump1.@P1:@P;
        valve1.@V1:@V <-> controller1.@CV1:@CV;
        levelmeasuring1.@L1:@L <-> controller1.@CL1:@CL;
        controller1.@CT1:@CT <-> timer1.@TC1:@TC;
end
```

5.3.8 Sample Simulation Event List

The event list provides a timeline of external stimuli used in the Steam Boiler system. Two execution cycles are listed in this simulation sample. 2 Level shared events are issued by the level measuring object that is instantiated in the SCS. These events simulate a scenario involving 2 execution cycles at 2 different times. At each time cycle (0 and 5), the level measuring measures the current quantity of water and send it to the controller through the shared event Level at port @L. The quantity of water is sent as a parameter in the event (90 and 120).

```
SEL: SteamBoiler
    levelmeasuring1, Level[quantity=90], @L1, 0;
    levelmeasuring1, Level[quantity=120], @L1, 5;
end
```

Figure 27 shows the sequence diagram for the sample simulation event list.
The formal solution of the Steam Boiler Controller case study is entered processed by the TROMLAB framework tools, the Interpreter and the Simulator, to produce simulation results. Those simulation results are used to test the VAT.

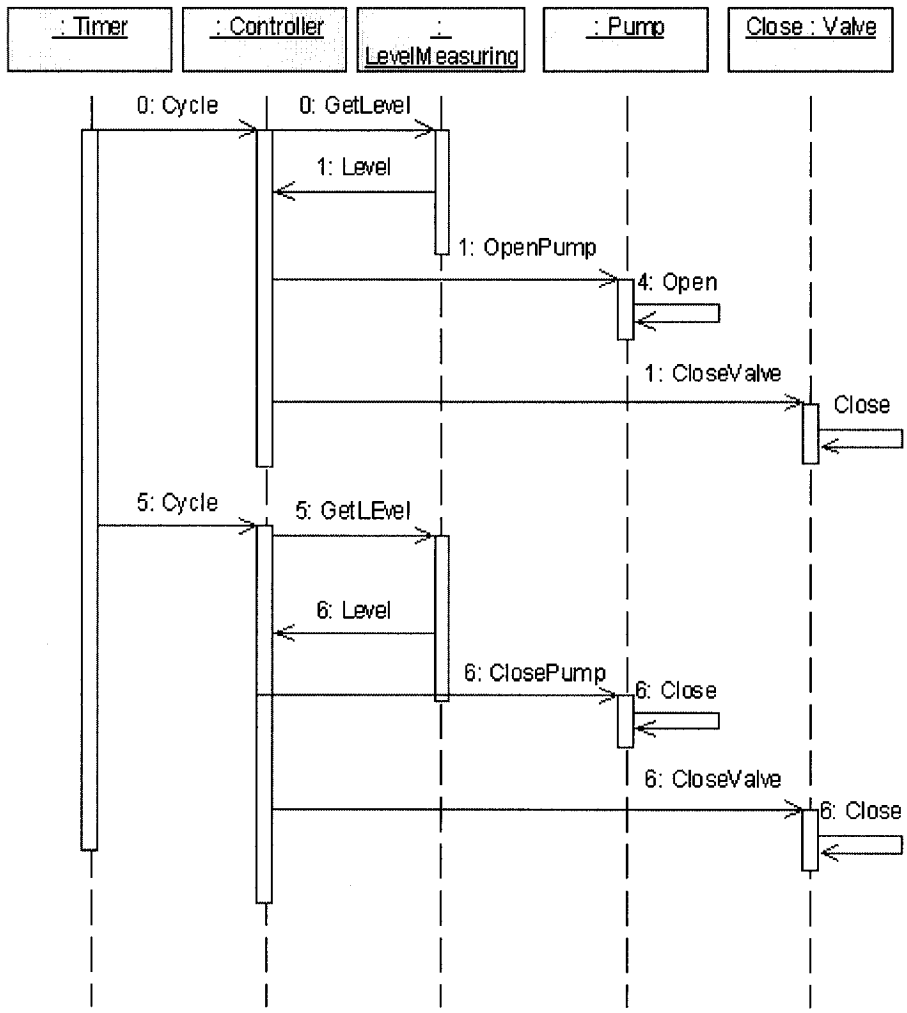

Figure 27: Sequence diagram

Chapter 6

Visualization Animation Tool

This chapter introduces developed Visualization Animation. It gives an overview about the graphical user interface and gives examples of different real-time reactive case studies used to test the tool.

6.1 VAT

There are two different ways to run the Visualization Animation Tool. The first way is during the simulation process when the simulator instructs VAT to visualize and animate the current simulation results. The second way is disconnected from the simulator and allows the user to run different kinds of real-time reactive case studies that were designed using TROM formalism and simulated using the TROMLAB Framework previously. In this chapter, only the second way of running VAT is presented because the concentration is on VAT and not on the simulation process; also, all that presented applies to the connected mode where the same interfaces and components are used to visualize and animate in both modes.

