

**A VISUAL MODELING TOOL FOR THE
DEVELOPMENT OF TRUSTWORTHY
COMPONENT-BASED SYSTEMS**

YUN ZHOU

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

APRIL 2009
© YUN ZHOU, 2009



Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*
ISBN: 978-0-494-63312-0
Our file *Notre référence*
ISBN: 978-0-494-63312-0

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.


Canada

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Yun Zhou**

Entitled: **A Visual Modeling Tool for the Development
of Trustworthy Component-based Systems**

and submitted in partial fulfillment of the requirements for the degree of
Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
_____ Examiner
_____ Examiner
_____ Examiner
_____ Supervisor

Approved _____
Chair of Department or Graduate Program Director

_____ 20 _____

Dr. Nabil Esmail, Dean
Faculty of Engineering and Computer Science

Abstract

A Visual Modeling Tool for the Development of Trustworthy Component-based Systems

Yun Zhou

A rigorous development of Trustworthy Computing System (TCS) is an active research area in TROMLAB research group since TCS concept was initiated in 2002 by Microsoft Corporation. In TROMLAB research group a component-based development of TCS was initiated in 2005. Several tracks of research activities, including formal specification of components and component-based systems, formal assessment of trustworthiness properties, and a framework construction for developing trustworthy systems are in different stages of development, and completion. It is in this context this thesis has evolved. It contributes a Visual Modeling Tool (VMT), the front-end to the development framework, in which the developer can construct visual models of system components without being burdened by complex formalisms. The thesis identifies the functional and performance requirements of the VMT tool from the goal of TCS research in TROMLAB, provides a detailed design which itself is based on component technology, and illustrates with two case studies, CoCoME and Mine Drainage, the modeling steps and its user-centric features.

Acknowledgments

I would like to express my sincere thanks to all those who supported and encouraged me during my studies and during the critical stages of complete this thesis. It would not have been possible for me to be here without their help.

First of all, I am deeply indebted to my supervisor, Dr. Vangalur Alagar, for getting me into TROMLAB to commence this thesis. His stimulating ideas and suggestions, as well as his constant support, patience and encouragement helped me in all the time of doing research and writing this thesis.

I owe my sincere gratitude to my colleagues in TROMLAB. Especially I want to thank Mubarak Mohammad, who offered valuable suggestions for improvement of the Visual Modeling Tool and also gave me helpful tutorials to guide me into TROMLAB. I also want to thank Naseem Ibrahim for all his assistance on testing the VMT and helping me with the case studies.

I am also grateful to my best friend Ying Fang. Her support and encouragement was of great help in my difficult times. Finally, I would like to give my special thanks to my parents and my fiance Yifan Sun for their support of my decision to study in Canada to commence this thesis, and their persistent love enabled me to complete this thesis.

Contents

List of Figures	viii
List of Tables	xi
1 Introduction	1
1.1 Trustworthy Computing Systems Development	3
1.1.1 Real-Time Reactive System(RTRS)	3
1.1.2 Trustworthiness Credentials	5
1.1.3 Component Based Development (CBD)	6
1.1.4 Scope of the Thesis	7
2 Basic Concepts-TADL	12
2.1 Introduction to Meta-architecture and TADL	12
2.2 Component Definition	15
2.3 Safety Contract	16
2.4 Component Architecture	20
2.5 Security mechanism	24
2.6 System Definition	28
3 Basic Concepts-XML Schema	29
3.1 Introduction	29
3.2 InterfaceType Schema	30
3.3 ComponentType Schema	34
3.4 ConnectorType Schema	39
3.5 ContractType Schema	42
3.6 PackageType Schema	48

3.7	RBAC Schema	50
3.8	System Schema	57
4	VMT Architecture: System Requirements	62
4.1	Introduction	62
4.2	Motivation	62
4.3	Purpose and Context	63
4.3.1	Design Stage	65
4.3.2	Implementation Stage	68
4.3.3	Run-time Stage	69
4.4	System Requirements of VMT	69
4.4.1	<i>Navigator Interface:</i>	71
4.4.2	<i>Palette Interface:</i>	71
4.4.3	<i>System Editor Interface:</i>	72
4.4.4	<i>Properties Editor Interface:</i>	74
4.4.5	<i>Error Message Interface:</i>	74
4.4.6	<i>Menu Bar:</i>	76
4.4.7	<i>Tool bar:</i>	79
4.5	Non-Functional Requirements of VMT	79
5	VMT Architecture: Design	81
5.1	Architectural Overview	81
5.1.1	Swing's MVC Architecture	82
5.1.2	Rationale behind selection	84
5.1.3	Architecture Diagram	84
5.2	Components	90
5.2.1	Navigator Component	92
5.2.2	Palette Component	93
5.2.3	System Editor Component	96
5.2.4	Properties Component	111
5.2.5	ErrorMessagePanel	117
5.2.6	MainMenuBar	117
5.2.7	MainToolBar	119
5.3	Development Platform	119

6	Modeling a Trustworthy System using VMT	121
6.1	GUI Overview	121
6.2	Navigator – refer to Section 4.4.1	121
6.3	Palette – refer to Section 4.4.2	126
6.4	System Editor – refer to Section 4.4.3	126
6.4.1	System Canvas	127
6.4.2	Realtime View	130
6.4.3	Trustworthy View:	130
6.4.4	Configuration View:	141
6.4.5	TADL Source View:	144
6.5	Properties Editor View – refer to Section 4.4.4	144
6.5.1	Component Properties Editor:	144
6.5.2	Interface Attributes Editor:	149
6.5.3	Connector/Connector Role Attributes Editor:	151
6.6	Error Message View:	151
6.7	Toolbar:	152
6.8	Menubar:	152
7	Conclusion and Future Work	153
7.0.1	Future Implementation Work	155
	Bibliography	156

List of Figures

1	Component Template	9
2	The building blocks of the meta-architecture	13
3	The component definition and specification	15
4	Safety Contract	16
5	Component Architecture	20
6	How two components are connected	22
7	Security Mechanism	25
8	System Definition	28
9	Framework for Developing TRTS - Context Diagram	64
10	Usecase Diagram of VMT	70
11	Model-View-Controller Architecture	83
12	Swing Architecture	83
13	The high-level Architecture Diagram	85
14	GUI System Package Diagram	86
15	GUI High-level Class Diagram	87
16	The First Level System Object Structure	89
17	The Second Level System Object Structure	90
18	The class diagram of Navigator component	91
19	The class diagram of Palette component	94
20	The class diagram of System Editor component	97
21	The class diagram of Real Time Panel component	102
22	The class diagram of Data Constraint Panel component	104
23	The class diagram of Service Panel component	105
24	The class diagram of Safety Property component	106
25	The class diagram of Contract Component	106
26	The class diagram of RBAC Component	107

27	The class diagram of Configuration Component	108
28	The class diagram of Source Editor component	109
29	The class diagram of properties component	111
30	The class diagram of component properties	113
31	The class diagram of component contract properties	114
32	The class diagram of interface properties	115
33	The class diagram of event properties	116
34	The class diagram of connector/connector role properties	117
35	The class diagram of ErrorMessagePanel component	118
36	The class diagram of MainMenuBar component	118
37	The class diagram of MainToolBar component	119
38	VMT Overview	122
39	Snapshot of Navigator Panel	123
40	Snapshot to create a new project/system	124
41	Snapshot of creating project/system errors	124
42	Snapshot of Palette	126
43	Snapshot of System Canvas	127
44	Snapshot to create a new element	128
45	Snapshot for a connector error	128
46	Snapshot for an interface error	129
47	Snapshot for a role error	129
48	Snapshot of Real-Time Panel	130
49	Snapshot of Data Constraint Panel	131
50	Snapshot of Service Panel	132
51	Snapshot of Safety Property Panel	132
52	Snapshot of Contract Panel	133
53	Snapshot for RBAC definition	133
54	Snapshot of managing RBAC – user definition	134
55	Snapshot of managing RBAC – group definition	134
56	Snapshot of managing RBAC – role definition	135
57	Snapshot of managing RBAC – privilege definition	135
58	Snapshot for assigning a user to a group	137
59	Snapshot for assigning a user to a role	138

60	Snapshot for assigning a group to a role	139
61	Snapshot for assigning privilege of events to a role	140
62	Snapshot for assigning privilege of data parameters to a role	141
63	Snapshot for System Hardware Component definition	142
64	Snapshot for System Configuration definition	143
65	Snapshot for TADL Source Panel	145
66	Snapshot for managing the properties of a component - contract definition .	145
67	Snapshot for managing the properties of a component - architecture definition	146
68	Snapshot for contract definition - service definition	147
69	Snapshot for contract definition - safety property definition	147
70	Snapshot for safety property definition	148
71	Snapshot for service definition	148
72	Snapshot for real-time definition	149
73	Snapshot for data constraint definition	150
74	Snapshot for managing the properties of an interface - event definition . . .	150
75	Snapshot for managing the properties of a connector/connector role - name definition	151
76	Snapshot for Error Message Interface	152
77	Snapshot for Toolbar	152
78	Snapshot for Menubar	152

List of Tables

1	Services provided by Navigator Interface	72
2	Services provided by System Canvas Interface	73
3	Services provided by Real-time Interface	73
4	Services provided by Trustworthy Interface	75
5	Services provided by System Configuration Interface	76
6	Services provided by TADL Source Interface	76
7	Services provided by Component Properties Editor Interface	77
8	Services provided by Interface/Connector/Connector Role Properties Editor Interface	78
9	Services provided by Error Message Interface	78

Chapter 1

Introduction

Research in the development of *Trustworthy Computing Systems*(TCS) is relatively new. In January 2002 Microsoft published a paper [Gat02a] on *Trustworthy Computing*. In October 2002 a revised version of this paper was made available on the web [Gat02b]. This white paper brought out the challenges in building a TCS which in turn created immense interest in the research community. In early 2005 a research team under the supervision of Dr. Alagar started investigating rigorous approaches to modeling and development of TCS. The goal is to build an environment in which trustworthy systems can be formally defined, designed, implemented, and certified. My thesis builds a visual front-end to that environment.

It is human nature to trust a technology when it becomes so dependable that its internal technical details become totally irrelevant when it comes to their daily use in life. Automobile, electricity, and telephones are some classic examples of broadly trusted technologies in our everyday life, because they work as advertised and they are almost there without fail when we need them. In comparison to this situation, although computers and its services

have intruded into our daily lives, we are not yet in a position to trust them as we trust the technologies born out of 19th century inventions.

Computing paradigm, in spite of its pervasive nature, has not changed in the last 30 or 40 years. There is a great expectation that the TCS initiative will change the computing paradigm through a combination of engineering principles, business practices, and regulatory service provision. Abstracted, these are the principles governing *safety*, *reliability*, and *business integrity*. The challenge is to ensure that these qualities are inherent in the artifacts produced throughout the different stages of the software development process, and procedures followed during deployment of the system, and the management of the operational system.

The TCS research group working under the supervision of Dr. Alagar is undertaking research in the *component-based development* of TCS, which involve the following topics:

1. rigorous process model, formal architecture and its description
2. formal definition of trustworthiness properties
3. formal definition of trustworthy components, and composition of components
4. languages and methods for specification and design,
5. formal techniques for design-time validation and verification
6. rigorous methods for implementation, reuse, and deployment
7. a framework for developing TCS based on the above techniques

The contribution of this thesis is the design and implementation of a *Visual Modeling Tool*(VMT), which provides precise interaction points for the user-centric tasks at the design, implementation, and run-time environment phases of the development framework.

1.1 Trustworthy Computing Systems Development

Alagar and Mubarak [MA08] have proposed a component-based development approach for TCS that is also real-time reactive systems (RTRS). An un-timed TCS is a special case of trustworthy RTRS. The VMT is tailored to the development of trustworthy RTRS. So, in this section we give a quick summary of RTRS, TCS definition, component models, and the development approach proposed in [MA08].

1.1.1 Real-Time Reactive System(RTRS)

Reactive systems maintain a continuous ongoing interaction with their environment. Such systems are event-driven, interact intensively with the environment through stimulus response behavior, and are regulated by strict timing constraints. Further, these systems might also consist of both physical components and software components controlling the physical devices in a continuous manner. Although reactive systems are interactive systems, there is a fundamental difference between these two systems. Whereas both environment and processes have synchronization abilities in interactive systems, a process in a reactive system is solely responsible for the synchronization of its environment. That is, a process in a reactive system is fast enough to react to stimulus from the environment, and the time

between stimulus and response is acceptable enough for the dynamics of the environment to be receptive to the response. For example, a human computer interface is an interactive system, whereas a controller that regulates the amount of steam escaping from a boiler is clearly reactive. In the case of real-time reactive systems, stimulus-response behavior is also regulated by timing constraints and the major design issue is one of performance. Examples of RTRS include telephony, air traffic control, nuclear power reactors, and avionics. The major factors that contribute to the complexity of RTRS are the following:

- *size*: telephony and air traffic control systems are made up of a large number of hardware and software components;
- *time constraints*: telephony imposes only *soft* time constraints, a violation of which may not cause a catastrophe but may reduce the amount of user trust; however, avionics and nuclear power control systems impose *hard* (strict) time constraints, which if violated will cause damage and injury to human safety, and perhaps shatter entirely the user trust in them;
- *criticality*: nuclear power reactor is a safety-critical systems, in the sense that its failure is unacceptable;
- *heterogeneity*: sensors, actuators, and system processes have different levels of functional and time sensitive synchronization requirements.

It is evident from the above discussion that RTRS must be trustworthy, and the essential features of RTRS that determine its trustworthiness are safety and reliability.

1.1.2 Trustworthiness Credentials

In general, trust is a social concept that is hard to define formally. However, in software industry [AB01, FA99], there is a consensus on the definition of trust. In software development community, the terms *trustworthiness* and *dependability* are used interchangeably. Trustworthiness is the system property that denotes the degree of user confidence that the system will behave as expected [FA99, IM02]. Dependability is defined as the ability to deliver trusted services [AB01]. A comparison between the two terms presented in [AB01] has concluded that the two terms are equivalent in their goals and address similar concerns. The goals of dependability are (1) providing justifiably trusted services, and (2) avoiding outage of service that is unacceptable to the consumer. Thus, dependability and trustworthiness involve achieving *availability*, *reliability*, *safety*, *security*, and *survivability*. Safety and security are non-functional requirements which can be formally specified as system properties at design time. The classical notion of reliability comes from statistics, generally interpreted as "mean time to failure". Software, considered as a product, can be subjected to this reliability model. This implies that reliability is measured on implemented code. Recently, reliability has been formally defined on design and is further related to the run-time measurable quantity. System availability and survivability are to be assessed in an operational environment under different load factor assumptions, and patterns of attacks on security of the system. If safety and security are ensured, they will eventually assure a higher rate of system availability and reliability. Therefore, Alagar and Mubarak [AM07b] have considered safety and security as the essential credentials of trustworthiness that should be specified during design stage and reliability and survivability as

properties to be assessed after implementing the system. Towards the later, the framework provides a comprehensive set of tools in the implementation and run time environment of the system. The VMT, being the front-end for all stages of system development, interaction points for such investigation and analysis are provided in the VMT.

1.1.3 Component Based Development (CBD)

CBD is the type of software engineering development in which systems are built by constructing units, called components, that perform simple tasks, and assembling them to create composite components that perform complex tasks. Some potential benefits of applying CBD for building TCS include complexity reduction, time and cost savings, predictable behavior, and productivity increase [IM02]. The discussion in [Moh09] brings out several inconsistencies in existing component definitions and the necessity to define components on a more formal footing. They have proposed a component model that collectively addresses the requirements of real-time reactive systems (RTRS) and credentials of trustworthiness. The rationale is that a RTRS must be trustworthy, and conversely trustworthiness invariably includes safety, an important requirement of RTRS.

A central challenge in building trustworthy systems using CBD method is composing trustworthy components so that the composed component is trustworthy . Their main contributions in [Moh09] are (1) a definition of the requirements of a component model for developing trustworthy RTRS, (2) a formal definition for trustworthy hierarchical RTRS components, and (3) a compositional theory for composing components so that safety and security are preserved in the composition. Because the component definition is richer and

more involved than existing component definitions, it is not possible to adapt the existing tools [MT00] in an implementation of new trustworthy component models. This sets the stage for discussing the scope and contribution of this thesis.

1.1.4 Scope of the Thesis

The primary goal of the development framework is to provide a framework for a rigorous development, analysis, and deployment of TCS. The application developer, who is normally an expert in the application domain, should be facilitated to focus on modeling and analysis aspects without being burdened by the formalism. The formalism should be working in the background, regulating that nothing improper is done in the construction of the system. Another goal is to reduce the complexity of understanding the results and behavior of the system through the introduction of easy to use and easy to learn task-oriented descriptions in the development framework. These are the strong motivations behind the development of VMT.

A question may be raised as to why a new tool is necessary, why not reuse an existing tool to create components? The answer is in the pudding: UML and tools based on it are suited only for object-oriented system development. Components are richer than objects. In particular, trustworthy components are very rich, as explained below. The component definitions[Moh09] are new, and the only existing tool ACMESTudio[GS06] is not sufficiently robust for our purpose. Acme is a second generation architectural description language. It provides support for specifying the canonical set of structural elements of an architectural design. It includes definitions of component, port (runtime interface),

connector, connector role, system, property (attribute), constraint, and representation (sub-structure of a component or a connector). Acme is not suitable for our purpose because of the following reasons [Moh09]:

- The rationale behind Acme is to provide support for the essential structural elements of architecture designs so that it can serve as a base for interchanging architectural designs. Since, most of ADLs lack support for non-functional requirements in their structural definition, Acme supports only structural constraints related to the architectural style of systems. It has no support for non-functional requirements at structural level.
- In order to define trustworthiness for a component, there is a need to specify the services provided and required at the interfaces of a component. This is essential because the requirements of trustworthiness are related to the services of a component. However, Acme does not include service definition.
- Acme defines only a structural composition of components using connectors. On the other hand, our component model defines a composition for both the structural and trustworthiness properties.
- A component in our model has a contract. All trustworthiness properties are defined at this contract. This enables managing the non-functional requirements in a central definition. Also, it enhances the reuse of component definition for different deployments. There is no concept of contract in Acme.

Figure 1 shows the *component template* proposed by Alagar and Mohammad [AM07a]. It

is composed of a *structure* part and a *contract* part. The structure of a template is an abstract external black-box view, called *frame*, and its internal hierarchical structure, called *architecture*. The frame consists of the interface types, where each interface type is associated with a set of services. A service may be parameterized with data types. An architecture is a collection of connector types, an abstract view of the tie-ins between interface types. The contract part of the template states the properties required of the system for which the structure is a blue print. A component is an *instance* of a component template. Every component instantiated from a template has one instance of the structure part defined for the template. The frame of the component is a set of interfaces, where each interface belongs to exactly one interface type in the template frame. An architecture instance corresponding to a component frame is an instance of the architecture corresponding to the frame in the template, having as many instances of connector types as are required for linking the interfaces in the component. A component's contract constrains the communication pattern at its interfaces and is faithful to the contract part in its template. The VMT enables the creation of components adhering to the above description. Component templates, interface types, connector types, and their instances can be constructed using the VMT. For composing large system configurations, the VMT is most helpful in breaking the complexity barrier. Concrete graphical representations offer traceability between structural and behavioral units of the system under design.

The VMT has been conceived and designed to provide interaction points to design, implement, and deploy a TCS. It is sufficient for a developer to know these interaction points. It is not necessary to know the underlying formalism. The basis for creating the

models in VMT is a meta-architecture, which is discussed in Chapter 2. The TADL format will look similar to ACME language description; yet there are many syntactic and semantic differences. This is also presented in Chapter 2. From the visual model, the tool generates automatically the architecture descriptions in XML and TADL, an architecture description language developed by Mohammad and Alagar [Moh09]. The XML description is for "internal consumption", such as static analysis. These are presented in Chapter 3. The requirements and a formal design of VMT are discussed in Chapter 4 and Chapter 5. In Chapter 6 we discuss the VMT features, and provide screen shots while discussing how to use the VMT. Chapter 7 concludes the the thesis with a summary of the contributions and a list of future enhancements.

Chapter 2

Basic Concepts-TADL

In this chapter we review the component meta-model, the architectural elements, and TADL (an architectural description language to specify trustworthy component-based systems), the architectural description language from Alagar and Mohammad [Moh09]. The VMT is designed to construct components and build systems that are consistent with the definitions given in this chapter.

2.1 Introduction to Meta-architecture and TADL

A meta-architecture is used to specify a specific architecture. It serves as an architecture type for creating different component-based system models in VMT. Figure 2 presents the meta-architecture defined by Mohammad [Moh09]. The high level view of the meta-architecture is composed of Component definition, Component structure definition, Safety contract, Security mechanism, and System definition. TADL is a new architecture description language to specify trustworthy component-based systems. The meta-architecture and TADL serve the following purposes.

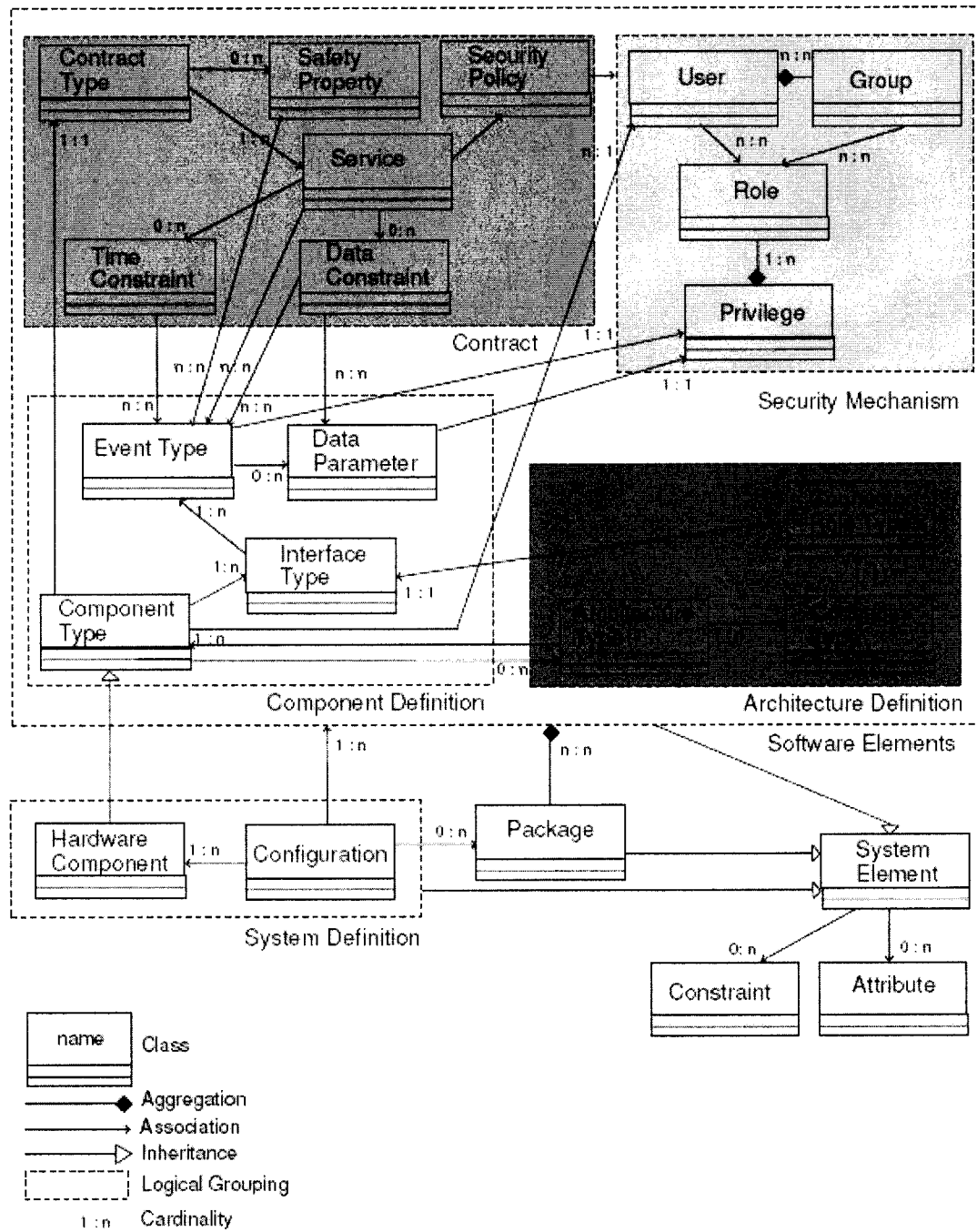


Figure 2: The building blocks of the meta-architecture

- They specify a unified component model that embodies the features of CBD, real-time systems, and trustworthiness.
- A unified method to define and analyze trustworthiness properties at the architectural level is given by them.
- They specify a component model that is able to translate descriptions of system from other component models to the meta-architecture.
- The TADL uses simple representations to describe architectural elements. Therefore, the developers are able to understand the system definitions without knowing formal methods.

For the convenience of reuse and reconfiguration of different system specifications, TADL defines each of the meta-architecture elements individually. The specification of the element is composed of three parts, namely: element type, element name, and specification of the contents of the element.

```
ElementType <name>{  
    (Attribute <name>)*;  
}
```

In the following sections, we give a detailed description of the meta-architecture elements, along with the TADL syntax of these elements.

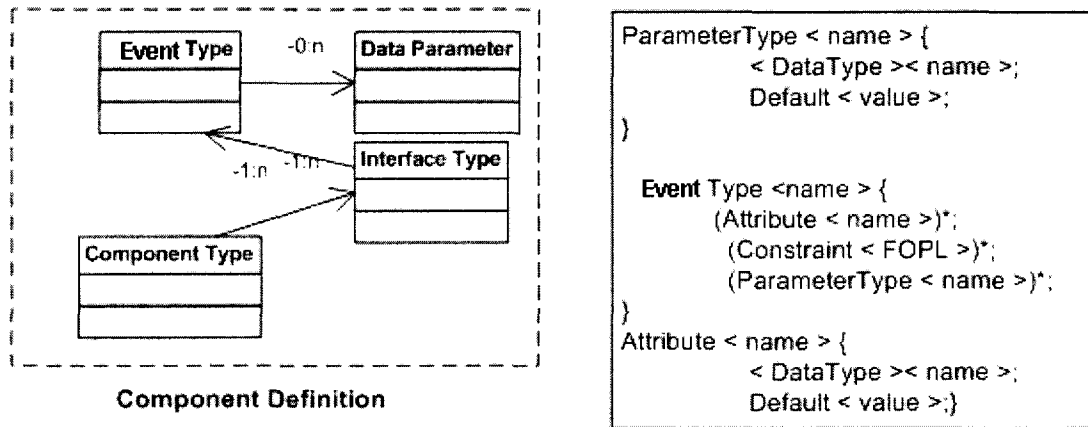


Figure 3: The component definition and specification

2.2 Component Definition

Figure 3 describes the elements contained in component definition, and the the TADL description of parameter type, and event type.

- Components offer/request events through interfaces. The syntax of interface type and component type will be described in Section 2.4.
- Events represent the capabilities offered by a component, or the functionalities required by a component. The events are provided at interfaces. The TADL specification of events may include *attributes*, *constraints*, and *parameter types*. Constraints describe the restrictions of an architectural design. The statements in the constraints should be maintained to be true. In TADL, a first-order predicated logic (FOPL) is used to define constraints.
- A data parameter is a variable that is transmitted between components, when these

components request events from or provide events to each other. The TADL specification of a data parameter includes name, data type, and default value.

- The TADL specification of an attribute includes name, data type and default value. The difference between an attribute and a data parameter is that data parameters are only related to events, while attributes can be included in any meta-architecture element.

2.3 Safety Contract

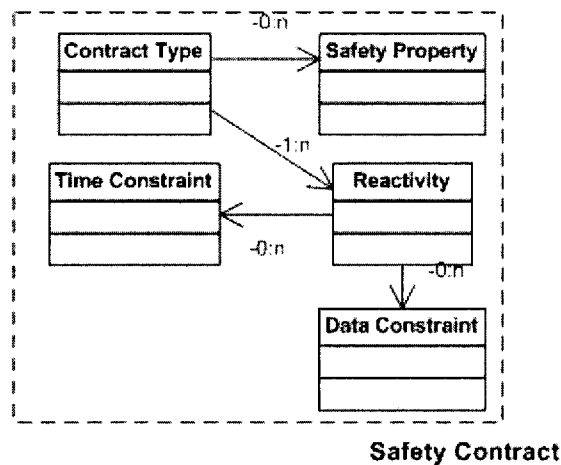


Figure 4: Safety Contract

Safety contract defines a set of properties that describe the correct and safe interactions between components. Figure 24 describes the elements of safety contract. These are *contract type*, *safety property*, *service*, and *time constraint*, and *data constraint*.

- *Time constraint*: A time constraint regulates the responses of a component. Time

constraints define the maximum response time of a component. The following is the TADL specification of the time constraint.

```
TimeConstraint <name> {  
    (Attribute <name>)*;  
    (Constraint <FOPL>)*;  
    EventType <request-name>;  
    RequestEvent  
    (<request-name>);  
    EventType <response-name>;  
    ResponseEvent  
    (<response-name>);  
    float MaxSafeTime;  
}
```

The definition includes attributes, constraints, the type and name of the event that makes the request, the type and name of the event that responses, and the maximum safe time of response.

- *Data constraint:* Data security requirements are formalized as data constraints. A data constraint will restrict the responses of a component. It determines whether or not a component should send a response when other component requests events from it. The TADL syntax for data constraint specification is given below.

```
DataConstraint <name> {
```

```

(Attribute <name>)*;
EventType <request-name>;

RequestEvent
(<request-name>);

EventType <response-name>;

ResponseEvent
(<response-name>);

Constraint <FOPL>;
}

```

When the constraint is assessed to true, the response will include the requested information , otherwise the response will not include the expected information.

