A Visual Meta-Language for Generic Modeling

Hakan Canli

A VISUAL META-LANGUAGE FOR GENERIC MODELING

THESIS

Hakan Canli, 1st Lieutenant, TUAF

AFIT/GCE/ENG/02M-1

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF

TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

Report Documentation Page

Report Date	Report Type	Dates Covered (from... to)
10 Mar 02	Final	Mar 2001 - Mar 2002

Title and Subtitle A Visual Meta-Language for Generic Modeling	**Contract Number**
	Grant Number
	Program Element Number
Author(s) 1st Lt Hakan Canli, TUAF	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Bldg 640 WPAFB OH 45433-7765	**Performing Organization Report Number** AFIT/GCE/ENG/02M-1
Sponsoring/Monitoring Agency Name(s) and Address(es) AFRL/SNZW ATTN: Mike Foster Bldg 630 S1D34, 2241 Avionics Circle WPAFB OH 45433-7303	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)

Distribution/Availability Statement
Approved for public release, distribution unlimited

Supplementary Notes
The original document contains color images.

Abstract
This research examines the usefulness of a visual meta-language (VLGM Visual Language for Generic Modeling) developed for the specification of components and relations in a modeling domain. The language is designed to allow software tools to interpret specifications and automatically provide modeling environments. VLGM makes use of the object-orientated software engineering methodology. It defines four types of special classes and three types of relations between them. Data types and primitive types are allocated with several attributes to provide restrictions and enable consistency checks over models. As part of this research a software tool was designed. The tool provides a workspace for creating VLGM specifications. It interprets VLGM designs and provides a generic modeling environment. An XML document format is used as a persistence mechanism to promote reusability and sharing. Four case studies from different modeling domains are used to explore the applicability of the idea.

Subject Terms VLGM, Visual Languages, Modeling, Object-Orientation, Simulation, UML, XML Document	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 186	

AFIT/GCE/ENG/02M-1

A VISUAL META-LANGUAGE FOR GENERIC MODELING

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Hakan Canli, B.S.

1st Lieutenant, TUAF

March 2002

AFIT/GCE/ENG/02M-1

A VISUAL META-LANGUAGE FOR GENERIC MODELING

Hakan Canli, B.S.
1st Lieutenant, TUAF

Approved:

[illegible] 6 MAR 02
Karl S. Mathias, Major, USAF date
Chairman

[illegible] 8 MAR 02
Timothy M. Jacobs, Lieutenant Colonel, USAF date
Member

Raymond R. Hill 6 Mar 02
Raymond R. Hill, Lieutenant Colonel, USAF date
Member

Acknowledgments

"I find that the harder I work, the more luck I seem to have."

Thomas Jefferson (1743-1826)

I owe much appreciation to those who have contributed to the achievement of my work at AFIT. I thank Major Karl Mathias for his expert guidance, understanding, amazing enthusiasm and curiosity about “How are the boxes doing?” Without his support, my academic journey at AFIT would not have been as regarding. I also thank my committee members, Lieutenant Colonel Tim Jacobs and Lieutenant Colonel Raymond Hill for their constructive comments. Special thanks to Dr. Henry Potoczny for reminding me of the joy of learning with his open mind and unique sense of humor.

Above all, my wife deserves the greatest praise for her patience and selflessness during her pregnancy while we worked through the AFIT experience. Her dedication and love made our son and my career possible. I dedicate this work to my wife and my son, whose little smile always managed to remind me what the important things are in life.

Table Of Contents

List Of Figures

List Of Tables

AFIT/GCE/ENG/02M-01

Abstract

This research examines the usefulness of a visual meta-language (VLGM – Visual Language for Generic Modeling) developed for the specification of components and relations in a modeling domain. The language is designed to allow software tools to interpret specifications and automatically provide modeling environments.

VLGM makes use of the object-orientated software engineering methodology. It defines four types of special classes and three types of relations between them. Data types and primitive types are allocated with several attributes to provide restrictions and enable consistency checks over models.

As part of this research a software tool was designed. The tool provides a workspace for creating VLGM specifications. It interprets VLGM designs and provides a generic modeling environment. An XML document format is used as a persistence mechanism to promote reusability and sharing. Four case studies from different modeling domains are used to explore the applicability of the idea.

A VISUAL META-LANGUAGE FOR GENERIC MODELING

1. INTRODUCTION

Simulation systems serve decision-makers as support mechanisms by testing the anticipated behavior and performance of real-life entities, concepts, or systems on models developed in suitable controlled mediums, and lead to reliable and more complete risk management. It is a process of capturing the state and dynamics of the system into a model, and deriving results and ideas by means of executing operations on the model.

Simulation is widely used for industrial, scientific, economic, educational, and military purposes since it provides cost-effective ways to test, train, and design. As the costs of these activities get higher and more companies count on simulation results, more is expected from the process. These expectations focus on how accurately and how quickly it can provide results.

The modeling aspect of the simulation process requires a series of communications between the analyst and computer. The way this communication occurs heavily depends on the simulation tool used and can be more efficient with the use of visual languages instead of textual ones. Visual languages not only provide parallel interpretation of model elements and their dependencies, but also grouping mechanisms to ease management of large-scale projects. On the other hand, it’s difficult to trace, search and debug through the model elements and handle large-scale projects with a textual language. If designed properly, visual languages can also be used to communicate models between analysts. To make use of these advantages, this research effort focuses on the usage of visual languages to support modeling and communication of simulation entities.

1.1. Background

Modeling is the process of capturing the important aspects of real world entities. Once a model is built, computations can be derived. For instance, “shape” is the property captured in a proportionally-scaled small-wing model to test and explore its aerodynamic properties in a wind tunnel, which provides an easier, cheaper and safer environment than building and testing a full size aircraft.

The models used in computer simulations are data structures defined in the simulation language used. Scenario design patterns for simulations depend on the simulation tool used. Some simulation tools may require structured text definitions, others may help the user to select basic pre-built structures and set their initial properties in a graphical environment. In both cases, the data structures are interpreted by the simulation tool prior to execution.

Considering today’s computational capabilities of computers, many simulations run in a reasonable time. However, this is not the case for the modeling required to prepare scenarios for them. In most cases, the bottleneck of the simulation process occurs in modeling since it requires detailed preparation of the data structures representing real world entities and their behaviors.

The solution to similar problems in software engineering has been to use visual modeling languages. The Unified Modeling Language (UML) has been proposed as a standard modeling language for object-oriented systems. UML is based on a unification of different object-oriented software development approaches developed in the last decade. It most directly unifies the methods of Booch, Rumbaugh and Jacobson [FOW99]. Although UML provides a unified modeling method for object-oriented systems, it is designed to support software engineering tasks and not suitable to meet specific requirements of simulation modeling. Reasons supporting this assertion are presented later in Chapter Three.

Object-orientation is an approach to problem-solving which seeks to identify the relevant objects in the problem domain. These objects are then defined and employed to solve the problem. It is a relatively recent paradigm, which provides a closer match to real-world entities and provides modularity by decomposing the problem into components called objects. Object-orientation makes reuse much more attainable and can greatly reduce complexity by reducing coupling. The object-oriented paradigm is becoming widely used in simulation tools to make use of its advantages.

1.2. Problem Description

Software vendors offer various kinds of tools providing environments to develop models and run simulations for different domains. These environments may be textual, visual, or mixed. Although textual languages provide more flexibility to the designer, it may be very challenging to textually define and keep track of entities and their relations for complex domains when compared to the simplicity of schematic ones.

The DoD employs simulation systems to test the interactions between weapons and the tactics to deploy them. These simulation tools include, but are not limited to, the Extended Air Defense Simulation (EADSIM), Suppressor Composite Simulation System, Joint Interim Mission Model (JIMM), and Simulated Warfare Environment Generator (SWEG).

In order to provide reuse of components and interoperability between DoD's simulation systems, there has been considerable research within AFIT. Each of these simulation systems has its own specific textual description language and modeling techniques, with some overlap between them. The nature of these languages requires the use of text editors, which lead to substantial user interaction, interpretation, and time-consuming manual generation of scenario files. Because of the unacceptable development times and costs, the Sensors Directorate of the Air Force Research Laboratory (AFRL) requires modeling tools that accelerate the creation and manipulation of objects and relations in a graphical user interface environment.

1.3. Research Focus

This section discusses the objectives, scope and approach of the thesis.

1.3.1. Objectives

Simulation tools with textual languages have several usability problems. First of all, the designer has to learn a textual language and understand the underlying architecture of the simulation tool like a software engineer. This results in a steep learning curve. Large-scale simulation scenarios sometimes require thousands of lines of textual definitions. This makes it difficult to achieve some design tasks such as managing dependencies between simulation entities, tracing, and debugging. Reusability of the scenario elements depends on manual copy and paste methods in text editors. These problems lead to a time-consuming design process, which may mean higher cost or mission failure.

Because of these problems, the simulation community requires graphical user interface environments that are designed to meet the cognitive requirements and the tasks of scenario development. This research asserts that a common object-oriented modeling pattern can be the basis for a generic user interface that will provide a graphical design environment for modeling, including simulation tools with textual languages. Modeling process is in fact an instantiation of component and relation types into a workspace and parameterization of their attributes. A visual meta-language may be designed to specify component and relation types of a modeling domain. If the language has the “transformability” property a software tool may interpret these specifications and automatically provide the design environment. This approach will not only solve the problems of using simulation tools with textual languages but also provide a generic modeling tool for any modeling environment. The objective of this research is to examine the applicability of this approach.

1.3.2. Approach

The visual meta-language, if designed properly, can be represented in a software tool and drawn on a piece of paper or blackboard. This will enable designers to communicate and discover ideas easily. It will help the designer as an external aid, which will increase human processing memory by reducing search and enhancing the detection of design patterns. It will simplify the model into a diagram, which would take pages to define textually.

This research effort focuses on developing a visual, structured, and modeling domain-independent notation. Using the visual meta-language, Visual Meta-Language for Generic Modeling (VLGM) specified in this thesis, an analyst can define the visual modeling environment for a specific modeling domain, including simulations. VLGM serves as a framework for modeling tools and is a basis for a common visual language among the modeling community. The intention is to use the VLGM design to create a domain-specific modeling environment. Once the domain-specific modeling environment is provided, an analyst can design models. For simulation systems, these models can be parsed into the scenario files to be executed in the simulation tool.

This research asserts that this approach addresses the usability problems of textual simulation languages and benefits the overall simulation design effort by reducing the time and cost for design and user training. It also provides a generic modeling environment for any kind of modeling-domain and a communication platform between analysts from different disciplines

1.3.3. Scope

Since three-dimensional or colorful structures and animations are difficult to present on a paper or a board, the language does not include complex visual structures. VLGM is generic enough to support any type of modeling domain. All types of simulations including continuous, discrete-event, deterministic, and stochastic simulations are supported. However, real-time simulations like flight simulators cannot be supported since they are too specific and have visualization, interaction, and performance issues that require a different engineering process.

1.4. Structure of the Thesis

This document is composed of seven sections. In Chapter Two, a summary of the current literature is presented. The disciplines involved include simulation, visualization, cognition, modeling, communication, and language theory. In Chapter Three, the thesis discusses the methodology used by means of a detailed examination of the Unified Modeling Language (UML) including its drawbacks and how the proposed language addresses these issues in the domain of modeling. In Chapter Four, the syntax and semantics of the proposed notation are introduced. Chapter Five explains the research software tool that demonstrates the model development environment. Chapter Six is composed of several case studies each representing problems from different modeling domains. Chapter Seven summarizes the study and presents suggested future research directions in this area.

2. LITERATURE REVIEW

The proposed approach to the problem of simulation scenario modeling requires integration of ideas from various disciplines including simulation, visualization, modeling, language theory, and software engineering. This chapter presents previous studies from these areas.

2.1. Simulation

Simulation is a discipline for developing a level of understanding of the interaction of the parts of a system, and of the system as a whole. As described in Webster's Dictionary, it is "imitation or enactment, as of something anticipated or in testing" or "the representation of the behavior and characteristics of one system through the use of another system, especially a computer program designed for the purpose" [WEB96]. System, in this context, means an entity which maintains its existence through the mutual interaction of its parts. A system exists and operates in time and space. In many respects, simulation is a daily-life experience for humans. As humans think on anything, unintentionally they develop a mental model and run a few different scenarios on that model in their mind. Similarly, computer simulations are achieved by modeling the behavior of a system and running tests on that model.

2.1.1. The Simulation Process

The simulation process involves understanding and modeling a system by defining its attributes and behaviors, validating the models, and performing statistical analysis of its inputs and outputs. Avni Tayfun defines "model" as a manifestation of reality in a controlled environment [TAY99]. A model possesses the prominent characteristics of the object, concept, or system it represents in some detail. The simulation modeling process requires a combination of art and science. To quote Tayfun, “Just like an artist, the simulation analyst develops skills to observe and translate events, ideas, and attributes of pertinent surroundings into a model” [TAY99].

Jerry Banks and Randall R. Gibson identify two skills that are required to be successful at simulation: The ability to understand a complex system and its relationships, and the ability to translate this understanding into an appropriate logical representation recognized by simulation software [BAN97]. Banks and Gibson propose twelve guidelines for industrial engineers who are getting started in simulation modeling as summarized below [BAN96].

1. **Define the problem:** Like other computer applications, a simulation model can only do what it was designed to do – and it is impractical to design it to do everything.
2. **Understand the system:** Be familiar with the procedures of the real system.
3. **Determine your goals and objectives:** Write down the goals and objectives. Based on the objectives, decide the resolution level of the model.
4. **Learn the basics:** Try to obtain training for the simulation.
5. **Confirm that simulation is the right tool:** Other cheaper analytical solutions may be possible.
6. **Attain support from management:** If the results will not be used, the project effort is irrelevant to the system.
7. **Learn about software tools for simulation:** Determine the correct tool.
8. **Determine what data is needed and what’s available:** Assumptions might be required for unavailable data.
9. **Develop assumptions about the problem:** Assumptions are required to optimize the simulation scenario and to simplify the model.

10. **Determine the outputs needed to solve the stated problem:** Define measurements, including how to collect and use them.
11. **Simulation conducted internally or externally:** Decide to use or not to use a consultant from outside of the company.
12. **Kick off the project:** Plan a meeting to present the results of the simulation.

Based on the survey by Tayfun the benefits of simulation modeling are managing change, minimizing risk, promoting creativity, enhancing communication, and providing accelerated testing and quantitative solutions. Simulations also avoid disturbance of the real system [TAY99].

Another study by Banks and Gibson presents the “evolving” characteristics of the simulation model. They concluded that the demand to use the same model that had been built early, in later project stages is an unrealistic expectation, since industrial systems and engineering projects evolve. Some of these changes include equipment, location, operating details, layout, control rules, operating procedures, material arrival profiles, material quantities, material sizes, order mix, order size, order profiles, operating assumptions, operating hours, staff shifts, breaks, labor work standards and practices. Any change to the actual system may have a significant effect on system operation, throughput, and other parameters. In order to provide accurate, useful results, a simulation model must evolve to keep up with the simulated system as it changes [BAN98].