6.1.1 The Start Page

When the user opens the tool's web page, he will get the interface shown in Figure 28. The interface includes a list box and command button. The list box contains four

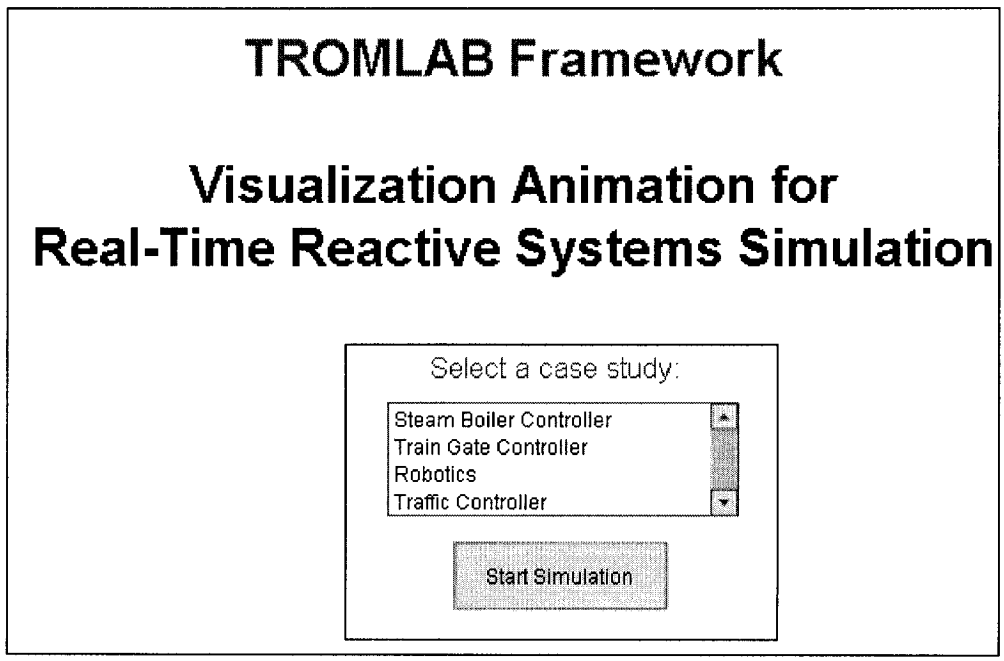

Figure 28: VAT start page

case studies that the user can select any one of them to see a visualization animation of their simulation. After selecting the case study, the user should press the Start Simulation command button to start the visualization process.

6.1.2 The Simulation Details

Figure 29 shows a snap shot of the visualization animation tool. The interface is divided into several parts each of which is presenting different type of information and contributes to painting the big picture of the simulation status. The rules of notation, grouping, and abstraction that aids in achieving the highest possible degree of understandability are applied in the design. Different notations are used to present different types of information. Also, related information are grouped into relatively closed and framed sections and separated from the others. Moreover, abstraction is used and only the important and relative information are presented and any unnecessary distracting things are avoided. Details about these will be clarified in the

following parts of this chapter. The interface consists of three sections: TROM objects with their sequence of active states that participated in the simulation results, current interaction part, and details section. Details about each section are described next. The Details section is provided for any future improvement to display details about the current event or any help information regarding what is displayed in the current status section.

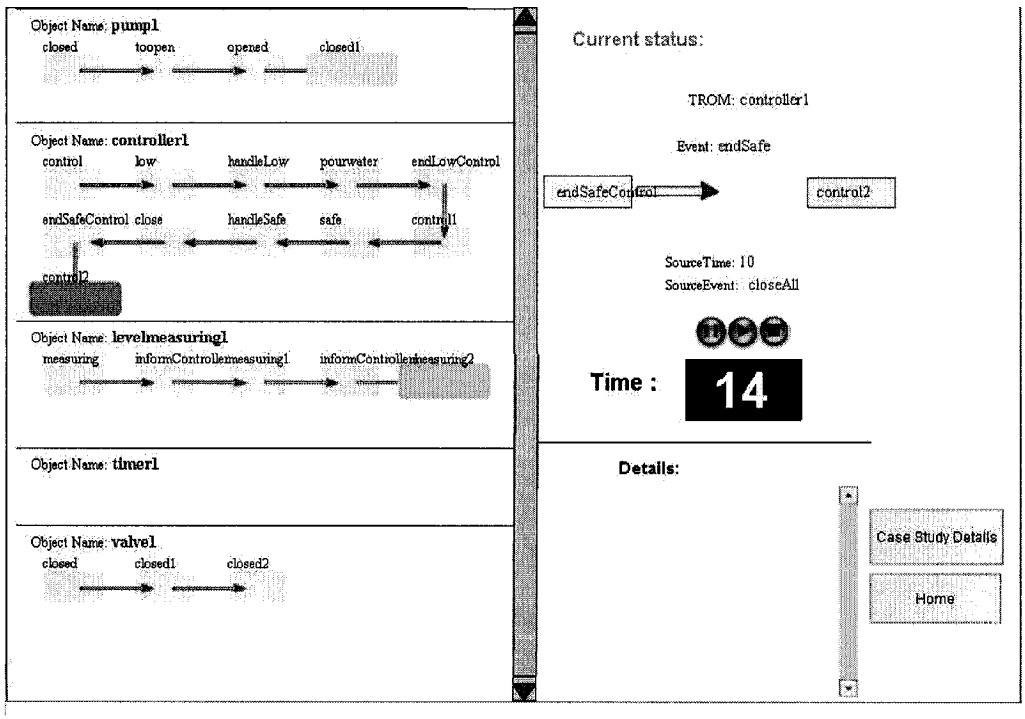

Figure 29: Simulation Visualization Animation

6.1.3 The TROM Objects Section

The interface consists of two parts divided by the scrollbar. The left part contains the TROM Objects participating in the simulated scenario of the presented case study. For each one of those objects, the active part of the state chart diagram is presented to describe the behavior of that object using the state transitions happened during the simulated scenario. Figure 30 shows a sample object state chart diagram.

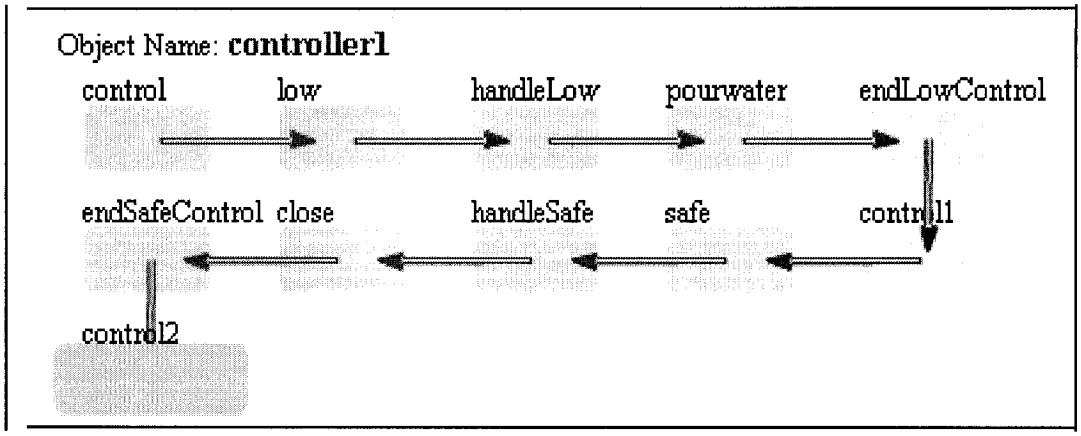

Figure 30: TROM object state chart diagram

The name of the TROM object is printed in the left most top part of each section. States are visualized as yellow rectangles. Each state has its name printed right on top of it. The state transition is visualized as an arrow starting from the source state and ending at the destination state. The use of those notations helps the user understand the presented information easily.