- *Service*: The relationships between requests for events and their responses are called Service constraints. Time constraint can be added to a service to set the maximum safe time of a stimulus-response relation. Service specification can also include a data constraint to ensure that only one response is selected from among several possible responses, when a stimulus is triggered. The TADL service specification is given as follows.

```

Service <name> {
    EventType <request-name>;

    RequestEvent
    (<request-name>);
}

```

```

    EventType <response-name>;
    ResponseEvent
    (<response-name>);
    (DataConstraint <name>)*;
    (TimeConstraint <name>)*;
    (Update statements)*;
}

```

- *Safety Property*: Informally, safety means that "nothing bad happens". It is a particular kind of constraint that is defined at the interfaces of a component. A component behavior should always respect the safety property. The TADL syntax for specifying safety property is given below.

```

SafetyProperty <name> {
    (EventType <name>)*;
    Constraint <FOPL>;
}

```

- *Contract type*: A contract is a package that includes services and safety properties. The TADL syntax for a contract type is given below.

```

ContractType <name> {
    (Service <name>+;
    (SafetyProperty <name>)*;
}

```

2.4 Component Architecture

As illustrated in Figure 5, the structural specification of a component is composed of interface type, connector role type, connector type, architecture type and component type.

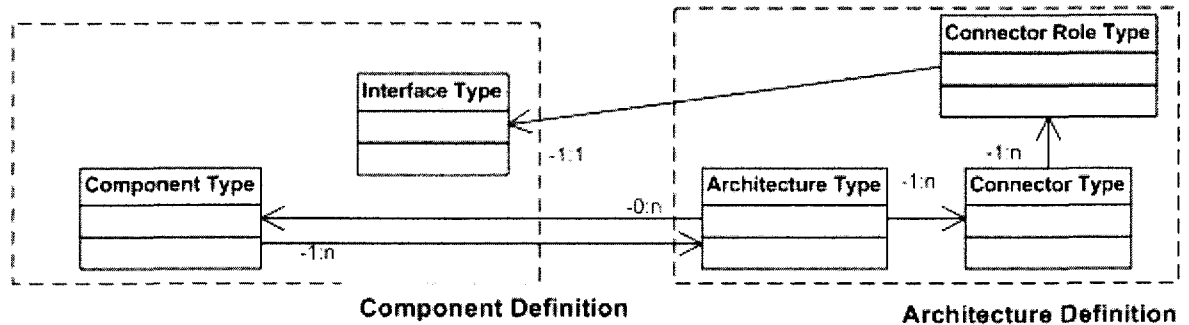


Figure 5: Component Architecture

- *Interface type:* An interface type is an abstraction that models the access point to the events provided/requested by components [Moh09]. The TADL definition of interface type contains an arbitrary number of event types and attributes. In addition, interface type specification can also have a protocol specification that is stored in an external file.

```
InterfaceType <name> {  
    String Protocol;  
    (Attribute <name>)*;  
    (EventType <name>)*;  
}
```


- *Connector type*: Communication mechanism between two interacting components is modeled by a connector. Components that are not connected to each other by connectors can not interact directly. The definition of connector type includes one or more connector role types, defined next.

```
ConnectorType <name> {
    (ConnectorRoleType <name>)+;
    (Attribute <name>)*;
    (Constraint <FOPL>)*;
}
```

- *Connector role type*: Each connector type is specified by a role. The connector type role is "an interface" of that connector type. It is linked with an interface type of a component. The TADL definition of connector role type is described as follows:

```
ConnectorRoleType <name> {
    (Attribute <name>)*;
    (Constraint <FOPL>)*;
    InterfaceType <name>;
}
```

Figure 6 [Moh09] presents how to link two components together, with interfaces, connectors and connector roles. A connector defines the connectivity between two or more components. A connector type definition includes a non-empty finite set of

connector role types in addition to attributes and constraints. A connector role type serves as an interface to a connector. It links a connector to a component interface. An attachment specification defines how two components can be attached together using a connector, two interfaces, and two connector role types. More precisely, it defines which connector role of a connector is connected to which interface of a component. Abstracting the connector role from the connector specification enables abstracting the communication method used in the connector from its access points. Therefore, it is possible to define different communication methods such as RPC, HTTP, and SOAP using the same access points (connector roles). Also, introducing connection points at the ends of a connector can help to reason about the integrity of the communication method by comparing representations of the data before and after the communication.

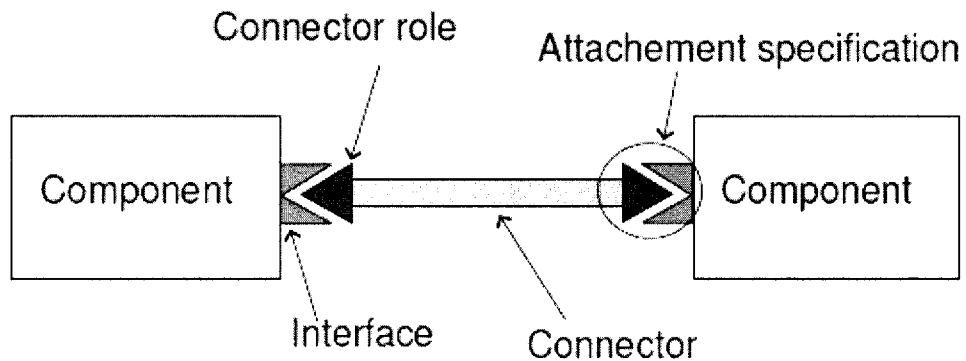


Figure 6: How two components are connected

- *Component type*: There are two types of components, called *primitive component*

type and *composite component type*. An architecture type, defined within a component type, distinguishes the two concepts. The structure of a composite component is defined by specifying one or more architectures. On the other hand, if there is no architecture in the specification, then the component is considered to be primitive. Below is the TADL specification of a component type.

```
ComponentType <name> {  
    (Attribute <name>)*;  
    (Constraint <FOPL>)*;  
    User u;  
    (InterfaceType <name>)+;  
    (ArchitectureType <name>)*;  
    ContractType <name>;  
}
```

The "User" section in the definition above will be described in Section 2.5.

- *Architecture type*: As described before, an architecture type specifies the internal architecture of a component. The TADL definition of an architecture type is illustrated as follows:

```
ArchitectureType <name> {  
    (ComponentType <name>)+;  
    (ConnectorType <name>)+;
```

```

(Attribute <name>)*;

(Constraint <FOPL>)*;

(Attachment

(ConnectorType.RoleType.InterfaceType,

ComponentType.InterfaceType))*;

}

```

In the "Attachment" section of the above definition a formal description of linking two components is given. Two components are linked together by attaching the interface type of a connector role type to the interface type of a component type. The attachment specification should cover every connecting point of an interface type and a connector role type.

2.5 Security mechanism

Figure 7 presents the elements in the specification of the security mechanism. The TCS model uses role-based security access control (RBAC) to enforce security at component interfaces. The mechanism ensures that the access to component events are regulated and data parameters are safe-guarded at interfaces. RBAC includes the following four concepts.

User: It is an attribute defined in a component type. The value of the user is assigned when a component is instantiated. The value represents that the component is executing on behalf of the 'user'. The formal TADL description of *User* section is given below.

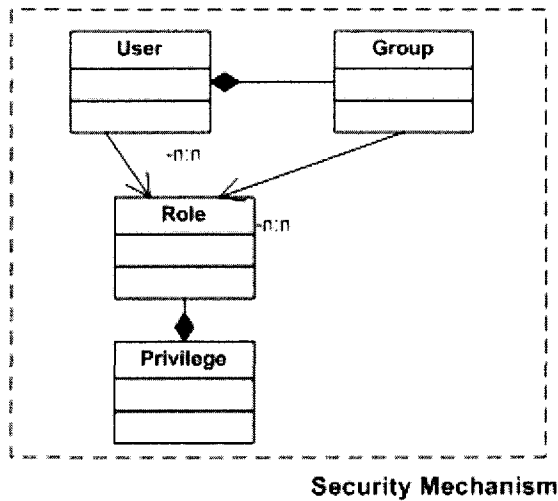


Figure 7: Security Mechanism

```

User <name> {
    (Attribute <name>)*;
    (Constraint <FOPL>)*;
}
  
```

Group: A role may be played by several users. So, a group defines the domain of users.

A user can be assigned to an arbitrary number of groups using the function 'User - Groups - Assignment' defined in the TADL specification of RBAC. The TADL description for *Group* is given below.

```

Group <name> {
    (Attribute <name>)*;
    (Constraint <FOPL>)*;
}
  
```

Role: A role is an association between a user and a set of actions that the user is permitted

to do in the system. As long as a user or a group is assigned to a specific role using the functions "User - Roles - Assignment(User,Role)" and "Group - Roles - Assignment(Group,Role)", the user and the group should take the security responsibilities of the role. The TADL description of *Role* is given below.

```
Role <name>{  
    (Attribute <name>)*;  
    (Constraint <FOPL>)*;  
}
```

Privilege: A privilege is a right that determines whether or not a user can access a specific event or a data parameter. Privileges are assigned to roles. One role can have many privileges. One privilege can be assigned to many roles. There are two kinds of privileges.

- a) *Event Privilege:* This defines the right to access a event, using the function 'Privileges - for - events'. Only a role who has a privilege of the event can request the event at the interfaces of a component and receive the responses.
- b) *Data Parameter Privilege:* It defines the right to access a data parameter, using the function 'Privileges - for - data - parameters'. Only a role who has a privilege of the data parameter can request the data parameter and receive the responses.

The following illustrates the TADL syntax of a privilege:

```

Privilege <name> {
    (Attribute <name>)*;
    (Constraint <FOPL>)*;
}

```

The TADL descriptions given above are "plugged-in" the following TADL format, which fully describes the security mechanism.

```

RBAC <name> {
    (User <name>)*;
    (Group <name>)*;
    (Role <name>)*;
    (Privilege <name>)*;
    (User - Groups - Assignment (User, Group))*;
    (User - Roles - Assignment (User, Role))*;
    (Group - Roles - Assignment (Group, Role))*;
    (EventType <name>)*;
    (ParameterType <name>)*;
    (Privileges - for - events
        (Event, Privilege, Role))*;
    (Privileges - for - data - parameters
        (DataParameter, Privilege, Role))*;
}

```

2.6 System Definition

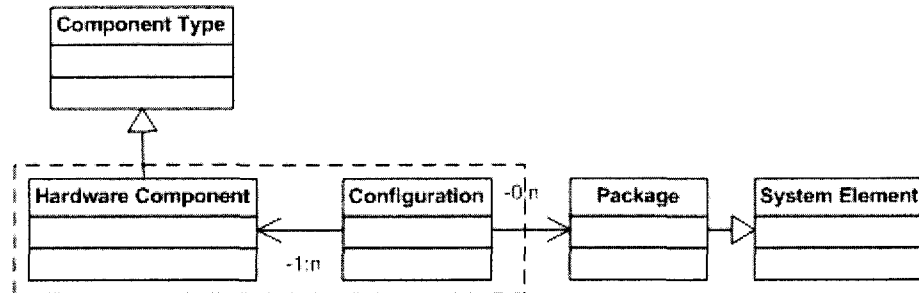


Figure 8: System Definition

A system is composed from software and hardware components. Figure 8 shows the elements of system definition. It includes hardware components, and software configurations. Hardware component is a particular type of component. The TADL specification of system configuration includes the instances of the defined software and hardware component types, along with a deployment plan. The deployment specification uses the function "Deploy (HardwareComponentType, ComponentType)" to configure the software components on the hardware components.

```
Configuration <name>{  
    (SystemElement <name>)+;  
    (Deploy (HardwareComponentType, ComponentType)) +;  
}
```

The package, in the diagram, stores a number of the related architectural elements to simplify future reuse of these elements.

Chapter 3

Basic Concepts-XML Schema

3.1 Introduction

The description of the visual models in TADL is formal, and is used for presentation to the users of the system. We generate the textual description of the visual model as XML Schemas, which are used for internal processing of the system development. The TADL description and XML description are equivalent.

The advantage of having XML Schema for the visual model is that it can be exported and if necessary converted to other presentation format. Based upon the XML Schemas, Ibrahim [Ibr08], has implemented a translator that produces the UPPALL extended state machines, a behavior model of the visual models, and verify the trustworthy properties of the modeled system. The compiler in VMT produces the XML and TADL descriptions for further use. In this section, we explain the XML Schemas.

An XML Schema is an XML schema language, recommended by W3C(World Wide Web Consortium) [Rec05], which defines a number of constraints to restrict the structure of an

XML document. In the following sections, we will review the pre-defined XML Schemas.

There are in total seven schemas to define the TADL. These are explained next.

3.2 InterfaceType Schema

The interface type schema includes an ordered sequence of the following sub-elements:

- *name*: It is a simple element to specify the name of the interface.
- *protocol*: It is a simple element to specify the protocol of an interface.
- *Attribute*: It is a complex element to specify the attributes of an interface. It is an ordered sequence of four simple elements, namely *name*, *datatype*, *value* and *description*.
- *EventType*: It is a complex element which includes an ordered sequence of the following sub elements:
 - *name*: It is a simple element to specify the name of the event.
 - *id*: It is a simple element to specify a unique id for the event. For the same event, its id number should be the same.
 - *type*: It is a simple element to specify the event type.
 - *Attribute*: It is a complex element to specify the attributes of a event. The schema of an Attribute has been discussed above.
 - *constraint*: It is a simple element to specify the constraints in a event.

- *ParameterType*: It is a complex element to specify the parameters in a event. It is an ordered sequence of four simple elements, namely name, datatype, value, and description.
- *Property*: It is a complex element to specify the properties of an interface. It is composed of an ordered sequence of two simple elements, namely name and value.
- *description*: It is a simple element to store the notes by users.
- *description*: It is a simple element to store the notes written by user for a specific element.

The `minOccurs` and `maxOccurs` in the XML Schema respectively specify the minimum and maximum occurrences of an element. The schema is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com)
by bg (bg) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:complexType name="Property">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="value" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

```

</xs:complexType>

<xs:complexType name="EventType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="id" type="xs:string"/>
    <xs:element name="type" type="xs:string"
      minOccurs="0"/>
    <xs:element name="attribute" type="Attribute"
      minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="constraint" type="xs:string"
      minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="parameterType" type="ParameterType"
      minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="property" type="Property"
      minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="description" type="xs:string"
      minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ParameterType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>

```

```

        <xs:element name="datatype" type="xs:string"/>
        <xs:element name="value" minOccurs="0"/>
        <xs:element name="description" type="xs:string"
            minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Attribute">
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="datatype" type="xs:string"/>
        <xs:element name="value" type="xs:string"
            minOccurs="0"/>
        <xs:element name="description" type="xs:string"
            minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="InterfaceType">
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="protocol" type="xs:string"
            minOccurs="0"/>
        <xs:element name="attribute" type="Attribute"

```

```

        minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="eventType" type="EventType"
        minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="description" type="xs:string"
        minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:schema>

```

3.3 ComponentType Schema

The component type schema includes an ordered sequence of the following sub elements:

- *name*: It is a simple element to specify the name of a component.
- *Property*: It is a complex element to specify the properties of a component. The schema of the property has been discussed in 3.2.
- *Attribute*: It is a complex element to specify the attributes of a component. The schema of the attribute has been discussed in 3.2.
- *constraint*: It is a simple element to specify the constraints of a component.
- *user*: It is a complex element to define the users of a component. The schema of the user will be discussed in section 3.7.

- *InterfaceType*: It is a complex element to specify the interfaces of a component. The schema of the interface type has been discussed in section 3.2.
- *ArchitectureType*: It is a complex element to specify the architectural structure of the component. It is composed of an ordered sequence of the following elements:
 - *name*: It is a simple element to specify the name of an architecture.
 - *ComponentType*: It is a complex element to specify the components in an architecture. The schema of the component type is discussed in section 3.3.
 - *ConnectorType*: It is a complex element to specify the connectors in an architecture. The schema of the connector type is discussed in section 3.4.
 - *Attribute*: It is a complex element to specify the attributes of an architecture. The schema of the attribute type is discussed in section 3.2.
 - *constraint*: It is a simple element to specify the constraints in an architecture.
 - *Attachment*: It is a complex element to define the connection of a connector role and the interface of a component. It is composed of an ordered sequence of the following elements:
 - * *name*: It is a simple element to specify the name of an attachment.
 - * *ConnectorType*: It is a complex element to specify the connector in an attachment. The schema of the connector type is discussed in section 3.4.
 - * *ConnectorRoleType*: It is a complex element to specify the connector role in an attachment. The schema of the connector role type is discussed in section 3.4.

- * *InterfaceType*: It is a complex element to specify the interface in an attachment. The schema of the interface type is discussed in section 3.2.
 - * *ComponentType*: It is a complex element to specify the component in an attachment. The schema of the component type is discussed in section 3.3.
 - * *description*: It is a simple element to store the notes by users.
- *ContractType*: It is a complex element to specify the safety contract of a component. The schema of the contract type is discussed in section 3.5.
 - *description*: It is a simple element to store the notes by users.

The XML schema conforming to the above specification is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com)
by bg (bg) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:include schemaLocation=".\\interfaceType.xsd"/>
    <xs:include schemaLocation=".\\connectorType.xsd"/>
    <xs:include schemaLocation=".\\contractType.xsd"/>
    <xs:include schemaLocation="RBAC.xsd"/>
    <xs:complexType name="ComponentType">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>

```



```

    <xs:element name="property" type="Property"
      minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="attribute" type="Attribute"
      minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="constraint" type="xs:string"
      minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="user" type="User" minOccurs="0"/>
    <xs:element name="interfaceTypes" type="InterfaceType"
      minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="architectureType"
      type="ArchitectureType" minOccurs="0"/>
    <xs:element name="contract" type="ContractType"
      minOccurs="0"/>
    <xs:element name="description" type="xs:string"
      minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ArchitectureType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="componentType" type="ComponentType"
      minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>

```

```

    <xs:element name="connectorType" type="ConnectorType"
        minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="attribute" type="Attribute"
        minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="constraint" type="xs:string"
        minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="attachments" type="Attachment"
        minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="description" type="xs:string"
        minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="Attachment">
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="connectorType-from"
            type="ConnectorType"/>
        <xs:element name="roleType-from"
            type="ConnectorRoleType"/>
        <xs:element name="interfaceType-from"
            type="InterfaceType"/>
        <xs:element name="componentType-to"

```

```

        type="ComponentType"/>
    <xs:element name="interfaceType-to"
        type="InterfaceType"/>
    <xs:element name="description"
        type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:schema>

```

3.4 ConnectorType Schema

The connector type schema includes an ordered sequence of the following sub elements:

- *name*: It is a simple element to specify the name of a connector.
- *ConnectorRoleType*: It is a complex element to specify the roles of a connector. It is composed of an ordered sequence of the following elements:
 - *name*: It is a simple element to specify the name of the connector role.
 - *Attribute*: It is a complex element to specify the attributes of a connector role. The schema of the attribute has been discussed in 3.2.
 - *constraint*: It is a simple element to specify the constraints of a connector role.

- *InterfaceType*: It is a complex element to specify the interface attached to the connector role. The schema of the interface type has been discussed in 3.2.
- *description*: It is a simple element to store the notes from users.
- *Attribute*: It is a complex element to specify the attributes of a connector. The schema of the attribute has been discussed in 3.2.
- *constraint*: It is a simple element to specify the constraints of a connector.
- *description*: It is a simple element to store informal notes by users.

The XML schema conforming to the above specification follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com)
by bg (bg) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:include schemaLocation=".\\interfaceType.xsd"/>
  <xs:complexType name="ConnectorRoleType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="attribute" type="Attribute"
        minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="constraint" type="xs:string"
```

```

        minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="interfaceType" type="InterfaceType"/>
    <xs:element name="description" type="xs:string"
        minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="ConnectorType">
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="connectorRoleType"
            type="ConnectorRoleType"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="attribute" type="Attribute"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="constraint" type="xs:string"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="description" type="xs:string"
            minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
</xs:schema>

```

3.5 ContractType Schema

Contract is a new element type in the component modeling. A contract is a specification that can be associated to one or more components. The contract type schema includes an ordered sequence of the following sub elements:

- *name*: It is a simple element to specify the name of a contract.
- *DataConstraint*: It is a complex element to specify the data constraints in a service.

It is composed of an ordered sequence of the following elements:

- *name*: It is a simple element to specify the name of the data constraint.
 - *Request EventType*: It is a complex element to specify the stimulus event of a data constraint. The schema of the event type has been discussed in 3.2.
 - *Response EventType*: It is a complex element to specify the response event of a data constraint. The schema of the event type has been discussed in 3.2.
 - *constraint*: It is a simple element to specify the constraints in a data constraint.
 - *description*: It is a simple element to store the notes from users.
- *TimeConstraint*: It is a complex element to specify the time constraints in a service.

It is composed of an ordered sequence of the following elements:

- *name*: It is a simple element to specify the name of the time constraint.
- *Attribute*: It is a complex element to specify the attributes of a time constraint.
The schema of the attribute is discussed in 3.2.

- *constraint*: It is a simple element to specify the constraints in a time constraint.
 - *Request EventType*: It is a complex element to specify the stimulus event of a data constraint. The schema of the event type has been discussed in 3.2.
 - *Response EventType*: It is a complex element to specify the response event of a data constraint. The schema of the event type has been discussed in 3.2.
 - *maxSafeTime*: It is a simple element to specify the maximum safe time of a time constraint.
 - *description*: It is a simple element to store the notes from users.
- *Service*: It is a complex element to specify the safe reactivities in a contract.
 - *name*: It is a simple element to specify the name of a service.
 - *id*: It is a simple element to identify a unique id for each service.
 - *Request EventType*: It is a complex element to specify the stimulus event of a data constraint. The schema of the event type has been discussed in 3.2.
 - *Response EventType*: It is a complex element to specify the response event of a data constraint. The schema of the event type has been discussed in 3.2.
 - *DataConstraint*: It is a complex element to include data constraints in a service. The schema of the data constraint has been discussed above.
 - *TimeConstraint*: It is a complex element to include time constraints in a service. The schema of the time constraint has been discussed above.

- *Update*: It is a complex element to describe the updates of the design. It is composed of an ordered sequence of two simple elements, namely *toBeUpdated* and *value*.
- *description*: It is a simple element to store the notes from users.
- *SafetyProperty*: It is a complex type to specify the safety property in a contract. It is composed of an ordered sequence of the following elements:
 - *name*: It is a simple element to specify the name of a safety property.
 - *EventType*: It is a complex element to specify the events that is restricted by the safety property. The schema of the event type is discussed in 3.2.
 - *constraint*: It is a simple element to specify the constraints in a safety property.
 - *description*: It is a simple element to store the notes from users.
- *description*: It is a simple element to store the notes by users.

The XML Contract Type schema conforming to the above specification is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com)
by bg (bg) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:include schemaLocation=".\\interfaceType.xsd"/>
  <xs:complexType name="SafetyProperty">
```



```

<xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="eventType" type="EventType"
        minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="constraint" type="xs:string"/>
    <xs:element name="description" type="xs:string"
        minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="ContractType">
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="dataConstraint" type="DataConstrain"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="timeConstraint" type="TimeConstrain"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="service" type="Service"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="safetyProperty" type="SafetyProperty"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="description" type="xs:string"
            minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

```

```

        </xs:sequence>
</xs:complexType>
<xs:complexType name="Service">
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="id" type="xs:string"/>
        <xs:element name="event-request" type="EventType"/>
        <xs:element name="event-response" type="EventType"/>
        <xs:element name="dataConstraint" type="DataConstraint"
            minOccurs="0"/>
        <xs:element name="timeConstraint" type="TimeConstraint"
            minOccurs="0"/>
        <xs:element name="update" type="Update"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="description" type="xs:string"
            minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="TimeConstraint">
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="attribute" type="Attribute"

```

```

        minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="constraint" type="xs:string"
        minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="event-request" type="EventType"
        minOccurs="0"/>
<xs:element name="event-resonse" type="EventType"
        minOccurs="0"/>
<xs:element name="maxSafeTime" type="xs:int"/>
<xs:element name="description" type="xs:string"
        minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="DataConstraint">
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="event-request" type="EventType"/>
        <xs:element name="event-response" type="EventType"/>
        <xs:element name="constraint" type="xs:string"/>
        <xs:element name="description" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Update">

```

```
<xs:sequence>
    <xs:element name="toBeUpdated" type="xs:string"/>
    <xs:element name="value" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
```

3.6 PackageType Schema

Package is a model element. The package type schema includes an ordered sequence of the following sub elements:

- *name*: It is a simple element to specify the name of a package.
- *Version*: It is a simple element to specify the version of a package.
- *InterfaceType*: It is a complex element to specify the interfaces in a package. The schema of the interface type has been discussed in 3.2.
- *ContractType*: It is a complex element to specify the contracts in a package. The schema of the contract type has been discussed in 3.5.
- *ConnectorType*: It is a complex element to specify the connectors in a package. The schema of the connector type has been discussed in 3.4.

- *ComponentType*: It is a complex element to specify the components in a package.

The schema of the component type has been discussed in 3.3.

- *description*: It is a simple element to store the notes by users.
- *PackageType*: It is a complex element to specify the sub packages in a package.

Below we give the package type XML schema conforming to the above specification.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2008 sp1 (http://www.altova.com)
by N alani (western) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:include schemaLocation="componentType.xsd"/>
    <xs:include schemaLocation="interfaceType.xsd"/>
    <xs:include schemaLocation="connectorType.xsd"/>
    <xs:complexType name="PackageType">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="Version"/>
            <xs:element name="interfaceTypes" type="InterfaceType"
                minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="contractTypes" type="ContractType"
                minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

```

    <xs:element name="connectorTypes" type="ConnectorType"
        minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="componentTypes" type="ComponentType"
        minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="description" type="xs:string"
        minOccurs="0"/>
    <xs:element name="packages" type="PackageType"
        minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:schema>

```

3.7 RBAC Schema

Role-based access control (RBAC) is the security model imposed in trustworthy component modeling. RBAC schema includes the information necessary to enforce RBAC methods.

RBAC schema is an ordered sequence of the following elements:

- *name*: It is a simple element to specify the name of a RBAC.
- *User/Group/Role/Privilege*: It is a complex element to specify the users/groups/roles/privileges in a RBAC. The schema of *User/Group/Role/Privilege* is composed of an ordered sequence of the following:

- *name*: It is a simple element to specify the name of the users/groups/roles/privileges.
 - *Attribute*: It is a complex element to specify the attributes of a users/groups/roles/privileges.
The schema of the attribute has been discussed in 3.2.
 - *constraint*: It is a simple element to specify the constraints of a users/groups/roles/privileges.
 - *description*: It is a simple element to store the notes from users.
- *UserGroupAssignments*: It is a complex element to specify the user-group assignments. It is composed of an ordered sequence of two complex elements, namely user and group.
 - *UserRoleAssignments*: It is a complex element to specify the user-role assignments. It is composed of an ordered sequence of two complex elements, namely user and role.
 - *GroupRoleAssignments*: It is a complex element to specify the group-role assignments. It is composed of an ordered sequence of two complex elements, namely group and role.
 - *EventType*: It is a complex element to include the events that is restricted by the RBAC. The schema of the event type has been discussed in section 3.2.
 - *ParameterType*: It is a complex element to include the parameters that is restricted by the RBAC. The schema of the parameter type has been discussed in section 3.2.
 - *PrivilegesForEvents*: It is a complex element to assign privileges of events to specific roles. It is composed of an ordered sequence of three composite elements,

namely EventType, Privilege, and Role.

- *PrivilegesForDataParameters*: It is a complex element to assign privileges of data parameters to roles. It is composed of an ordered sequence of three composite elements, namely ParameterType, Privilege, and Role.
- *description*: It is a simple element to store the notes by users.

The XML schema for the above RBAC specification is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com)
by bg (bg) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:include schemaLocation=".\\interfaceType.xsd"/>
  <xs:complexType name="RBAC">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="users" type="User" minOccurs="0"
        maxOccurs="unbounded"/>
      <xs:element name="groups" type="Group" minOccurs="0"
        maxOccurs="unbounded"/>
      <xs:element name="roles" type="Role" minOccurs="0"
        maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```



```
<xs:element name="privileges" type="Privilege"
    minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="userGroupsAssignments"
    type="UserGroupAssignments"
    minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="userRolesAssignments"
    type="UserRolesAssignments"
    minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="groupRolesAssignments"
    type="GroupRolesAssignments"
    minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="eventType" type="EventType"
    minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="parameterType" type="ParameterType"
    minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="privilegesForEvent"
    type="PrivilegesForEvents"
    minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="privilegesForDataParameters"
    type="PrivilegesForDataParameters" minOccurs="0"
    maxOccurs="unbounded"/>
<xs:element name="description" type="xs:string"
```

```

        minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="User">
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="attribute" type="Attribute"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="constraint" type="xs:string"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="description" type="xs:string"
            minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Group">
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="attribute" type="Attribute"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="constraint" type="xs:string"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="description" type="xs:string"

```

```

        minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Role">
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="attribute" type="Attribute"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="constraint" type="xs:string"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="description" type="xs:string"
            minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Privilege">
    <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="attribute" type="Attribute"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="constraint" type="xs:string"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="description" type="xs:string"

```

```

        minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="UserGroupAssignments">
    <xs:sequence>
        <xs:element name="user" type="User"/>
        <xs:element name="group" type="Group"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="UserRolesAssignments">
    <xs:sequence>
        <xs:element name="user" type="User"/>
        <xs:element name="role" type="Role"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="GroupRolesAssignments">
    <xs:sequence>
        <xs:element name="group" type="Group"/>
        <xs:element name="role" type="Role"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="PrivilegesForEvents">

```