Banks and Gibson also surveyed the danger of the assumption that complex simulations can be carried out using software without some degree of programming. Non-programming simulation software systems are based on pre-built constructs for typical activities and are often too generic. If the level of simulation software interface is scaled from total programming through non-programming, as the level of simulation software interface increases, the less flexibility is given to the analyst. In software with high-level user interfaces, most modeling decisions are made by the developers of the software, not by the user. Simulation software should ideally allow users to operate at different levels and to change smoothly between them. Otherwise, the analyst may produce overly simplified and invalid models. Non-programming software packages hide critical details that the analyst needs to see in order to understand and verify the model's behavior and results. Although this kind of software provides ease of use, it may endanger the fidelity of the model [BAN97].

2.1.2. Multi-domain Simulation

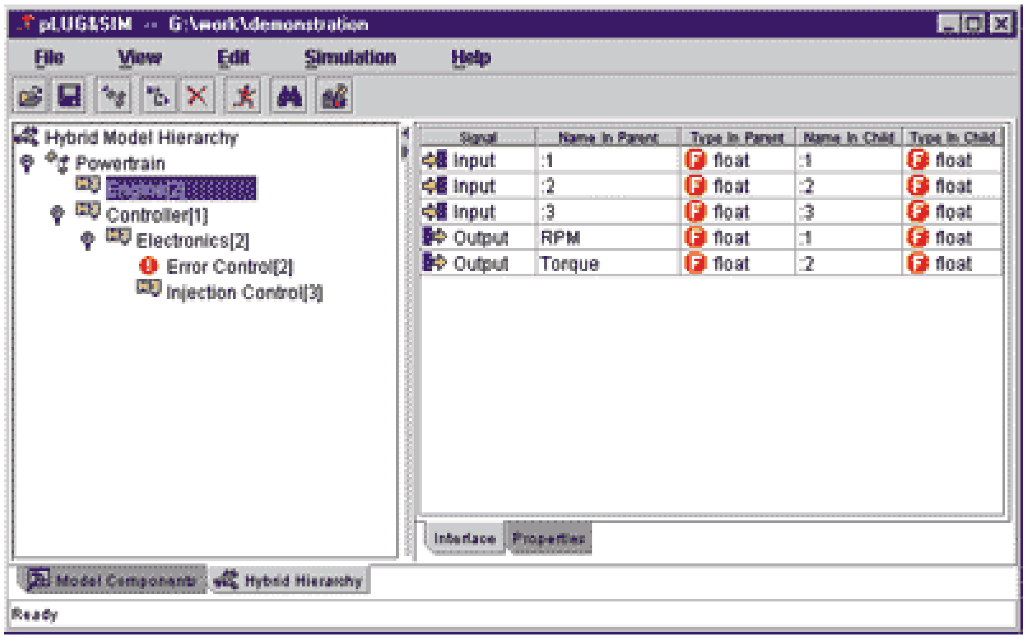
Andrzej Bargiela categorizes the strategic directions in simulation and modeling into three categories [BAR00]:

1. **Abstraction:** A scientific tool for coping with the complexity of the systems.
2. **Uncertainty processing:** Human-induced uncertainty effects on systems and modeling methodologies.
3. **Simulation Paradigms and Architectures:** Large-scale adaptive systems with agent-based modeling and simulation paradigms, distributed and global simulation paradigms, and effective visualization and interaction techniques.

Bargiela emphasizes the importance of standards in distributed communication objects (e.g., CORBA, Java), distributed simulation (e.g., HLA/RTI), and distributed collaborative modeling (e.g., DEVS/CDM). These standards offer the potential for simulations to be constructed by interconnecting various models. These improvements and the widespread availability of digital communication lines and the Internet provide the technical opportunity to develop large-scale distributed simulations [BAR00].

Philip Clarke states that a variety of simulation software have recently been created that allows the user to design in an environment closer to the application domain and removes the low-level details of the implementation. These commercial simulations offer simulation to non-programmers. However, this type of software provides domain-specific solutions. Modern systems have many facets. This implies the requirement to integrate these different software packages [CLA99].

Clarke introduced the generic co-simulation tool called pLUG&SIM. Developed by simulation tools company, Integrated Systems Inc. (ISI) [CLA99], this tool provides an environment to build interfaces between models of different simulation software. Once the interfaces have been built between models, different simulation software can run concurrently and simulate a heterogeneous system by sharing data among each other. This solution also provides the flexibility to execute different software packages on different machines and benefit from the advantages that distributed software systems provide. The tool is based on the Common Object Request Broker Architecture (CORBA), which enables users to work in a distributed computing environment. Figure 1 is a screenshot of the pLUG&SIM user interface [WIN98].



Signal	Name In Parent	Type In Parent	Name In Child	Type In Child
Input	:1	float	:1	float
Input	:2	float	:2	float
Input	:3	float	:3	float
Output	RPM	float	:1	float
Output	Torque	float	:2	float

Figure 1. pLUG&SIM Interface

Another study that addresses multi-domain simulation requirements is presented by Ali Goucem. Goucem introduced the language design group created within the ESPIRIT "SiE-WG" project in September 1996. Twelve months later, the group published the specification of a system modeling language called "Modelica" [GOU99].

Modelica is an object-oriented language of large, complex, and heterogeneous physical systems. Models in Modelica are described by differential, algebraic, and discrete equations. Modelica is a textual language, and the tools supporting Modelica provide a graphical modeling environment. Currently, the Dymola software package by Dynasim, a Swedish company, supports the use of Modelica language [MOD00]. Modelica language is important because of its resemblance to this research. The comparison of Modelica and this research is presented in Table 1.

Table 1. Modelica Vs. VLGM

Aspect	Modelica	VLGM
Structures	Special types of classes and primitive types	Special types of classes and primitive types
Representation	Textual	Visual
Behavior	Described by differential, algebraic and discrete equations. This limits the language to physical systems.	None. Application domain is larger but lacks behavioral description.
Relations	Composition, inheritance	Composition, inheritance, port type selection
Visual specifications	None	Able to specify visual properties of components relations and ports
Constraints	None	Able to specify constraints over relation cardinalities and value ranges

2.2. Modeling and Object Orientation

This section focuses on general concepts of modeling and the object-oriented paradigm.

2.2.1. Modeling

James Rumbaugh, Ivar Jacobson and Grady Booch define a model as a representation of something in the same or another medium. Models capture important aspects of a system, use a medium that is convenient for working, and can be used for engineering calculations [RUM99]. Models are intended to be easier to design and use than the final system. Rumbaugh, Jacobson and Booch list the uses of models as follows [RUM99]:

- To capture and state domain knowledge so that all stakeholders may understand and agree on them.
- To think about the design of a system.
- To capture design decisions.
- To generate usable products.
- To organize, find, filter, retrieve, examine and edit information about large systems.
- To explore multiple solutions economically.
- To master complex systems.

Models are composed of semantics, presentation, and context. Textual or visual notation determines how to represent a system, semantics are the meaning of notational expressions, and context is the internal decomposition of the system represented. The amount of detail in a model is the analyst's choice and should be based on one of the following purposes [RUM99]:

- As a guide to the thought process (Hierarchical top-down decomposition).

- Abstract specification of the essential structure of a system (Intended to be evolved later).
- Full specification of a final system (Enough information to build the system).
- Examples of typical or possible systems (Provides comparison between options).
- Complete or partial descriptions of systems.

A modeling notation is also called a “modeling language” since models provide a communication mechanism. Modeling languages can be textual or visual. Kim Marriot and Bernd Meyer describe visual languages as some set of diagrams, which are a collection of symbols in a two or three-dimensional space [MAR98A]. Visual modeling languages have a wide-range of application areas such as circuit design, software engineering, aviation charts, maps, and sign language.

2.2.2. The Object-Oriented Paradigm

Object-orientation is an approach to problem solving which seeks to identity the relevant objects in the problem domain. These objects are then defined and employed to solve the problem. James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy and William Lorensen defined the term “object-oriented” as follows:

> Superficially the term "object-oriented,” means that we organize software as a collection of discrete objects that incorporate both data structure and behavior. This is in contrast to conventional programming in which data structures and behavior are only loosely connected. [RUM91]

An “object” is the most fundamental concept in the object-oriented paradigm. It is a conceptual (logical or physical) entity composed of attributes and methods. Attributes hold the data that determine the state of the object, and methods determine the behavior of the object based on its current state. An object is normally referred to by a name and has an "identity." Attribute values of an object might change in time, perhaps as a result of performing a behavior, but it would still be the same object [STE99]. A UML representation of object is shown in Figure 2 [MUL97].

A "class" is an abstract representation for some particular type of object. Often described as a blueprint for an object, it defines objects of that type. Objects are built from the class by a process named "instantiation." As a result, any object is an instance of a class. Figure 3 shows the UML representation of class. Different types of relationships are applicable between classes. An "association" relationship is a semantic connection shown with a line between classes or objects as in Figure 4 [MUL97].

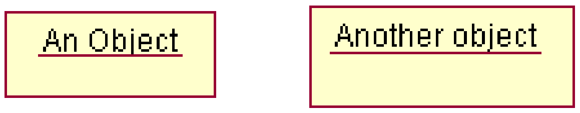


Figure 2. UML Object Representation

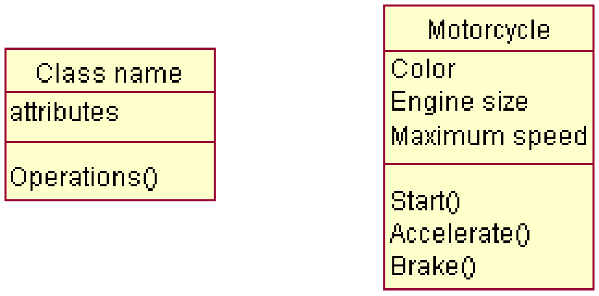


Figure 3. UML Class Representation

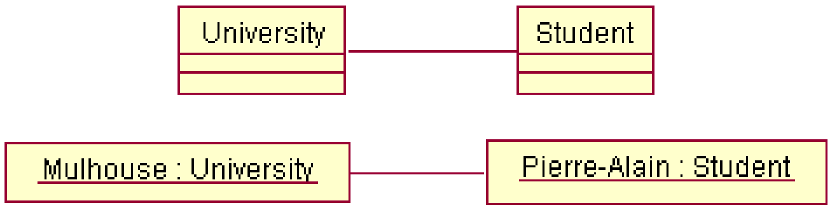


Figure 4. UML Association Relationship

By default, an association expresses a weak coupling between abstractions. An "aggregation" is a special type of association expressing a strong coupling. Aggregation indicates relationships like "part of," "composed of," or "master and slave." It is represented with a diamond. UML also defines even stronger coupling, "composition," meaning that when the owner object is deleted it results in the deletion of its composite objects. Composition is represented with a filled diamond [MUL97].

Inheritance is a relation where one class has all the properties and methods of its parent and extends it by including additional methods or variables. Classes are ordered within an inheritance hierarchy. A "superclass" is an abstraction of its "subclasses." The UML representation of an inheritance relation is shown in Figure 6 [MUL97].

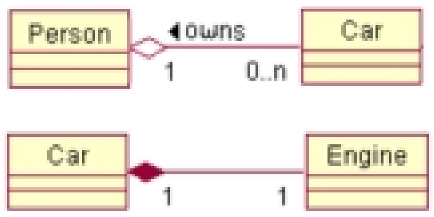


Figure 5. UML Aggregation and Composition Relationships

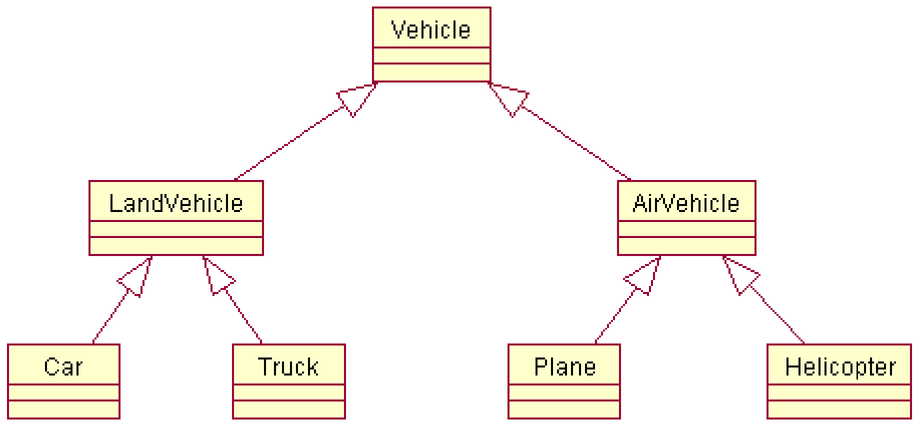


Figure 6. UML Inheritance Relationship

Abstraction, encapsulation, inheritance, reuse, and emphasis on the object structure instead of the procedural structure are themes well supported by the object-oriented paradigm. Rumbaugh, Blaha, Premerlani, Eddy and Lorensen define “abstraction” as focusing on the essential, inherent aspects of an entity and ignoring the accidental properties [RUM91]. Use of abstraction during analysis means concentrating on application domain concepts and not making low-level design decisions. “Encapsulation” (information hiding) is achieved by differentiating accessible and inaccessible properties of objects from outside of the object. Details of an object can be changed while its interface remains the same.

The object-oriented paradigm promises improvement in productivity by being a natural match between implementation and problem. It promotes reuse of objects and increases quality by reducing errors and coupling. It provides better maintainability by encapsulation, and ease of extensibility by simply adding another object or feature to an existing object.

2.2.3. Object-Oriented Modeling Approaches and UML

Object-oriented modeling languages emerged in the 1970’s and different approaches to object-oriented analysis and design have been proposed. In the 1990’s, more than 50 different object-oriented methods were available. The confusion caused by different interpretations limited the progress of these methods. Stronger versions of these methods began to appear by late the 1990s, including OOSE (Object-Oriented Software Engineering) by Ivar Jacobson, OMT (Object Modeling Technique) by Jim Rumbaugh, and Grady Booch’s method. OOSE provided a use-case-oriented approach supporting requirements analysis based on interactions between users and systems. OMT was especially expressive for analysis and information systems. Booch’s method was particularly expressive for system partitioning.

Table 2. Origins of UML

Origin	Element
Booch	Categories and subsystems
Embley	Singleton classes and composite objects
Fusion	Operation descriptions, message numbering
Gamma et. al.	Frameworks, patterns and notes
Harel	State charts
Jacobson	Use Cases
Meyer	Pre- and post-conditions
Odell	Dynamic classification, emphasis on events
OMT	Associations
Shlaer-Mellor	Objects' lifecycles
Wirfs-Brock	Responsibilities and collaborations

The unification of Booch and Rumbaugh resulted in the release of a draft version 0.8 of UML in October 1995. In fall 1995, Jacobson joined the unification process [OMG01]. Table 2 presents the previous efforts that have influenced the unification [MUL97].