The first state in each section represents the initial state of the relative TROM object and the last state represents the current object state. The current state is always blinking so that the user can on the glance know in which state the relative TROM object is.

If there are more TROM objects than what the interface can present, the scrollbar can be used to navigate between the state charts up and down.

6.1.4 The Current Status Section

On the right hand side of the scrollbar, the current interaction during the system simulation is presented. There are two different possible types of interaction: object interaction through shared events and state transition.

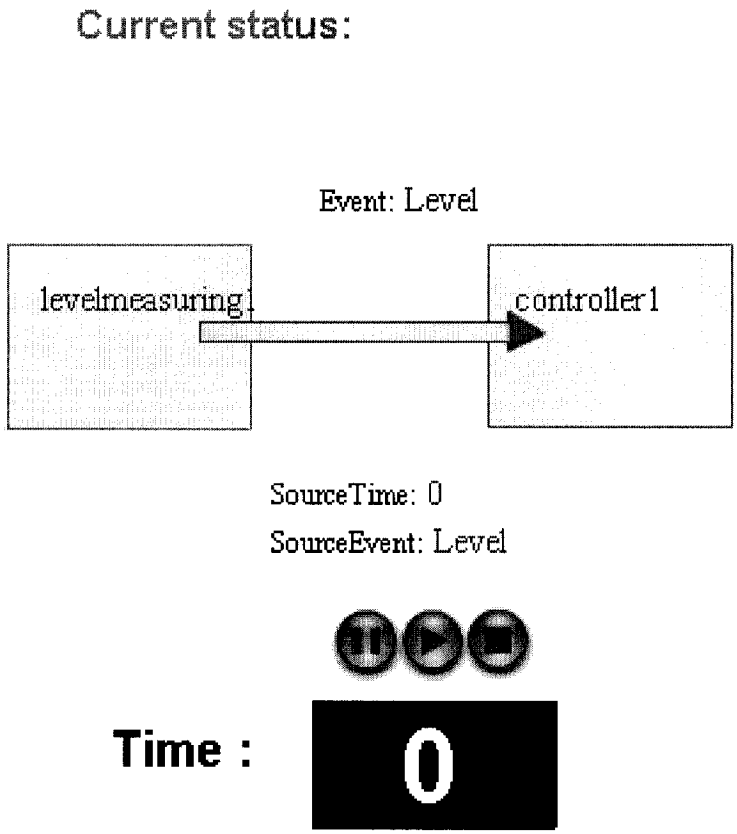

Figure 31: TROM object state chart diagram

Figure 31 shows two TROM objects, levelmeasuring1 and controller1, interacting with each other by the shared event "Level". The interaction details presented along with the two objects are:

- Event name: the shared event that is sent from the source to the destination object.
- Source Time: the time at which the source event synthesizing this event happened.
- Source Event: the source event that caused this event to happen, more details will be provided in the state transition.

- Time: the current time. Because the time plays an important role in the interactions of real-time reactive systems, the time is presented in a big font. Concept of color contrast, using black box and white colored number, is applied in the presentation of time to make it clear and hence grasped at a glance.

Three buttons are provided to pause, play, and stop the interaction in order to give the user controllability over the animation process.

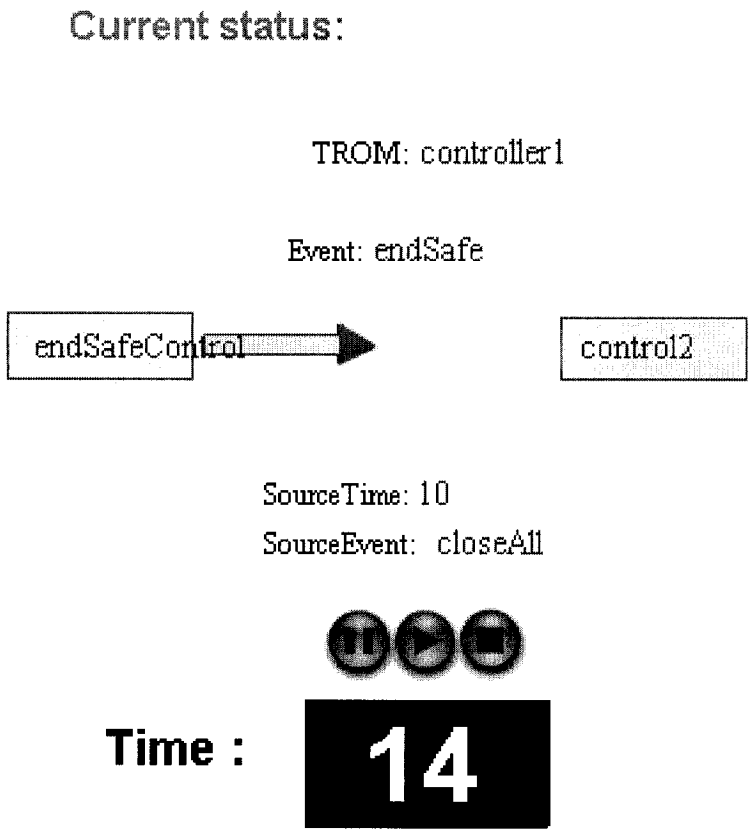

Figure 32: TROM object state chart diagram

The second type of interaction is the state transition and is shown in Figure 32.

The transition consists of source and destination states along with textual descriptions

to present the details of the transition event. The text includes:

- TROM: the object at which the current transition is happening.
- Event: the event that is causing the transition to happen.