```

    <xs:sequence>
        <xs:element name="event" type="EventType"/>
        <xs:element name="privilege" type="Privilege"/>
        <xs:element name="role" type="Role"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="PrivilegesForDataParameters">
    <xs:sequence>
        <xs:element name="dataParameter"
            type="ParameterType"/>
        <xs:element name="privilege" type="Privilege"/>
        <xs:element name="role" type="Role"/>
    </xs:sequence>
</xs:complexType>
</xs:schema>

```

3.8 System Schema

A system configuration includes hardware and software components. The System Configuration schema includes an ordered sequence of the following sub elements:

- *name*: It is a simple element to specify the name of a system.

- *Attribute*: It is a complex element to specify the attributes of a system. The schema of the Attribute has been discussed in section 3.2.
- *ComponentType*: It is a complex element to specify the components in a system. The schema of the component type has been discussed in section 3.3.
- *Deploy*: It is a complex element to deploy software components to hardware components in a system. It is composed of an ordered sequence of two complex elements, namely *HardwareComponentType* and *ComponentType*. The schema of *ComponentType* has been discussed in section 3.3, and the the schema of a hardware component type is composed of a sequence of the following:
 - *name*: It is a simple element to specify the name of the hardware component.
 - *Attributes*: It is a complex element to specify the attributes of a hardware component. The schema of Attribute has been discussed in section 3.2.
 - *constraints*: It is a simple element to specify the constraints of a hardware component.
 - *InterfaceType*: It is a complex element to specify the interfaces of a hardware component. The schema of an interface type has been discussed in section 3.2.
 - *description*: It is a simple element to store the notes from users.
- *description*: It is a simple element to store the notes from users.
- RBAC: a complex element to include a security mechanism for a system. The schema of the RBAC has been discussed in section 3.7.

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2008 rel. 2 sp2 (http://www.altova.com)
by Moe Mawlana (Concordia.) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:include schemaLocation=".\\componentType.xsd"/>
    <xs:include schemaLocation=".\\RBAC.xsd"/>
    <xs:complexType name="SystemElement">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="description" type="xs:string"
                minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="Deploy">
        <xs:sequence>
            <xs:element name="hardwareComponentType"
                type="HardwareComponentType"/>
            <xs:element name="componentType" type="ComponentType"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="HardwareComponentType">

```

```

<xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="attributes" type="Attribute"
        minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="constraint" type="xs:string"
        minOccurs="0"/>
    <xs:element name="interface" type="InterfaceType"
        minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="description" type="xs:string"
        minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:element name="Configuration">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="attributes" type="Attribute"
                minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="components" type="ComponentType"
                minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="deploy" type="Deploy"
                minOccurs="0" maxOccurs="unbounded"/>