The unified methodology is designed to provide guidance to the order of team activities, to direct the task of individual developers and the team as a whole, to specify what artifacts should be developed, and to offer criteria for monitoring and measuring a project's products and activity. Jacobson, Booch and Rumbaugh list the four goals of UML as follows [MUL97]:

1. To represent complete systems using object-oriented concepts.
2. To take into account the scaling issues.
3. To establish an explicit coupling between concepts and implementation.
4. To create a modeling language usable by both human and machines.

The unified development process met the requirements of the software development community with a generic process framework that can be specialized for a variety of software systems, application areas, organizations, competence levels, and project sizes. The distinguishing aspects of UML are the ability to provide a use-case driven, architecture centric, iterative, and incremental design process [JAC99].

2.2.4. Modeling – The Big Picture

When Peter Chen introduced the entity-relationship model, as a special diagrammatic modeling technique for database design, he also explained the problems with the current techniques and integrated his model into a design process. His model addressed weaknesses and strengths in three major data models: the network model, the relational model, and the entity-set model. Chen identified four levels of logical views of data: [CHE76]

1. Information concerning entities and relationships, which exist in one's mind.
2. The information structure.
3. The access-path independent data structure.
4. The access-path dependent data structure.

In the hierarchy of abstractions, an entity-relationship model presents the first and second level. Then, task dependent implementation specifications follow the entity-relationship model. Chen also proposed a four-step design methodology [CHE76]:

1. Identify the entity sets and relationship sets of interest.
2. Identify semantic information in the relationship sets (Cardinality).
3. Define the value sets and attributes.
4. Organize data into entity-relationship relations and decide primary keys.

Likewise, when Ivar Jacobson, Grady Booch, and James Rumbaugh developed UML (Unified Modeling Language), they integrated the technique into the software development process [JAC99]. These two cases imply that the modeling technique should explicitly determine the pitfalls of the former techniques, objectives of the new technique, where to place the technique into the big picture of development process, and some design patterns to be used.

2.3. Facets of Modeling

This section covers the facets of modeling: abstractions, relations, and behaviors. Issues in dynamic models are also covered in this section.

2.3.1. Abstraction

Diana Kao and Norman P. Archer defined modeling as an iterative process that involves constant generation of sub-design tasks and constant moving among such tasks. They proposed a framework suggesting that the completeness of the design output can be enhanced by effective use of abstraction techniques. The object of creating abstractions is to reduce complexity, which is related to the number of objects, their attributes, and dependencies caused by the relations among objects. Generally, abstractions are created by reducing the number of objects and their associated values or by simplifying the relations among objects. Using abstractions, designers can handle the complexity of a problem so that they can focus on certain problem facets, deal with problems at a desired level of complexity, and think about the problem rather than being occupied by the details. Kao and Archer categorized types of abstractions as follows [KAO97]:

- **Horizontal Abstraction:** Horizontal abstractions include many facets at a particular level of detail and deal with breadth of a design problem.
- **Vertical Abstraction:** Vertical abstractions involve several levels of detail along a particular problem dimension and decomposition of design tasks.
- **General Abstraction:** General abstractions are relations or dependencies among ideas that are represented vertically and horizontally.

A top-down design approach to a problem can be either breadth-first or depth-first. A top-down breadth-first approach usually involves identifying the key problem facets (horizontal abstractions) before working any detail of those facets. A top-down depth-first approach focuses on one problem facet from the more general to the detailed level (vertical abstractions). A bottom-up approach deals with the problem at the detailed level first [KAO97].

Kao and Archer also observed the techniques used in modeling a problem, the information chosen by the designer to be included in the model, and the number of abstraction levels identified. Their study, as summarized in Table 3 reveals the differences between domain expert designers and non-domain expert designers [KAO97].

Table 3. Modeling Abstraction

Design Dimension	**Domain Expert**	**Non-domain Expert**	**Comments**
Completeness	Able to identify and state which facets were most crucial	Unclear which facets were most important	Domain experts applied horizontal abstraction more effectively
High level of Abstraction	Demonstrate good ability in clustering ideas into high level facet	Often list many ideas, but failed to group the ideas into meaningful problem facets	Domain experts applied vertical abstraction more effectively
Organization	Tended to use high level concepts and principles to guide the entire design Domain knowledge internally organized into a design schema	Lack of goals and objectives to guide design Lack of design schema	Domain experts generated cohesive designs Domain experts were able to organize design with fewer hanging ideas

2.3.2. Relations

Conrad Bock and James Odell published a series of articles on relations that offer more complete models by exploring different aspects of relations [BOC97A, BOC97B, BOC98A, BOC98B]. Their suggestions are based on treating associations as first-class object types. The first article showed that the current modeling of cardinalities is ambiguous, since the models cannot distinguish between the number of objects in a single link and the number of links in which the same object can participate [BOC97A]. The second article asserts that the standard model of navigation does not include the navigation from an object to the links in which it participates and vice versa [BOC97B]. The third article suggests that the modeling of a "role" will be more complete if it is always taken in the context of relation [BOC98A]. The final article extends the aggregation model by allowing relations to be involved in an aggregation on either side - both as an owner and as a part [BOC98B].

Bock and Odell identify two types of cardinality constraints and discuss how these cardinality constraints might resolve ambiguities in existing methods [BOC97A]:

Single Tuple Cardinality: Single tuple cardinality specifies the minimum and maximum number of objects that can participate in a single *tuple* (or *link* in UML) of the relation. For example, a marriage relation involves one man and one woman.

Multiple Tuple Cardinality: Multiple tuple cardinality specifies the minimum and maximum number of tuples in which an object can participate at a time. A man or a woman can only have one marriage for each.

Most modeling languages combine these cardinality constraints into one constraint. This situation not only introduces loss of some information but also ambiguity during the transition from the modeling phase to the implementation. Bock and Odell suggested a recursive (non-mathematical) technique, which covers the missed information and avoids infinite recursion. Figure 7 demonstrates how to model both single and multiple tuple cardinalities.

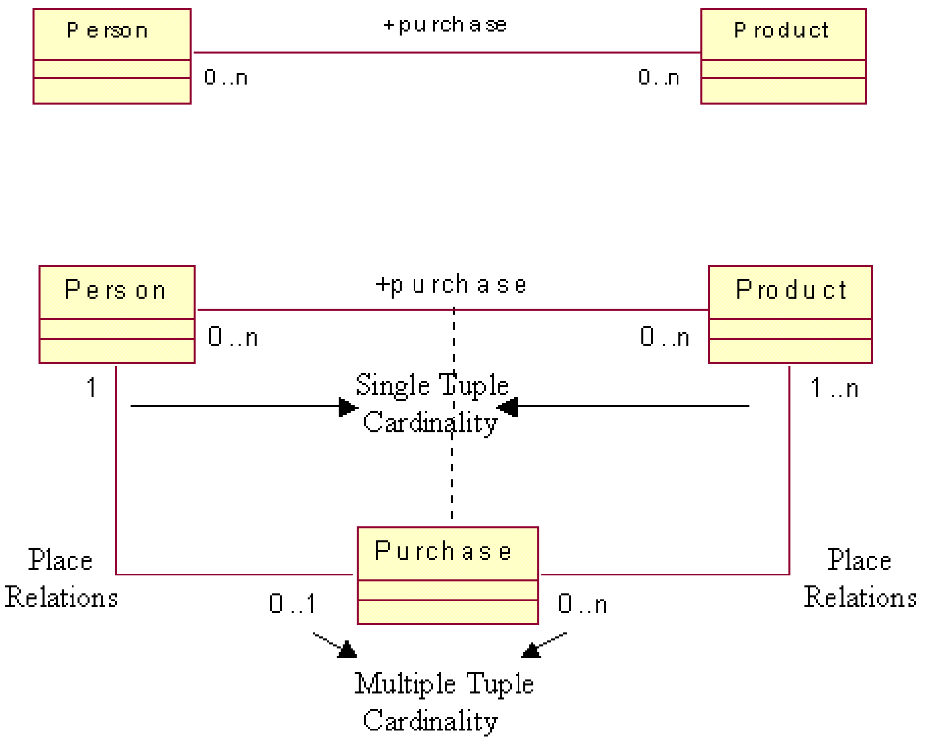


Figure 7. Expressing Cardinalities

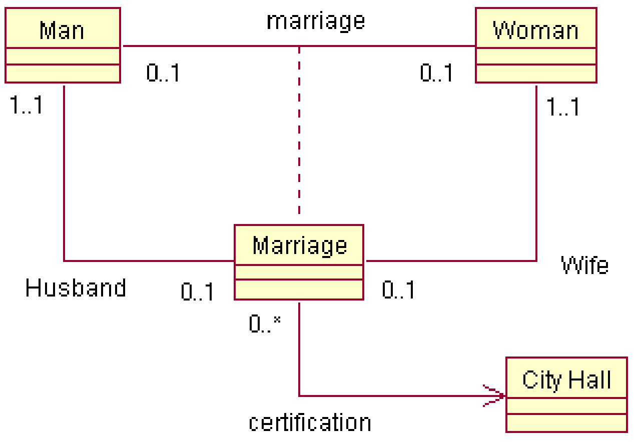


Figure 8. Relations as Objects

As stated earlier, Bock and Odell’s model centers on treating relations as object types (Purchase in Figure 7 and Marriage in Figure 8) and relating them to their participating types (Person and Product). This causes an infinite recursive structure. This problem is addressed by defining a default object, called “place” that keeps only the participant types of the place relation when there’s no user-defined feature (attributes or methods) for the place relation. If the designer decides to define features to the place relation, places are then transformed into full objects. This procedure can be repeated to the level that the user requires [BOC97A].

Navigation between objects is simply a mapping, which is analogous to the mathematical concept of a “function.” The difference between mapping and a relation is that mappings have “directionality.” [BOC97B] For each relation there are navigation choices to implement in the design stage. These kinds of choices are called “design templates.”

Design templates delay the decision of navigation specification. Using design templates and delaying the decision to a later design stage allows designers to focus on optimizing usage scenarios. This also improves usability by allowing selection from different navigation possibilities in each use.

This approach becomes more complicated in the case of place relations. Most current modeling methods cannot model mappings between relation participants and the relation itself, since they do not treat relations as objects. Bock and Odell identified two different types of mappings [BOC97B]:

Optimized mappings: The navigations are directly implemented and answers are obtained quickly.

Derived mappings: The navigation occurs by a defined routine. Since it is based on a calculation or a search, it is slower.

The following considerations are suggested for derived mappings [BOC97B]:

- It is not space-efficient to allow fundamental types to be input to the optimized mappings such as to store a reference to the sum of two integers.
- The designer should keep track of validity of the derived mappings such as a discount’s affect on the price of an item for specific purchase.
- Some mappings might be a hybrid of optimized and derived. For example, the total price of a purchase is calculated by addition of the price of each item in the chart, but, a discount might be applied over some of the items.

Bock and Odell identified two kinds of information that help manage the creation and destruction of tuples (links) [BOC98A]:

Allowed types: Restriction on the objects that can be connected by the relation over given time. For a marriage relation allowed types are a man and a woman.

Current types: Identification of the objects that are currently connected by the relation. Current types for a marriage relation are a husband and a wife. This technique reveals the information missing in most of the current modeling methods, as shown in Figure 9 [BOC98A].

This cardinality restriction allows a better clarification of the model. Bock and Odell also introduced “non-current types,” which covers the objects that could be related, but currently are not, such as a bachelor and a bachelorette for a marriage relation. They mentioned that this type also has similar services and benefits as current types.

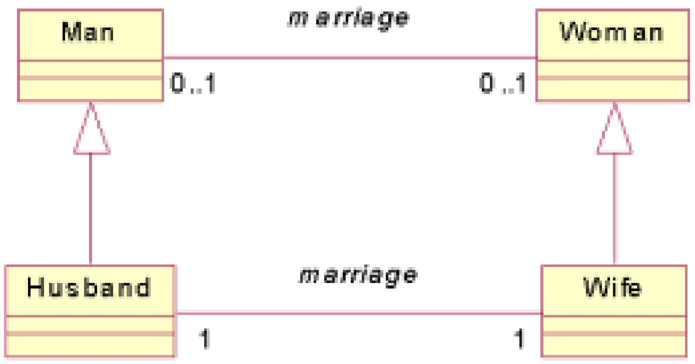


Figure 9. Allowed and Current Types

This model provides several benefits. First of all, the inheritance hierarchy simplifies modeling and provides a closer match to real life. The attributes and methods defined for subtype are effective only in the case of a link between objects. From a conceptual point of view, this model helps clarification between what's possible and what's actual, thus, it expands expressiveness and flexibility. This technique also benefits in aggregations. For example, aggregation between a vehicle and an engine can be better specified by differentiation of car engine and boat engine.

However, this technique requires implementation of run-time reclassification meaning that the class of an object might be changed without affecting its identity. Although most programming languages do not support reclassification, it can be achieved by object-slicing which is casting an object to a base object. This extra effort is paid back by simplification and scalability. This method conflicts with UML in which a role is only an interface that must be supported by the allowed type rather than being a type under which instances can be reclassified [BOC98A].

Bock and Odell also showed that relations might be parts of an aggregate, as well as aggregates themselves. Most modeling methods omit the following uses of relations in aggregations [BOC98B]:

Relations as parts of aggregate: Since a relation is treated as an object, both objects and relations can be a part of an aggregation.

Relations as aggregates: Aggregation may also relate two relations. There are two kinds of aggregates: objects and relations.

The diagrams in Figure 10, Figure 11, and Figure 12 present the usages of the aggregation.