- Source Event: the event that triggered the current event to happen. This means that the after the source event happened at the source time, this event was triggered to happen as a consequence. For example, if the event CloseAll happens at time 10, the endSafe event will be triggered to happen at time 14.
- Source Time: the time at which the source event occurred.

After introducing the VAT, 4 case studies will be used to test the functionality of the VAT and prove its capability in visualizing the simulation results of different types of real-time reactive systems.

Chapter 7

Testing

7.1 Testing Case Studies

The main purpose behind the design and implementation of this tool is to create a general mechanism to visualize and animate simulated real-time reactive systems using TROMLAB. This goal motivated the consideration of having the flexibility of handling different kinds of case studies with different complexities and specifications. Beside the Steam Boiler Controller case study, VAT has been tested on visualizing and animating the simulation of 3 other well known real-time reactive case studies: Railroad crossing, Robotic Assembly, and Traffic Control. The following sections will introduce briefly each case study and show a sample VAT snapshot during the process of visualizing and animating it.

7.2 Railroad Crossing Case Study

This problem was introduced by [HL94] as a benchmark for comparing different approaches for specifying and verifying real-time systems. A modeling solution for this case study was introduced by [MAKS00]. The problem definition includes trains passing through gates by coordinating with controllers. The trains communicate with the

controllers by sending messages, and the controllers control the gates by issuing commands. When a train approaches a gate, it sends a message to the controller of that gate. The controller, in turn, instructs the gate to close. After the train has passed through the gate, it sends another message to the controller, who then commands the gate to open. Timing constraints are defined for the communication messages and event firing. The complexity of this problem lies in the system configuration. The solution allows different numbers of trains to be defined interacting with different number of controllers and gates. This flexibility hides a relative complexity in the design. By running this model in VAT, it proves its ability to handle and process

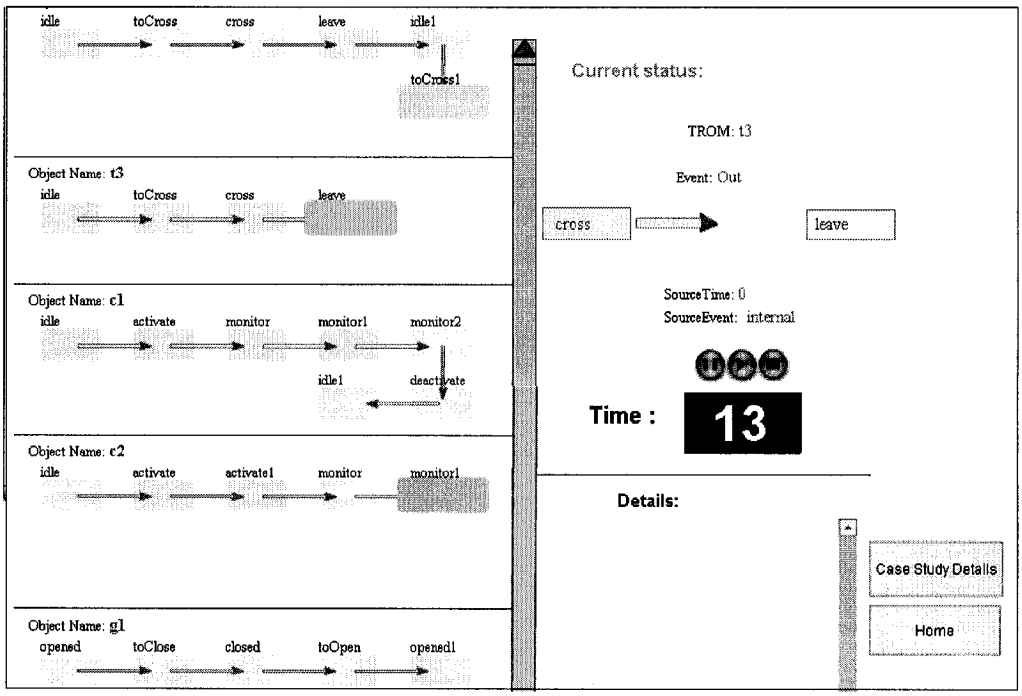

Figure 33: Railroad Crossing Visualization Animation

any real-time reactive system no matter how complex the system configuration is in its formalized model. Figure 33 shows a screen shot of the visualization animation of the simulation of the Railroad Crossing problem for a configuration of three trains, two controllers, and two gates.

7.3 Robotic Assembly Case Study

A design for this problem was introduced in [AM00] and reviewed in [Liu03]. The assembly unit consists of a user, a conveyor belt, a vision system, a robot with two arms, and a tray for assembling. The user places two kinds of parts: a dish and a cup onto the conveyor belt. Then the belt conveys the parts toward the vision system. Whenever a part enters the sensor zone, the vision system detects it and informs the belt to stop immediately. Next, the vision system recognizes the type of the part and communicates to the robot so that the robot can pick it up from the stopped conveyor belt. After the robot picks up a part, the belt resumes moving. The assembly finishes when a dish and a cup are separately placed in the tray by two arms of the robot. Figure 34 shows the simulation of the Robotic Assembly.