```



```
        <xs:element name="description" type="xs:string"
            minOccurs="0"/>
        <xs:element name="rbac" type="RBAC" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Chapter 4

VMT Architecture: System Requirements

4.1 Introduction

Chapter 4 and Chapter 5 describe the architecture of the VMT. We follow the template [MOB00] for documenting the VMT architecture. This chapter introduces the context in which the tool is to be used, and the functional and non-functional requirements of the VMT. Chapter 5 will present the design of the tool.

4.2 Motivation

By looking through the recently adopted proposal for the revised version of UML [OMG06], called UML 2.0, it seems that it can be claimed to be a component modeling language. A critical review of UML 2.0 [JO06] states that while the added features of UML 2.0 lends support to the expressive power at the architecture level, due to the lack of tool support it is impossible to evaluate such a claim. The new constructs introduced in UML 2.0 are Part,

Connectors, and Ports. `BasicComponents` and `PackagingComponents` are new names to indicate the capabilities of components modeled in separate packages, primarily to help different implementations. They do not have any precise semantics. The introduction of `StructuredClass` as a first class element and the distinction between required interface and provided interface are not fully supported by precise semantics. To a port an interface can be attached only if the protocol state machine of the port and the protocol state machine of the interface are *compatible*. There is no tool in UML 2.0 to prove compatibility between state machines.

The distinguishing feature of our research group is on a formal development of trustworthy systems. Component definitions are to be formal, composition rules are formal, verification of trustworthy properties are to be formally done, and the code development with run time analysis are to be conducted within our formal framework. As such UML 2.0, which is still in an infant stage, cannot be used for our project. This necessitated the development of a tool that will create the visual models of components, classes, and behaviors, for which formal definitions exist. The tool will hide the formalism, yet provide assistance for the correct models to be created.

4.3 Purpose and Context

Figure 9 [MA08] is the context diagram of the VMT. It describes the full development framework of TCBS. It shows the tools that our group members are designing to support

the formal development of TCBS. The visual modeling tool is the fundamental tool that developers will use at different stages of system development, in particular during the design stage of TCBS.

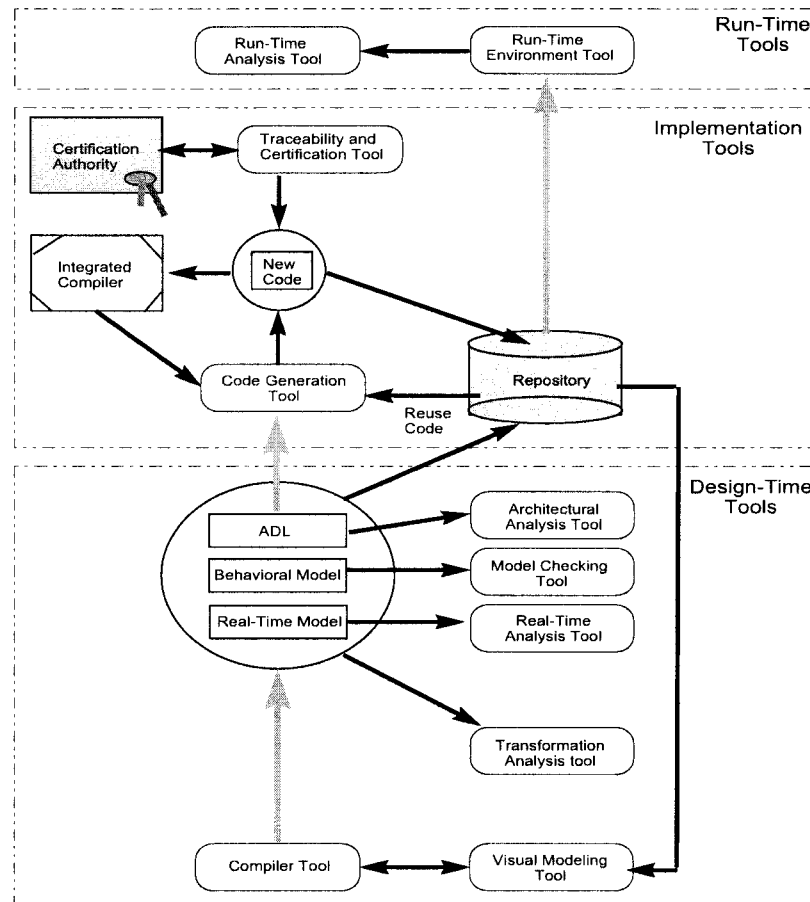


Figure 9: Framework for Developing TRTS - Context Diagram

The context diagram shows that there are three important stages in the development of TCBS. The VMT is to support the activities during *Design*, *Implementation*, and *Deployment (Run-time)*. By analyzing the functionalities of the tools in Figure 9 we identify the requirements of the VMT. A process model and a rationale for conceiving the current

development framework consistent with the process model are discussed in [MA08].

4.3.1 Design Stage

In the design stage a developer uses the VMT to create, edit, and compile visual models of the elements of TCBS and systems composed with the elements. The tool provides the facility to access a repository of design artifacts which can be reused by the developer. The VMT enables a model to be compiled and checked for syntactic correctness before generating a textual description of the model in TADL, the *trustworthy architectural description language* [Moh09].

Visual modeling tool (VMT) VMT provides a friendly graphical user interface for developers to construct TCBS. The goal of VMT is to ease the complexity inherent in system developer's work. Instead of being directly exposed to the formal notation, system developers can model the elements of trustworthy component models using a high-level graphical notation. Both functional and non-functional properties can be specified succinctly in the visual models. The tool serves the following three purposes.

- it acts as an interface to design components, connectors, and system configurations along with their attributes and properties;
- it enables to view the textual and visual representations of the design; and
- it provides different views of the system design for different users. We identify at least three views.

Component Based Development (CBD) view This view is a projection of the model on the architecture, characteristics of components, connectors, system configurations, and global architectural constraints.

Real-time view This view shows the real-time features of the design, projecting the time constraints on services.

Trustworthiness view This view highlights the safety, liveness and security properties imposed on a component.

The VMT provides the access points to the rest of the tools.

Compiler tool (CT) The compiler checks the validity of the model, and supports different types of output by transforming the design according to formally defined transformation rules. The following three types of output are generated by the current version of compiler.

- performs lexical and syntactic analysis of the visual models with respect to its abstract definition, and generates a formal textual description in an architectural description language (TADL),
- checks the real-time and trustworthiness elements of the model by comparing the corresponding views defined in the visual modeling tool, and generates:
 - a behavioral model descriptions in different formal notations, and
 - a description of real-time models.

Transformation analysis tool (TAT) This tool will be used to inspect the correctness, completeness and compatibility of the transformation process.

- **Completeness:** A view is complete with respect to the visual model if every feature in the view corresponds to a feature in the visual model.
- **Correctness:** A view is correct with respect to the visual model if the view is complete and every feature in the visual model has a corresponding feature in the view.
- **Compatibility:** Two views are compatible if and only if both views are correct with respect to the visual model.

We emphasize that the translation need not be one-to-one. However, it is expected that the tool will not miss out on any feature in the visual model, and ensure that there is no ‘junk’ in the output.

Real-time analysis tool (RAT) This tool conducts a real-time scheduling analysis from the description of real-time models that are generated by compiler. The purpose is to check whether or not the verified model can meet the hardware performance constraints.

Model checking tool (MT) This is an important tool at the design stage. It translates the behavioral model descriptions that are generated by Compiler into UPPAAL descriptions and verify the correctness of trustworthy view of system design in the visual modeling tool. The rationale to have this tool in the design stage is to allow developers conduct early verification of their models. The tool, in conjunction with the compiler and transformation analysis tool, makes the formal verification transparent to the developer.

Architectural analysis tool (AAT) This tool will analyze the textual description in TADL that is generated by Compiler and verify the correctness of CBD view of system design in the visual modeling tool.

4.3.2 Implementation Stage

The tools to be used during the implementation stage are discussed below.

Component repository The repository is the storage place for the development source code of a system and a project. It also stores each defined system element specification in a separate file, so that the developers can retrieve and reuse the developed trustworthy components from the repository.

Code generation tool (CGT) This tool analyzes the system design specifications, and produces source code. For every component or connector, if the source code already exists in the repository, the tool will reuse the code, otherwise it will generate the source code. The tool supports different programming languages.

Traceability and certification tool (TCT) This tool scrutinizes the newly generated components code, and verifies that the code conforms with CBD, real-time, and trustworthiness specifications. A detailed introduction of the techniques to do the verification is described in [MA08]. After traceability analysis, the tool obtains a certificate from a *certification authority*. The certificate indicates the trustworthiness of the component and the level of development conformity to design and quality attributes stated in its specifications.

A detailed introduction of how the certificate is issued is described in [MA08].

4.3.3 Run-time Stage

Two tools have been planned for this stage. These are as follows:

Run-time environment In order to support running systems and dynamically reconfiguring executions, this tool is introduced to perform as an intermediary between the language of the run-time environment that communicates with the operating system and the language for running component assemblies. It loads component assemblies from the component repository.

Run-time analysis/visualization This tool supports run-time analysis during system execution. It guarantees that the system behavior is in conformance with the defined functional and nonfunctional properties.

4.4 System Requirements of VMT

From the discussion in previous sections, we can see that VMT serves as the basis for using the other tools in the development framework. A good VMT is expected to enhance the usability of the entire system. In the following sections, we divide the interface of VMT into seven essential parts and describe the functional requirements of each of these interfaces. We explain the functionalities through use cases. A more detailed description of the use cases and their screen shots are given in Chapter 6.

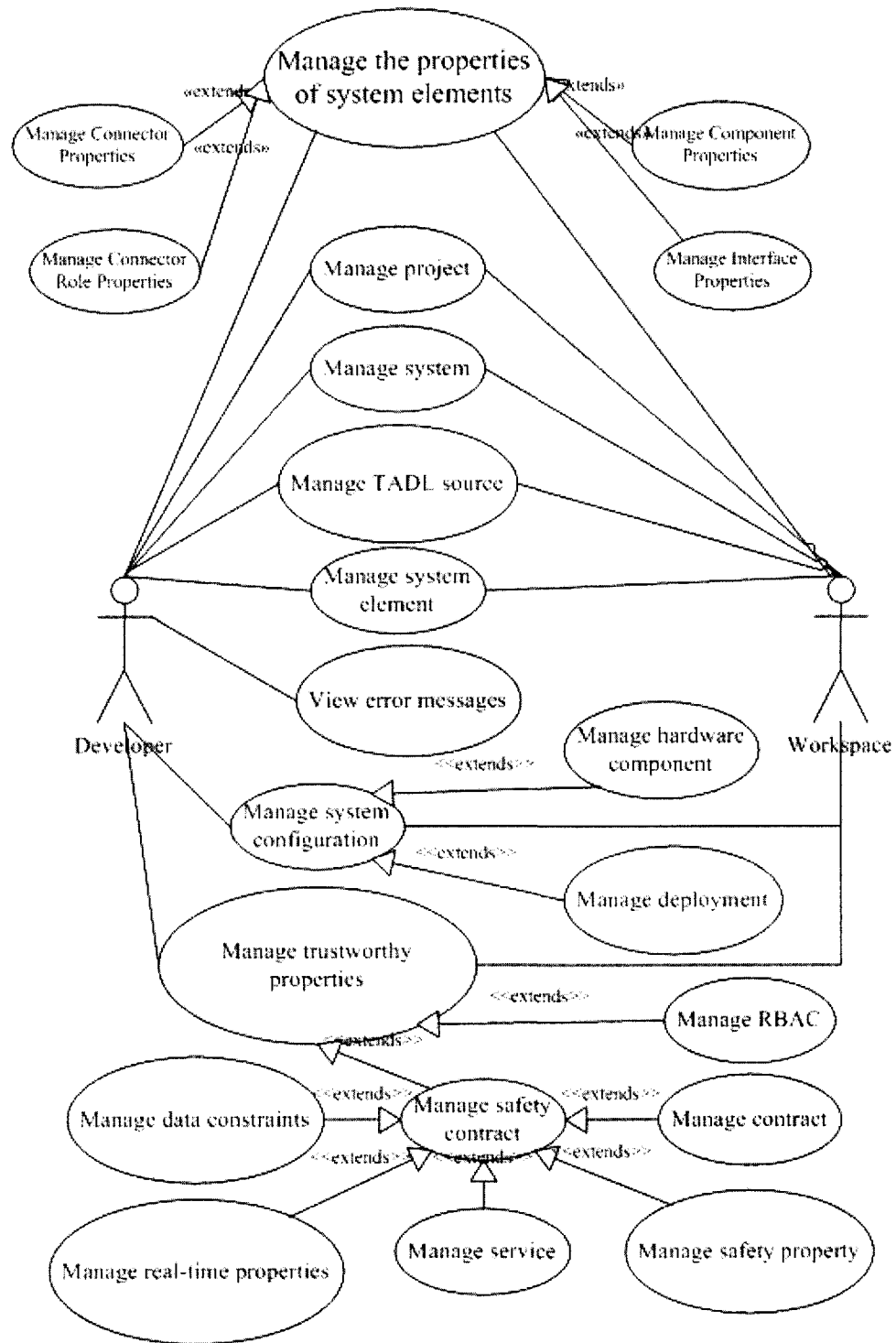


Figure 10: Usecase Diagram of VMT

Figure 10 shows the general use case diagram for VMT. The use case diagram contains two actors (Developer and Workspace), and seventeen major use cases (View Error Messages, Manage project, Manage system, Manage TADL source, Manage system element, Manage Component Properties, Manage Interface Properties, Manage Connector Properties, Manage Connector Role Properties, Manage hardware component, Manage deployment, Manage RBAC, Manage data constraints, Manage real-time properties, Manage service, Manage safety property, and Manage contract). Workspace is the directory where the project stores.

4.4.1 *Navigator Interface:*

This interface allows developers to manage the TADL projects and TADL systems. Each project is represented as a directory, with the tree node to represent a TADL system in this project. This interface is updated dynamically whenever a system/project is added or deleted. The detailed services provided by Navigator Interface are described in Table 1.

4.4.2 *Palette Interface:*

This interface provides visuals for initial system element. The Palette items are composed of multiple visuals of initial system elements, which are classified into two groups: (1) Simple palette items, including component type, interface type, connector type, and connector role type. (2) Composite palette items: packages. The VMT should enable a developer to select a palette item and draw the figure of selected palette item in system editor.

Interface	Navigator Interface
Usecase: Manage Project	This service allows the developers of a TCBS to create a new project, open an existing project, and save a project. When a project is created/opened, a project tree will be generated to allow the developers to view the hierarchy of projects and systems in the current work space. The root of the tree is the project name, with each tree node representing a system in the current project.
Usecase: Manage System	This service allows the developers to create a new system in a specific project, open/delete/save a system. When a system is created, a tree node will be added to the corresponding project tree. When a system is deleted from a specific project, the tree node to represent the system will be deleted correspondingly from the project tree. When a system is saved, an external XML file of the system will be saved in the project directory. The XML file is generated automatically using the predefined XML schema in Chapter 3. And this XML file will be updated dynamically if a developer saves changes in the system.

Table 1: Services provided by Navigator Interface

4.4.3 *System Editor Interface:*

This facility allows developers to visually construct a system. Every system has multiple views, and hence the editor will have multiple sub-interfaces, as discussed below.

- **System Canvas Interface:** This interface allows the elements from the Palette to be used to visually construct a system. The detailed services provided by System Canvas Interface are described in Table 2.
- **Real-time Interface:** Real-time interface collects every real-time feature defined in different components to allow developers to have a general look of all the real-time features of the design. The detailed services provided by Real-time Interface are described in Table 3.
- **Trustworthy Interface:** This interface allows a developer to have a general view of

Interface	System Canvas Interface
Usecase: Manage Sys- tem Element	This service allow the developers to visually create a new system element (component/connector/interface/connector role), to be able to select a specific system element, to move the selected system element within system canvas, save a system element in the system, and to delete the selected system element from the system. In addition, this interface also provides services to attach/remove an interface to a specified component, to attach/remove a connector role to a specific connector, to attach/remove a connector role to a specific interface of a component, and to save the changes.

Table 2: Services provided by System Canvas Interface

Interface	Real-time Interface
Usecase: Manage real-time properties	As discussed in Chapter 2, a time constraint regulates the responses of a component. Therefore, real time properties of a system is defined inside a component. A developer can create/modify/delete a real-time feature in the corresponding component properties definition interface. We will discuss it in the later section of this chapter. In this interface, the developers can view the pre-defined real-time features as well as modify a the properties of a pre-defined real-time.

Table 3: Services provided by Real-time Interface

trustworthy properties of a design. The detailed services provided by Trustworthy Interface are described in Table 4.

- **System Configuration Interface:** This interface shows the features related to system configuration, and deployment. The detailed services provided by TADL Interface are described in Table 5.
- **TADL Source Interface:** This interface shows the textual description of model in TADL. The detailed services provided by TADL Interface are described in Table 6.

4.4.4 *Properties Editor Interface:*

This interface allows a user to edit the attributes and properties of system elements. The detailed services provided by Properties Editor Interface are described in Table 7, and Table 8.

4.4.5 *Error Message Interface:*