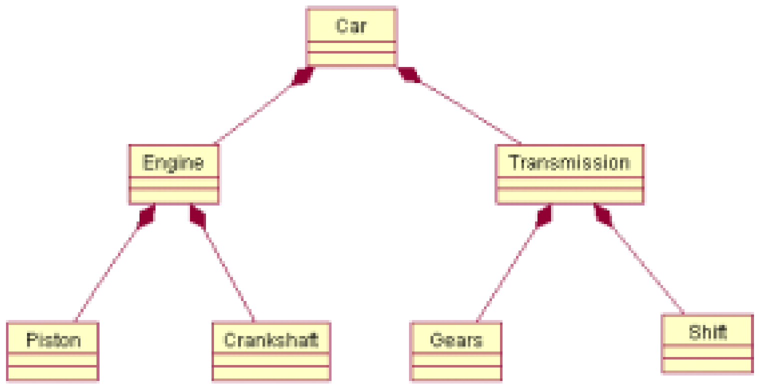


Figure 10. Standard Part Hierarchy

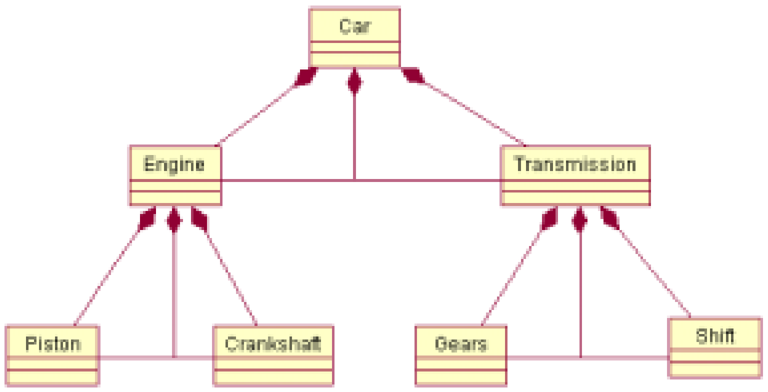


Figure 11. Part Hierarchy with Relations as Parts

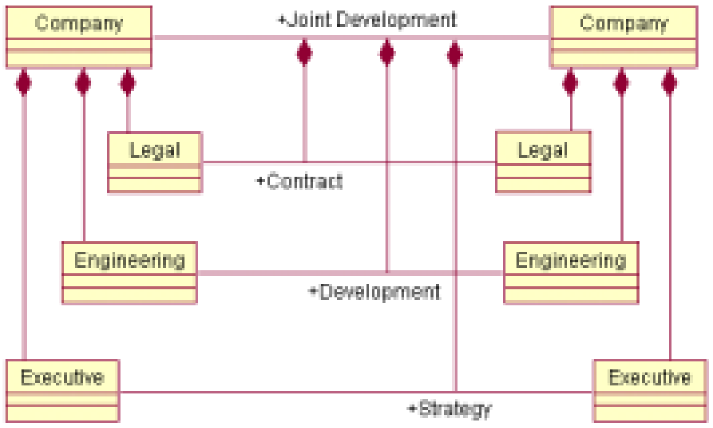


Figure 12. Aggregate Relation Between Relations

2.3.3. Spectrum of Dynamic Systems and Models

Hartmut Bossel explains two different approaches to simulate a behavior: by description and by explanation. The first approach, description of behavior, is based on observing the behavior outputs under different input conditions. This approach treats a system as a "block box." The second approach, explanation of behavior, is based on understanding the parts of a system and interactions among them, thus, a system is treated as a "glass box" [BOS94].

Bossel also provides a specification of dynamic models in a spectrum based on the following terms [BOS94]:

Explanatory-Descriptive: If the goal is not only to mimic the behavior but also to understand how it works, explanatory models are used. These models are also called "process models," "mechanistic models," "real-structure models," or "structural models."

Real Parameter-Parameter Fitting: Real parameters involve using the actual parameters that can be measured directly in the system. If the real parameters are not enough to describe the system, parameter fitting is necessary and the parameters should be designed so that quantitative results of simulations won't conflict with the empirical observations on the system.

Deterministic-Stochastic: This specification is based on the presence (or absence) of random variables involved as an input to the system behavior. Stochastic models produce different results for each run.

Constant Parameters-Time Variant Parameters: The parameters of the system may be constant or a function of time. This means the behavior of the system changes by time or stays constant.

Non-linear-Linear: This mathematical distinction is based on the change rate of the state variables. Analytical methods can be used for linear systems where numerical simulation is usually necessary for non-linear systems because of the complexity involved.

Time-Continuous-Time-Discrete: For continuous models the state of the system can be measured at any instant or time interval, where the state variables in discrete models are observable at certain discrete time intervals.

Space-Discrete-Space-Continuous: Real systems cannot be located at a single point, but, in some cases spatial distribution of the system does not affect their behavior. Airflow distribution on an aircraft wing is essential for wing design, but pressure distribution in a closed hydraulic system is not important since it will be identical at each point. It is obvious that simulation of systems with spatial gradients demands a lot more computation time.

Autonomous-Exogenously Driven: Systems operate in an environment, which they receive inputs, and produce outputs based on their state. Systems that are not subject to exogenous inputs are autonomous. Real systems cannot be autonomous in long run, but often, autonomous properties dominate their behavior.

2.3.4. Behavioral Models

Conrad Bock shows how behavior models can be used in an object-oriented way. Behavior models coordinate steps and are concerned with when to take each execution step and when the inputs are determined [BOC99A].

- **Control Flow:** The control flow model takes each step when another one is complete and does not require any input to be ready. Flow Charts used in programming are an example of this kind of model.
- **Data Flow:** This model takes each step when other steps provide inputs such as functional languages, assembly lines.
- **State Machines:** State machines take each step based on outside events. The inputs are calculated as part of the step itself. Vending machines are this kind of system.

This characterization of behavior models leads to the chart in Table 4. These models can be unified into one at the expense of losing expressibility since each concentrate on different aspects [BOC99A]. Control flow emphasizes the sequencing of steps, data flow emphasizes the calculation of inputs and state machines emphasize a response to an external stimuli.

Bock also identified three different ways to integrate these models into the object-oriented paradigm. Steps in these models can be mapped to the methods of classes and operations written in non-object-oriented way can be invoked. A relatively harder way might be associating each step with the changes to objects that the step is intended to cause, and associating these changes with the steps they initiate. [BOC99A]

In another study, Bock discussed the difficulties and trade-offs in language unification based on their observations on unification of behavior models in UML. Easy-to-use languages are generally designed for particular applications and different languages cannot be simply put into one large one, since it requires integration and harmony of artifacts. A trade-off between generality and application-specific power occurs in unification. [BOC99B]

Table 4. Characterization of Behavioral Models

	Control Flow	**Data Flow**	**State Machine**
Inputs Determined	At start	Before start	At start
Start Conditions	Internal	Internal	External

The authors of the UML integrate three behavior models starting with state machines, since it's the most familiar one to object-oriented designers. Control flow and data/object flow were added later to support business modeling. UML also has a collaboration diagram which shows the interactions of objects performing a task. Examples of UML behavior models are presented in Figure 15 through Figure 14 [OMG01]. Bock indicated the following problems with the behavior model integration in UML [BOC99B]:

Comprehensibility issues. State machine users expect events to come from the outside, not the inside, and expect states to be states of the object, not the behavior. Control and data/object flow users do not see how state machines apply to their models

Emphasis on state machines. A particular event may only be executed once by a state machine. But this is not acceptable in business modeling which treats an event as a persistent object.

Notation for data/object flow. State diagrams cannot be directed to or from particular inputs and outputs of states. If two inputs are of the same type, it is ambiguous in the notation which takes which. Tool vendors are forced to invent or adjust their own notation and consequently will fragment the standard.

State machines do not have parameters. Users requiring functional decomposition cannot effectively reuse state machines. The user is forced to assign the business function to an object to achieve it.