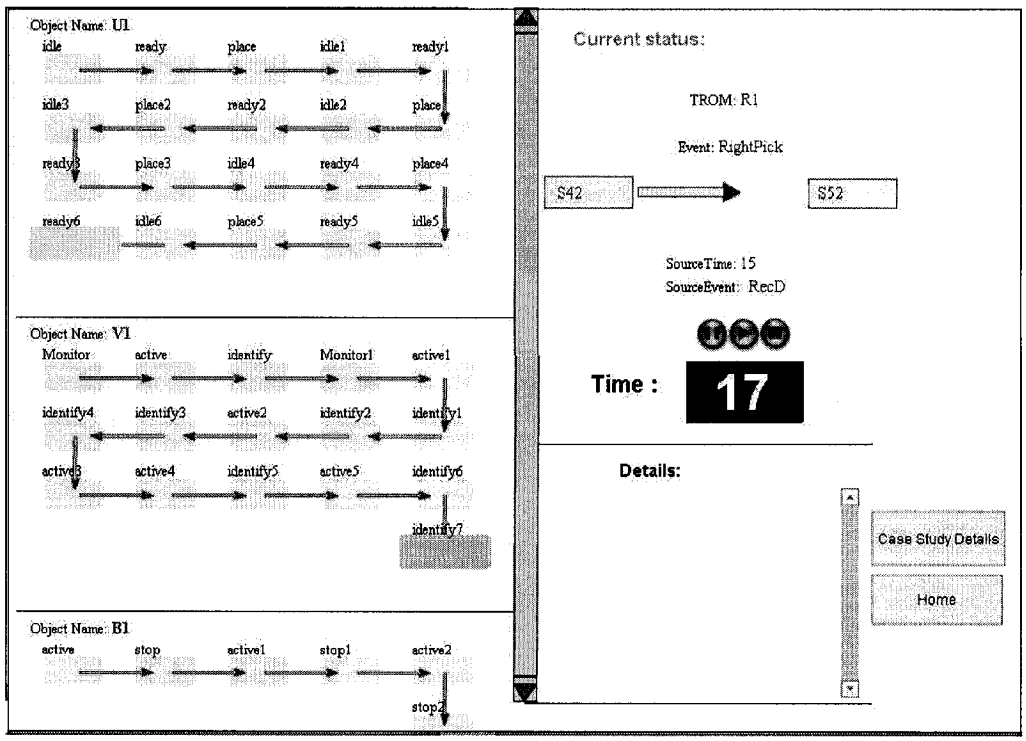

Figure 34: Robotic Assembly Visualization Animation

7.4 Traffic Control Case Study

A model for Autonomous Traffic Control systems was described in [AM03]. The traffic model consists of divided highways running in perpendicular directions. At the proximity of the intersection between any two highways, each road is divided into several lanes. There are three lanes for incoming traffic in each of the northbound, southbound, eastbound, and westbound directions, and three such lanes for outgoing traffic. In every direction, vehicles in the right lane turn right, vehicles in the middle lane go straight, and vehicles in the left lane turn left. The intersection is a shared resource that is allocated by the traffic controller to vehicles in such a way that every vehicle at the intersection obtains the resource within a finite amount of time; there is neither deadlock nor starvation and vehicles do not collide while crossing the intersection. The complexity of this model lies in the design that handles a dynamic resource allocation for vehicles. By being able to visualize and animate such case

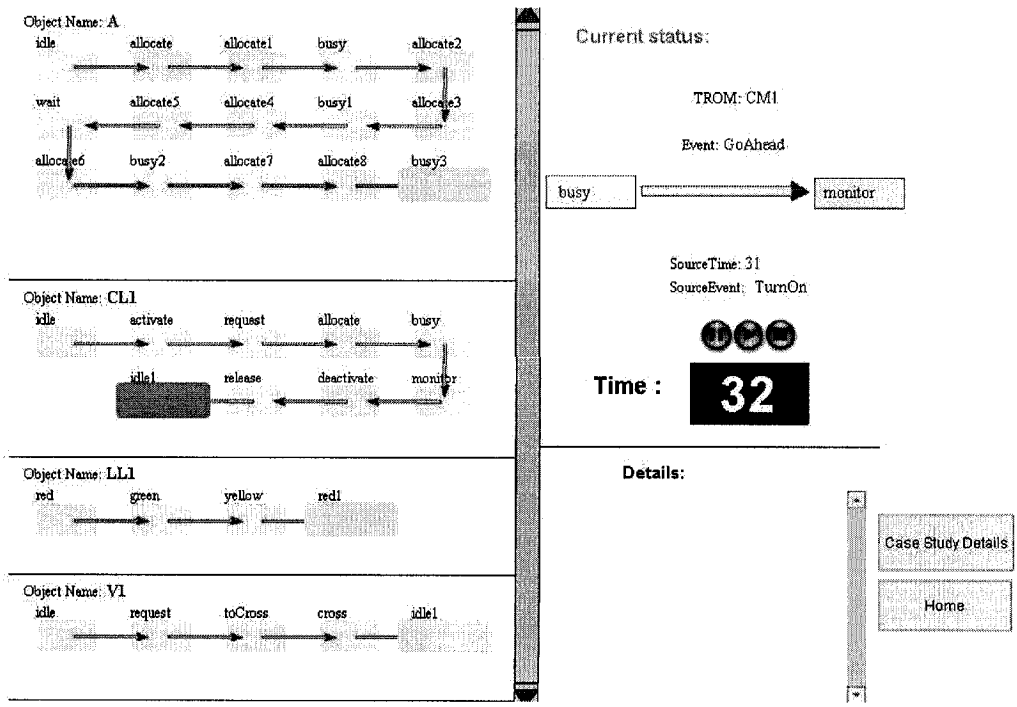

Figure 35: Traffic Control Visualization Animation

study, VAT is proving to be able to handle any real-time reactive system simulation no mater how complex the design is. Figure 35 shows a snapshot of the visualization animation of the Traffic Control case study.

After testing the VAT, it will be compared to related works in the same field.

Chapter 8

Related Work

The domain of Real-Time reactive systems has become a rich area for research and study because of the vast development and usage of such systems in the daily life. Therefore, many institutes and research labs have been working on the validation of real-time system design. The interest resulted in the development of different tools to simulate the expected behavior of the produced design before implementing it. This section introduces two examples of related work in the areas of simulation and visualization of real-time reactive systems along with comparison with VAT. Furthermore, the understandability guidelines provided in this thesis are applied to the Rational Rose Real-Time visual modeling.

8.1 CD++ Real-Time Simulation

CD++ is a toolkit developed to implement specifications of DEVS formalism, Discrete EVents Systems specifications [GW03]. DEVS formalism provides a framework for building layered real-time models. Beside the toolkit, simulation mechanisms were developed to execute the DEVS independent from the models. During the execution of simulation processes, models receive external events within a given deadlines from an external event file through input ports. The simulator keeps track of missed deadlines and worst case response time of the specifications. Finally, the simulation results are

produced in an output file. Figure 36 shows an example simulation result output file taken from [GW03] for an Alarm Clock model. The abstraction of simulation

actual time	message time	port	value
01:00:000	01:00:000	DISPLAY_TIME	00:01
02:00:000	02:00:000	DISPLAY_TIME	00:02
03:00:000	03:00:000	DISPLAY_TIME	00:03
...			