This view is used to display the problems, or errors of a design. The detailed services provided by Error Message Interface are described in Table 9. There are various constraints while a developer is making a design using the VMT. The constraints are described as follows:

- Every interface element in System Canvas must be attached to a component element.
- Every connector role element in System Canvas must be attached to a connector element.

Interface	Trustworthy Interface
Usecase: Manage Data Constraint	This service allows a developer to view all the data constraints defined in a design. As discussed in Chapter 2, a data constraint restricts the responses of a component. Therefore, data time properties of a system is defined inside a component. A developer can create/modify/delete a data constraint in the corresponding component properties definition interface. This interface provides services to view and modify the pre-defined data constraint features.
Usecase: Manage Service	This service allows a developer to view all the services in a design. As discussed in Chapter 2, a service is included in the definition of a contract, and a contract is included in the definition of a component. Therefore, service properties are defined inside a component. A developer can create/modify/delete a service in the corresponding component properties definition interface. This interface provides services to view and manage the pre-defined service features.
Usecase: Manage Safety Prop- erty	This service allows a developer to view all the safety properties defined in a design. As discussed in Chapter 2, a safety property is included in the definition of a contract, and a contract is included in the definition of a component. Therefore, safety properties are defined inside a component. A developer can create/modify/delete a safety property in the corresponding component properties definition interface. This interface provides services to view and manage the pre-defined safety property features.
Usecase: Manage Contract	This service allows a developer to view all the contracts defined in a design. As discussed in Chapter 2, a contract is included in the definition of a component. Therefore, safety properties are defined inside a component. A developer can create/modify/delete a contract in the corresponding component properties definition interface. This interface provides services to view and manage the pre-defined contract features.
Usecase: Manage RBAC	This service allows a developer to create/save/view a User/Group/Role/Privilege in RBAC, to delete an existing User/Group/Role/Privilege from RBAC, to add/edit/delete/save/view the properties of a User/Group/Role/Privilege. The properties of a User/Group/Role/Privilege include the name, attribute and constraint. In addition, VMT should also provide services to assign a user to a specific group and remove a user from a group, to assign a user to a specified role and dismiss a user from a role, to assign a group to a specific role and dismiss a group from a role, to assign privileges of events to a role and cancel the privileges of events of a role, and to assign privileges of data parameter to a role and cancel the privileges of data parameters of a role.

Table 4: Services provided by Trustworthy Interface

Interface	System Configuration Interface
Usecase: Manage Hardware Component	This service allows a developer to create/save/view a hardware component in a design, to delete an existing hardware component from a design, and to add/edit/delete/save/view the properties of a hardware component. The properties of a hardware component include: the name, attribute, constraint, and interface.
Usecase: Manage Deployment	This service allow a developer to add/edit/delete/save/view the mapping of a software component to a hardware component.

Table 5: Services provided by System Configuration Interface

Interface	TADL Source Interface
Usecase: Manage TADL Source	This service allows a developer to view textual description of the a design in TADL. The TADL text is automatically translated from the model, and it can not be edited manually. They will be updated dynamically whenever a developer saves a change in a design.

Table 6: Services provided by TADL Source Interface

- Every connector role should be attached to an interface element.
- The name of the system should begin with lower-case or upper-case letters(a-z).
- The developer must select a directory from the navigator interface to create a project or a system.
- The two interface types which are used to connect two components must be compatible [MA08]. That is, if two interfaces are connected by a connector, the two interfaces should contain the same events - one is input event, and the other is output.

4.4.6 *Menu Bar:*

Menu bar is required to be displayed as a horizontal row on top of the GUI. It provides its services from a series of selectable pull-down menu items. The services include create a

Interface	Properties Editor Interface
Usecase: Manage Component Properties	This service allows a developer to add/edit/delete/save/view the properties of a selected component in System Canvas Interface. The properties of a component include: the name, type name, attribute, constraint, user, architecture, and contract.
Usecase: Manage Component Properties - Manage Contract	Contract is defined as a component property. This service allows a developer to add/edit/delete/save/view the properties of a contract in a component. The properties of a contract include: the name, service, and safety property. When a contract is modified in a component, the trustworthy contract view should be updated automatically to show the changes.
Usecase: Manage Con- tract - Manage Safety Prop- erty	Safety Property is defined as a contract property. This service allows a developer to add/edit/delete/save/view the properties of a safety property in a contract. The properties of a safety property include: the name, attribute, event type, and constraint. When a safety property is modified in a contract, the trustworthy safety property view should be updated automatically to show the changes.
Usecase: Manage Con- tract - Manage Service	Service is defined as a contract property. This service allows a developer to add/edit/delete/save/view the properties of a service in a contract. The properties of a reactivity include: the name, attribute, request event, response event, data constraint, time constraint, and update statements. When a service is modified in a contract, the trustworthy service view should be updated automatically to show the changes.
Usecase: Manage Ser- vice - Manage Real-time Properties	Real-time properties is defined as a service property. This service allows a developer to add/edit/delete/save/view the properties of a real-time property in a service. The properties of a time constraint include: the name, attributes, constraint, request event, response event, and MaxSafeTime. When a real-time property is modified in a service, the Real-time interface should be updated automatically to show the changes.
Usecase: Manage Service - Manage Data Constraints	Data constraint is defined as a service property. This service allows a developer to add/edit/delete/save/view the properties of a data constraint in a service. The properties of a time constraint include the name, attributes, constraint, request event, response event, and MaxSafeTime. The properties of a data constraint include the name, attributes, request event, response event, and constraint. When a data constraint is modified in a service, the trustworthy data constraint interface should be updated automatically to show the changes.

Table 7: Services provided by Component Properties Editor Interface

Usecase: Manage Inter- face Properties	This service allows a developer to add/edit/delete/save/view the properties of a selected interface in System Canvas Interface. The properties of an interface include: the name, interface type name, attribute, event, and protocol. Events are defined at interfaces, and therefore, developers should also be allowed to add/save/view an event in a specific interface, and to add/edit/delete/save/view the properties of an event. The properties of an event include: the name, Type of event, attribute, parameter, and constraint.
Usecase: Manage Connector Properties	This service allows a developer to add/edit/delete/save/view the properties of a selected connector in System Canvas Interface. The properties of a connector include: the name, type name, attribute and constraint.
Usecase: Manage Con- nector Role Properties	This service allows a developer to add/edit/delete/save/view the properties of a selected connector role in System Canvas Interface. The properties of a connector role include: the name, type name, attribute and constraint.

Table 8: Services provided by Interface/Connector/Connector Role Properties Editor Interface

Interface	Error Message Interface
Usecase: View Error Messages	This service allows a developer to view the warnings and errors of the current design. The messages are generated automatically when a developer made an operation which contradicts the above mentioned rules, and can not be edited manually. If there is no error or warning, this interface will stay blank.

Table 9: Services provided by Error Message Interface

new project, create a new system, open an existing project, open an existing system, save changes in the VMT and delete.

4.4.7 *Tool bar:*

Tool bar contains a number of icons to provide developers a fast way to access the most frequently used functions of the VMT. It is usually displayed as a horizontal row on top of the GUI directly beneath the Menu bar. The items in the toolbar might be enabled or disabled depending on the status of the System Editor.

4.5 Non-Functional Requirements of VMT

The basic requirement for the design of a visual model tool is to satisfy the needs of the users of the TADL systems. In addition to this list of functionalities, we add the following non-functional aspects.

- *Usability:* The interfaces of the tool should be user-friendly, in the sense a user would need little or no training to use the tool. The visuals for the component, the connector, the interface and packages should be easy to recognize.
- *Reactivity:* The tool should respond in a timely manner for every stimulus from the user.
- *Portability:* The interface should work under different operating systems.

- *Extensibility*: The design would allow developers to easily add new functionalities and components to the tool.

Chapter 5

VMT Architecture: Design

In this chapter we describe the design of the VMT. First, the rationale behind the selected architectural style for the VMT will be detailed. Second, the architecture diagram of the system will be given. Third, in the components section we use class diagrams to describe the detailed design of each component. Fourth and last we discuss the development environment of the tool.

5.1 Architectural Overview

This section demonstrates the Logical View of the VMT. Software architecture is itself based on components, and the design describes the classes that make up a component's internal details. In essence, VMT architecture is a collection of interrelated software components, and the externally visible properties of these components together with a description of the interactions between these components (called connector). There are a number of distinct architectural styles to choose from. By architectural style we mean a set of design rules that identify the kinds of components and connectors that may be used, along with

constraints for combining them. In this thesis, we choose the *Swing's MVC Architecture* – a modified MVC (model-view-controller) style – for the VMT.

5.1.1 Swing's MVC Architecture

The MVC [KP88] is a well-known architecture for graphical user interface (GUI) designs to separate the presentation of data from applications that manage the data. As shown in Figure 11, it breaks each component of a system into the following three parts:

- **Model (Processing):** It manages the behavior and data of a component, provides information to the View when the View requests for a data, and updates the data when it receives a notification from the Controller.
- **View (Output):** It manages the visual representation of a component.
- **Controller (Input):** It decides how components interact with events. It interprets the mouse and keyboard events from the user, and notifies the model and/or the view to update accordingly.

Swing's MVC Architecture is basically a modified version of the MVC design pattern. It combines the View and Controller into a single user interface (UI) object, as shown in Figure 12 [Fow99]. The Model remains as a separate object in Swing's Architecture, the same as the traditional MVC architecture.

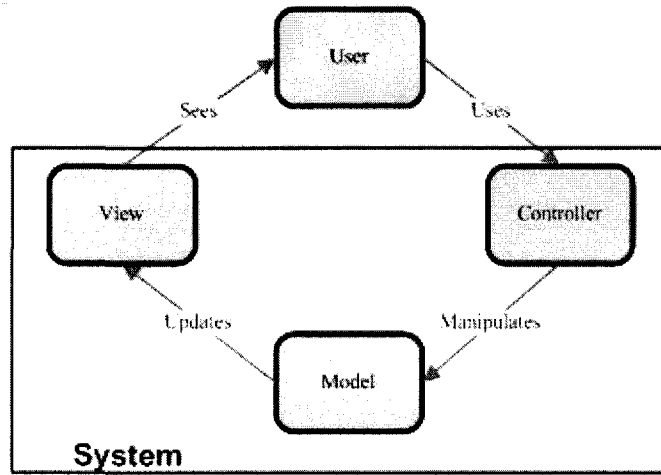


Figure 11: Model-View-Controller Architecture

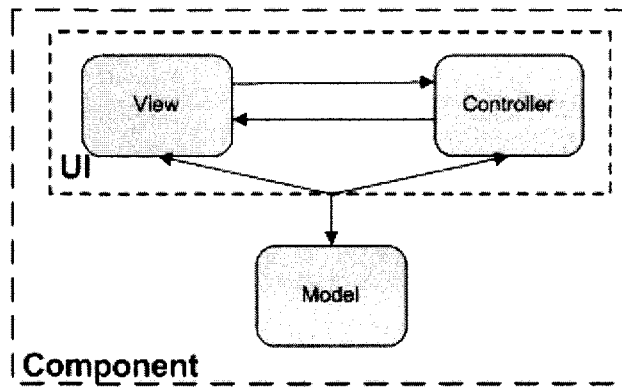


Figure 12: Swing Architecture

5.1.2 Rationale behind selection

A traditional MVC architecture does not suit well in our application for the following reason: In real GUI programming, the view and the controller are always closely related. It is very difficult to write a generic Controller that does not know the specifics about the View [Fow99].

The advantages of Swing's MVC Architecture are described as follows [Fow99] [KP88]:

- It facilitates model-driven programming by treating the model definition as a separate element.
- It facilitates the dynamic changing of the interfaces by using a separate UI object to represent a component's view/controller responsibilities.

5.1.3 Architecture Diagram

Figure 13 shows the structure of the VMT. There are two packages in the diagram, namely GUI and SystemObject. GUI is the UI-Delegate of the system, and it takes care of the outlook as well as the event handling of a system model; System Object represents the model of the system, and it stores and operates the data. A user uses GUI to modify the SystemObjects. GUI shows the visual representations of SystemObjects. There are two external system outputs: 'systemobject.system', and 'system.xml'.

1. *Systemobject.system*: It is an automatically generated XML file by GUI to keep the drawing information of the system. The GUI can also import an XML file, and draw corresponding pictures.

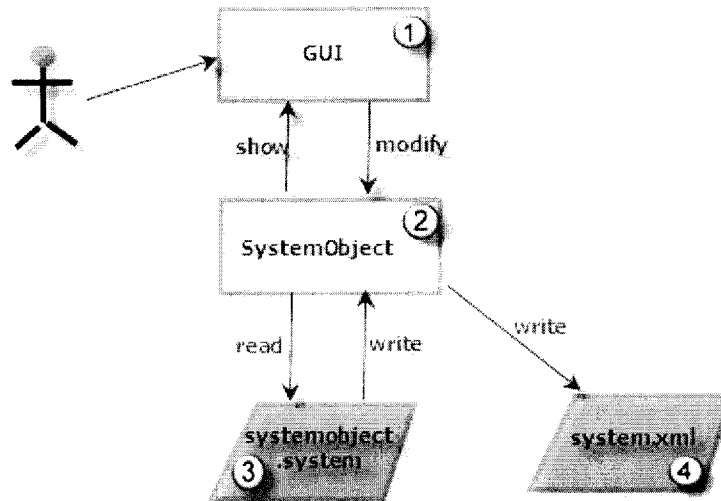


Figure 13: The high-level Architecture Diagram

2. *System.xml*: It is another automatically generated XML file, according to the pre-defined XML Schema. It is used for TADL syntax verification by other tools.

Figure 14 shows the package diagram of GUI and SystemObject. We can see that GUI contains eleven components, according to the requirements described in Section 4.4. Each UI Delegate component in GUI package is connected to its model in SystemObject package, using a dashed line. Error Message UI-D, ToolBar UI-D and MenuBar UI-D represent the error message panel, tool bar and the menu bar of the system, and they do not have corresponding models.

- *The structure of GUI*: Figure 15 shows the high-level class diagram of the GUI. The MainFrame is the window of the VMT. It is composed of the following six main panels:

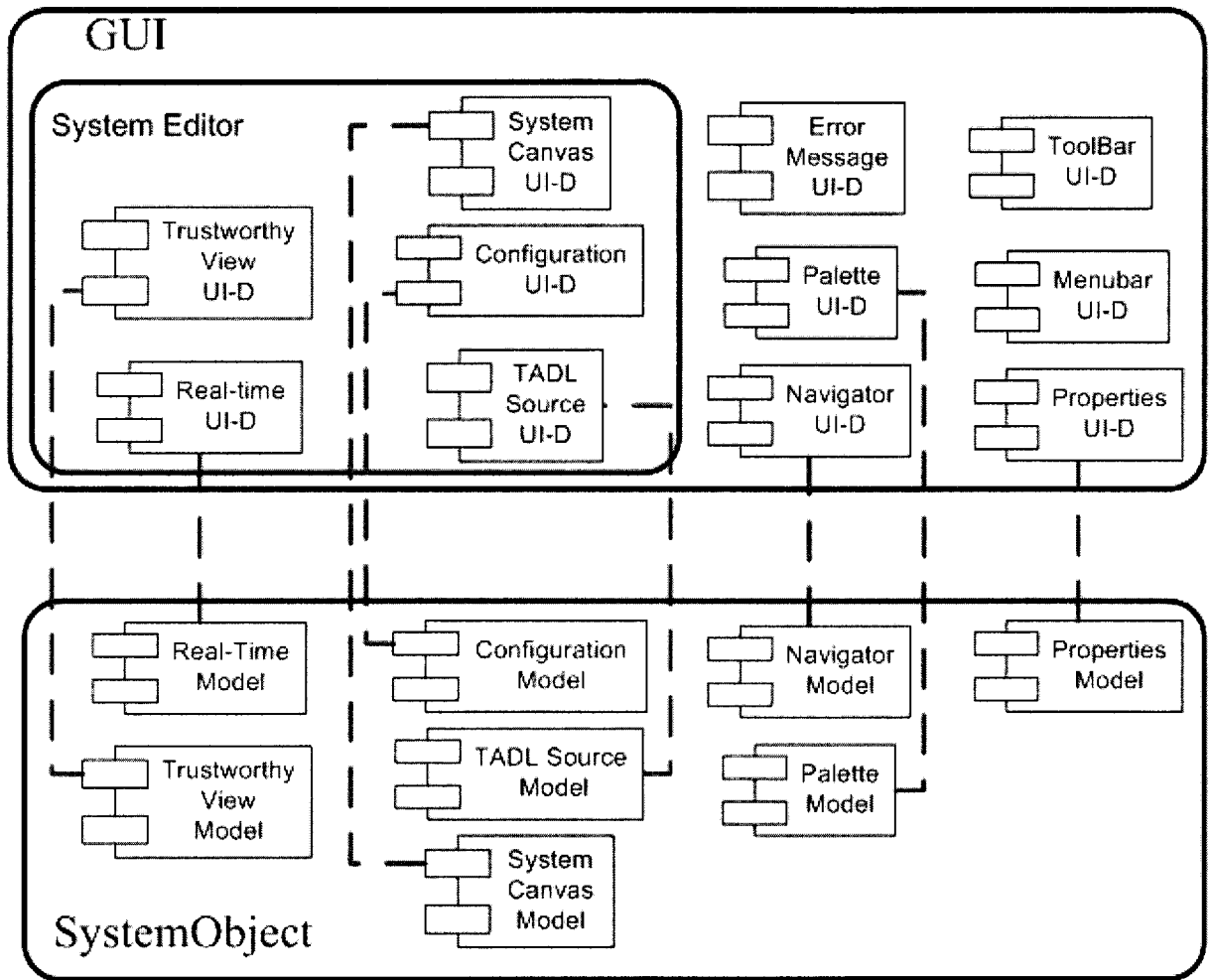


Figure 14: GUI System Package Diagram

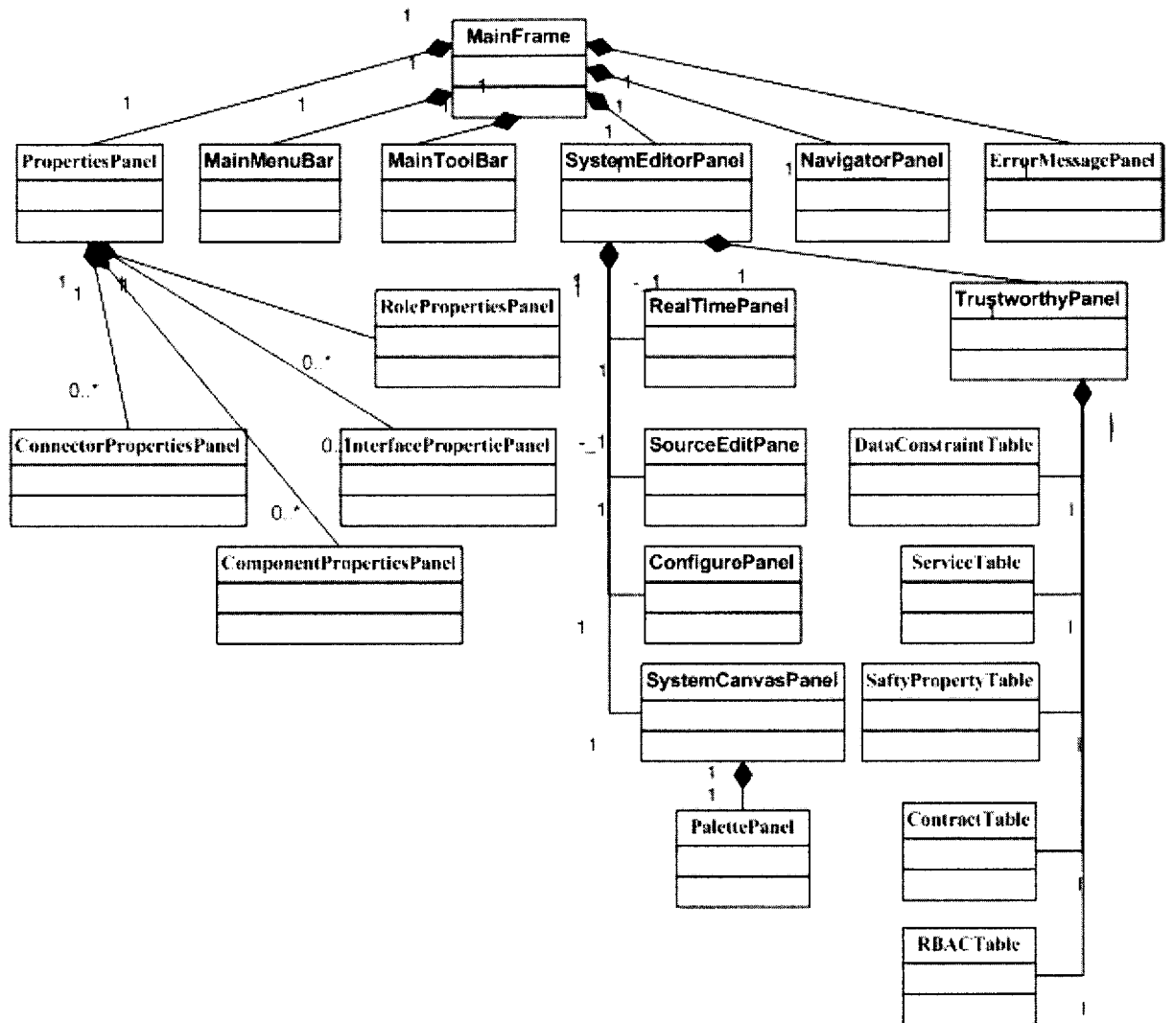


Figure 15: GUI High-level Class Diagram

- Navigator panel: It manages the functionalities provided by Navigator interface.
 - Palette panel: It manages the functionalities provided by Palette interface. Palette is closed related with System Canvas. Therefore, we include the Palette inside a System Canvas in the System Editor panel.