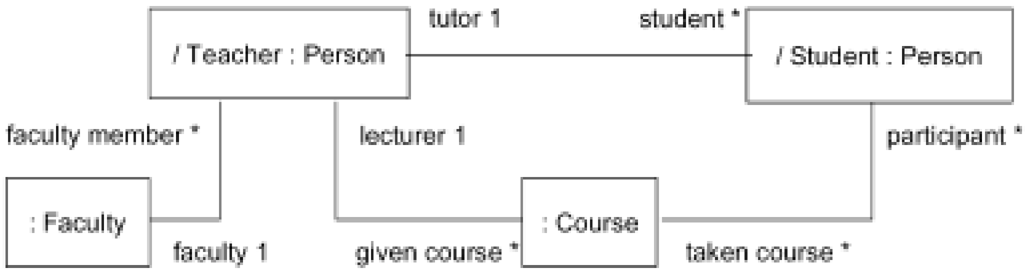


Figure 13. UML Collaboration Diagram at the Specification Level

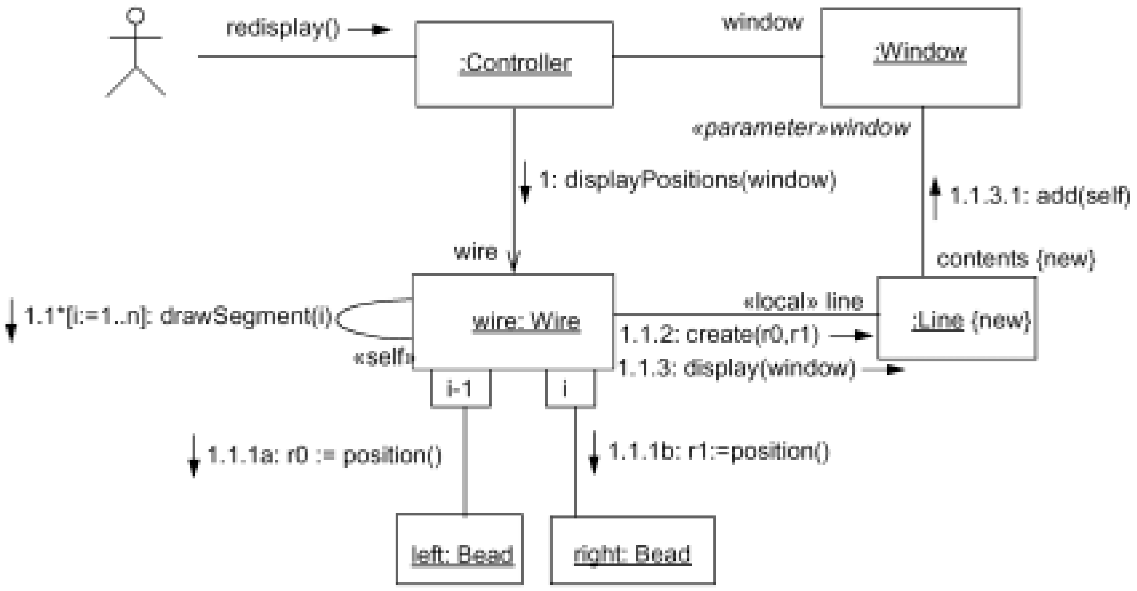


Figure 14. UML Collaboration Diagram at the Instance Level

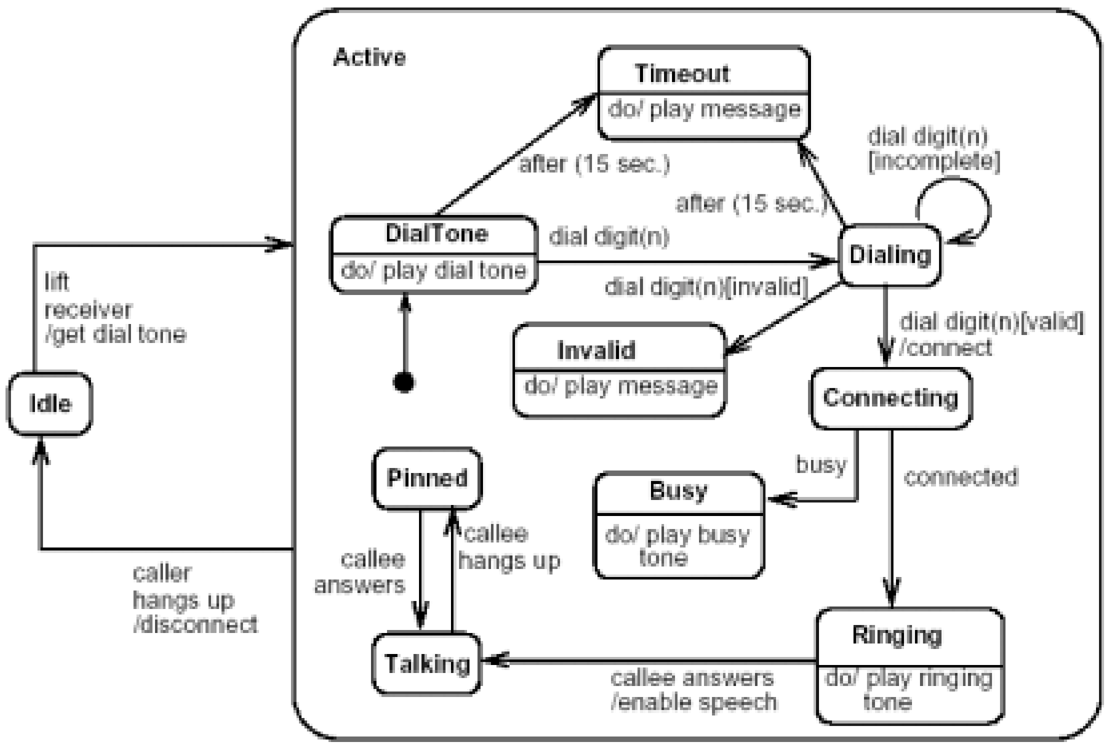


Figure 15. UML Statechart Diagram

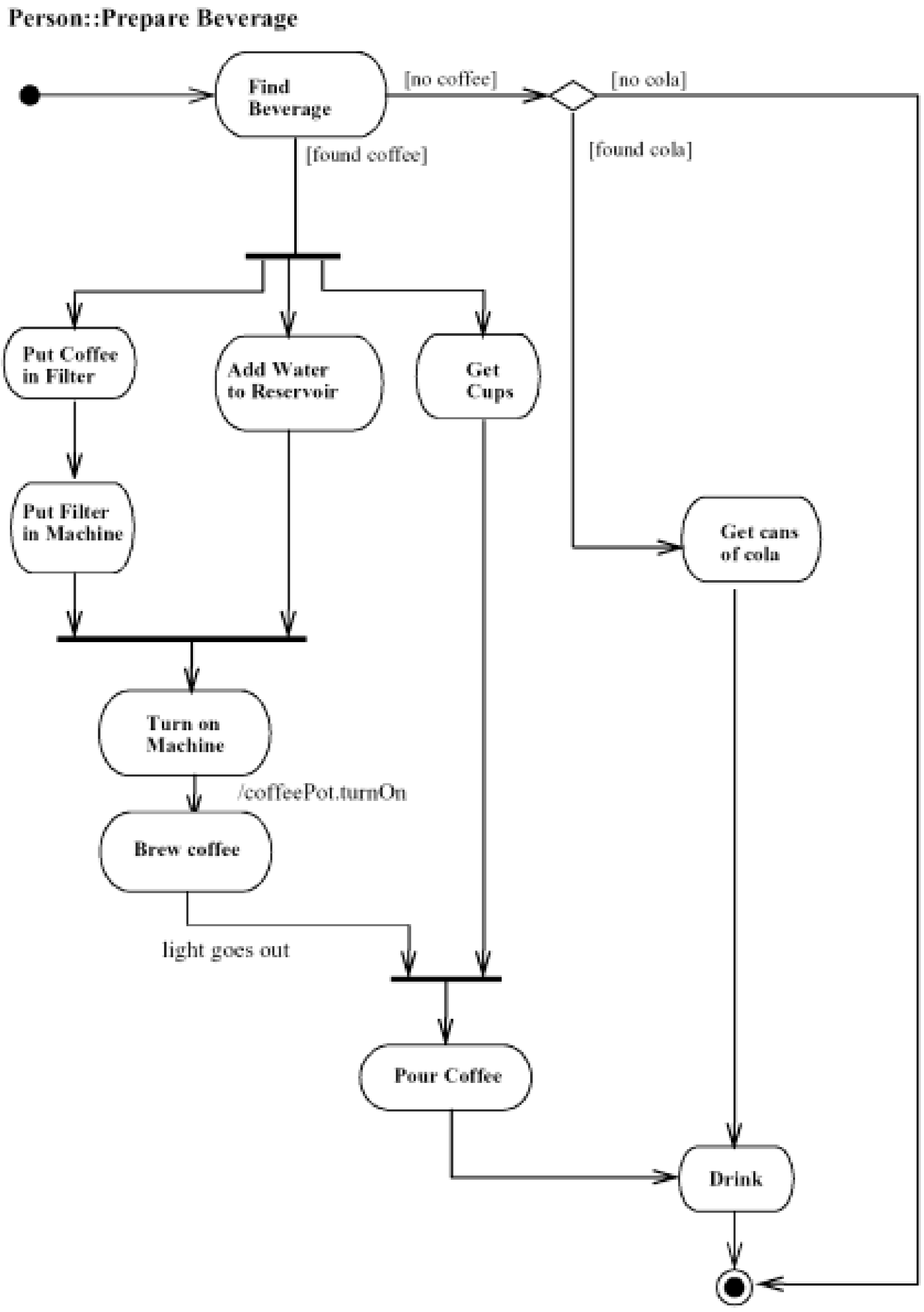


Figure 16. UML Activity Diagram

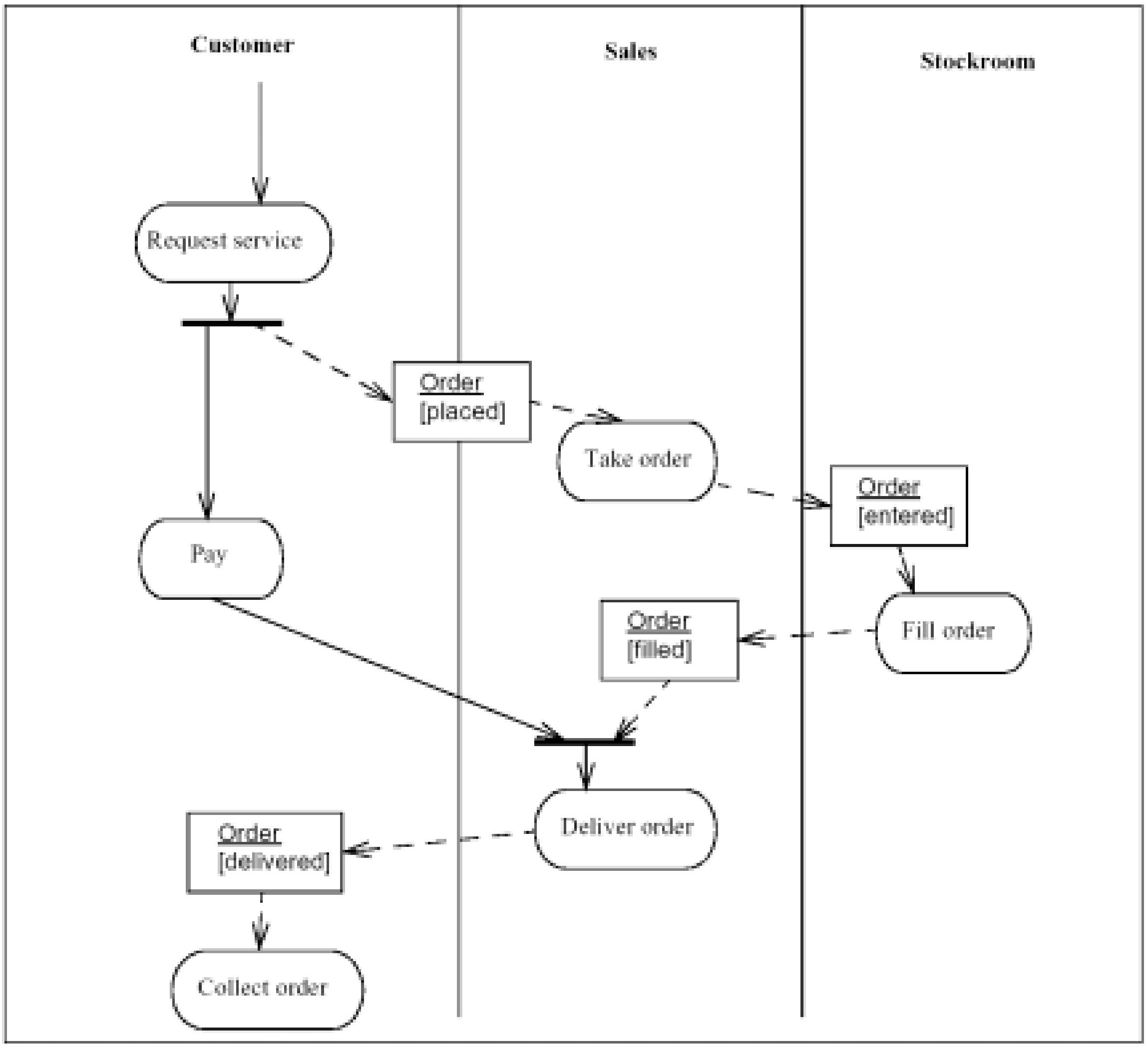


Figure 17. UML Action-Object Flow Relationships

Bock also suggested Gamma's State Pattern as an extension to UML to preserve the benefits of two common approaches to state modeling [BOC00, GAM95]. Bock's object-oriented state is a combination of both behavior state and feature state. The approaches identified by Bock are as follows [BOC00]:

Behavior state. A machines reaction to incoming events is based on its state.

Feature state. This refers to constraints on the attribute values of an object and its links to other objects. An object changes its state based on the constraints it satisfies.

Proposed object-oriented state. This model treats states as objects that are instantiated and deleted at run time as state machine executes, following and extending Gamma's State Pattern.

The state pattern presented by Bock is only aimed at applications in which the operations of an object have state-dependent methods [BOC00]. Operations invoked on the object are delegated to state instances, which have their own methods for the operations.

Figure 18 shows the proposed notation as an extension to UML [BOC00]. "Person's" states are shown as "Sick" and "Well." Sick state is defined as a state class. State classes can participate in associations to model application-specific information. State instances provide a place to record the information as the execution proceeds. The designer can define associations between state classes and various kinds of resources. Figure 18 reveals that state classes provide two separate places to record the following two aspects of feature states [BOC00]:

- A state class specifies general requirement for being in that state.
- The state instance records a set of specific feature values that justify the object being in that state.

Bock also suggests using UML's Object Constraint Language (OCL) to express the constraints as shown in the Figure 19. The OCL is a formal language used to specify invariant conditions that must hold for the system being modeled. In Figure 19, the feature state "married" has a constraint defined via OCL. Being in "married" state is limited to one "spouse," not more or less.

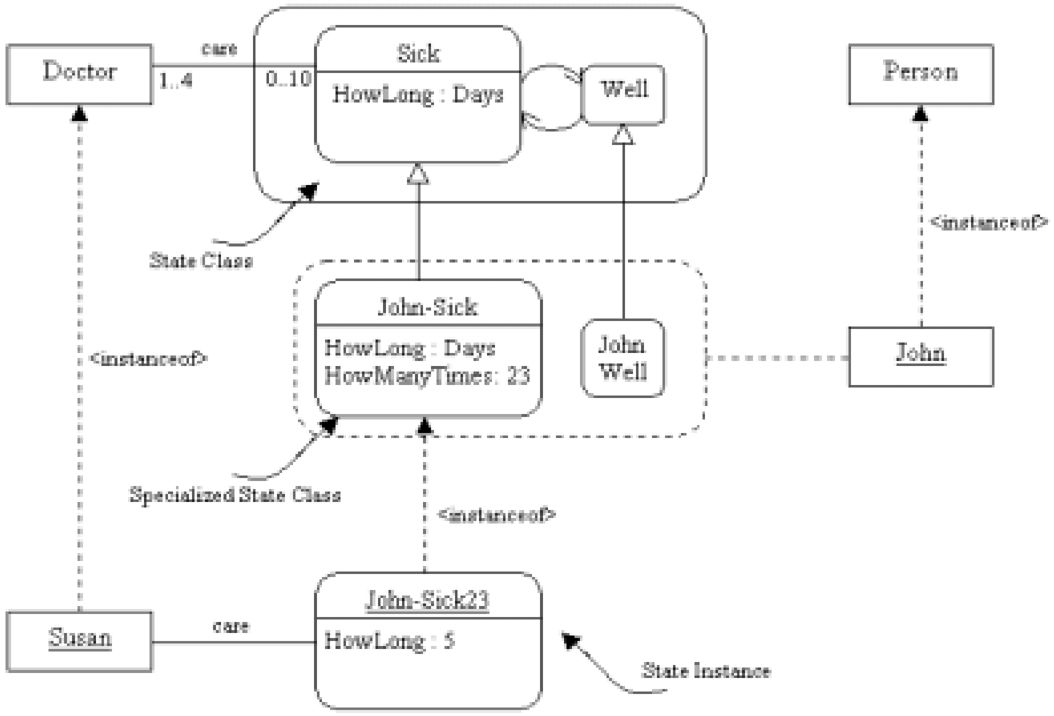


Figure 18. Object-Oriented States

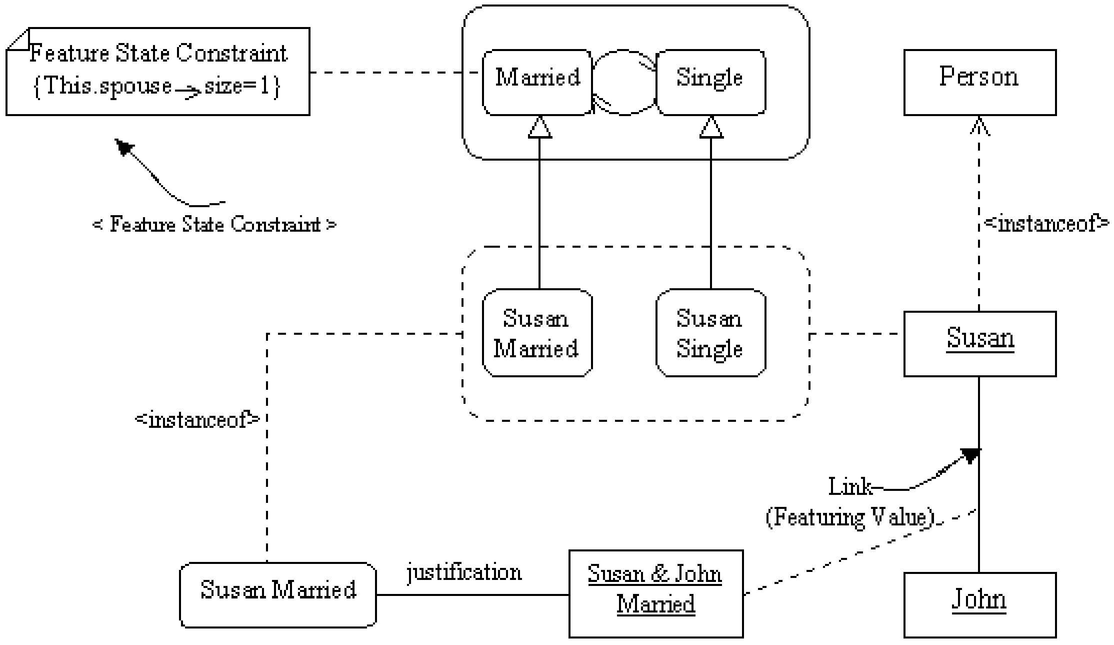


Figure 19. Formalization of Feature States using OO states

2.4. Information Visualization

Expressions in visual languages involve the use of visual structures designed to carry information within their variety of properties such as shape, color, dimension and relative location. Interpretation of the information in visual structures depends on human perception system. Since one of the main motivations to use visual modeling language is to provide human-computer communication, the visual modeling languages should be designed by considering both computational issues and information visualization techniques.

2.4.1. The Human Perception System

The human perception system is composed of a three-level hierarchical organization: Non-foveal portions of the retina, foveola, and receptors in the foveloa [CAR99]. The retina is good at detecting movement or other changes in the visual environment, by maintaining a rough representation of the location of shapes previously examined. However, it cannot hold detail. The foveola provides high resolution movement and focus of the eye and depth information. Two encoding systems are applied to the information: spatial properties such as location, size, and orientation, and object properties such as shape, color, and texture. Receptors in foveloa provide a computationally parallel surveillance structure that moves in the visual field to catch areas with high information content like moving objects.

There are two ways to process visual information: controlled processing and automatic processing. Controlled processing is detailed, serial, slow, and conscious. Reading is an example of this kind of processing. It provides low capacity and can be inhibited. Automatic processing is superficial, parallel, fast, independent of load, and unconscious. While driving, visual information is processed automatically. Automatic processing provides high capacity and cannot be inhibited [CAR99]. Coding techniques to help search and pattern detection should use features that can be automatically processed.

2.4.2. Benefits of Visualization Techniques

External aids serve two purposes: Communicating the idea and discovering the idea itself. During thinking process, human uses internal representations. External aids help expand capacity of thinking, memory, and reasoning. For example, the use a piece of paper as an external aid eases multiplication of larger numbers. The notion of external cognition is used to express the value of external aids and how we map between external and internal representations [CAR99].

Mike Scaife and Yvonne Rogers surveyed how external aids help reasoning [SCA96]. They used three central characteristics to explain aspects of external cognition. "Computational offloading," which is reducing amount of cognitive effort required to solve the problem, "re-representation," which is the representation of problem in the external aid, and "graphical constraining" which is application of constraints graphically. Scaife and Rogers offered a list of general conceptual design issues of which designers should be aware [SCA96]:

Explicitness and visibility. The designer should aim to facilitate higher levels of understanding by means of explicitness and visibility.

Cognitive tracing and interactivity. Designers should pay attention to cognitive traces and interactivity to facilitate ease of use and problem solving.

Ease of production. Designers should consider the ease of production of graphical representation.

Combining external representations. Designers should decide to use different types of external representations from textual to symbolic structures, whichever is suitable.

Distributed graphical representations. Collaborative construction of graphical representations might be an issue.

The following are the six proposed major ways that visualization can amplify cognition [CAR99]:

1. By increasing the memory and processing resources available to the users.
2. By reducing the search for information.
3. By using the visual representations to enhance the detection of patterns.
4. By enabling perceptual inference operations.
5. By using perceptual attention mechanisms for monitoring.
6. By encoding information in a manipulable medium.