30:00:000	30:00:000	DISPLAY_TIME	00:30
30:00:000	30:00:000	BUZZER_ON	1
31:00:000	31:00:000	DISPLAY_TIME	00:31
32:00:000	32:00:000	DISPLAY_TIME	00:32

Figure 36: Simulation Output

result representation makes the tool capable of simulating various real-time models. However, the usability of the results is very low compared with the usability of VAT that uses graphical metaphors to represent the interaction between entities.

8.2 Simulation of Steam Boiler

A simulation with graphical visualization for the steam boiler was implemented by [Lot96]. The tool simulates the behavior of the simulator while reacting to different messages that can be controlled by the user. The Tool Command Language (Tcl) and the Widget Toolkit were used to implement the simulation. Control panels are provided to allow the user to control and direct the execution of the simulation. Figure 37 show a snapshot of the steam boiler taken from [Lot96].

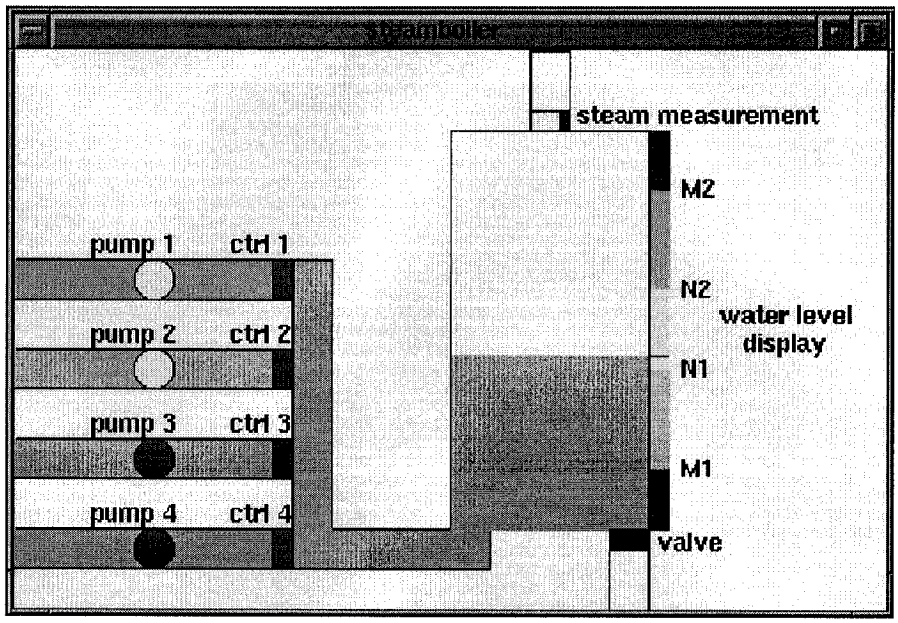

Figure 37: Steam Boiler Simulator

From the Figure we can see that the usability and understandability of the tool is very high; the user can perceive easily the status of the system. However, this tool is very specific and is used only to visualize the steam boiler while VAT, on the other hand, is very general and can be used to visualize and animate any real-time system.

8.3 Rational Rose RealTime

IBM Rational Rose RealTime is a software development environment developed to model real-time system designs using the Unified Model Language(UML), generate implementation code using several programming languages such as C++ and Java, run the system, and debug the whole system application with visual presentation. Hence, it supports the whole system development life cycle. Some of the types included in Rational Rose RealTime are:

- Capsules: the fundamental modeling basis of UML2.0 "classes with structure"

that encapsulates all its attributes and operations and keep them logically protected [Wor03].

- Ports: the medium for communication between capsules. Ports connect capsules together allowing messages to be sent from one capsule to another.
- Protocols: interaction messages are defined in protocols.
- States.
- Transitions.
- Events.

By applying the understandability guidelines that are introduced in this thesis to the visual modeling of Rational Rose RealTime, we got the following results:

- "Every TROM Object and State should be presented in a way so that it is clearly identified and perceived by the user.": Each type is presented in a different visual notation than the other. Rational Rose RealTime uses the standard UML notations to present each type. Therefore, it supports the guideline of using different notations for different concepts so that the user can understand the presented information easily.
- "The relationship between TROM Objects and their corresponding states should be made clear.": States for a particular capsules are grouped in state chart diagrams which support the guideline of easing the perception of the relationship between objects and states. VAT, on the other hand, groups the states of each object in one frame so that the user can on a glance understand the relationship between the states and the TROM Object.
- "States of a particular TROM Object should be presented in the same sequence of execution that happened during the simulation process.": During the execution of the model, all the states of a particular capsule are presented. The

execution shows the transition from one state to another. The difference between this presentation and the one provided in the VAT is that the Rose RealTime shows all the states while the VAT shows only the active states that participated in the sequence of execution. The tradeoff is that showing only the active states reduces the mental overhead on the user while monitoring the state transitions. On the other hand, showing all the states will allow the user to see the other transition possibilities that didn't happen in the execution.

- "The Type of event should be identified.": Rose RealTime allows the user to debug through multiple capsules using a debugging facility; therefore, the user can monitor the shared events that allow the communication between two capsules through the defined ports.

- "The causing event should be mentioned.": Rose RealTime has event traces that allow the user to trace message transmission; hence, it facilitates the identification of the causing event. In VAT, the causing event is clearly presented with a caption titled SourceEvent so that it will be easy for the user to understand it from the presentation.

- "Time should be always presented.": Rose RealTime allows the user to know the time at which each event has happened during the model execution using the tracing facilities. On the other hand, VAT clearly presents the time in a clear format to enable the user to find it easily.

- "System's current status and history should be always available to the user.": The debugging and tracing facilities allows the user to know the current status of the model and the history of interactions between objects. Also, the current state is displayed with a frame to help the user identify it easily. In VAT, the current state is presented using blinking colors.

- "Help and support should be available to the user.": Help and support are provided through tutorials, examples, and animated demos to aid the user in

understanding the presented information. In VAT, help facility is provided by giving a legend page that describe the different notations used.