 - System Editor panel: It is the container to hold different views of a system. It is composed of a real time panel to view real time properties, a source edit pane to show the TADL representation of the system model, a configuration panel to manage the system configuration properties, a system canvas panel—which includes a palette panel—to visually construct a system model, and a trustworthy panel to view and manage the trustworthy properties of a system.
 - Properties panel: It is the interface to operate the properties of the selected system element. It is composed of connector properties panel, component properties panel, interface properties panel, and connector role properties panel.
 - Error message panel: It shows the error of the current design.
 - Main menu bar: It is the interface of the system’s menu bar.
 - Main tool bar: It is the interface of the system’s tool bar.
- *The first level structure of the VMT model*: Figure 16 shows that a SystemObject is composed of one to many system items, trustworthy objects, and deployment objects.
 - *The second level structure of the VMT model*: As shown in Figure 17, a trustworthy object is composed of one to many RBAC objects, and contract objects. A contract

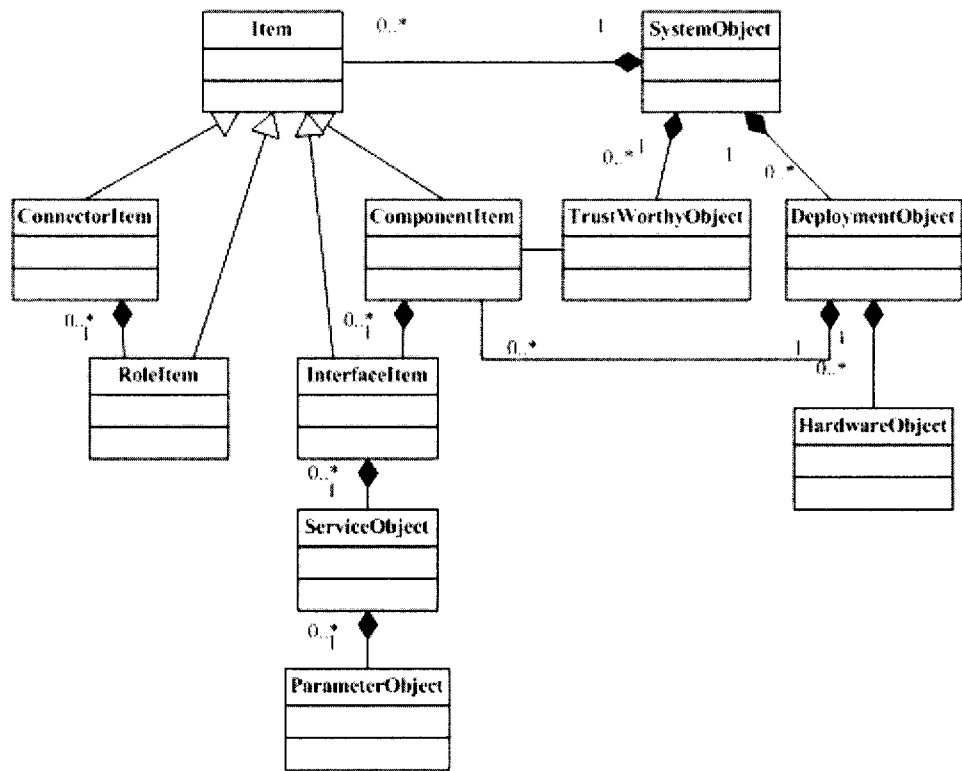


Figure 16: The First Level System Object Structure

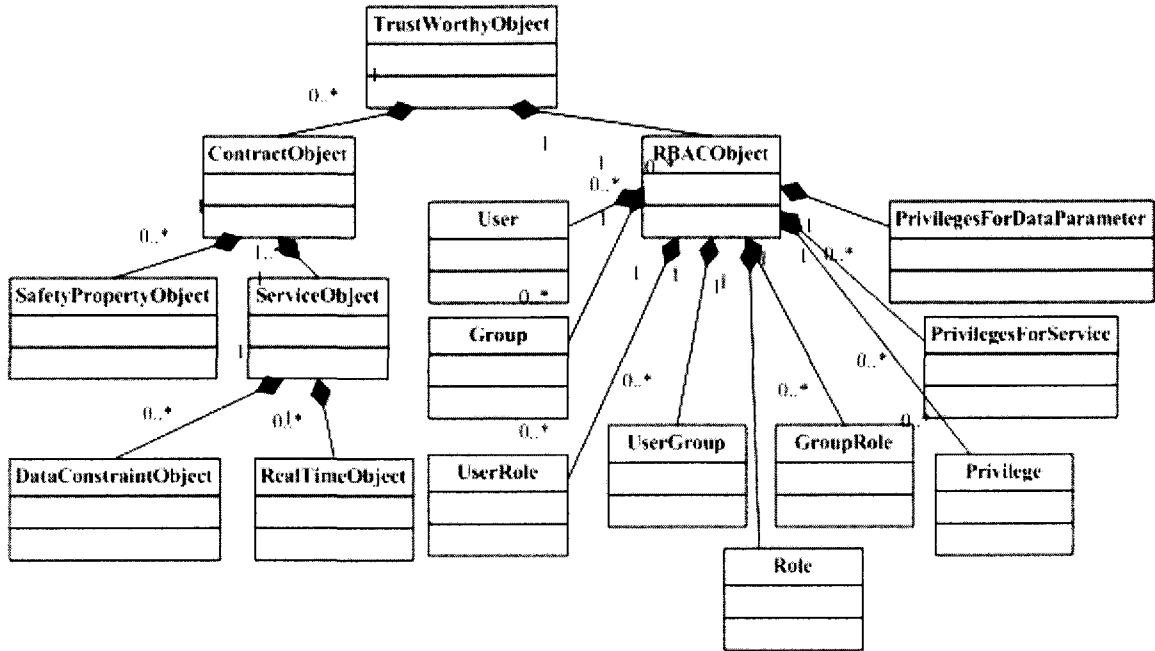


Figure 17: The Second Level System Object Structure

object is composed of one to many safety property objects and service objects. Data constraint objects and real time objects are included in service objects.

5.2 Components

This section describes a detailed description of each component in Figure 14. We use class diagrams to describe the internal structure for each component. In the following sections, we will combine the class diagrams of the UI-Delegate and the Model of a same component together.

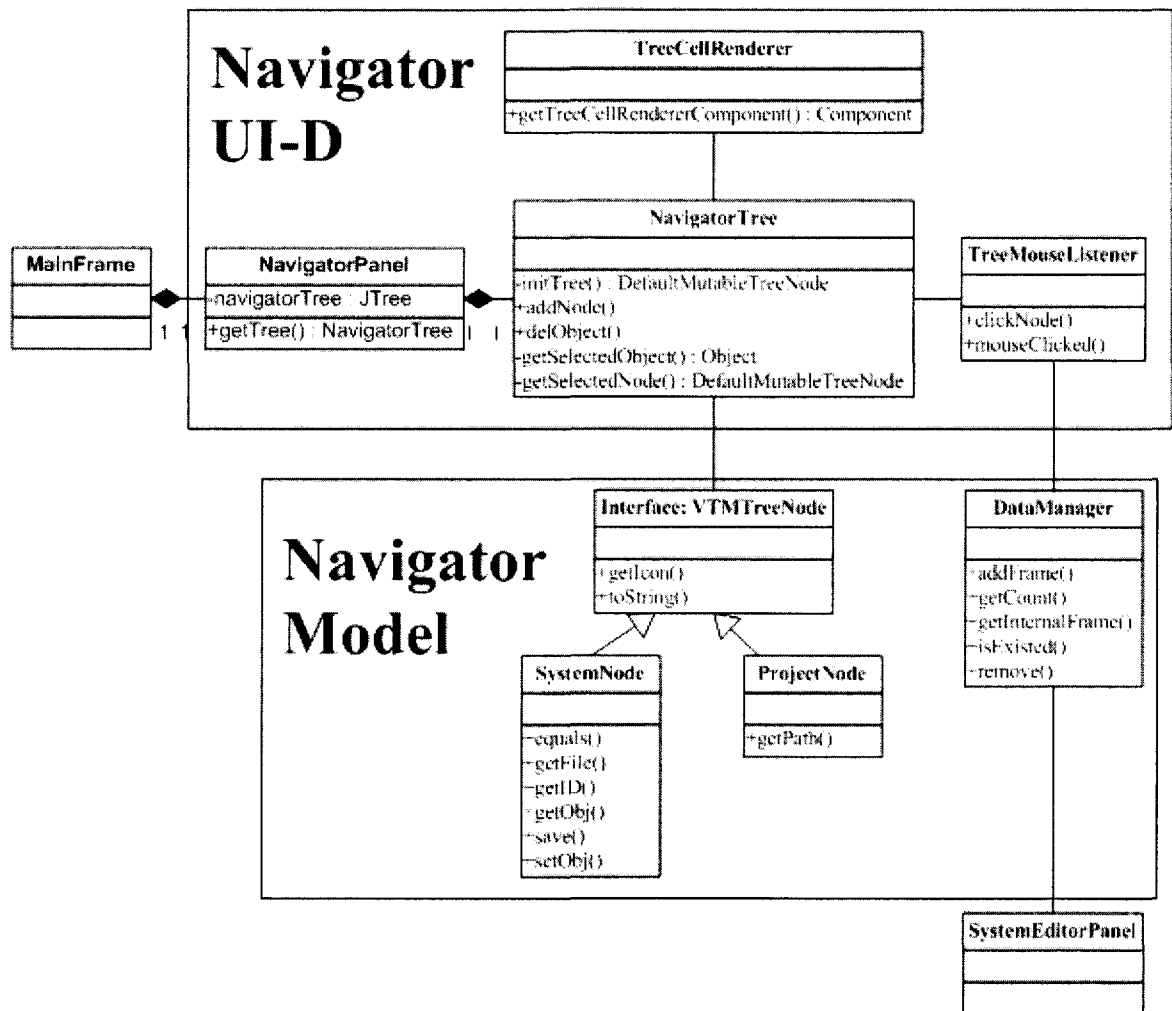


Figure 18: The class diagram of Navigator component

5.2.1 Navigator Component

Figure 39 illustrates the class diagram for the Navigator Component. As we can see from the architecture diagram that the component has been divided into two parts, namely:

- *Navigator UI-Delegate*: This component is responsible for managing the outlook and the events of Navigator component.
 - Navigator Panel: It is the container of Navigator tree. It manages the outlook of the Navigator panel, and gets the Navigator Tree using `getTree()` method.
 - Navigator Tree: This class manages the outlook of the Navigator tree. It updates the tree when a project or a system is added or deleted. It also contains methods to get the current selected tree node and to get the selected object (system or project) at the current node.
 - TreeCellRender: It customizes the display of the navigator tree node, such as customizing the display icons for a system node and a project node.
 - TreeMouseListener: This class defines the behavior of the Navigator when a mouse event is captured. When a developer clicks on a node, the `mouseClicked()` function will capture the current selected object. `clickNode()` method specifies that when the developer double clicks the tree node:
 - * if the system node has already been defined before, then this system will be open in the System Editor Panel.
 - * if the system node is a new system, then this method will create a new frame for the system in the System Editor Panel.

- *Navigator Model:* It is responsible for managing the data of Navigator component.
 - *VTMTreeNode:* It is an interface which defines the common methods that every VMT tree node must have. `getIcon()` is to get the display icon for the tree node, and `toString()` is to get the name of the tree node.
 - *SystemNode:* It is an instance of class `VTMTreeNode`. It manages the display picture, and the data of a system. It contains `save()` method to output an XML file which keeps the drawing information of the system. Here we used an online open source 'Xstream1.3' [Xst08] to serialize objects to XML and transform them back again [Xst08].
 - *ProjectNode:* It is an instance of class `VTMTreeNode`. It manages the display picture, and the data of a project node.
 - *DataManager:* It provides functions to open a selected system node in the `SystemEditorPanel`. `SystemEditorPanel` determines whether the XML of the system already exists or not. If it exists, then read the XML to re-draw the system; if not, then create a new XML file to keep the drawing information of the system.

5.2.2 Palette Component

Figure 42 represents the class diagram for the Palette component.

- *Palette UI-Delegate:* It is responsible for managing the outlook and the events of Palette component.

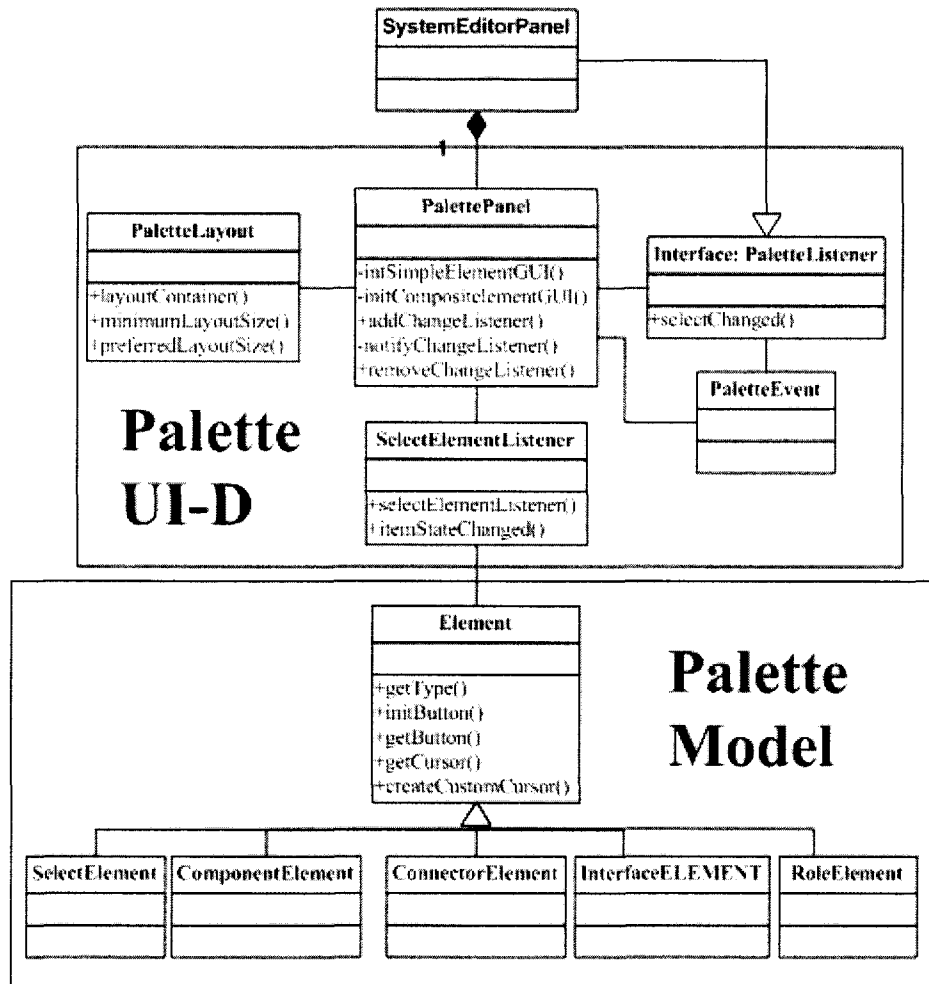


Figure 19: The class diagram of Palette component

- PalettePanel: It is the container of palette items. Palette can not be separated from system canvas, therefore, it should be part of it. Palette panel is partitioned into three parts: select, simple elements, and composite elements. This class initializes the look of the three parts. It also contains `addChangeListener()` method to add listeners to the palette items, `notifyChangeListeners()` method to notify changes to all the listeners, and `removeChangeListener()` method to remove a listener of palette items.
 - PaletteLayout: It is a class to set the lay out for Palette panel. That is, to set the positions of every palette items.
 - Interface PaletteListener: This interface manages the listeners of palette item event. It defines that every palette event listener should have a `selectChanged()` method to receive the currently selected palette element type.
 - PaletteEvent: Whenever a palette item is selected, it will trigger out a palette event.
 - SelectElementListener: This class manages the events when a palette item is selected or deselected by a user.
- *Palette Model*: It is responsible for managing the data of Palette component.
 - Element: It is an abstract class to define the general methods of palette items, such as `getType()` method to get the type of the selected element, `initButton()` method to initialize the position of palette items, `getButton()` method to get the current selected palette items, `getCursor()` method to get the current look

of cursor, and `createCustomCursor()` method to customize the look of cursors when different palette items are selected.

- ComponentElements: This class extends Element class. It customizes the look of component palette item, and the look of cursor when a component item is selected.
- InterfaceElements: This class extends Element class. It customizes the look of interface palette item, and the look of cursor when an interface item is selected.
- ConnectorElements: This class extends Element class. It customizes the look of connector palette item, and the look of cursor when a connector item is selected.
- ConnectorRoleElements: This class extends Element class. It customizes the look of role palette item, and the look of cursor when a role item is selected.
- SelectElement: This class extends Element class. It customizes the look of select palette item, and sets the look of the cursor to system default look when a user presses Select button.

5.2.3 System Editor Component

- *System Canvas*: Figure 20 represents the class diagram for System Editor Component.
 - System Canvas UI-Delegate: It is responsible for managing the outlook and the events of system canvas component.

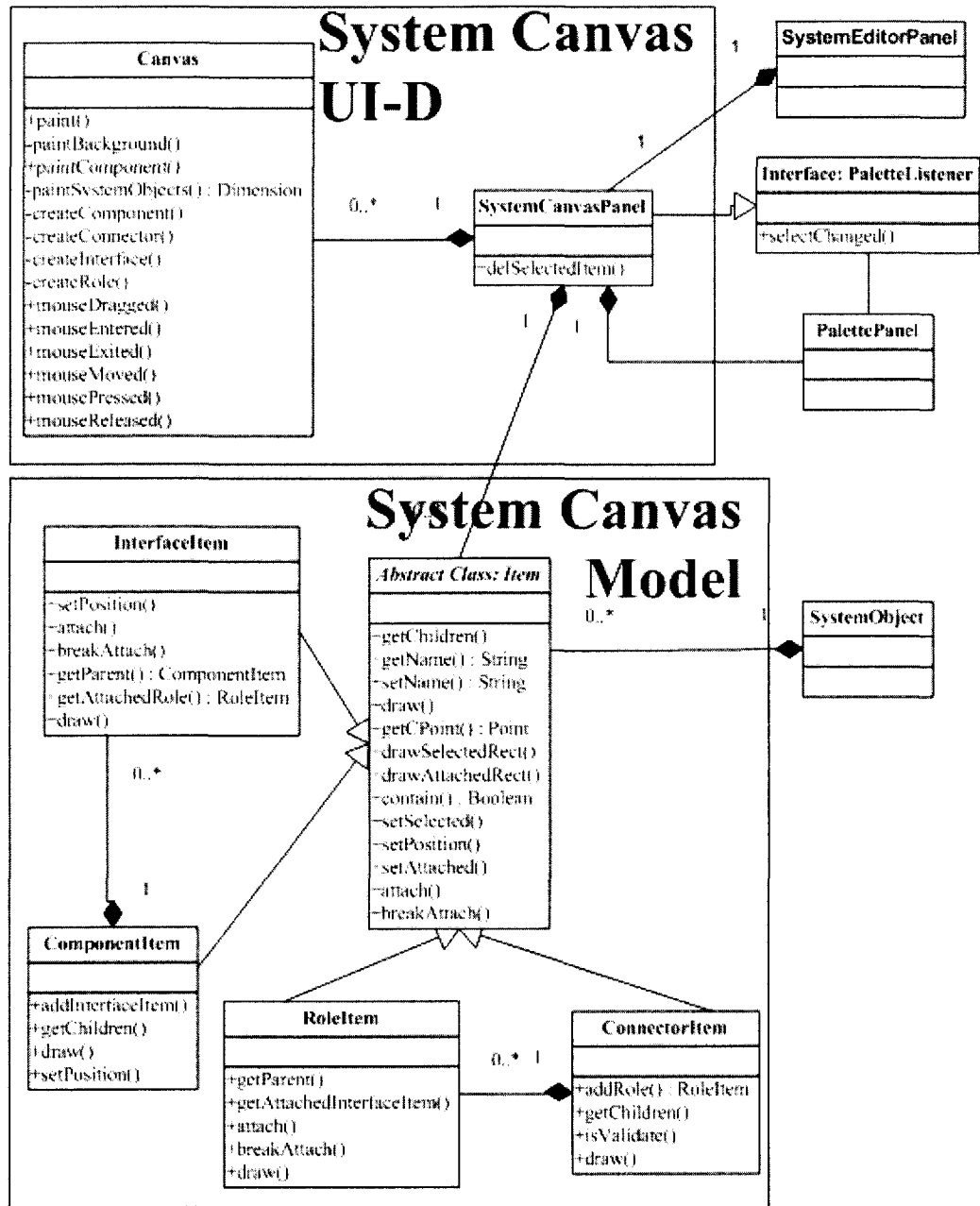


Figure 20: The class diagram of System Editor component

- * **SystemCanvasPanel:** It is the container of system Canvas. It is a listener of palette component. When a palette item is selected, system canvas will receive a notification from palette component about which palette item is selected. It also contains a `delSelectedItem()` method to allow developers to delete a selected item from the system canvas.
- * **Canvas:** It provides a canvas where developers can visually create system components. It contains the following methods: (i) `init()` to initiate the look of canvas; (ii) `paint()`, `paintBackGround()` and `paintComponent()` are intended to draw the canvas component; `paintSystemObjects()` to return the dynamic dimension of canvas.

The drawing canvas will be automatically expanded with many functionalities to show the whole system model: (i) `mouseEntered()` - to set the look of the cursor when the mouse enters system canvas; (ii) `mouseExited()`- to set the look of the cursor to default system mouse look when the mouse exits system canvas; `mousePressed()` to compare the mouse position with the dimension of system elements, if mouse position is included in one of the already defined system elements, then highlight the element, and update the properties view to show the properties of the currently selected element; (iii) `mouseReleased()` - to specify that if a developer selects an item from the palette and releases mouse in system canvas, the graph of the corresponding system element will be drawn in system canvas. This method also determines whether two elements are attached by checking

whether the center point of one element is included in the dimension of another element; (iv) - createRole() to determine whether the position of the role, which is being created, is inside one of the defined connectors. If yes, add the role to the connector; createComponent() to add a component to the system model; (v)createInterface() - to determine whether the position of the interface, which is being created, is inside one of the defined components. If yes, add the interface to the component; (vi) createConnector() - to add a connector with its default two roles to the system model; (vii) mouseDragged() - to specify that when a developer drags a system element, the element graph will move with the mouse. If a developer drags a role and drops it within an interface, then this role and interface are set to be attached; (viii) mouseMoved() to specify that if an interface palette element is currently selected, and when a developer moves the mouse over an already defined component, then this interface is attached to the component.

– System Canvas Model: It is responsible for managing the data of System Editor component.

* Item: It is an abstract class to define the general methods of system elements. It contains the following methods: (i) getChildren() - to return the lists of children of an item; (ii) getName() - to return the name of an item; (iii) setName() - to set the name for an item; (iv) draw()- to draw the graph of an item; (v) getCPoint() - to get the center point of an item;

(vi) drawSelectedRect() - to draw a highlight rectangle outside an item, when it is being selected; (vii) drawAttachedRect() - to draw a highlight rectangle when an interface item is within the area of a component, or a role item is within the area of an interface; (viii) attach() - to attach two items; (ix) constrain() - to determine whether an element is located inside another element; (x) setSelected() - to set an item to the selected status; (xi) setPosition() - to return the new position for an item; (xii) setAttached() - to set the relationship of two items to be attached; (xiii) - breakAttach() to unattach two attached items.

* ComponentItems: It is an instance of Item. It contains the following methods: (i) addInterfaceItem() - to add an Interface into a Component; (ii) draw() - to define the way of drawing a component graph; (iii) setPosition() - to return the current position for the component, and its related interfaces; (iv) getChildren() - to return the lists of interfaces that are contained in this component.

* InterfaceItems: It is an instance of Item. It contains the following methods: (i) setPosition() - to return the current position for the interface and its attached connector; (ii) getParent() - to return the name of the component that the interface is contained in; (iii) draw() - to define the way of drawing an interface graph; (iv) getAttachedRole() - to return the name of the role that the interface is attached to; (v) attach() - to set the attachment between an interface item and a role item; (vi) breakAttach() - to break the

attachment between an interface item and a role item.

- * **ConnectorItems:** It is an instance of Item. It contains the following methods: (i) `addRole()` - to add a role to a connector; (ii) `getChildren()` - to return the list of roles of a connector; (iii) `isValidate()` - to check if all the connector roles are attached to an interface; (iv) `draw()` - to define the way of drawing a connector graph.

- * **RoleItems:** It is an instance of Item. It contains the following methods: (i) `getParent()` - to return the name of the connector that the role is connected to; (ii) `getAttachedInterfaceItem()` - to return the name of the interface that the role is attached to; (iii) `attach()` - to attach an interface item to a connector role item; (iv) `breakAttach()` - to break the attachment between a role and an interface; (v) `draw()` - to define the way of drawing a role graph.

- **Realtime Component:** Figure 21 illustrates the class diagram for the Realtime Panel Component. As we can see from the diagram the component has been divided into two part, whose descriptions are given below.

- Real-time UI-Delegate: It is responsible for managing the outlook and the events of real time panel component.

- * **RealTimeTable:** It manages the outlook, and the behavior of the real time interface. It is implemented as the listener of trustworthy object. which means if a change has been detected in trustworthy object the real time panel will receive a notification. The `getModel()` method in this class is

used to get the model of real time table, which is the RealTimeObject.

The update() method is used to update the RealTimeTable automatically whenever a notification is received from TrustWorthyObject.

- * RealTimeAttributesManager: It defines a pop-up dialog to manage the properties of a RealTimeObject. As is shown in the diagram, AddRealTimeDialog is composed of five sub-interfaces to manage the five kinds of attributes in a RealTimeObject.

- Real-time Model: The only class contained in Real-time Model is RealTimeObject.

- *Trustworthy Panel Component*: Figure 22, Figure 23, Figure 24, Figure 25, and Figure 26 respectively illustrate the class diagrams for the Data Constraint, Service, Safety Property, Contract, and RBAC Panel Component. As we can see from the diagram that every component has been divided into two parts.

- UI-Delegate: It is responsible for managing the outlook and the events of data constraint component.

- * Table: It manages the outlook, and the behavior of the Data Constraint/Service/Safety Property/Contract/RBAC panel. Data Constraint/Service/Safety Property/Contract table is implemented as the listener of trustworthy object, and will update automatically whenever it receives change notifications from TrustWorthyObject. RBAC table provides an interface to manage a RBAC Object.

- * AttributesManager: It defines a pop-up dialog to view and manage the

properties for a Data Constraint/Service/Safety Property/Contract/RBAC Object.

- Model: There is only one class in the model.

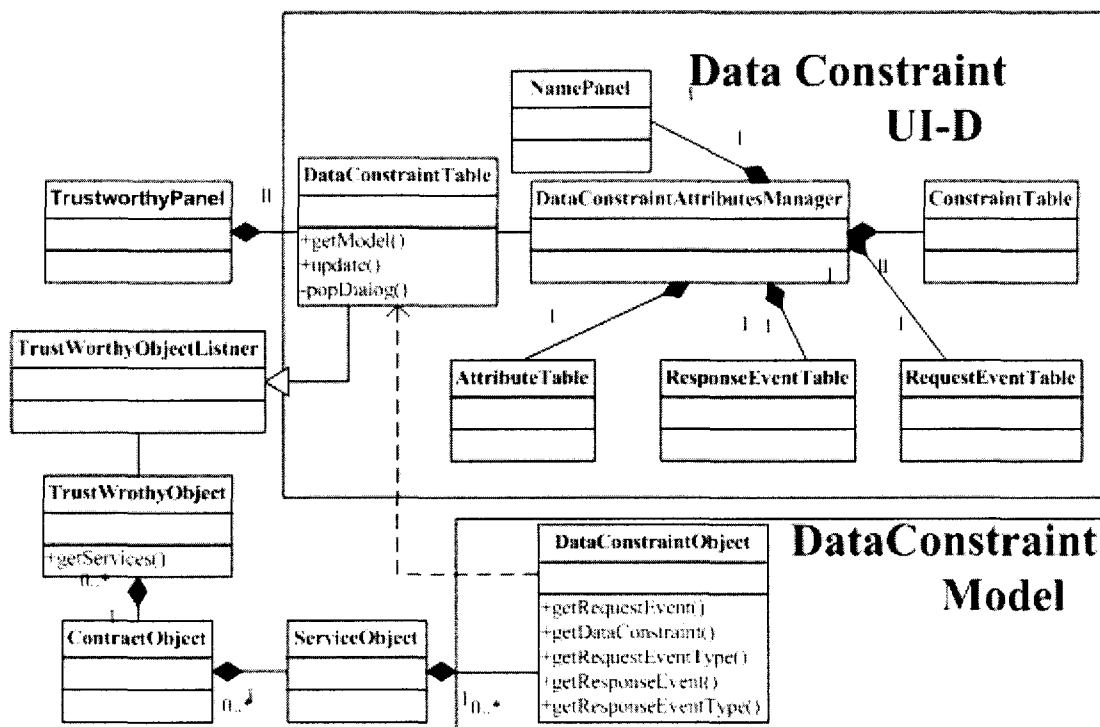


Figure 22: The class diagram of Data Constraint Panel component

- Configuration Panel Component: Figure 27 illustrates the class diagram for the Configuration Component. As we can see from the diagram that the component has been divided into two parts.

- Configuration UI-Delegate: It is responsible for managing the outlook and the events of configuration panel component.