2.5. Visual Language Theory

This section covers the specification and proposed frameworks for visual languages and two example studies.

2.5.1. Specification of Visual Languages

Kim Marriott, Bernd Meyer, and Kent B. Wittenburg surveyed the formalisms that have been suggested for visual languages over the last 30 years. The main motivation of specification is to facilitate communication and interaction between humans and computers [MAR98B].

Visual languages are not necessarily sequential, meaning that drawing and interpretation order is irrelevant. Sequential languages like textual languages only have the relation “immediately proceeds” in their grammar where diagrams may have relations such as “above,” “below,” or “adjacent to.” With these differences, it is not always easy to specify visual languages. Currently, there are three main approaches to specification of visual languages. Marriott, Meyer, and Wittenburg also presented other kinds of formalisms that do not fall into one of these categories [MAR98B]:

- **Grammatical approaches:** Based on the grammatical formalisms of textual language specifications. The difference is dealing with sets instead of sequences to specify geometric relations other than sequential.
- **Logical Approaches:** Uses first-order mathematical logic or other forms of mathematical logic, which often stem from artificial intelligence. These approaches are usually based on spatial logic and axiomatization of the different possible geometric relations. One advantage of this is the same formalism can be used to specify both syntax and semantics of the language.
- **Algebraic Approach:** Uses an algebraic specification that consists of composition functions which construct complex pictures from simpler picture elements. Parsing is typically achieved by finding a function sequence that constructs the picture.

Application areas for visual language specification might be graphical user interfaces and interpretation of low-level media such as handwriting, sketch recognition and image processing. Specification might be useful in graphical user interfaces for interpretation of user input, design support and interaction with visual and multi-media databases [MAR98B].

2.5.2. Human Computer Interaction Framework

Hari Narayan and Roland Hubscher proposed a theoretical framework for visual languages that emphasize human-computer interaction and addresses both computational and cognitive issues [NAR98]. Visual languages are intended for use both by computers and humans. They should be designed and analyzed based on both computational and cognitive requirements. This implies that theoretical analysis should address issues of comprehension, reasoning, and interaction as well as issues of visual program parsing, execution, and feedback. Narayan and Hubscher also provide the following definitions [NAR98]:

Visual languages: Languages with alphabets consisting of visual representations that are used for human-human or human-computer interaction.

Diagrammatic representations: Visual representations that encode and convey information about their referents without being true analogs of the entities being represented.

Diagrammatic reasoning: The process of comprehending and making inferences from diagrammatic representations.

Visual representations are designed to explicitly show the relations in the domain by spatial and visual organization of information. This allows a viewer to recognize relevant patterns, to detect emergent properties, and to derive meaning and inferences [NAR98].

Some application areas for visual languages include visualization of information, graphical simulations, and direct manipulation of visual languages by graphical user interfaces to enhance diagrammatic reasoning. Visual languages might also be used for software visualization to enhance programming and debugging environments [NAR98].

Narayan and Hubscher build a framework for analysis and synthesis of visual languages on three objects of interest to any theoretical or practical investigation: A computational system, a cognitive system, and the language itself. The success of a visual language in their framework depends on two criteria: Computational tractability and cognitive effectiveness [NAR98]. Figure 20 shows the exchange of information between human and computer by means of computation and cognition processes. The computer parses, interprets and executes user inputs while the user uses perception and reasoning processes [NAR98].

The visual language analysis framework is divided into three subsections: Representation of information, cycle of interaction, and evaluation. The framework structure is presented in Table 5.

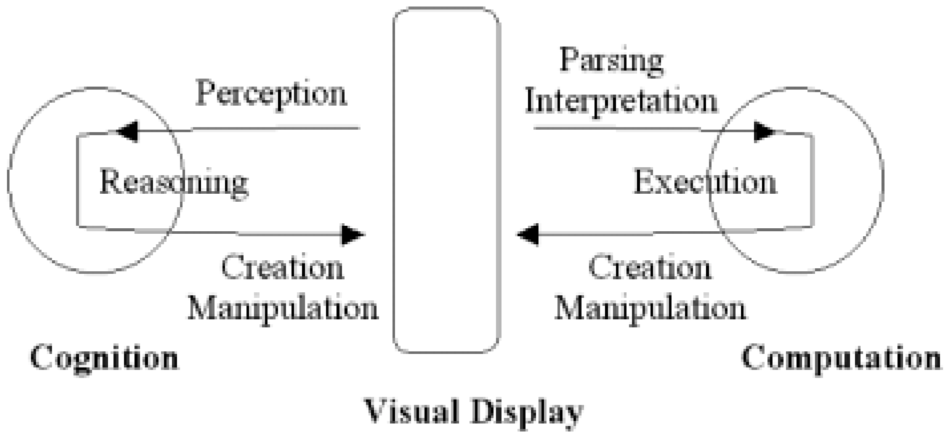


Figure 20. HCI Framework

Table 5. Taxonomy for Visual Language Research

Representation of Information	Cycle of Interaction	Evaluation
Application Domain Static Syntax Static Semantics Dynamic Syntax Dynamic Semantics	Granularity Visual Communication Computational Aspects Cognitive Aspects	Computational Evaluation Cognitive Evaluation

Representation depends on the information being represented. The application domain typically consists of objects, relations between objects and attributes of objects depicting their state. Dynamic processes result in state changes. The language is a set of valid sentences syntax is the rules for creating valid sentences and semantics stands for the meanings of the sentences. Thus, a language has static and dynamic properties in their syntax and semantics.

The cycle of interaction describes the cognitive processes between human and computer. The granularity of the cycle of interaction is a criterion to compare visual languages. A visual language may be used for one or two-way communication. Depending on the visual language, different computational processes may involve such as parsing, interpretation, execution and generation. The cognitive aspects involved are perception, comprehension, inference, and creation. The visual languages are evaluated by their computational efficiency and cognitive effectiveness. The analysis can be done either at individual language or as a comparison of two or more languages.

2.5.3. Representation Framework

Marc Andries, Gregor Engels, and Jan Rekers studied the representation of a visual specification in a computerized environment. They stated that this software environment should represent the specification at four levels in order to perform its tasks: Physical layout, pictorial structure, abstract structure, and representation of the meaning [AND98].

The physical layout and the meaning of the diagram are important to the users of visual languages. Andries, Engels, and Rekers proposed two intermediate representations, Spatial Relations Graphs and Abstract Syntax Graphs to connect physical layout to the meaning as shown in Figure 21 [AND98].

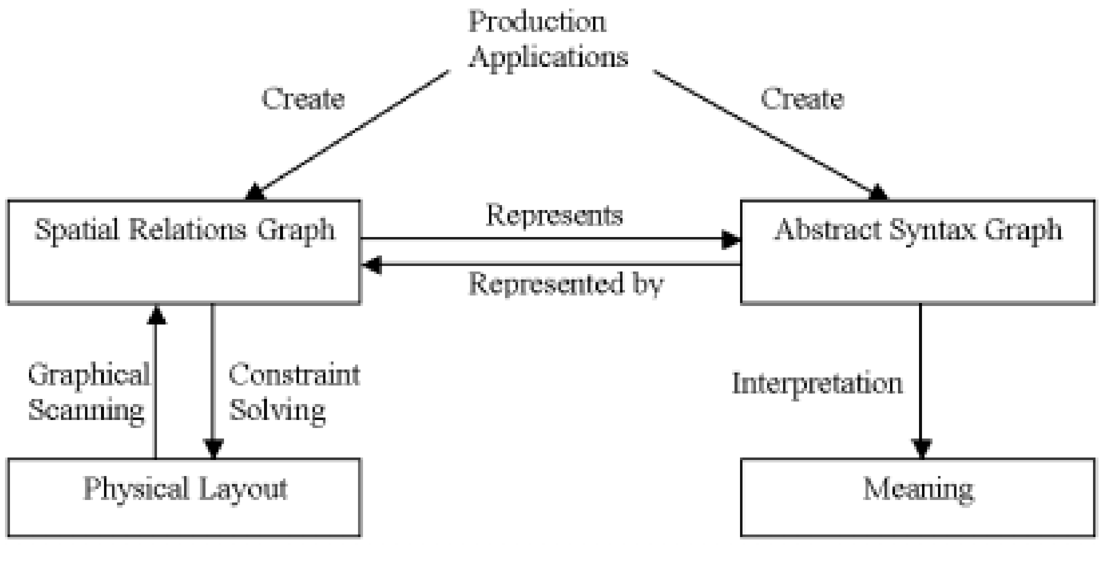


Figure 21. Representation Framework

The Physical Layout consists of graphical objects (lines, circles, rectangles, text) that are not interpreted yet. The Spatial Relations Graph is an abstraction of physical layout and interprets diagram as spatial relations and objects. The spatial relations graph is graphical and defines pictorial structure. It is generated by means of graphical scanning. The constraint solver can generate a physical layout from the spatial relations graph. The Abstract Syntax Graph describes visual sentences consisting of nodes and edges. A semantic processor can be used to interpret the meaning of the abstract syntax graph.

2.5.4. Example Applications of Visual Languages

Simon J. Buckingam Shum, Allan Maclean, Victoria M. E. Bellotti, and Nick V. Hammond surveyed the use of graphical notations to support argument construction and communication. They present a use-oriented analysis of a graphical argumentation notation named QOC (Questions, Options, and Criteria). Their study focuses on the specific domain of software design, which has following problems [SHU97]:

- Some decisions that have been made early may be unclear to the subsequent designers.
- It is hard to keep track of discussions, decisions, and the criteria for decisions.
- These conditions block communication, reuse and recovery of reasoning previously made.

These problems of the software design process can be assisted by an argumentation-based design rationale. This method clarifies vague requirements and tracks their evolution by means of representing multiple viewpoints and trade-offs. It offers consistency in decision making, documentation of decision processes, and building cumulative design knowledge through systematic reuse [SHU97].

QOC notation is based on four structures: Questions are used to encapsulate key issues that shape the design, Options are alternative answers to Questions, Criteria are used in assessing one Option over another, and Assessments are the relations between Options and Criteria. Figure 22 [SHU97] presents the vocabulary of QOC, Figure 23 [CRE98] and Figure 24 [SHU97] show typical usages of the structures in discussion graphics and Figure 25 [SHU96] shows a screenshot of a software implementation.

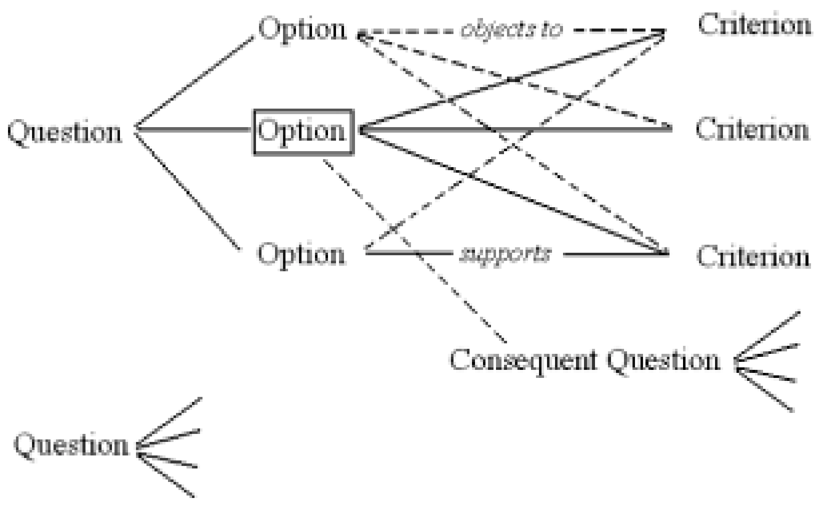


Figure 22. QOC Grammar

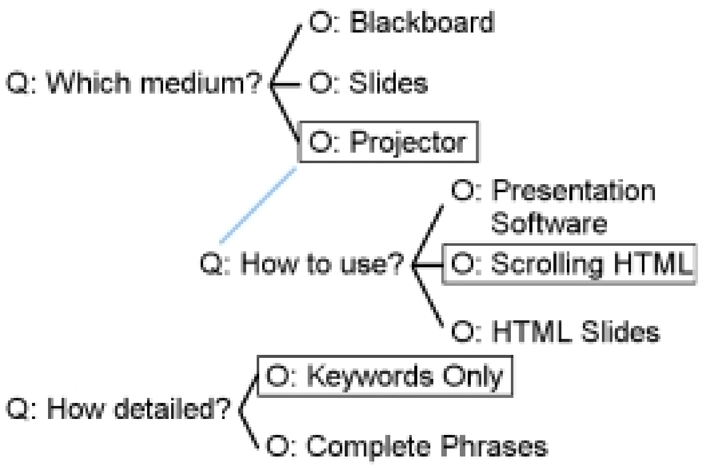


Figure 23. QOC Example - Presentation Preparation

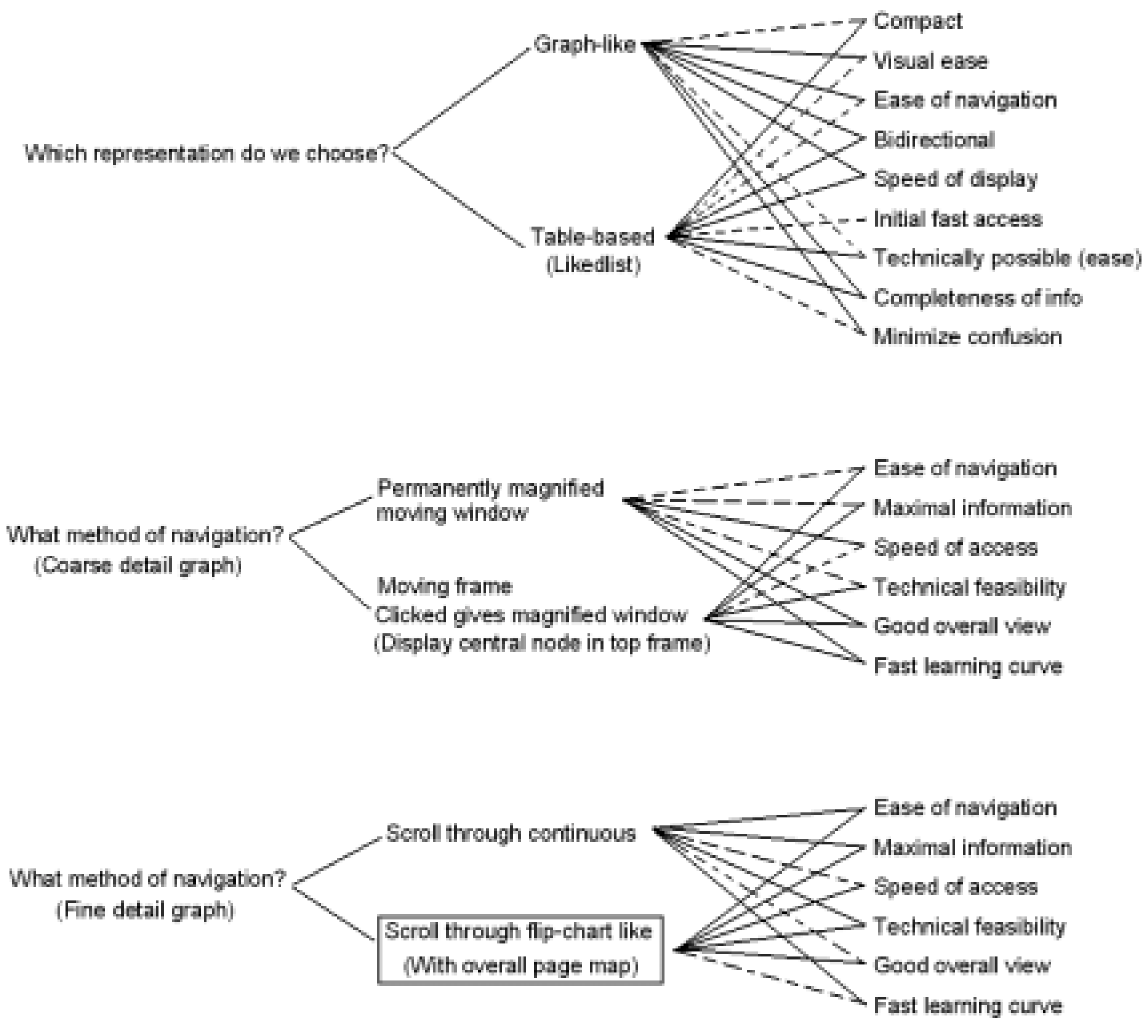


Figure 24. QOC Example - Network Browser User Interface Design

Shum, Maclean, Bellotti, and Hammond analyzed their notation with three empirical studies [SHU97]. They presented data drawn from video-based observations of designers using QOC while solving problems. Four steps are watched in expressing ideas using QOC notation: Identifying and separating elements of ideas (Unbundling), deciding whether a contribution is a Question, an Option or a Criterion (Classification), labeling (Naming), and linking to the other ideas (Structuring).