From the previous results we conclude that the stated understandability guidelines are applied in Rational Rose RealTime. Differences of applying the guidelines are found between Rose RealTime and VAT due to the differences in the complexity of the two applications. The debugging and tracing facilities that are provided in Rose RealTime do exist in the TROMLAB framework Simulator tool.

Chapter 9

Conclusion

This thesis worked on improving the usability of TROMLAB tools. General guidelines for solving the understandability problems for real-time reactive systems simulation results were introduced. Those guidelines can be used not only for TROMLAB environment but also for any other Real Time Reactive environment.

A new Visualization Animation tool was designed and implemented by this thesis to complete the TROMLAB Framework architecture and to aid in the improvement of TROMLAB quality in use. This tool can be used to simulate the results of any Real Time Reactive simulation provided that it conforms to the defined XML schema for translating simulation results. Hence, the implementation of this tool will take TROMLAB steps forward to communicate with other research labs in the same field of study. Also, the work produced in this thesis will encourage further utilization of TROMLAB simulation because of the usability improvements achieved.

9.1 Future Work

9.1.1 Simulator

The current version of Simulator contains a very powerful simulation, querying, and reasoning facilities that were developed iteratively by TROMLAB research members. The limitation of it is that it can be used only through the defined GUI. This means that in order to get use of those tools the Real Time Reactive practitioner should have the source code of those tools in hand and run it on windows environment. This thesis suggests converting those powerful tools into web services that can be used by any other tool or user. This can be done by building a web service interface that can handle any request from any source. The interface should be able to receive input TROM formalism and produce output XML simulation results remotely from any web source. This improvement will take TROMLAB steps ahead for internationalizing its components and benefiting other labs from its resources.

9.1.2 VAT

When the suggested work in the previous section is built, VAT can serve as the GUI interface that can be used by both web and windows users to interact with the back end TROMLAB tools. Additional functionalities such as querying the simulation and getting more details about the current interaction and event specification can be implemented.

Bibliography

[AAG93] G. Abowd, R. Allen, and D. Garlan. Using style to understand descriptions of software architecture., 1993.

[AAM96] V. S. Alagar, R. Achuthan, and D. Muthiayen. Tromlab : A software development environment for real-time reactive systems. October 1996.

[Abr91] J.-R. Abrial. Steam boiler control specification problem, August 1991.

[AM98] V. S. Alagar and D. Muthiayen. Specification and verification of complex real-time reactive systems modeled in uml. Submitted for publication in IEEE Transactions on Software Engineering (Being revised), July 1998.

[AM00] V.S. Alagar and D. Muthiayen. Towards a mechanical verification of real-time reactive systems modeled in uml, 2000.

[AM03] V.S. Alagar and D. Muthiayen. A rigorous approach to modeling autonomous traffic control systems. 2003.

[Bha99] V. Bhaskaran. Graphical user interface for tromlab environment. Master's thesis, Department of Computer Science, Concordia University, Montreal, Canada, December 1999.

[Cio91] Frank A. Cioch. Measuring software misinterpretation. *Journal of Systems and Software*, 14(2):85–89, 1991.

[GH93] J. V. Guttag and J. J. Horning. *Larch: Languages and Tools for Formal Specifications.* Springer Verlag, 1993.

[GS94] D. Garlan and M. Shaw. An introduction to software architecture, January 1994.

[GW03] Ezequiel Glinsky and Gabriel Wainer. Definition of real-time simulation in the cd++ toolkit, 2003.

[Hai99] G. Haidar. Simulated reasoning and debugging of tromlab environment. Master's thesis, Department of Computer Science, Concordia University, Montreal, Canada, December 1999.

[HL94] C. Heitmeyer and N. Lynch. The generalized railroad crossing: A case study in formal verification of real-time systems. volume RTSS94, pages 120–130, San Juan, Puerto Rico, December 1994. 15th IEEE Real-Time Systems Symposium.

[IEE98] Draft for Standard IEEE. Ieee p1471/d5.1 draft recommended practice for architectural description, December 1998.

[Kru95] Philippe Kruchten. Architectural blueprintsthe 4+1 view model of software architecture, November 1995.

[Liu03] S. Liu. Simulated validation of real-time reactive systems with parameterized events. Master's thesis, Department of Computer Science, Concordia University, Montreal, Canada, August 2003.

[Lot96] Annette Lotzbeyer. Simulation of a steam-boiler, 1996.

[MAKS00] D. Muthiayen, V.S. Alagar, F. Khendek, and A. Sefidcon. An approach to a synthesis of formal and visual description techniques for the development of real-time reactive systems, 2000.

[Mir98] B. Mirel. Visualizations for data exploration and analysis:a critical review of usability research. *Technical Communication*, 45(4), 1998.

[Mut96] D. Muthiayen. Animation and formal verification of real-time reactive systems in an object-oriented environment. Master's thesis,

Department of Computer Science, Concordia University, Montreal, Canada, October 1996.

[OCB00] M. A. Ogush, D. Coleman, and D. Beringer. A template for documenting software and firmware architectures, March 2000.

[Sri99] V. Srinivasan. An intelligent graphical interface system for tromlab. Master's thesis, Department of Computer Science, Concordia University, Montréal, Canada, March 1999.

[Tao96] H. Tao. Static analyzer: A design tool for trom. Master's thesis, Department of Computer Science, Concordia University, Montreal, Canada, August 1996.

[Wil94] Lloyd G. Williams. Assessment of safety-critical specifications. 11(1):51–60, 1994.

[Wor03] Developer Works. Ibm rational rose realtime: A guide for evaluation and review, 2003.

Appendix A

Simulation Results in XML Formt

the following example presents the XML format for the Steam Boiler Controller case study's simulation results for the scenario of having a alow level of water at the first cycle of execution and a safe water level in the second cycle.