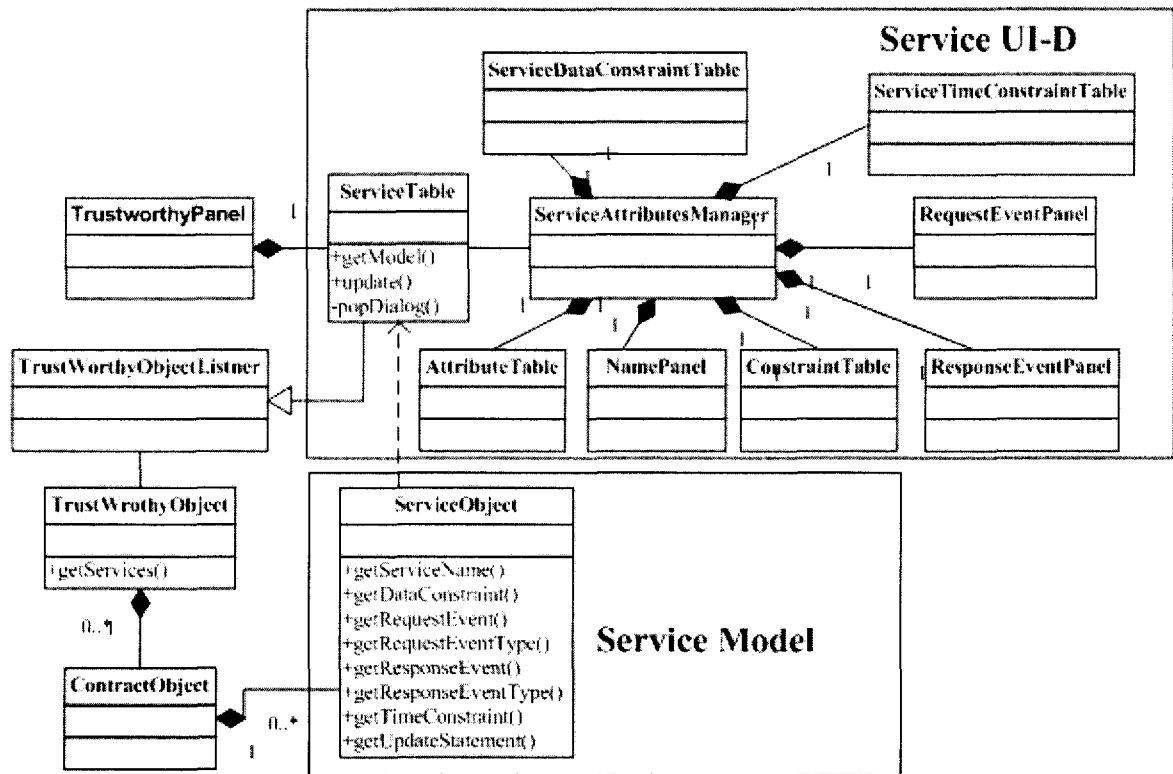


Figure 23: The class diagram of Service Panel component

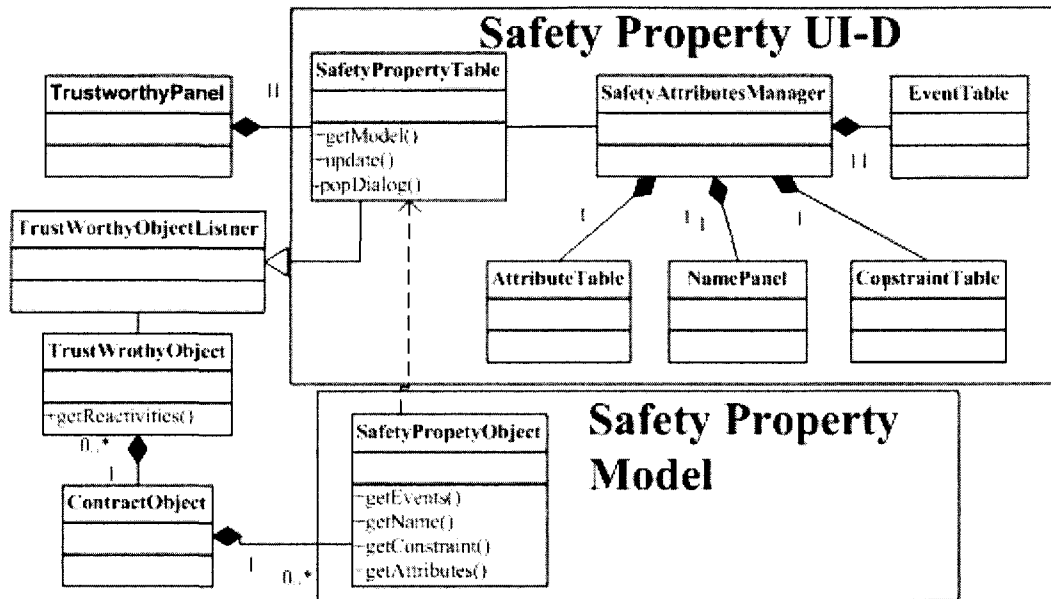


Figure 24: The class diagram of Safety Property component

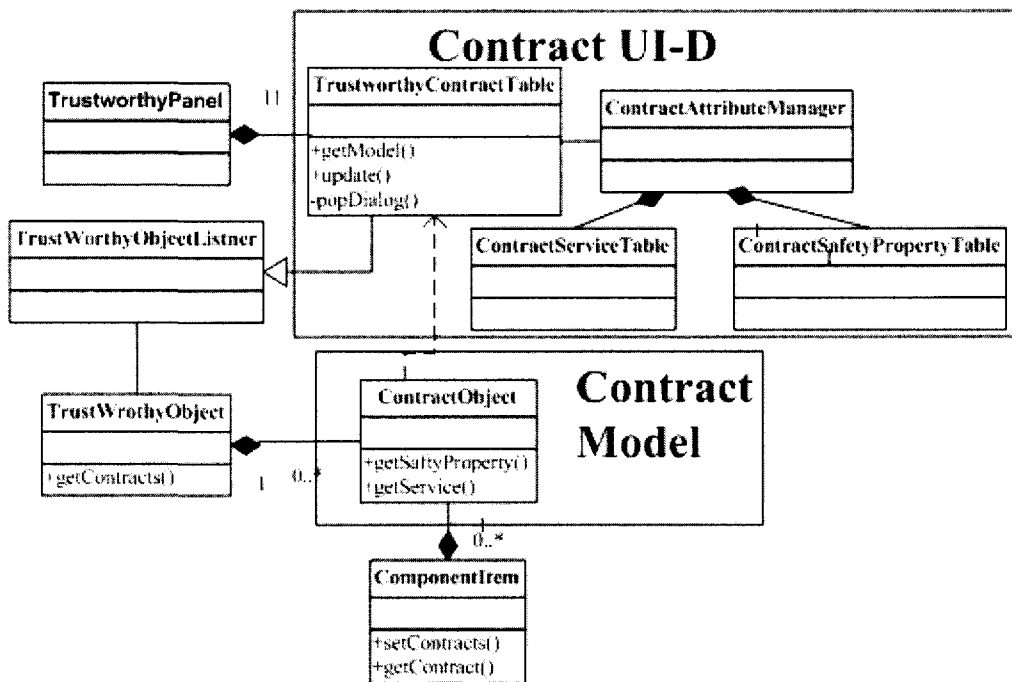


Figure 25: The class diagram of Contract Component

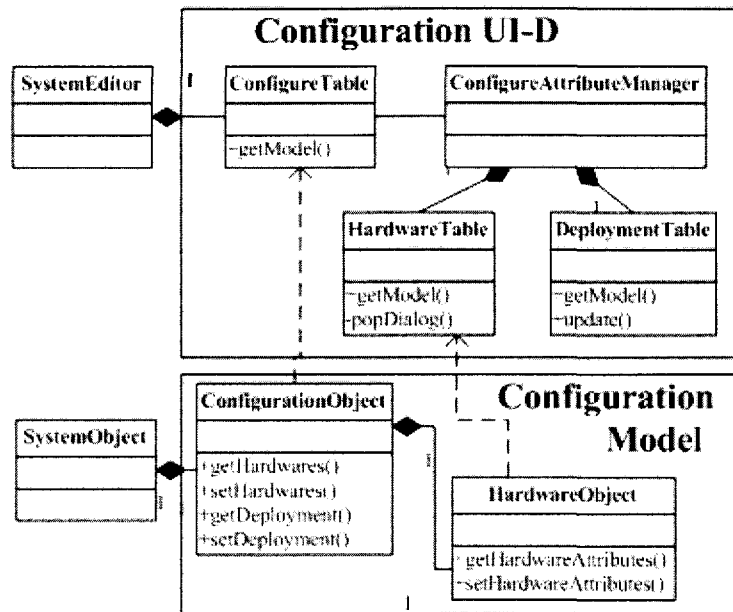


Figure 27: The class diagram of Configuration Component

- * **ConfigureTable**: It provides interfaces to add and delete a ConfigurationObject.
- * **ConfigureAttributesManager**: It defines a pop-up dialog to manage the properties of a ConfigurationObject. As is shown in the diagram, it is composed of two sub-interfaces.
- **Configuration Model**: It manages the data in Configuration interface.
 - * **Configuration Model**: It is the model for ConfigureTable. It includes one to many hardware components.
 - * **HardwareModel**: It is the model for HardwareTable.
- *TADLSourceEditorPanel*: Figure 28 represents the class diagram for the Source

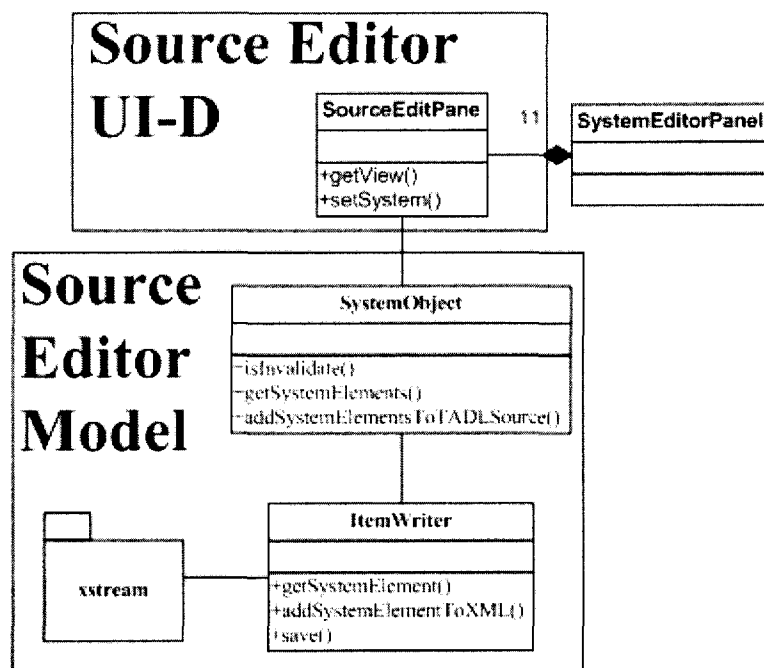


Figure 28: The class diagram of Source Editor component

Editor Panel.

- Source Editor UI-Delegate: It is responsible for managing the outlook and the events of source editor component. It displays the generated TADL of system model, and also generates an external XML file which is conform with the pre-defined XML Schema.
 - * SourceEditPane: This class contains the following methods: (i) getView() - to return the source view to system editor; (ii) setSystem() - to show the translated TADL in source editor panel.
- Source Editor Model: It is responsible for managing the data of Source Editor component.
 - * SystemObject: The structure of System Object has been mentioned earlier in this chapter. It contains the following methods: (i) isInvalidate() - to check the whether system satisfies with the validation rules, if not, error messages will show in the Error Message Panel; (ii) getSystemElements() - to return every system element in the system; (iii) addSystemElementsToTADLSource() - to add every system element to TADL file.
 - * ItemWriter: The methods are: (i) getSystemElement() - to return every system element in the system; (ii) addSystemElementToXML() - to translate every system element to XML element, which is conform with the pre-defined XML Schema. (iii) save() to output the XML file. Here we used an online open source 'dom4j-1.6.1' is reused [Sou08] to create an XML

document.

5.2.4 Properties Component

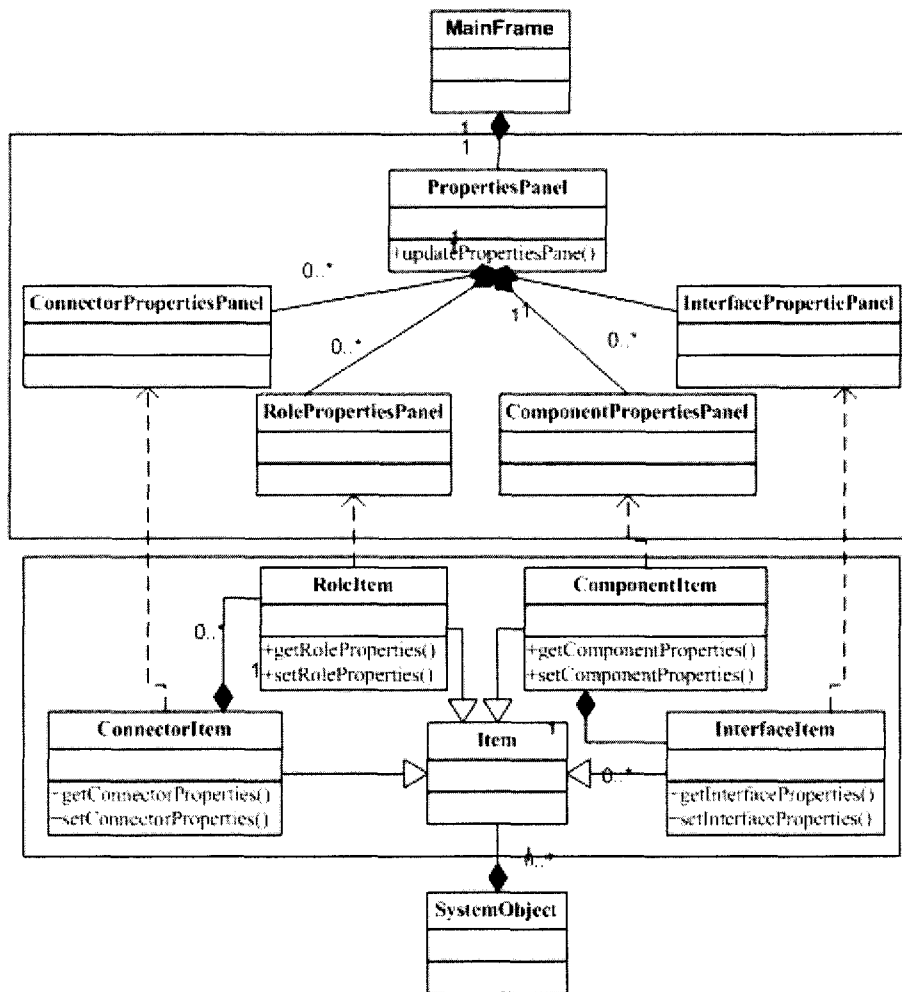


Figure 29: The class diagram of properties component

Figure 29 represents the high level class diagram for the properties view.

- *Component Properties Panel Component:* Figure 30 represents the class diagram

for the component properties panel. It is composed of the several sub panels to manage the properties of a component item. The model of Component Properties Panel is ComponentItem. A ComponentItem includes multiple ContractObjects. ContractObject is a shared data between Trustworthy Contract interface and component properties interface, but only the ComponentItem has setContractObject() method, which means that the contract can only be modified in the component properties panel.

- *Component Contract Panel Component:* Figure 31 represents the class diagram for the component contract properties. The UI-D contains ContractTable to manage the look and events of contract panel. Whenever there is a change in the ContractObject, ContractTable will send a notification to every TrustWorthyObject listener.

- ContractTable contains a ContractServiceTable/ContractSafetyTable to manage the look and events of service/safety property panel in a contract. The corresponding model of ContractServiceTable/ContractSafetyTable is ServiceObject/SafetyObject. ServiceObject/SafetyObject is a shared data between Trustworthy Service/Safety interface and component contract properties interface, but only the Component Contract has setServices()/setSafeties() method, which means that the services/safeties can only be modified in component contract properties panel. The same for DataConstraintObject and RealTimeObject.

- *Interface Properties Panel:* Figure 32 represents the class diagram for the interface properties panel. It is composed of the several sub panels to manage the properties of an interface item. The model of Interface Properties Panel is InterfaceItem. An

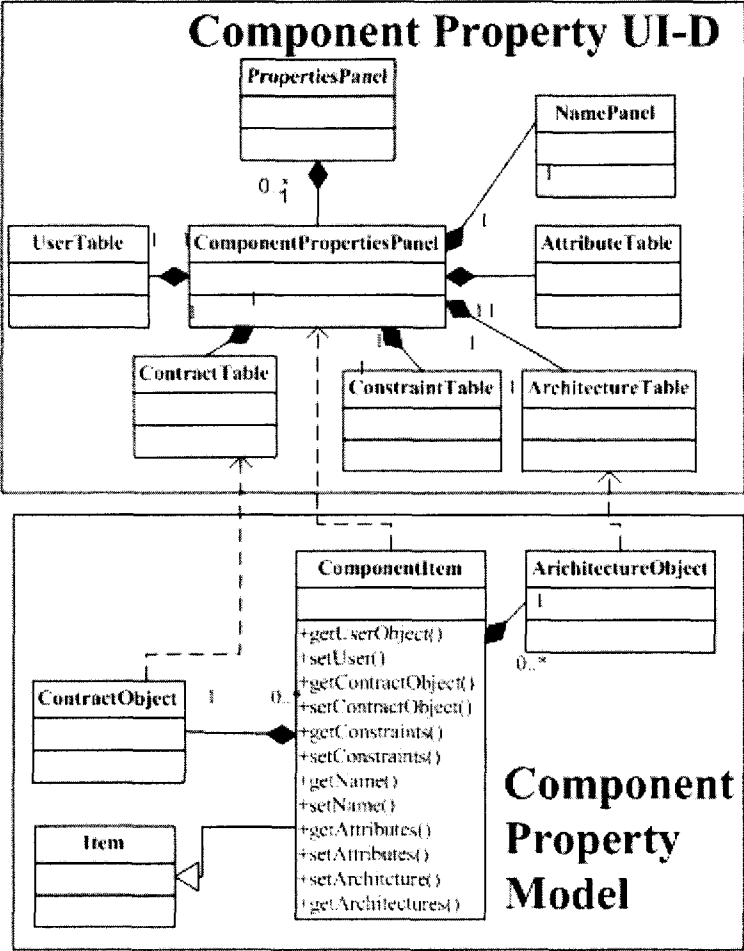


Figure 30: The class diagram of component properties

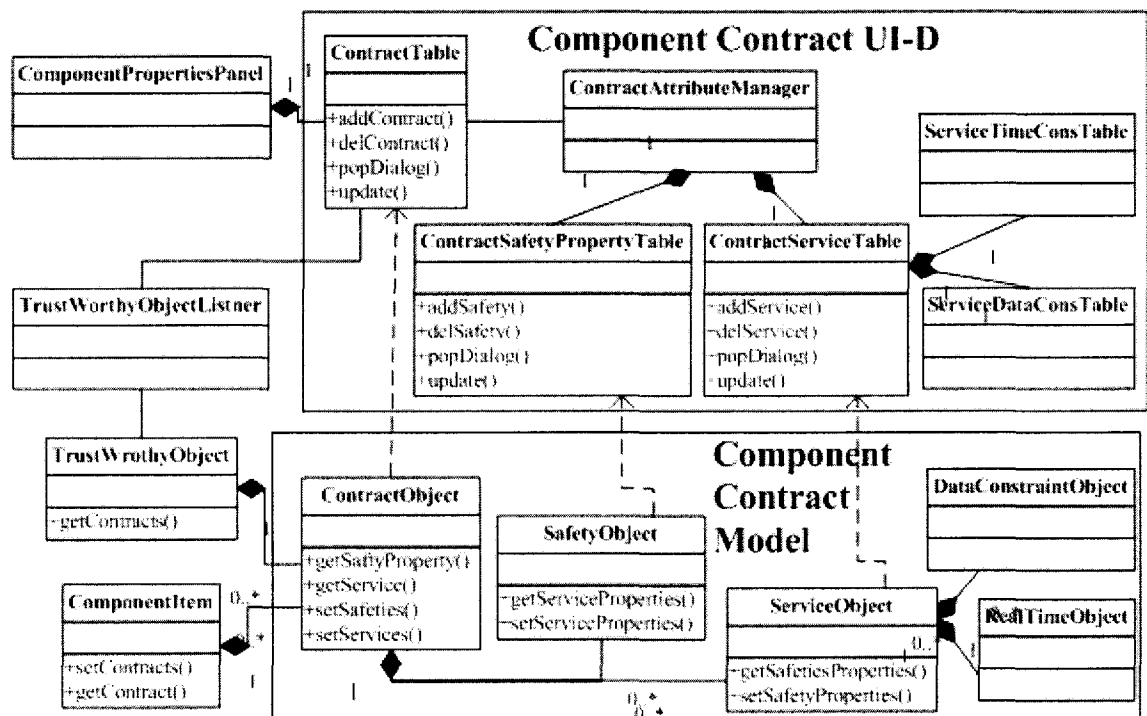


Figure 31: The class diagram of component contract properties

InterfaceItem includes multiple EventObjects. Figure 33 represents the class diagram for the event properties panel.

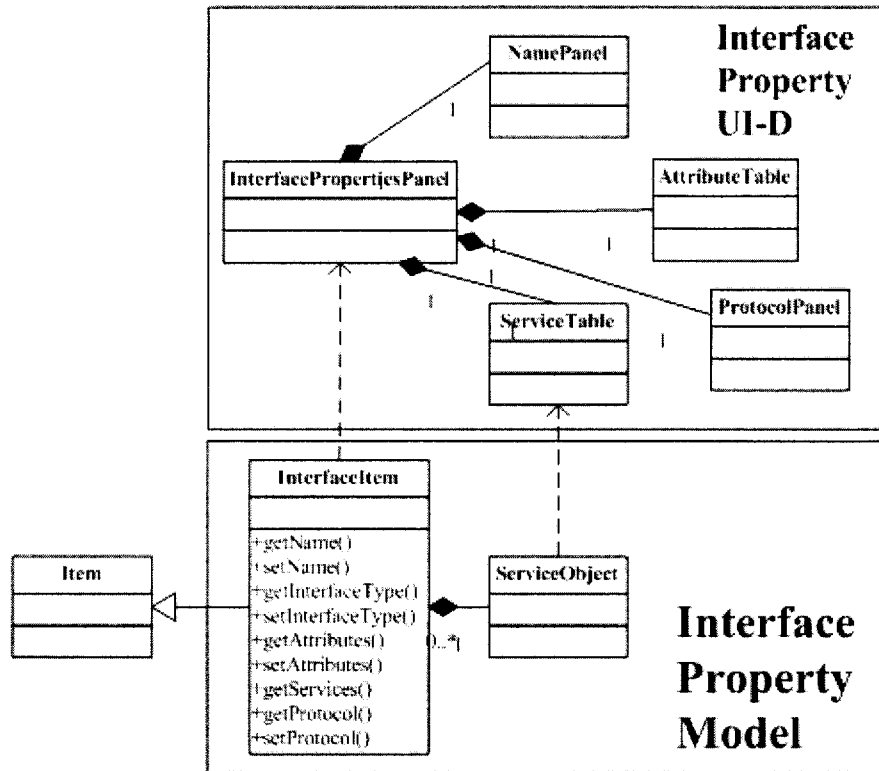


Figure 32: The class diagram of interface properties

- Connector/Connector Role Properties Panel: Figure 34 represents the class diagram for the connector/connector role properties panel. It is composed of the several sub panels to manage the properties of a connector/connector role item. The model of Connector/Connector Role Panel is ConnectorItem/RoleItem.

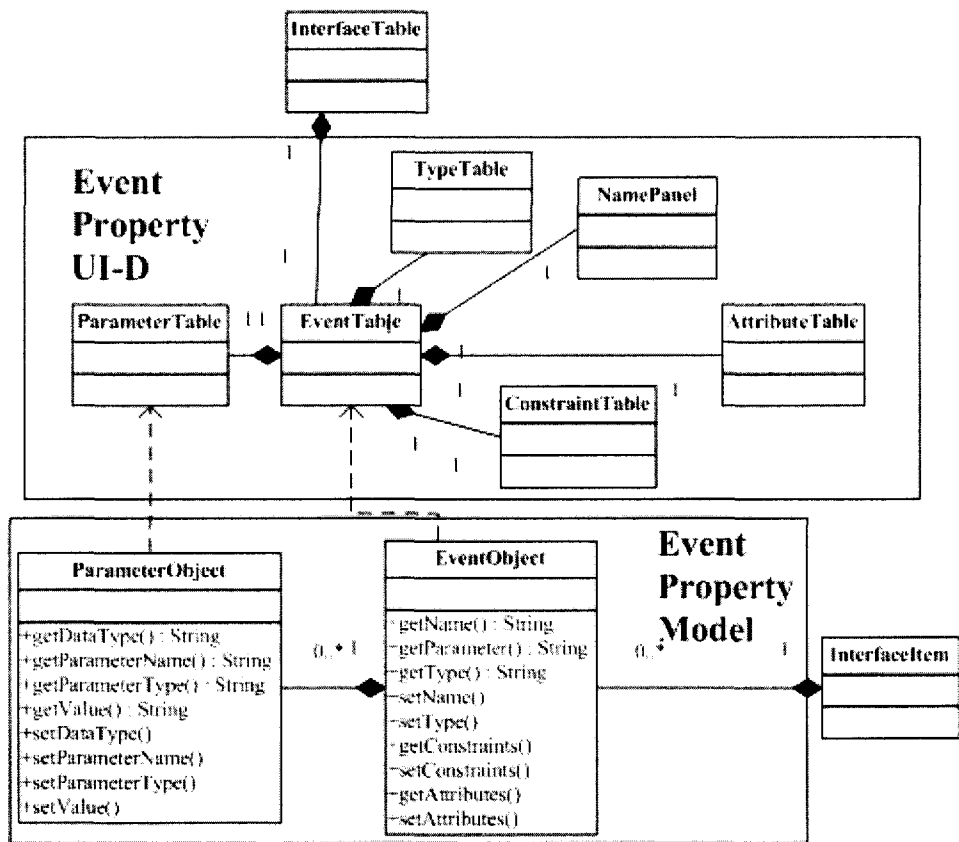


Figure 33: The class diagram of event properties

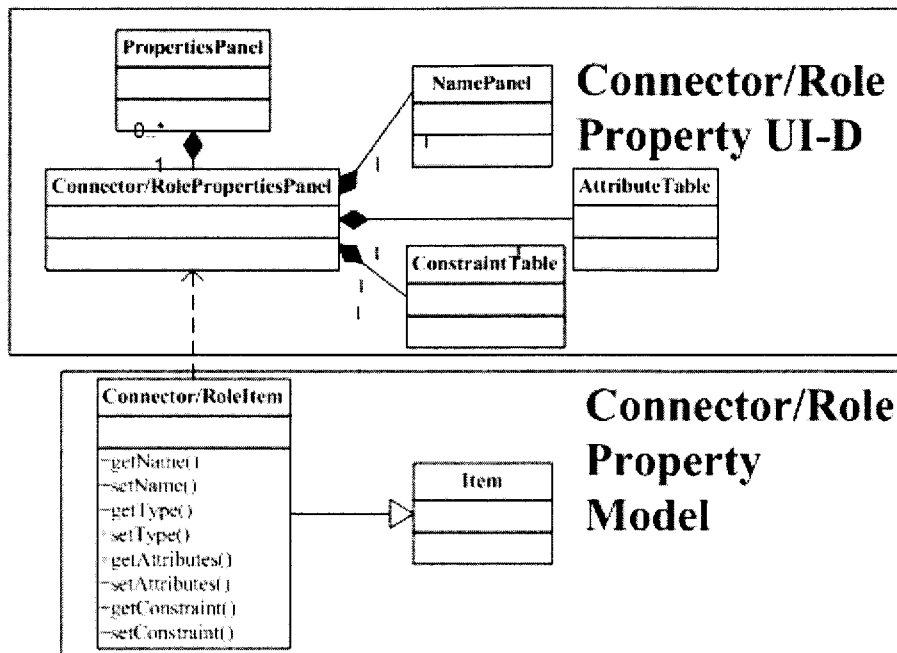


Figure 34: The class diagram of connector/connector role properties

5.2.5 ErrorMessagePanel

Figure 35 represents the class diagram for the ErrorMessagePanel. It detects every operation in the MainFrame of the VMT. If any operation violates the rules, an error message will show in this Panel.

5.2.6 MainMenuBar

Figure 78 represents the class diagram for the Menubar.

- MainMenuBar: is responsible for building the Menubar for the GUI window. It contains at least three menu items: DelButton to delete system elements from system canvas; SaveButton to save a system to an external XML file; NewButton to create a

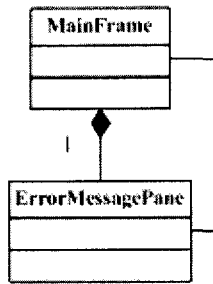


Figure 35: The class diagram of ErrorMessagePanel component

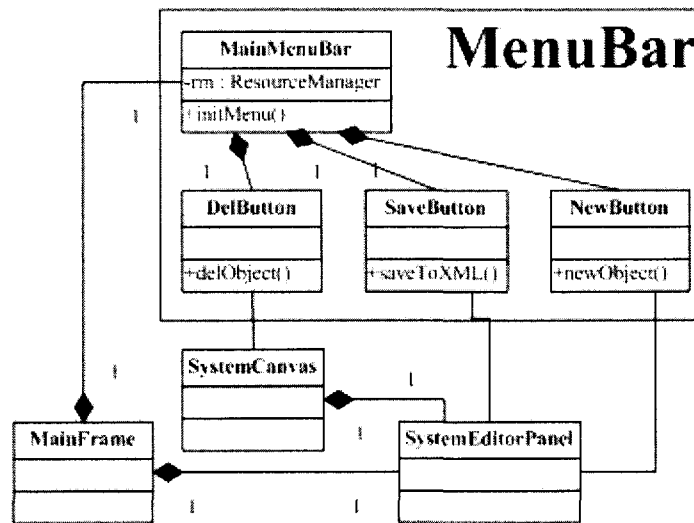


Figure 36: The class diagram of MainMenuBar component

new project or a new system.

5.2.7 MainToolBar

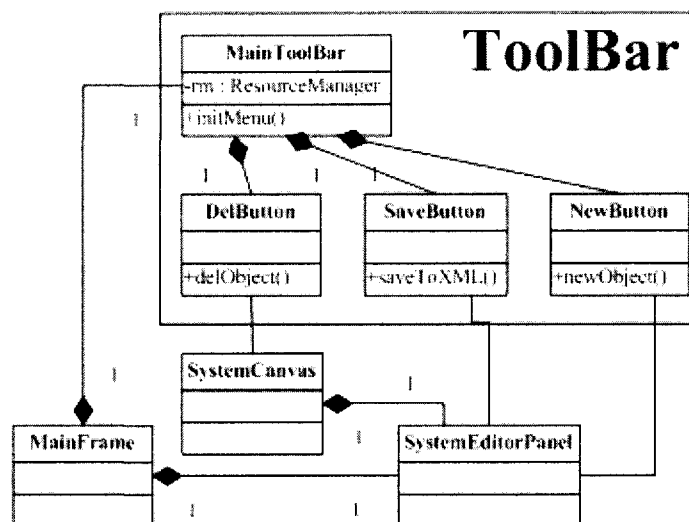


Figure 37: The class diagram of MainToolBar component

Figure 77 represents the class diagram for the Toolbar.

- MainToolBar: is responsible for building the Toolbar for the GUI window. Toolbar just provides a shortcut icon for the most frequently used methods in the MenuBar. So the class diagrams of Toolbar and MenuBar are very similar.

5.3 Development Platform

The VMT is implemented using Java Swing Component. The development environment used to develop the VMT is Eclipse3.3(JDK 1.5).

Swing is a very sophisticated component-based framework to develop rich graphical user interfaces (GUIs) in Java applications. Swing Library has a separately downloadable version for JDK1.1, and it is included as part of Java2(JDK1.2 and up).

The advantages of Java Swing that motivates the selection of it are [Fow99]:

1. Pluggable look and feel. The interface can have a cross-platform Java Look and Feel so the program looks consistent on all platforms (For example, Windows, Unix, and etc). The interface can also change dynamically on different platforms. We choose the later to implement the VMT. This makes the most sense, because VMT will look like the other programs in the system.
2. Separate model and view architecture. We mentioned the advantages of this architecture in Section 5.1.1

We reused the following online open sources while implementing the VMT. These open sources helped to save a lot of developing time.

1. XStream1.3 [Xst08]: As mentioned earlier, XStream is referenced to serialize objects to XML and transform them back again.
2. dom4j-1.6.1 [Sou08]: As mentioned earlier, dom4j is used to create an XML file from scratch.
3. JTableView [Sou07] by SOURCEFORGE.NET: to manage the table views in the VMT.

Chapter 6

Modeling a Trustworthy System using VMT

This chapter provides the snapshots of the VMT GUI, followed by an explanation according to how to use the tool. The order of the screenshots follows the sequence of functional requirements mentioned in Section 4.4.

6.1 GUI Overview

Figure 38 illustrates the organization of the VMT user interface. The framework is composed of seven major interfaces as illustrated in Chapter 4.

6.2 Navigator – refer to Section 4.4.1

Figure 39 is the Navigator Panel on the top left corner of the VMT window. It displays a directory tree of the file where the VMT is installed. The directory tree can be folded by pressing the button on the top of the Navigator Panel.

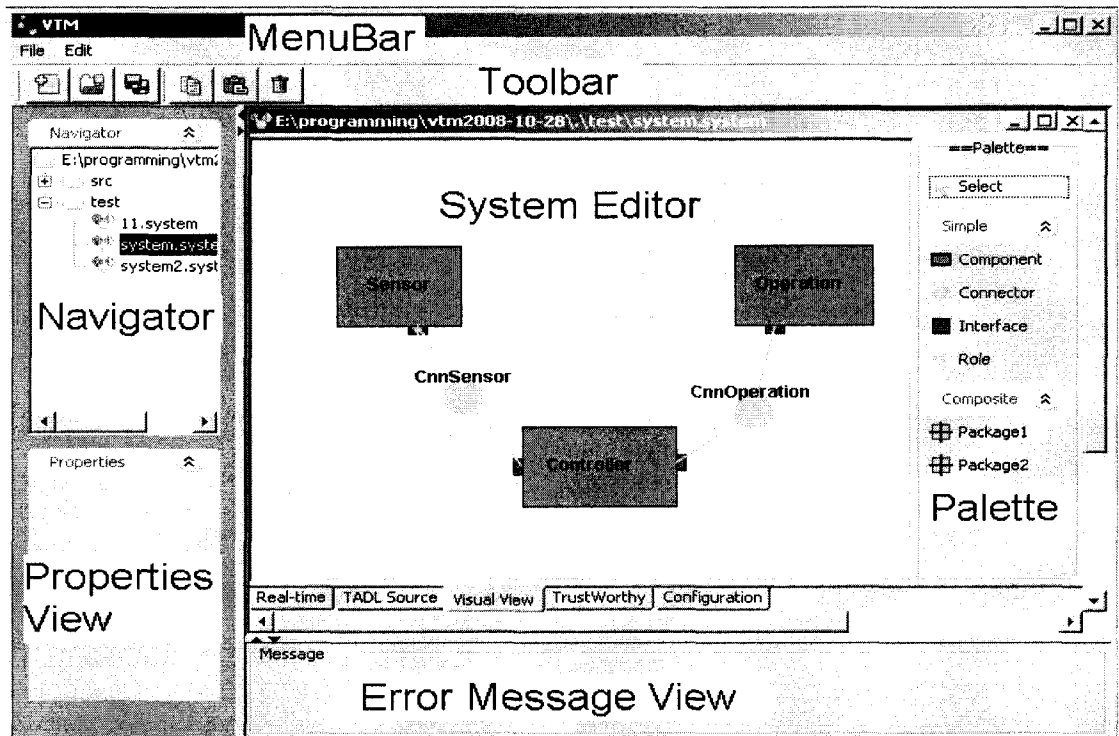


Figure 38: VMT Overview

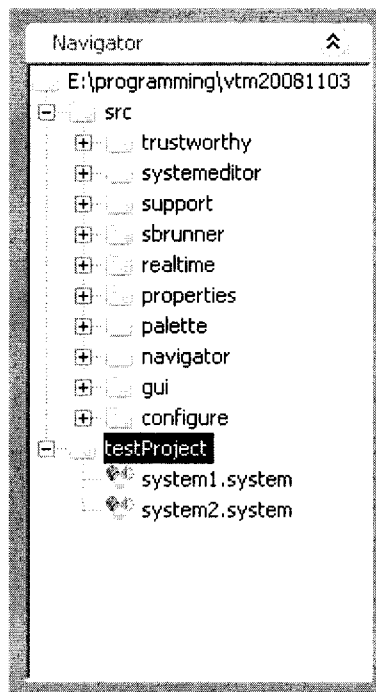


Figure 39: Snapshot of Navigator Panel

- *Use Case: Manage Project*

1. To create a new TADL project: Select a directory in the Navigator panel, and then press the New button from the Toolbar or select 'File–New' from the Menubar. From the pop up window, choose 'Project' and input the new project's name to create a new TADL project, as illustrated in Figure 40. After a project is created, a file node with the project name will be added to the Navigator panel. An external file with the project name as its file name, will be created in the selected directory. If no directory is selected for the project, an error message will show in the Error Message Interface, as in Figure 41

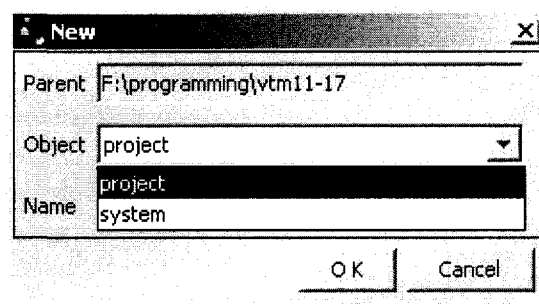


Figure 40: Snapshot to create a new project/system

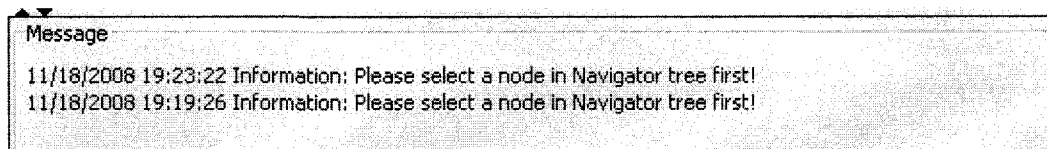


Figure 41: Snapshot of creating project/system errors

2. To open an existing TADL project: Double click the project name in the Navigator Panel. The project tree will be expanded, with the tree nodes to represent

the system in the current project.

3. To save a project, simply press the Save button from the Toolbar.

- *Use Case: Manage System*

1. To create a TADL system: Select a project node in the tree, and press the New button from the Toolbar or select 'File–New' from the Menubar. From the pop up window as illustrated in Figure 40, choose 'System' and input the new system's name to create a new TADL system. After a system is created, a system node will be added to the corresponding project tree. If no project is selected to create a system, an error message will show in the Error Message Interface, as illustrated in Figure 41

2. To open an existing TADL system: Double click the system node in the Navigator tree, and the five views of the system (real-time view, TADL source View, visual view, trustworthy view and Configuration View) will be open simultaneously in System Editor.

3. To delete an existing TADL system: Select a system node in the tree, and press the Delete button from the Toolbar. After a system is deleted, the system node to represent this system will be deleted from the project tree, along with the XML document.

4. To save a system: Simply press the Save button from the Toolbar. An external XML file of the system is generated using the predefined XML schema which has been discussed in Chapter 3.

6.3 Palette – refer to Section 4.4.2

Figure 42 is the Palette on the top right corner of the VMT window. It is composed of three parts: (1) Select item: This allows a developer to select an element in the system canvas. (2) Simple Palette items: This provides a developer the visuals to create simple system element. (3) Composite Palette items: This provides a developer the visuals to create a composite system element. We will discuss how to use the Palette later in Section 6.4.1.

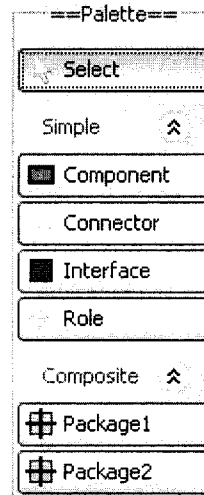


Figure 42: Snapshot of Palette

6.4 System Editor – refer to Section 4.4.3

System Editor lies in the center of the VMT window. It is composed of five views of a system. When a user opens a TADL system from the navigator view, the five views of System Editor will be open simultaneously; and when a user closes a system, the five views of System Editor will be closed simultaneously:

6.4.1 System Canvas

The left part of Figure 43 is the System canvas, and the right part is Palette. System canvas and Palette are closely related. The diagram in System canvas is constructed by selecting and displaying the elements from Palette.

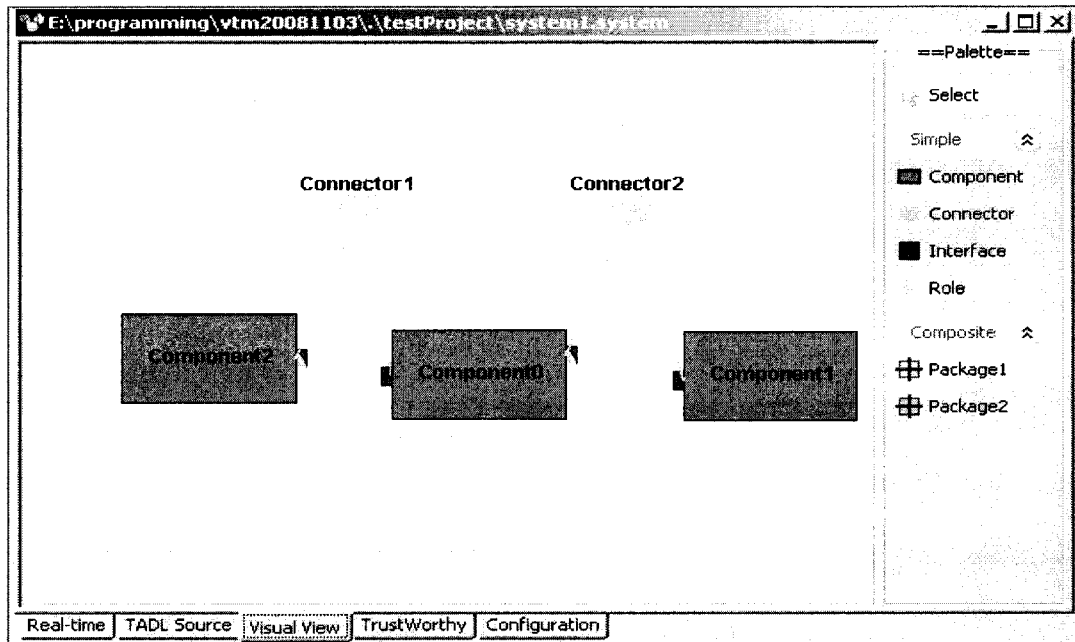


Figure 43: Snapshot of System Canvas

- *Use Case: Manage System Element*
 1. To create a component/interface/connector/connector role: Select the corresponding item in Palette, and drop the mouse in System Canvas. A pop-up window in Figure 44 will show. Fill in the name attribute for the element, and press OK button to finish creating an element.

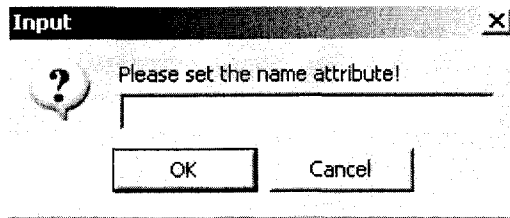


Figure 44: Snapshot to create a new element

2. To save a component/interface/connector/connector role: Press the 'save' button from Toolbar, or select 'File-Save' from Menubar. If a connector role is not connected to any interfaces, an error message should report at this time, as in Figure 45.

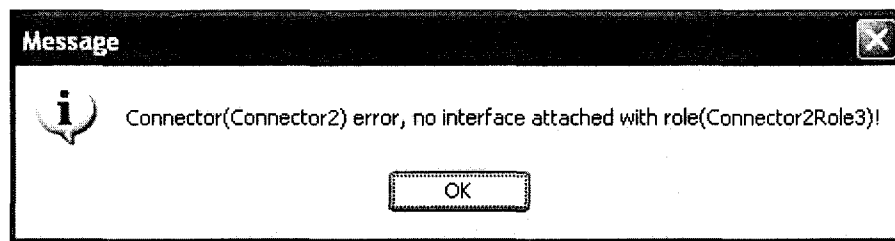


Figure 45: Snapshot for a connector error

3. To delete a component/interface/connector/connector role: Select the element to delete in System Canvas, and press the Delete button from Toolbar. The diagram of the deleted element will be removed from system canvas.
4. To select an element in system canvas: Press Select button in Palette, then a developer will be able to select any element in the system canvas.
5. To move a selected element within system canvas: Hold left mouse button, drag the mouse, and release the mouse at the target position.

6. To attach an interface to a specified component”: Select the interface, and release the mouse over an existing component in system canvas. An interface must be attached to an existing component. Otherwise, a warning window will show as illustrated in Figure 46.

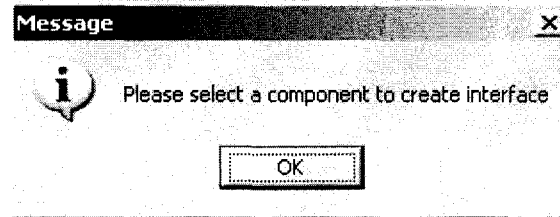


Figure 46: Snapshot for an interface error

7. To attach a connector role to a connector: Each connector contains two roles in default. To attach more roles to a connector, select the role item from Palette, and release the mouse over an existing connector diagram in system canvas. A role must be attached to an existing connector. Otherwise, a warning window will show as illustrated in Figure 47.

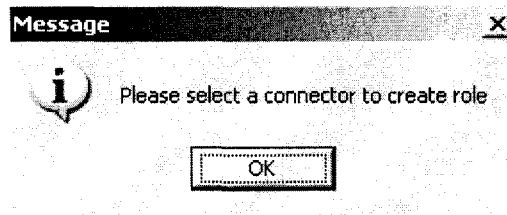


Figure 47: Snapshot for a role error

8. To attach a connector role to an interface of a component: Select the connector role, hold left mouse button, drag the mouse, and release the mouse over the

destination interface.

6.4.2 Realtime View

- *Usecase: Manage real-time properties* Figure 48 shows the real-time interface. Double click the selected real-time name, a pop-up window will show to allow developers to view and manage the properties of real-time properties.

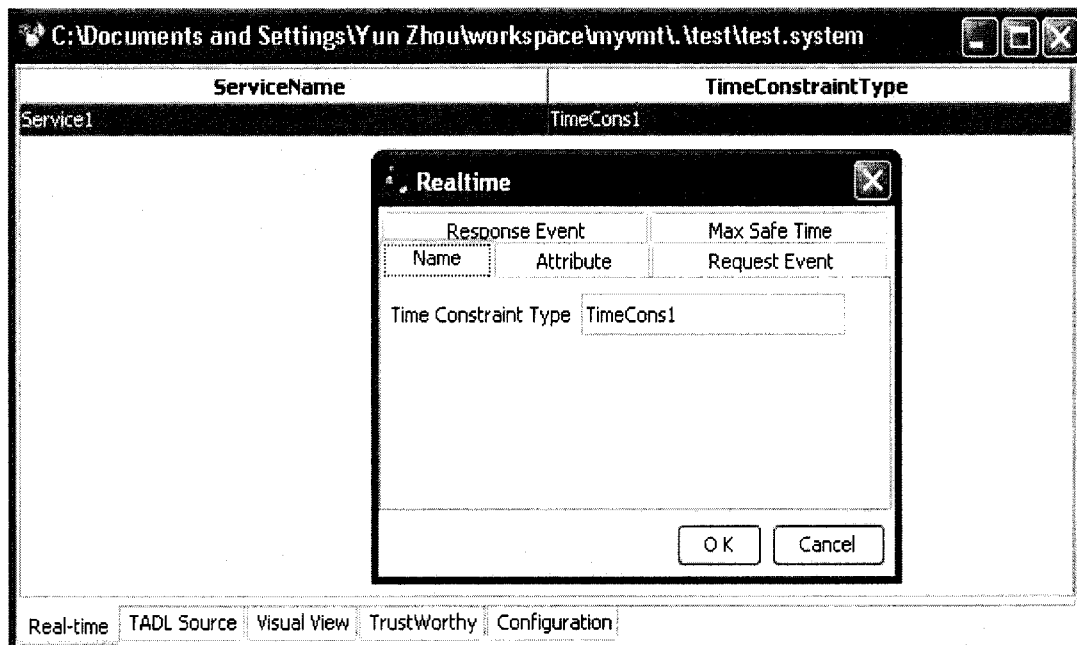


Figure 48: Snapshot of Real-Time Panel

6.4.3 Trustworthy View:

This sections covers the five use cases discussed in Section 4.4.3 Table 4.

- *Usecase: Manage Data Constraint/Service/Safety Property/Contract* Figure 49,

Figure 50, Figure 51, and Figure 52 respectively illustrate the window of managing the properties of data constraint, service, safety property and contract. Double click on the selected row, and a pop-up window will show to allow users to view and manage the properties of a data constraint/service/safety property/trustworthy contract.

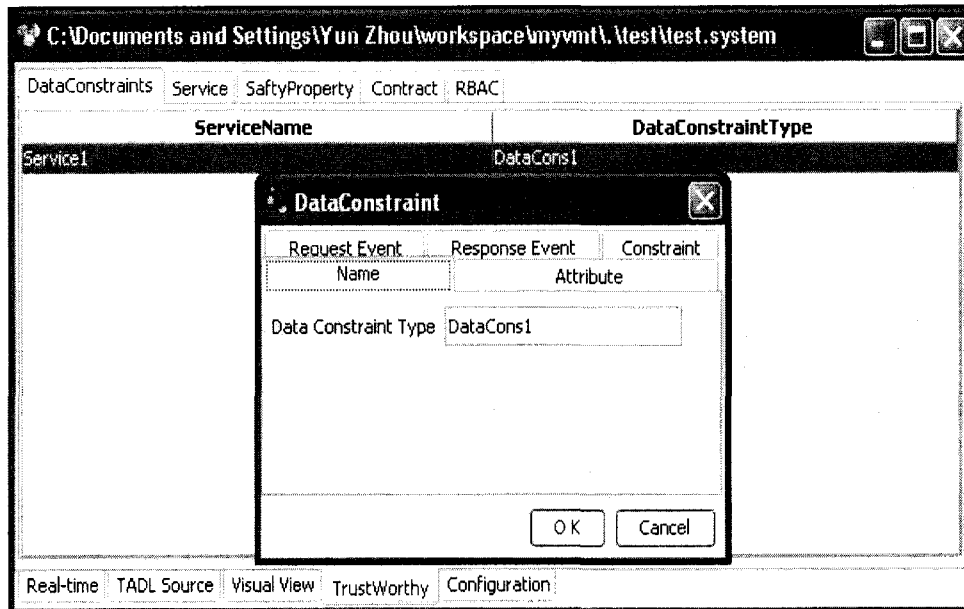


Figure 49: Snapshot of Data Constraint Panel

- *Usecase: Manage RBAC* Figure 53 illustrates the interface to manage the security mechanisms of a design.