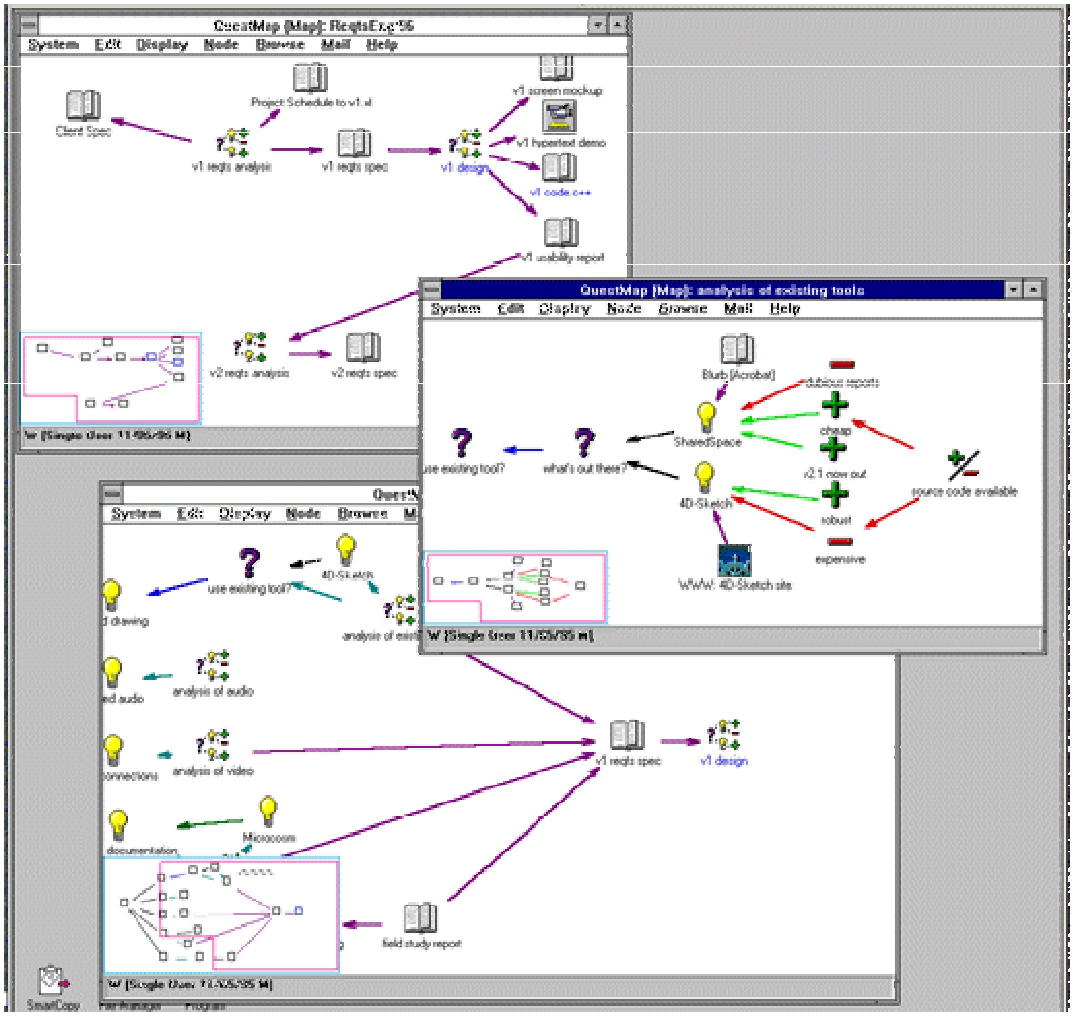


Figure 25. QOC Software Screenshot

The conclusion of the study was that the QOC notation provides the most support when elaborating poorly understood design spaces, but it creates a distraction for well-constrained and understood design spaces [SHU97].

D. Jager researched generation of tools from graph-based specifications. Jager's approach is based on formal meta-modeling of visual languages in the very high level programming language PROGRES (Programmed Graph Rewriting Systems). Tools for visual languages are constructed automatically from the meta-model. PROGRES offers a variety of features for manipulating graphs such as traversing paths within a graph, matching graph patterns, and it supports the graphical specification of graph patterns [JAG00]. Jager indicates that the resulting tool is not the kind of tool they would like to have. It is slow and the specification interpretation is environment dependent. Visualization is not suitable and the user interface has usability problems. Finally, the code is difficult to maintain, since it depends on the thesis students changing every year [JAG00].

2.6. Principles for Visual Language Design

This section presents the principles for visual language design.

2.6.1. Critical Tasks of Modeling Languages

R.F. Paige, J.S. Ostrof, and P.J. Brooke suggested that like programming languages, modeling languages should be designed if they are required to be practical, usable, and accepted. The design process should be based on principles. The starting point to derive these principles is to ask the intention to use modeling languages [PAI00].

The key question, the intention to use modeling languages, leads to the analysis of critical tasks required by users of modeling languages. The critical tasks identified by Paige, Ostrof and Brooke are architectural description, behavioral description, system documentation, and forwards and backwards generation [PAI00].

Architectural Description: Modeling languages are used to describe a system in terms of abstractions and relationships at appropriate levels of detail. Modeling languages should support development of large models and tracability between levels of abstractions.

While concentrating on large-scale model support, designers of the language should not compromise the applicability of the language to small systems otherwise the language might be difficult to learn for new users.

Behavioral Description: Behavioral descriptions capture the details of what each abstraction represent, what each does, and when the interactions occur. Some types of behavioral descriptions may be process algebras, state-based descriptions and natural language.

System Documentation: Modeling languages are also used to provide documentation on how the system works. Other than the model, which itself is a document, taking notes is considered documentation support. The modeling language may provide automatic report generation based on the notes and the model.

Forwards and Backwards Generation: One of the main interests in using modeling languages is the ability to transfer from visual to textual language and vice versa. This is sometimes called "round trip engineering." In order to achieve this kind of capability, the modeling structures of the language should be designed so that it will be easy to map them to the structures of the textual language. Synchronization between the model and textual definition becomes an issue, since the users often do not maintain both together.

Paige, Ostrof, and Brooke suggest that the goals defined above are not independent, so, like all other engineering problems, designers of the language will face tradeoffs and it will be difficult to satisfy each one of them [PAI00].

2.6.2. Design Principles

Paige, Ostrof, and Brooke state that a great deal of effort has been spent on setting up programming language design principles. Modeling languages and their tools should be designed with the same care. Techniques, criteria and principles for designing modeling languages should be produced and be validated by experiment. Design principles for modeling languages based on critical tasks are explained as follows [PAI00].

Simplicity: This should be one of the leading principles since the language is intended to be used as a communication aid among humans and between human and computer. If the language is simple then it will also be memorable, which is a desired property. Simplicity provides ease of learning, the ability to draw models by hand, and greater ease in creating software tools to support the language. There should be no unnecessary complexity in the language.

Uniqueness: If a language has the uniqueness property, it provides only one good way to express every concept. This prevents ambiguities and redundant overlapping expressions in the models.

Consistency: This points to the purpose of the language. Any feature in the language should address the purpose, otherwise it should be discarded. The authors mention that it is hard to determine whether UML is consistent, since there are no precise design goals other than standardization of modeling concepts. Consistency of language should not be confused with the consistency of the model. Consistency of the model is related to the reliability and will be discussed later.

UML allows users to describe a system with several different models. These models may be independent like class diagrams, deployment diagrams, use-case diagrams. Although they capture and emphasize different aspects of the system, consistency between models might become an important issue for a designer dealing with large-scale systems. It is questionable whether a consistency check for UML can be automated.

Seamlessness: This principle helps the ability to generate code from model. It involves using the same abstractions in the model and in the textual language. This avoids a logical “impedance mismatch.” UML is not seamless since some transformation mechanisms are required to generate code for behavioral models.

Reversibility: The ability to generate a model from code contributes to the production of maintainable code and to the documentation. This is a complex process since the textual definition might implement a structure that cannot be expressed in the visual modeling language.

Scalability: The language should provide mechanisms to handle large-scale problems. At the same time, these mechanisms should not detract from the design of small-scale models. To hold this principle, the language should provide concise mechanisms to define the fundamental abstractions, ways to hide details and grouping mechanisms.

Supportability: It should be suitable for humans, since it will often be used on a white-board or paper. It is also meant to be used by computerized tools. The language should be implementable and supportable by software tools. This principle introduces restrictions in syntax and semantics of the language.

Reliability: The goal is to produce quality models. To ensure reliability of the design, the language should provide support for automatic consistency checks via the grammatical rules of the language.

Space Economy: The models should take as little space on screen or page as possible to reduce distractions caused by search and browsing.

B. Henderson-Sellers, D. Firesmith, and I.M. Graham [HEN97] outline the characteristics of Common Object Modeling Notation, COMN, which is a notation of OPEN (Object-Oriented Process, Environment and Notation) Modeling Language (OML). The benefits of OML they mention reveal typical expectations from visual languages. Usability of the notation is improved by intuitive symbols that help learning the syntax and semantics of the language. The language should be simple, and consistent. Sellers, Firesmith, and Graham state that integration of semiotics (study of signs and symbols) into the syntax enhanced usability of COMN. Since a modeling language is intended to be used among humans it should be easy to draw by hand and avoid using features such as color, boldface, and italics. It should be able to handle large-scale projects and should not compromise usability for small-scale problems. The language's consistency with traditional notations also helps reduce misinterpretation [HEN97].

2.7. Evaluation Criteria

Frank van Harmelen, Manfred Aben, Fidel Ruiz, and Joke van de Plassche studied formal modeling languages that have begun to play an increasingly important role for knowledge-based system (KBS) modeling. These languages reduce the vagueness and ambiguity of informal descriptions, enable validation of completeness and consistency through formal proofs, and bridge the gap between the informal model and the system design. However, they suffer from usability problems. Harmelen, Aben, Ruiz, and Plassche took $(ML)^2$, a formal KBS modeling language, developed in 1990, as a case study and applied an evaluation. They used the following set of evaluation criteria, which can also be generalized for other languages [HAR96].

- **Expressiveness:** Were certain things impossible to express? Were some things difficult to express?
- **Frequency of errors:** What are the most common errors and what are the frequencies of those errors. Is there any way to identify and avoid them?
- **Redundancy:** Was redundancy present in models? Can we identify different type of redundancy? How can redundancies be avoided?
- **Locality of change:** Do changes propagate through the formal models? If so, what are the causes, and can they be avoided?
- **Reusability:** Do our models enable reusability?

- **Guidelines and tool support:** Are these guidelines proposed in earlier research helpful? Was the tool support useful?

T.R.G. Green and M. Petre [GRE96] identified that evaluation of a programming language requires both the psychologist's and computer scientist's point of view, however, it is difficult for psychologists to understand the design issues, and computer scientists might fail to see their creations through a psychologist's eyes. Green and Petre proposed a cognitive dimensions framework as an evaluation method to visual programming languages so that a programmer can concentrate on the standard tradeoffs by means of these dimensions [GRE96].

The cognitive dimensions framework defines a small set of terms. The dimensions are meant to be coherent with each other like physical dimensions. A programmer thinking his design along these dimensions will explore the tradeoffs involved. Any cognitive artifact can be described in these terms. Although that description might be at a very high level, it will predict some major aspects of the user activity. Green and Petre used two commercially available visual programming languages: LABVIEW and PROGRAPH to illustrate the framework and demonstrate the type of conclusions to which the framework leads. The list of dimensions is as follows [GRE96]:

Viscosity: How much effort is required to perform a single change? Does a local change in the model affect other parts of the model in an unnecessary user interaction?

Abstraction gradient: What are the minimum and maximum levels of abstraction? Can fragments be encapsulated? Introducing more abstractions might be a solution to viscosity problems. Well-chosen abstractions can also increase comprehensibility and protect against errors.

Closeness of mapping: What programming techniques need to be learned to map problem domain to program domain? The closer the real world is to the program world, the easier the problem solving is going to be.

Consistency: When some of the language has been learned, how much of the rest can be inferred? The language might be consistent for the designer but it might create problems to the user. Increasing abstractions can also change the closeness of mapping either for better or worse.

Diffuseness/Terseness: How many symbols or graphic entities are required to express a meaning? Some notations might be achieved more compactly by reducing the number of symbols used to solve the problem.

Error-proneness: Is there any ambiguity in the notation? Does the notation itself lead to errors?

Hard mental operations: Are there places where the user needs to resort to fingers or pencil annotation to keep track of what is happening? The language should avoid brain twisters. The problematic mental operations must lie at the notational level, not solely at the semantic level.

Hidden dependencies: Is every dependency overtly indicated in both directions? Is the indication perceptual or only symbolic? Hidden dependencies might be introduced by more abstractions. Browsers might be used to make hidden dependencies visible. But the distractions of invoking the browser break up the pattern of problem solving. Over-specialized views given by a browser often deprives the programmer of opportunistically taking advantage of information from other sources.