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <simulation>
    - <initial-states>
        <initial trom="pump1" state="closed" />
        <initial trom="controller1" state="control" />
        <initial trom="levelmeasuring1" state="measuring" />
        <initial trom="timer1" state="idle" />
        <initial trom="valve1" state="closed" />
    </initial-states>
    - <tromevent>
    - <event-set>
        - <event trom="levelmeasuring1"time="0">
        <name>Level</name>
            - <state-set>
            <state trom="pump1">closed</state>
            <state trom="controller1">control</state>
            <state trom="levelmeasuring1">informController</state>
            <state trom="timer1">idle</state>
            <state trom="valve1">closed</state>
            </state-set>
        <cause trom="user" time="0" causing="" />
        </event>
```

```
- <event trom="levelmeasuring1" time="0">
<name>Sendlevel</name>
    - <state-set>
    <state trom="pump1">closed</state>
    <state trom="controller1">control</state>
    <state trom="levelmeasuring1">measuring</state>
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="" time="0" causing="internal" />
</event>
- <event trom="controller1" time="0">
<name>Level</name>
    - <state-set>
    <state trom="pump1">closed</state>
    <state trom="controller1">low</state>
    <state trom="levelmeasuring1">measuring</state>
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="levelmeasuring1" time="0" causing="Level" />
</event>
- <event trom="controller1" time="0">
<name>increase</name>
    - <state-set>
    <state trom="pump1">closed</state>
    <state trom="controller1">handleLow</state>
    <state trom="levelmeasuring1">measuring</state>
    <state trom="timer1">idle</state>
```

```
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="" time="0" causing="internal" />
</event>
- <event trom="controller1" time="0">
<name>OpenPump</name>
    - <state-set>
    <state trom="pump1">closed</state>
    <state trom="controller1">pourwater</state>
    <state trom="levelmeasuring1">measuring</state>
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="controller1" time="0" causing="increase" />
</event>
- <event trom="pump1" time="0">
<name>OpenPump</name>
    - <state-set>
    <state trom="pump1">toopen</state>
    <state trom="controller1">pourwater</state>
    <state trom="levelmeasuring1">measuring</state>
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="controller1" time="0" causing="OpenPump" />
</event>
- <event trom="controller1" time="0">
<name>CloseValve</name>
    - <state-set>
```

```
    <state trom="pump1">toopen</state>
    <state trom="controller1">endLowControl</state>
    <state trom="levelmeasuring1">measuring</state>
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="controller1" time="0" causing="OpenPump" />
</event>
- <event trom="valve1" time="0">
<name>CloseValve</name>
    - <state-set>
    <state trom="pump1">toopen</state>
    <state trom="controller1">endLowControl</state>
    <state trom="levelmeasuring1">measuring</state>
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="controller1" time="0" causing="CloseValve" />
</event>
- <event trom="pump1" time="1">
<name>open</name>
    - <state-set>
    <state trom="pump1">opened</state>
    <state trom="controller1">endLowControl</state>
    <state trom="levelmeasuring1">measuring</state>
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="pump1" time="0" causing="OpenPump" />
```

```
</event>
- <event trom="controller1" time="3">
<name>endLow</name>
    - <state-set>
    <state trom="pump1">opened</state>
    <state trom="controller1">control</state>
    <state trom="levelmeasuring1">measuring</state>
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="controller1" time="0" causing="increase" />
</event>
- <event trom="levelmeasuring1" time="10">
<name>Level</name>
    - <state-set>
    <state trom="pump1">opened</state>
    <state trom="controller1">control</state>
    <state trom="levelmeasuring1">informController</state>
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="user" time="0" causing="" />
</event>
- <event trom="levelmeasuring1" time="10">
<name>Sendlevel</name>
    - <state-set>
    <state trom="pump1">opened</state>
    <state trom="controller1">control</state>
    <state trom="levelmeasuring1">measuring</state>
```

```
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="" time="0" causing="internal" />
</event>
- <event trom="controller1" time="10">
<name>Level</name>
    - <state-set>
    <state trom="pump1">opened</state>
    <state trom="controller1">safe</state>
    <state trom="levelmeasuring1">measuring</state>
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="levelmeasuring1" time="10" causing="Level" />
</event>
- <event trom="controller1" time="10">
<name>closeAll</name>
    - <state-set>
    <state trom="pump1">opened</state>
    <state trom="controller1">handleSafe</state>
    <state trom="levelmeasuring1">measuring</state>
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="" time="0" causing="internal" />
</event>
- <event trom="controller1" time="10">
<name>ClosePump</name>
```

```
    - <state-set>
    <state trom="pump1">opened</state>
    <state trom="controller1">close</state>
    <state trom="levelmeasuring1">measuring</state>
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="controller1" time="10" causing="closeAll" />
</event>
- <event trom="pump1" time="10">
<name>ClosePump</name>
    - <state-set>
    <state trom="pump1">closed</state>
    <state trom="controller1">close</state>
    <state trom="levelmeasuring1">measuring</state>
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
<cause trom="controller1" time="10" causing="ClosePump" />
</event>
- <event trom="controller1" time="10">
<name>CloseValve</name>
    - <state-set>
    <state trom="pump1">closed</state>
    <state trom="controller1">endSafeControl</state>
    <state trom="levelmeasuring1">measuring</state>
    <state trom="timer1">idle</state>
    <state trom="valve1">closed</state>
    </state-set>
```

```
            <cause trom="controller1" time="10" causing="ClosePump" />
            </event>
            - <event trom="valve1" time="10">
            <name>CloseValve</name>
                - <state-set>
                <state trom="pump1">closed</state>
                <state trom="controller1">endSafeControl</state>
                <state trom="levelmeasuring1">measuring</state>
                <state trom="timer1">idle</state>
                <state trom="valve1">closed</state>
                </state-set>
            <cause trom="controller1" time="10" causing="CloseValve" />
            </event>
            - <event trom="controller1" time="14">
            <name>endSafe</name>
                - <state-set>
                <state trom="pump1">closed</state>
                <state trom="controller1">control</state>
                <state trom="levelmeasuring1">measuring</state>
                <state trom="timer1">idle</state>
                <state trom="valve1">closed</state>
                </state-set>
            <cause trom="controller1" time="10" causing="closeAll" />
            </event>
        </event-set>
        </tromevent>
    </simulation>
```