 1. *To manage the properties of RBAC:* Figure 54, Figure 55, Figure 56, and Figure 57 respectively illustrate the window of managing user, group, role and privilege of a design.
 - To create a User/Group/Role/Privilege in RBAC: Press the 'Add' button on the bottom right corner of RBAC window. A pop up window will show

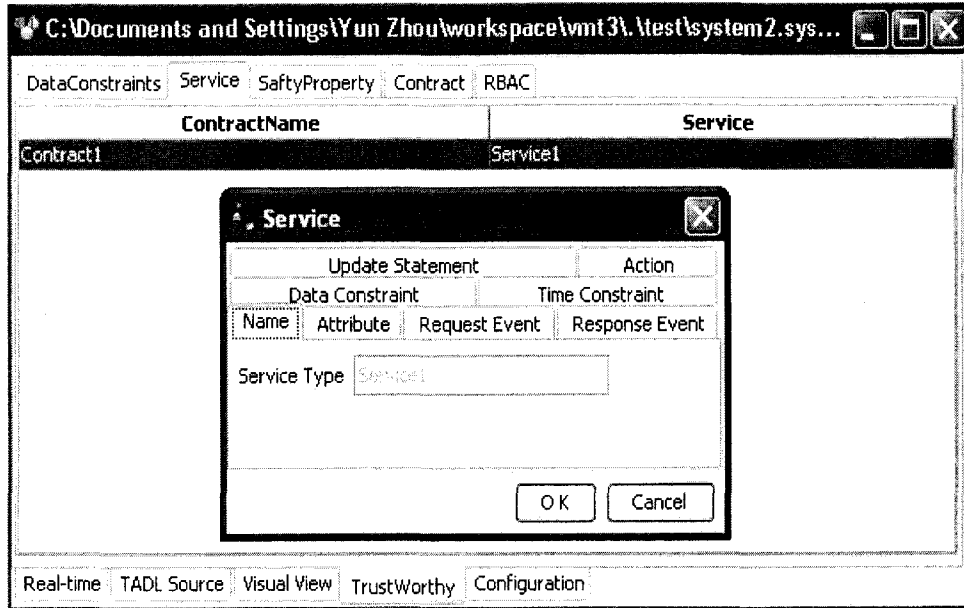


Figure 50: Snapshot of Service Panel

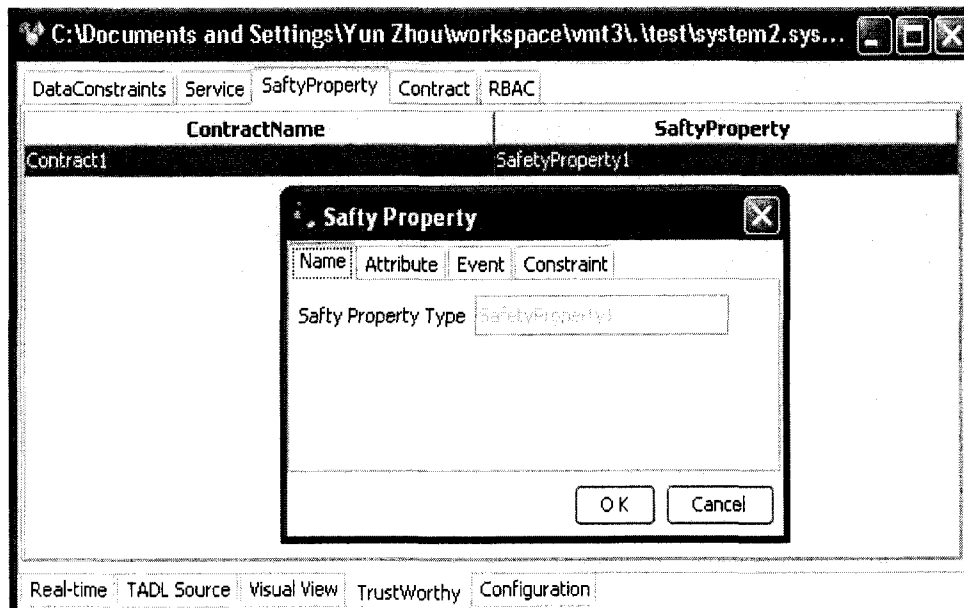


Figure 51: Snapshot of Safty Property Panel



Figure 52: Snapshot of Contract Panel

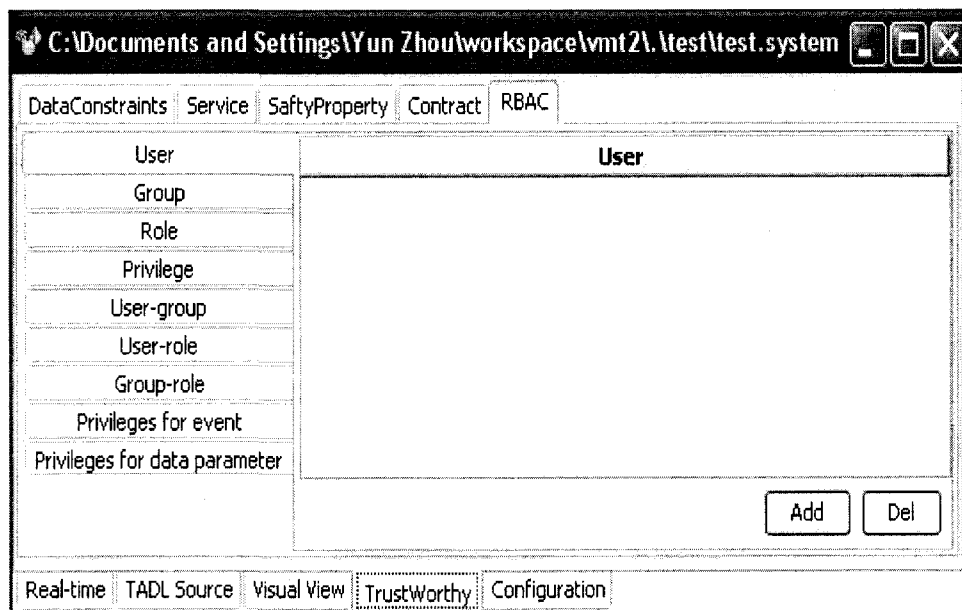


Figure 53: Snapshot for RBAC definition

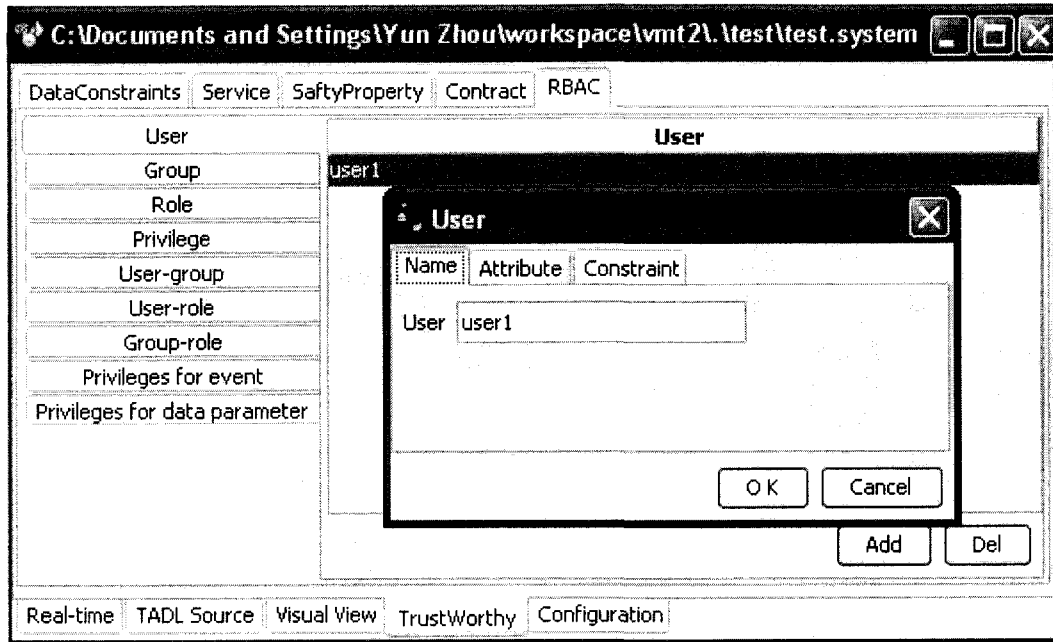


Figure 54: Snapshot of managing RBAC – user definition

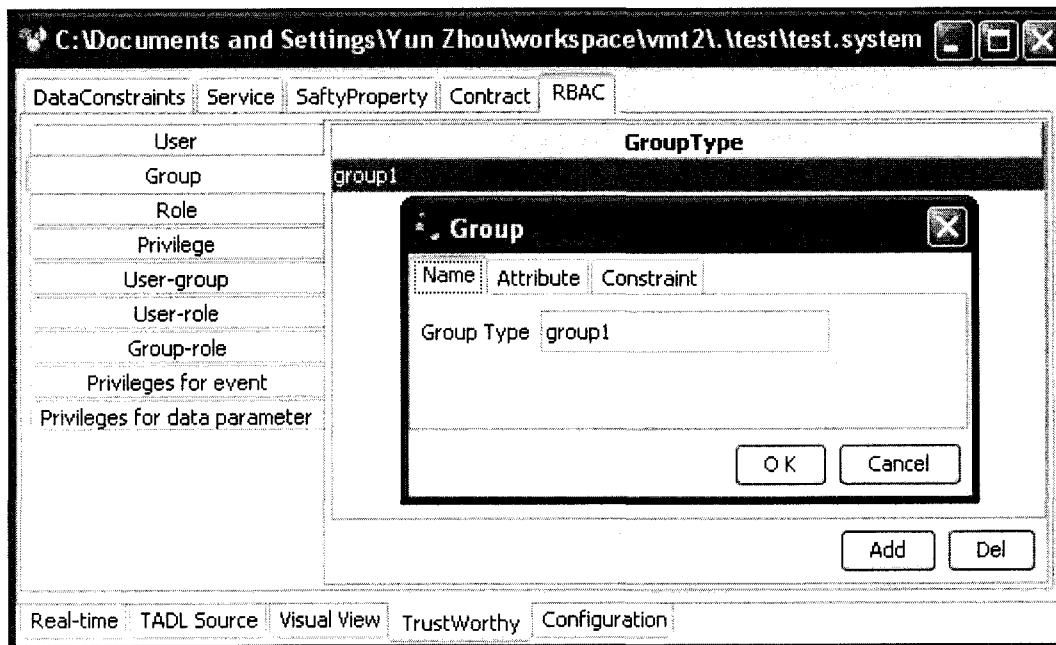


Figure 55: Snapshot of managing RBAC – group definition

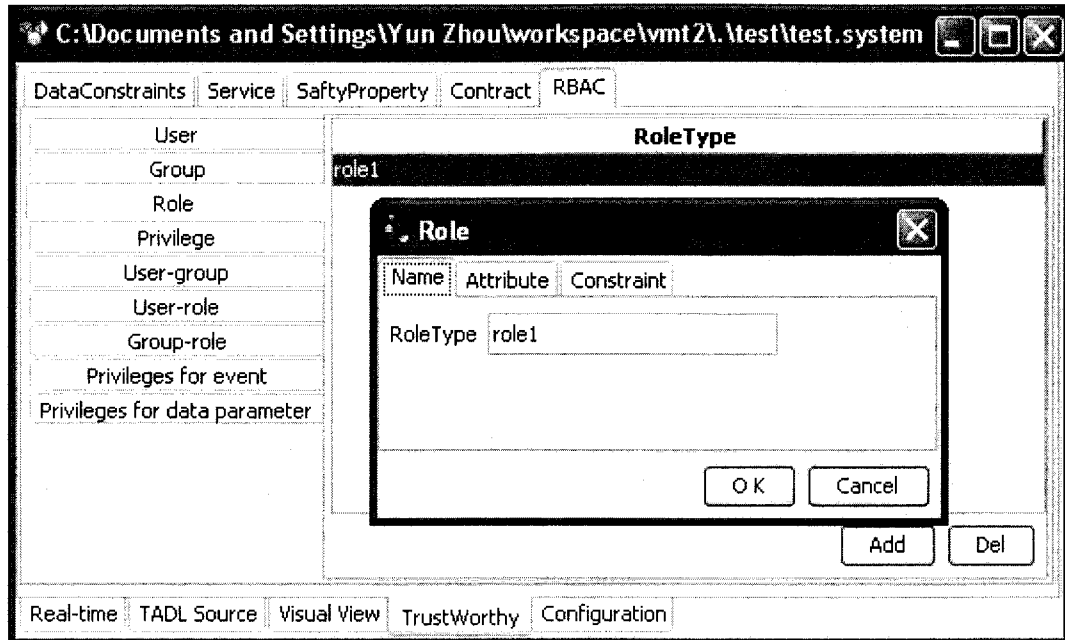


Figure 56: Snapshot of managing RBAC – role definition

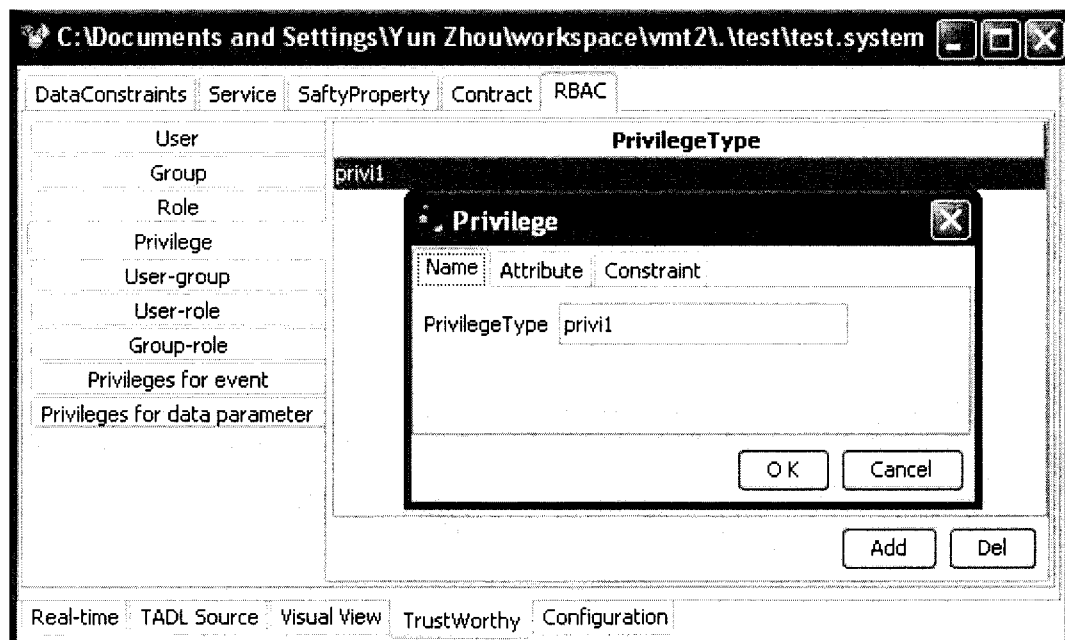


Figure 57: Snapshot of managing RBAC – privilege definition

to allow developers to fill in the properties. After a User/Group/Role/Privilege is created, its name will appear in the corresponding window.

- To modify a User/Group/Role/Privilege in RBAC: Double click on the name. A pop up window will show to allow developers to change the properties of a User/Group/Role/Privilege.
- To save a User/Group/Role/Privilege and its properties: Click 'OK' button in the pop-up window to save changes, and 'Cancel' button to undo changes.
- To delete a User/Group/Role/Privilege from RBAC: Select the name of User/Group/Role/Privilege, and press 'Del' button on the bottom right corner of the RBAC window. The name of the User/Group/Role/Privilege will be removed from the corresponding window.
- Figure 58 illustrates the interface to manage the relationship between a user and a group.
- Figure 59 illustrates the interface to manage the relationship between a user and a role.
- Figure 60 illustrates the interface to manage the relationship between a group and a role.
- Figure 61 illustrates the interface to manage the relationship between a privilege of service and a role.
- Figure 62 illustrates the interface to manage the relationship between a

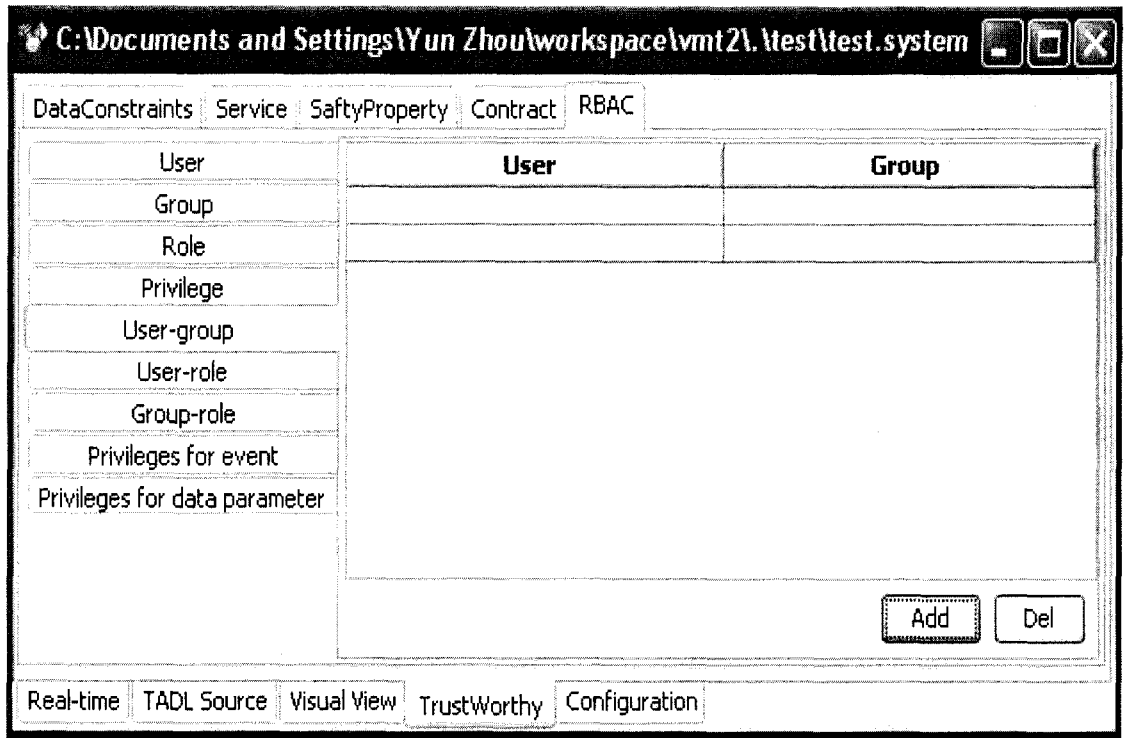


Figure 58: Snapshot for assigning a user to a group

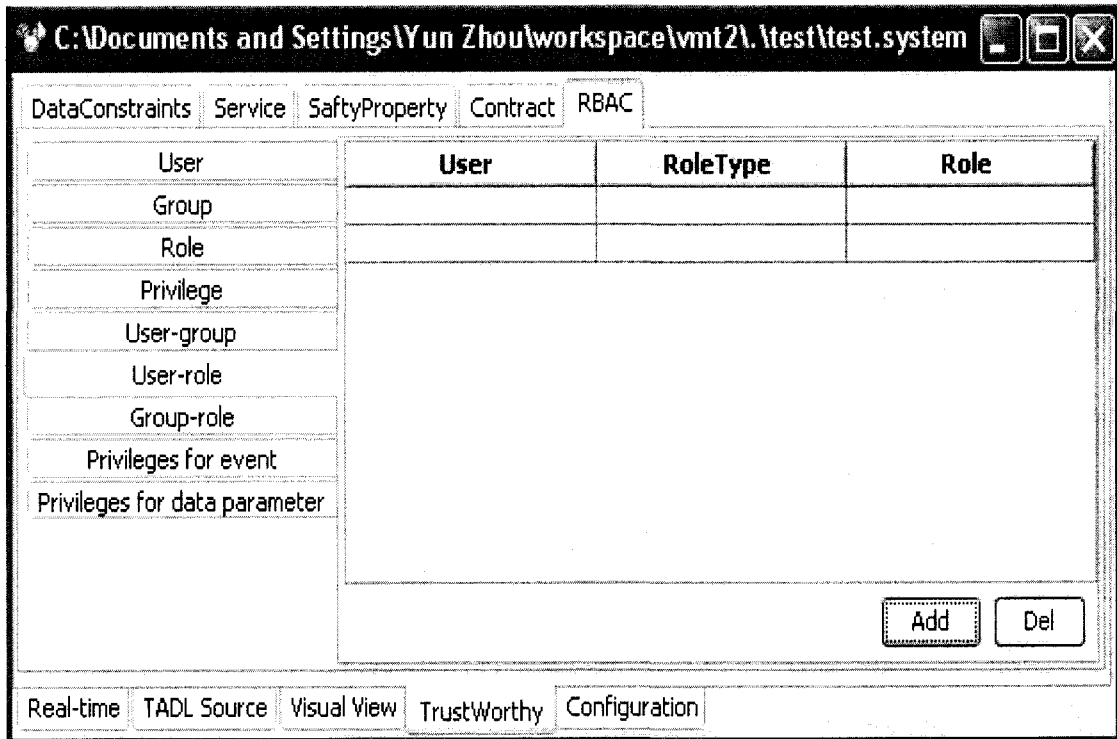


Figure 59: Snapshot for assigning a user to a role

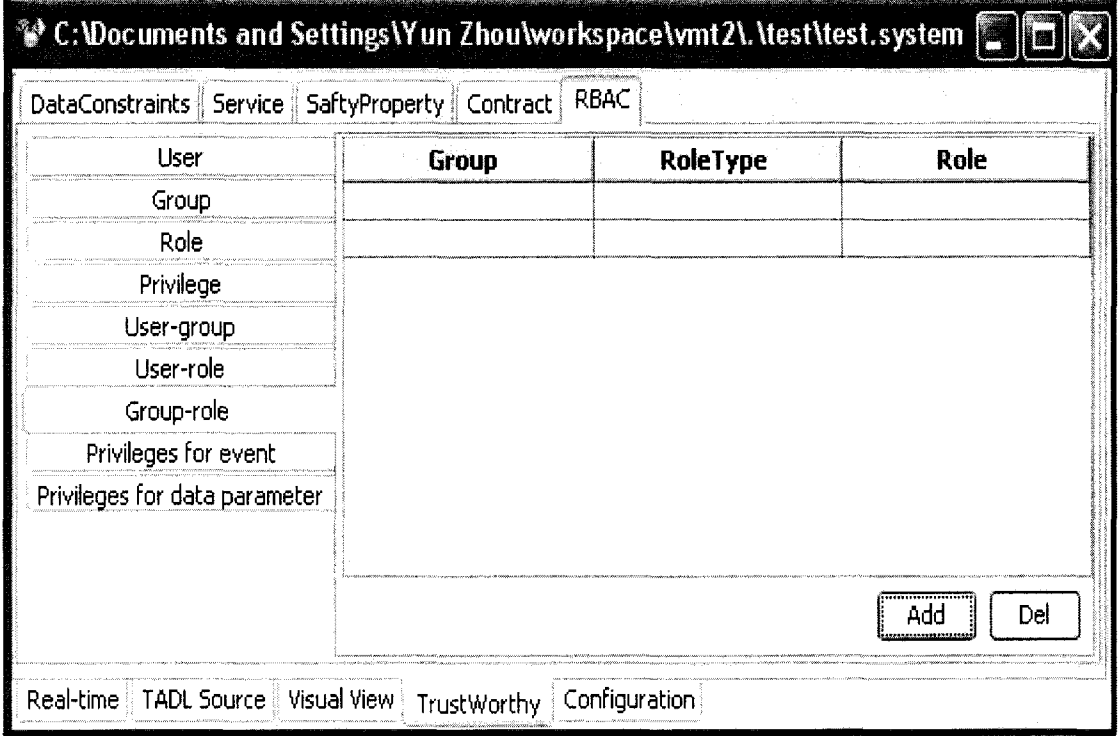


Figure 60: Snapshot for assigning a group to a role

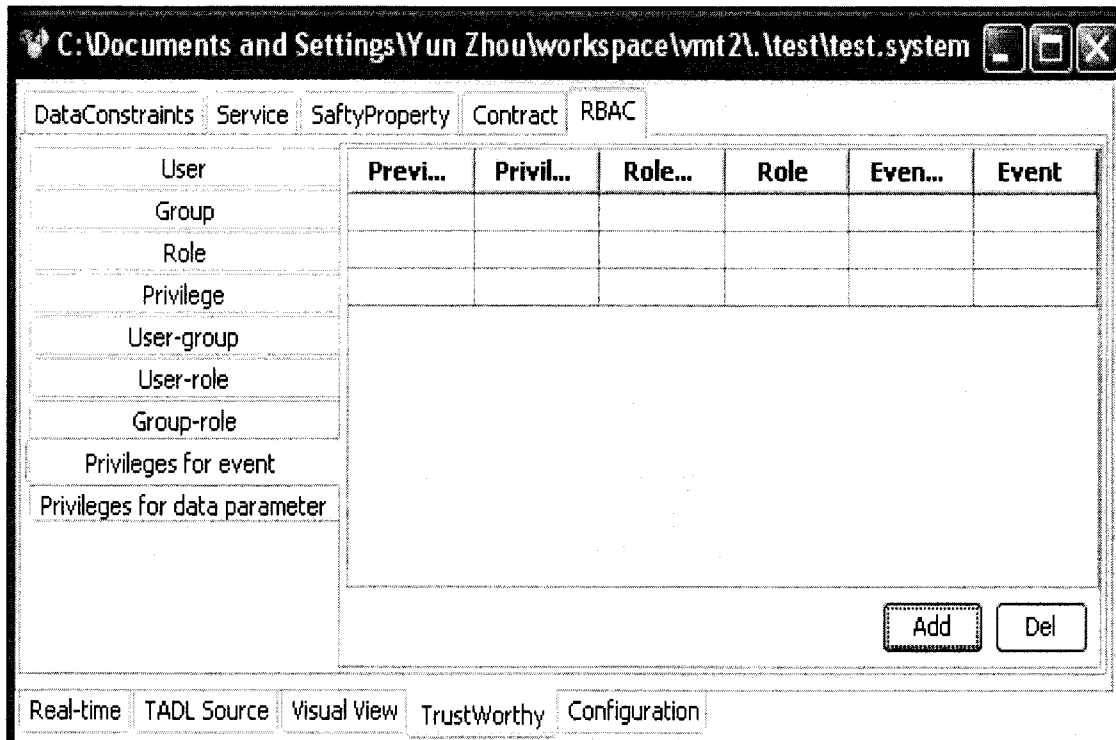


Figure 61: Snapshot for assigning privilege of events to a role

privilege of data parameter and a role.

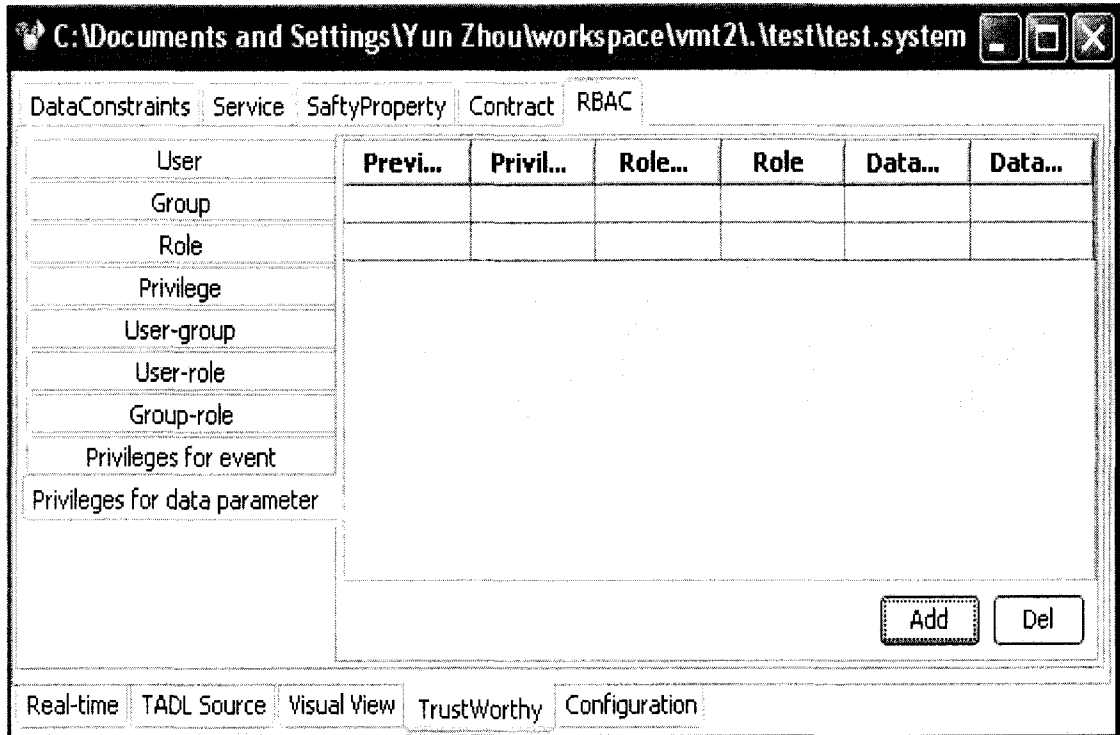


Figure 62: Snapshot for assigning privilege of data parameters to a role

6.4.4 Configuration View:

- *Use Case: Manage Hardware Component* Figure 63 shows the window to manage hardware component.
 1. To add a hardware component of a system: Press the 'Add' button on the bottom right corner of hardware component tab. A pop up window will show to allow developers to fill in the properties of a hardware component.
 2. To modify a hardware component: Double click on the name of the hardware

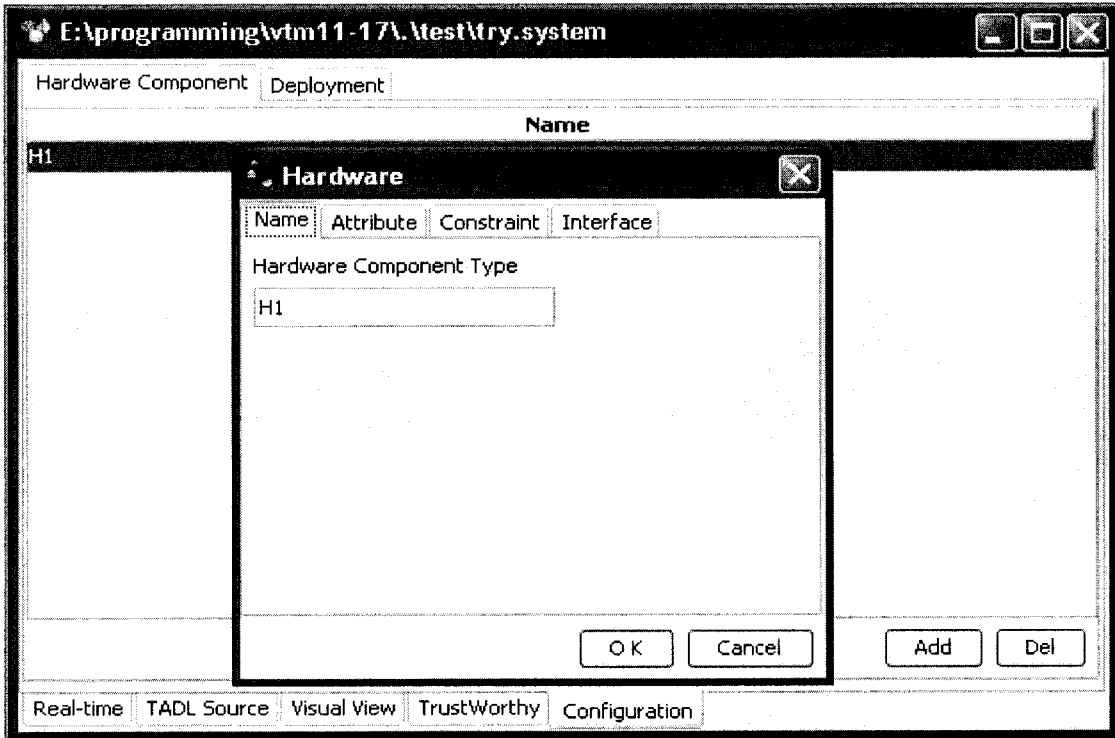


Figure 63: Snapshot for System Hardware Component definition

component. A pop up window will show to allow developers to change the properties.

3. To save a hardware component and its properties: Click 'OK' button in the pop-up window to save changes, and 'Cancel' button to undo changes. After the changes are saved, the name of the hardware component will appear in the hardware window.
 4. To delete a hardware component: Select the name of the hardware component, and press 'Del' button on the bottom right corner of the hardware component.
- *Use Case: Manage Deployment* Figure 64 shows the window to manage the deployment of a design.

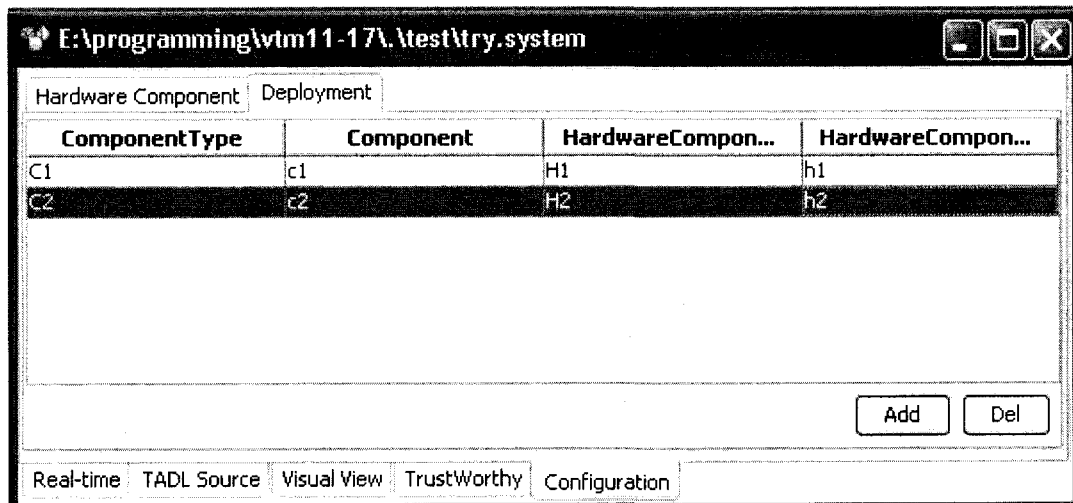


Figure 64: Snapshot for System Configuration definition

1. To add a deployment to a system: Press the 'Add' button on the bottom right corner of deployment tab. Fill in the properties in the new table.

2. To modify a deployment of a system: Double click the table cell to modify it.
3. To save a deployment: Press the Save button from Toolbar or select 'File–Save' from the Menubar.
4. To delete a deployment from a system: Select the row that is to be deleted, and then press 'Del' button on the bottom right corner of the deployment tab. The name of the deleted deployment will be removed from the deployment window

6.4.5 TADL Source View:

Figure 65 is the window to show the translated TADL Source of a design. It is updated dynamically when there is a change in the VMT. Whenever a save button is pressed, the external XML file is changed accordingly, and the TADL source interface is updated.

6.5 Properties Editor View – refer to Section 4.4.4

This section covers the usecases which are discussed in Section 4.4.4.

6.5.1 Component Properties Editor:

If a developer select a component in System Canvas, the Properties Editor window will be updated automatically to show the properties of the selected component.

- Usecase: Manage Component Properties. Figure 66, and Figure 67 respectively shows the window of managing the contract, and architecture properties of a component.

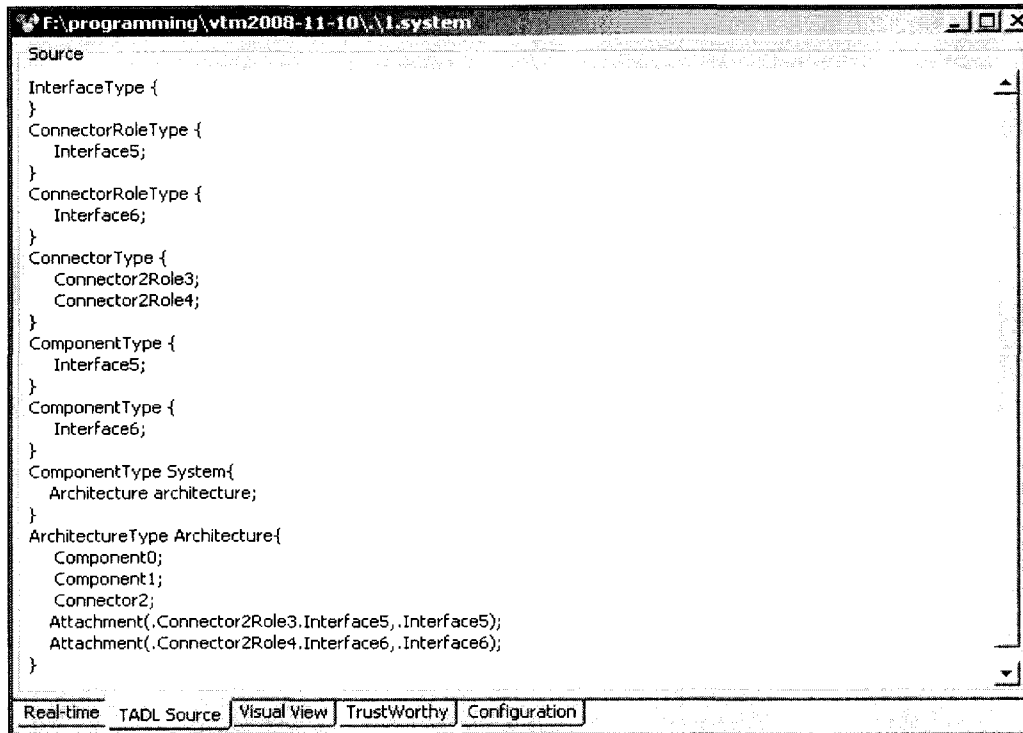


Figure 65: Snapshot for TADL Source Panel

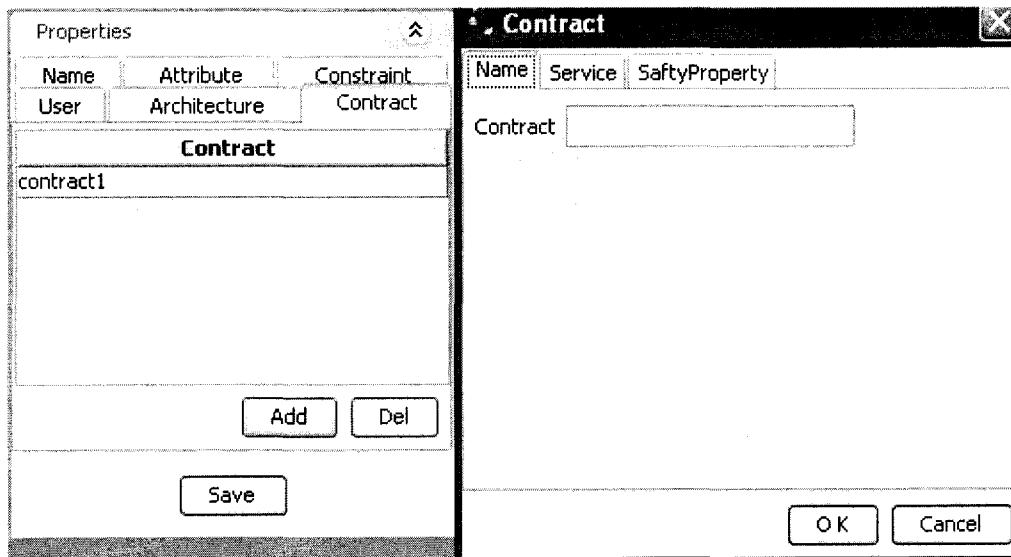


Figure 66: Snapshot for managing the properties of a component - contract definition

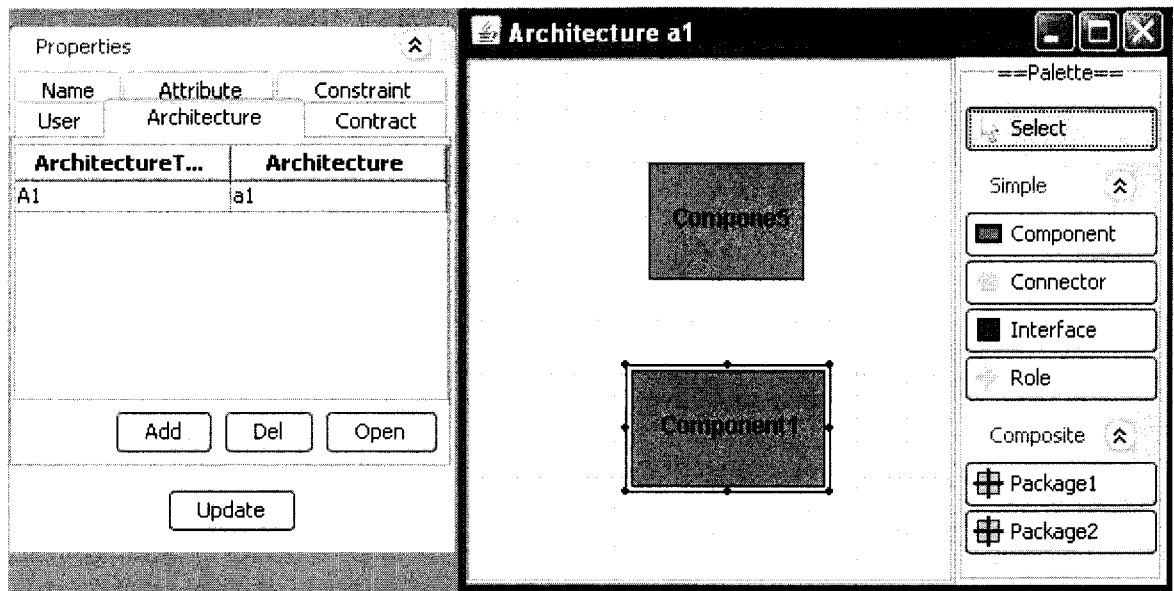


Figure 67: Snapshot for managing the properties of a component - architecture definition

- *Usecase: Manage Component Properties - Manage Contract* Figure 66, Figure 68, Figure 69 respectively shows the window to edit the name, service and safety property of a contract.
- *Usecase: Manage Contract - Manage Safety Property* Figure 70 shows the window to edit the name, attribute, event and constrains of a safety property.
- *Usecase: Manage Contract - Manage Service* Figure 71 shows the window to edit the name, attribute, request event, response event, data constraint, time constraint and update statements of a service.
- *Usecase: Manage Service - Manage Real-time Properties* Figure 72 shows the window to edit the properties of a time constraint, including the name, attribute, request event, response event and maximum safe time.

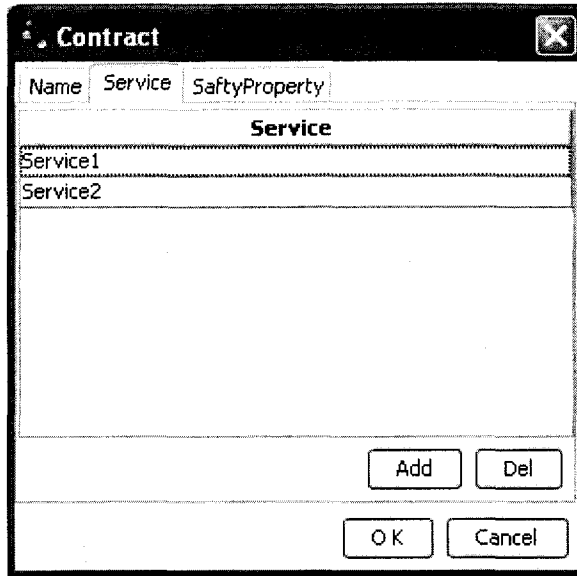


Figure 68: Snapshot for contract definition - service definition



Figure 69: Snapshot for contract definition - safety property definition



Figure 70: Snapshot for safety property definition

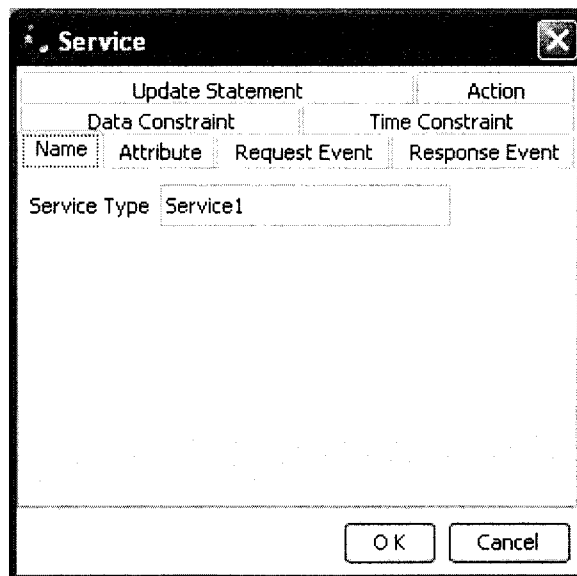


Figure 71: Snapshot for service definition

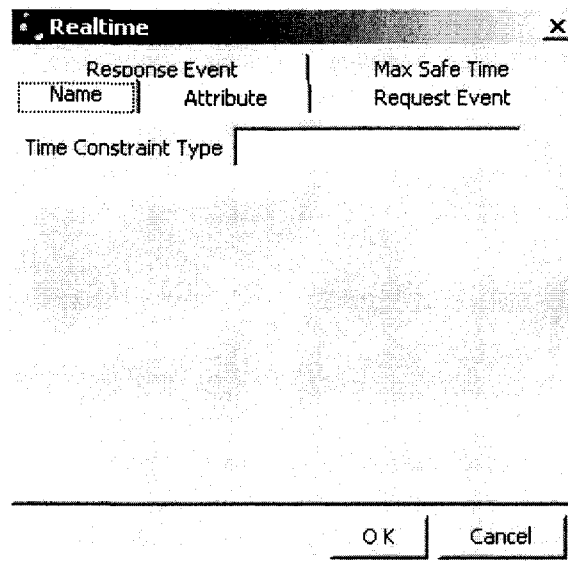


Figure 72: Snapshot for real-time definition

- *Usecase: Manage Service - Manage Data Constraints* Figure 73 shows the window to edit the data constraint properties, including the name, attribute, request event, response event and constraint properties.

6.5.2 Interface Attributes Editor:

If a developer selects an interface in System Canvas, the Attributes Editor window will update to show the properties of the selected interface.

- *Usecase: Manage Interface Properties* Figure 74 shows the window of managing the event attribute of an interface.

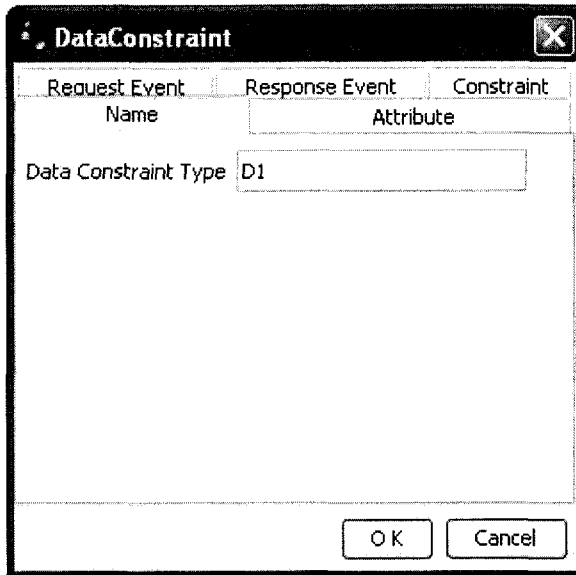


Figure 73: Snapshot for data constraint definition

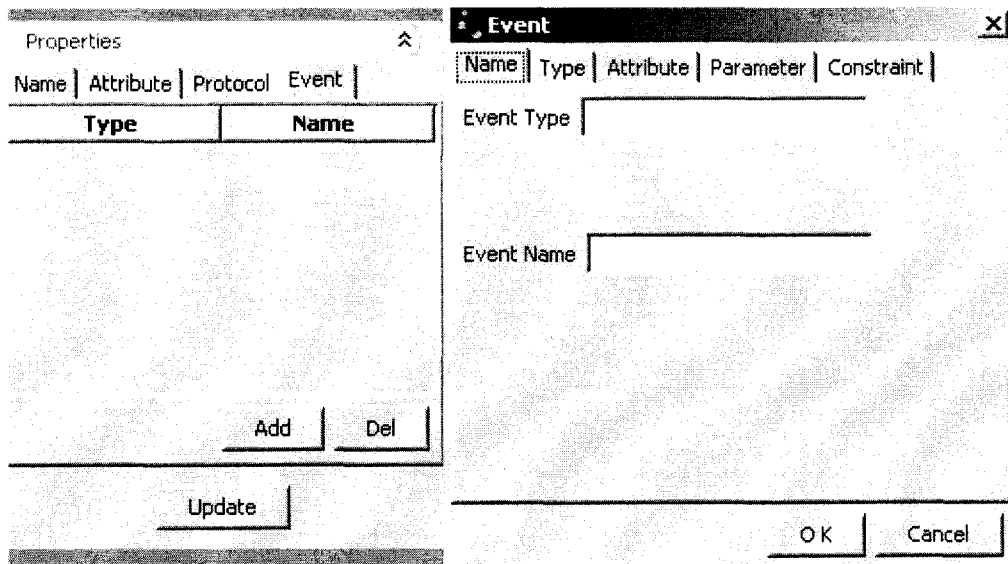
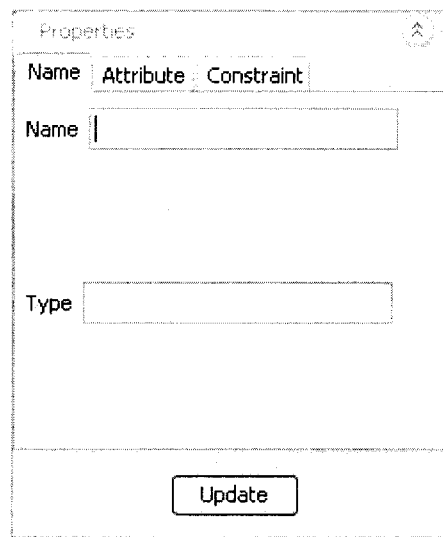


Figure 74: Snapshot for managing the properties of an interface - event definition

6.5.3 Connector/Connector Role Attributes Editor:

If a developer select a connector/connector role in System Canvas, the Attributes Editor window will update to show the properties of the selected connector/connector role.

- *Usecase: Manage Connector/Connector Role Properties* Figure 75 shows the window of managing the properties of a connector/connector role.



The image shows a dialog box titled "Properties" with a close button in the top right corner. Inside the dialog, there are two tabs: "Attribute" and "Constraint". The "Attribute" tab is selected. Below the tabs, there are two text input fields. The first is labeled "Name" and the second is labeled "Type". At the bottom center of the dialog is an "Update" button.

Figure 75: Snapshot for managing the properties of a connector/connector role - name definition

6.6 Error Message View:

Figure 76 lies in the right bottom of the VMT window to show errors of the current design.

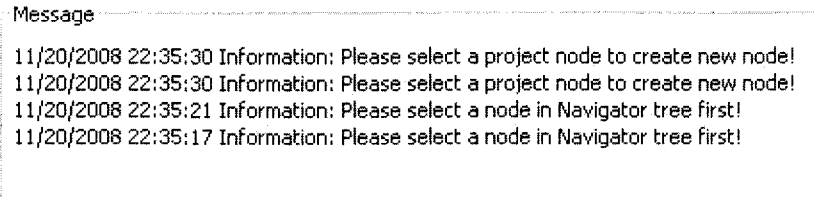


Figure 76: Snapshot for Error Message Interface

6.7 Toolbar:

Figure 77 is the window of toolbar. From left to right, the icons are: add, open, save, copy, paste, and delete.



Figure 77: Snapshot for Toolbar

6.8 Menubar:

Figure 78 is the window of menubar.



Figure 78: Snapshot for Menubar

Chapter 7

Conclusion and Future Work

The contribution of this thesis is the Visual Modeling Tool (VMT), the front-end for developers of trustworthy systems using the framework [put here reference to the paper]. The formalism of the component model is hidden behind the user-friendly interfaces offered by the VMT. The tool can be used for the entire life-cycle of the development including design, implementation, and deployment stages.

The tool provides seven essential interfaces.

- *Navigator interface* Using this interface a developer manages system evolution. New system or project can be added or existing project can be deleted.
- *Palette interface* This interface provides visuals for system elements.
- *System Editor interface* The editor provides the place and editing facilities where developers can visually construct a system. It is composed of five sub-interfaces, each projects one system view. These interfaces respectively are *System Canvas*, *Real-time View*, *Trustworthy View*, *System Configuration View*, and *TADL view*.

- *Properties Editor interface* Through this interface the attributes of a system element are managed.
- *Error Message interface* This message outputs the errors in the current design.
- *MenuBar* This is a general GUI component to provide services from drop-down menus.
- *ToolBar* This is another general GUI component to provide shortcuts of the frequently used methods in MenuBar.

The tool automatically translates the visual models in System Editor interface to TADL [Moh09], an architecture description language for trustworthy systems, and to an XML description. The TADL and XML descriptions are equivalent, because the XML schema is derived from TADL schema. XML schemas are developed by Ibrahim [Ibr08], and the compiler in VMT translates the visual models to the XML file. The translator will produce the XML file only if the visual model is syntactically and semantically correct with respect to their formal definitions. The XML file is used for formally analyzing the visual models. This is achieved by generating UPPAAL behavioral models, which are state machine descriptions, from XML description, and model checking them in UPPAAL [NI08]. Another purpose of translating the visual model to XML is that the translated XML can be exported to other platforms for verification.

The tool has been tested with two case studies, CoCoME and Mine drainage, discussed in [NI08]. For these case studies, the XML and TADL files produced by VMT are identical. A detailed description of the two case studies is not presented here, mainly because it is

purely an experimental work done following the step by step explanation given in Chapter 6.

7.0.1 Future Implementation Work

The design of VMT allows easy extensions to the tool. The two interesting extensions are (1) constructing a *package system element*, and (2) *building a data warehouse* for reusable artifacts. A package stores a number of related architectural elements. The package system element, when added, will help to identify related items in the architecture and help reuse. Each defined system element specification can be stored in a separate file in the repository, so that the developers can retrieve and reuse the defined trustworthy components from the repository. Adding the data warehouse with package system element will enhance the reuse at all stages of system development.

Bibliography

- [AB01] J.Laprie A.Avizienis and B.Randell. Fundamental concepts of dependability. Technical report, University of California, Los Angeles, LAAS-CNRS Toulouse, France, University of Newcastle upon Tyne, U.K., April 2001.
- [AM07a] Vasu Alagar and Mubarak Mohammad. A component model for trustworth real-time reactive systems development. *Theoretical Computer Science*, 2007. This paper is electronically published in *Electronic Notes in Theoretical Computer Science*. URL: www.elsevier.nl/locate/entcs.
- [AM07b] Vasu Alagar and Mubarak Mohammad. Specification and verification of trustworthy component-based real-time reactive systems. Cavtat near Dubrovnik, Croatia, September 3-4 2007. Six International Workshop on Specification and Verification of Component-Based Systems (SAVCBS 2007).
- [FA99] S.M.Bellovin F.B.Schneider and A.S.Inouye. Building trustworthy systems: Lessons from the ptn and internet. *IEEE Internet Computing*, November-December 1999.
- [Fow99] A. Fowler. A swing architecture overview: The inside story on jfc component design. URL: <http://java.sun.com/products/jfc/tsc/articles/architecture/>, 1999.
- [Gat02a] Bill Gates. *Trustworthy Computing*. Microsoft Corporation. URL: www.microsoft.com, January.15 2002.
- [Gat02b] Bill Gates. *Trustworthy Computing*. Microsoft Corporation. URL: www.microsoft.com, Oct. 2002.
- [GS06] D. Garlan and B. Schmerl. Architecture-driven modelling and analysis. In Tony Cant, editor, *Conferences in Resarch and Practice in Information Technology*,

volume 69. 11th Australian Workshop on Safety Related Programmable Systems (SCS'06), Melbourne, 2006.

- [Ibr08] Naseem Ibrahim. Transforming architectural descriptions of component-based systems for formal analysis. master thesis. Technical report, Concordia University, 2008.
- [IM02] I.Crnkovic and M.Larsson. *Building Reliable Component-Based Software Systems*. Artech House publisher, 2002.
- [JO06] J.M.Bruel and I. Ober. Components modeling in uml 2. *Studia Univ. Babeş-Bolyai, Informatica*, LI(1):79–90, 2006.
- [KP88] G. Krasner and S. Pope. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. *Journal of Object Oriented Programming*, 1:26–49, August/September 1988.
- [MA08] Mubarak Mohammad and Vasu Alagar. A framework for the component-based development of trustworthy systems. 2008.
- [MOB00] D. Coleman M. Ogush and D. Beringer. *A template for documenting software and firmware architectures*. Hewlett Packard, 1.3 edition, March 2000.
- [Moh09] Mubarak Sami Mohammad. *A Formal Component-Based Software Engineering Approach for Developing Trustworthy Systems*. Phd thesis, Concordia University, Montreal, Canada, 2009.
- [MT00] Nenad Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1), Jan. 2000.
- [NI08] V.S. Alagar N. Ibrahim, M. Mohammad. Tadi case studies. Technical Report ACT-Trust-08-04, Concordia University, November 2008.
- [OMG06] OMG. Unified modeling language: Infrastructure and superstructure, version 2.0. Technical Report OMG document formal/05-07-05 and formal/05-07-04, Object Management Group, March 2006.

- [Rec05] W3C Recommendation. Xml schema. <http://www.w3.org/2001/XMLSchema>, September 2005.
- [Sou07] SourceForge.net. tableview. URL: <http://sourceforge.net/projects/tableview/>, June 2007.
- [Sou08] SourceForge.net. dom4j. URL: <http://www.dom4j.org/>, October 2008.
- [Xst08] Xstream. Xstream. URL: <http://xstream.codehaus.org/index.html>, February 2008.