Premature commitment: Do programmers have to make decisions before they have the information they needed? Increasing abstractions might force the designer to guess ahead.

Progressive evaluation: Can a partially-complete program be executed to obtain feedback on "How am I doing?"

Role expressiveness: Can the reader see how each component of the program relates to the whole?

Secondary notation: Can programmers use layout, color, or other cues to convey extra meaning?

Visibility: Is every part of the code simultaneously visible? Introducing visual browsers may decrease visibility problems.

2.8. Literature Review Summary

This section presented the relevant current literature in the research disciplines of simulation, modeling, visualization, language theory, and software engineering. The literature study on simulations focuses on the modeling aspect of simulations. It included advantages of the simulation modeling as surveyed by Tayfun [TAY99] and general guidelines for simulation design by Banks and Gibson [BAN96]. Banks and Gibson also inform about the "evolving" characteristics of the simulation model [BAN98], and the danger of using simulation software without programming [BAN97]. Bargiela [BAR00] divided the strategic directions in simulation and modeling into three categories, emphasizing the importance of multi-domain and distributed collaborative simulations. Clarke introduced the generic co-simulation tool called pLUG&SIM [CLA99] that provides an environment to build interfaces between models of different simulation software. An object-oriented language, "Modelica" [GOU99, MOD00], published by design group created within the ESPIRIT "SiE-WG" project was surveyed. Modelica is a textual language to specify large, complex, and heterogeneous physical systems. The comparison of Modelica and VLGM was also provided.

Concepts and different aspects of modeling and object-orientation were examined from various sources [RUM99,FOW99, STE99, MUL97, RUM91]. The research also included the unification process of object-oriented modeling techniques by James Rumbaugh, Ivar Jacobson, and Grady Booch [OMG01, MUL97, JAC99]. The study by Peter Chen (entity-relationship model) [CHE76] and UML [JAC99] implied that the modeling technique should explicitly determine the pitfalls of the former techniques, objectives of the new technique, where to place the technique into the big picture of development process, and some design patterns to be used. Three facets of modeling, abstractions [KAO97], relations [BOC97A, BOC97B, BOC98A, BOC98B], behavioral models [BOC99A, BOC99B, OMG01, BOC00, GAM95], and specification of dynamic models [BOS94] were examined in detail.

The background study about visual language theory [MAR98B, NAR98, AND98, SHU96, SHU97, CRE98, JAG00] , human perception system [CAR99], and amplification of cognition [CAR99, SCA96] by visualization techniques formed the theoretical basis of the research. Finally, the design principles and evaluation criteria for visual languages were surveyed [PAI00, HEN97, HAR96, GRE96].

3. METHODOLOGY

This chapter explains the methodology used to solve the problems of textual simulation modeling.

3.1. Motivations to Develop a Visual OO Modeling Language For Modeling

Visualized information proved to be easier to manage than the textual information by increasing the (human's) working memory and processing resources. Visualization techniques reduce the search for information, enhance the detection of patterns, and enable perceptual inference operations [CAR99]. These properties of visual modeling make it preferable to textual modeling. Visual languages facilitate not only ease of development, but also communication of ideas and discovery of new ideas.

3.2. Drawbacks of UML for OO Simulation Modeling

The literature review in Chapter Two discussed the object-oriented paradigm and related visual modeling techniques. The Unified Modeling Language (UML), as defined in its specification, is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems [OMG01]. This section focuses on UML and its usability for simulation modeling.

3.2.1. Scenario Construction Scalability

Models, specifically simulation scenarios, are composed of instantiated objects and links between them. To be used in simulation modeling, the language should offer a means to group instantiated objects in a hierarchical manner. This grouping abstraction will be subject to relations in an upper-level hierarchy.

UML's emphasis is on the design of object types (classes) and relations between them. UML defines three mechanisms to group other model elements: "package," "subsystem," and "model" [OMG01]. "Packages" are non-instantiable and can be applied to all kinds of UML elements including instances. However, the semantics of packages only provide a name space for the elements they cover. "Subsystems" may be instantiable or non-instantiable and are used to provide a grouping mechanism for specifying a behavioral unit of a physical system. The semantics of an instantiable subsystem are similar to the semantics of a composite class. A composite class is composed of other classes forming a higher-level abstraction. Typically composite classes are defined and then instantiated. A grouping mechanism is predefined and not arbitrary, meaning that subsystems and composite classes are not usable for grouping arbitrary instances. A "model" in UML is a description of a physical system at a certain level of abstraction such as a use case, analysis, design, implementation, computational, engineering, or organizational model. A UML "model" does not provide suitable abstraction for grouping instantiated objects either.

3.2.2. Problems With Behavioral Diagrams and Code Generation in UML

Paige, Ostrof, and Brooke suggested "Forward and Backwards Generation" as one of the four critical tasks required by users of modeling languages [PAI00]. It is a required ability to transfer from visual to textual languages and vice versa. In order to achieve this kind of ability, modeling structures of the language should be designed so that it will be easy to map them to the structures of the textual language. To support forward and backwards generation in a modeling language "Seamlessness" is suggested as a design principle, which involves using the same abstraction in the model and in the textual language. UML is not seamless since some transformation mechanisms are needed to generate code for behavioral models [PAI00].

As detailed in Chapter Two, Bock explored the problems with behavior model integration in UML in four basic areas [BOC99B]:

- Comprehensibility issues;
- Problems caused by emphasis on state machines;
- Problems with notation for data/object flow; and
- State machines do not have parameters.

These problems not only lead to misunderstandings and inconsistent models, but also inhibit code generation.

3.2.3. Consistency Problems

“Consistency,” another design principle proposed, focuses on the purpose of the language. Any feature in the language should address the purpose, otherwise it should be discarded [PAI00]. Paige, Ostrof, and Brooke also mention that it is difficult to determine whether UML is consistent, since there are no precise design goals other than standardization of modeling concepts [PAI00]. As detailed in Chapter Two, it is questionable whether a consistency check for UML can be automated.

3.3. Design Objectives for the Visual Language for Generic Modeling

This section surveys the issues considered in the development of the proposed visual modeling language.

3.3.1. General Modeling Pattern

This research follows the general design pattern for modeling. In almost every kind of modeling environment, the analyst has a list of components and types of relations in a library structure. Modeling is achieved by simply selecting from list of components, instantiating them into the work area and setting allowable relations between them. Typically, these components and relations may have attributes associated with them that must be supplied by the analyst. For example, a digital circuitry design involves components like logic gates and signal generators. A single type of relation, cable, connects these components. Internally, these logic gates have propagation value that affects their timing behavior.

This thesis asserts that a UML-like object-oriented visual language can be used to define these kinds of component and relation libraries. If the language is designed properly, it should be possible to generate the modeling libraries for numerous problem domains. Then the analyst can design scenarios using these libraries. Consistency checking and graphical constraining might also be defined by the library and enforced by the development environment.

The visual language should be capable of defining any visual modeling environment. The specific intention of this study is to use the generic environment to provide a graphical user interface for simulation tools. Figure 26 presents the integration of the study into simulation process. First, libraries for the problem domain should be generated by means of the proposed visual language. Then a generic software tool interprets the components, relations and rules in the library and provides the scenario development environment. The libraries and the scenarios are saved as XML documents to enable sharing. Finally, the scenarios developed in the generic user interface environment are transformed into the textual format required by the simulation tool.

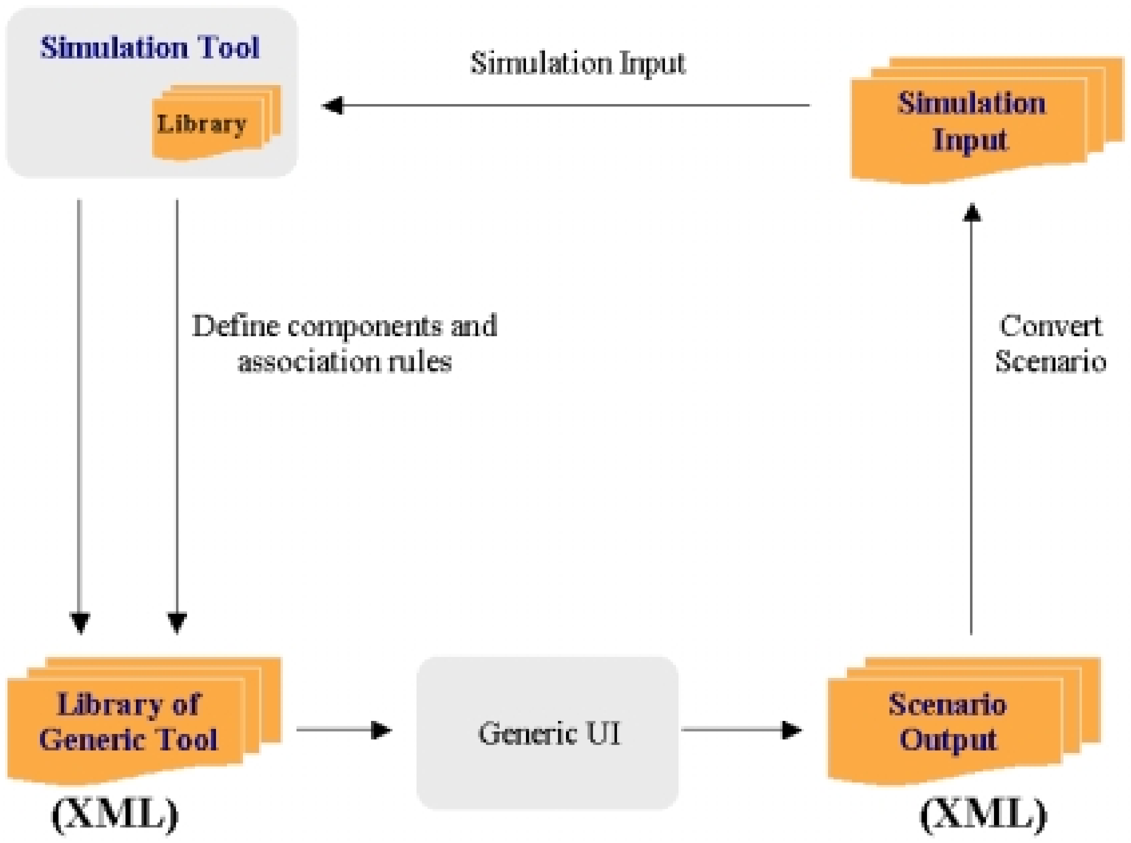


Figure 26. Simulation Modeling

How to achieve and implement the transformation of the scenarios into desired format depends solely on the purpose of the design, of which neither the VLGM language nor the tool is aware. As will be explained in Chapter Five, the parser within the implemented tool already has ability to load the design into memory, thus, an algorithm that can walk through the scenario design might be designed for the desired domain.

3.3.2. Critical Tasks and Implications

As detailed in Chapter Two, Paige, Ostrof, and Brooke, list four critical tasks required by the users of visual languages [PAI00]. The following list of critical tasks is determined to meet the requirements of the proposed visual language.

- Type definitions,

- Reuse and library development,
- Hierarchical scenario design,
- Consistency checking of the models,
- System documentation,
- Forwards and backwards translation, and
- Support for multi-domain modeling.

Type definitions involve structural definition of the abstractions used in the scenarios. These definitions should be organized as a library to promote reuse. The scenario description is the actual model where components are parameterized and the relations between them are set. The language should also support implementation of automatic consistency check mechanisms on scenarios. The model itself is considered documentation. By means of some note taking mechanisms, auto-report functionalities should be supported for software tools.

To support simulation modeling the software implementation of the language proposed should be able to translate scenarios developed in the language into the textual definitions to be used in the actual simulation tool. The language does not necessarily address the implementation of the translation, but it should be designed to allow that.

Different simulation application domains have different implementation requirements leading the design trade-offs between general and domain-specific approaches. Because of this, most industrial simulation tools on the market are domain-specific. However, this research is intended to be used in variety of simulation application domains.

3.3.3. Design Principles

Based on the study by Paige, Ostrof, and Brooke, this study derived the following design principles for the proposed visual meta-language [PAI00].

Simplicity: As mentioned previously, since the language is intended as a communication aid among humans, and between human and computer, simplicity is desired. It eases learning, drawing by hand and creating software tools to support the language. Simulation users may not be familiar to object-oriented modeling, thus, there should be no unnecessary complexity and ambiguity in the language. The visual aspects of structures other than its basic representation known as “secondary notation,” such as color variations and shading should not be used to convey extra meaning.

Users of simulation tools want to design in a notation that’s closer to the application domain, and do not want to be bothered with the low-level details of the simulation implementation [CLA99]. In order to meet these requirements of the users, the simulation tools should provide abstractions representing the real world entities.

Uniqueness: The language should provide only one good way to express every concept to prevent ambiguities and redundant overlapping expressions in the models. For instance, in a road map, existence of two different ways to represent a highway may cause ambiguities.

Consistency: Any feature in the language should address the purpose. In a road map, a line representing a gas pipeline does not relate with navigational purposes.

Seamlessness: Seamlessness involves using the same abstraction in the model and in the textual language, which helps forward translation. Since the modeling language proposed is intended to be used by analysts from different simulation application domains, the language should introduce ways to describe abstractions. This way the language and the simulation tool will use the same abstraction and forward generation will be guaranteed.

Scalability: The language should provide mechanisms to handle large-scale problems without compromising the usability for small-scale models. Encapsulation and grouping mechanisms should be used to provide better scalability.

Supportability: The language should be suitable to the development of computerized tools.

Reliability: To provide reliable models the language should provide support for consistency checks for the scenarios.

Space Economy: The models should take as little space on the screen or page as possible to avoid distractions caused by searching and browsing.

Reusability: Component-based development and reusability is crucial for simulation modeling. The language should support packaging mechanisms and library development.

3.4. Assumption on Behavioral Description

This study does not cover the behavioral description of the components used in the models. The language assumes the existence of behavioral descriptions of core components in the simulation tool, which is intended to run the scenario developed. The language only addresses the static modeling of the parameterized components connected on the working space.

3.5. Translation of Scenarios for Simulation Tool

To generate textual definitions as an input to a simulation tool, the abstractions defined and used in the scenarios should match the abstractions defined by the simulation tool that will run the scenario. The intention is that the simulation software vendor will provide definitions of the core components and the visual language tool will provide a working area to the analyst to select from these components, instantiate and parameterize them, set relations and create a scenario. From this, the tool will provide the scenario for the simulation software. The visual language supports the flexibility to define these abstractions but does not address how to achieve the translation to scenarios.

3.6. Success Criteria

Since the visual language design does not allow persuasive quantitative analysis, the language will be tested against the design principles with several case studies covering different modeling domains. The following evaluation criteria are developed based on the study by Frank van Harmelen, Manfred Aben, Fidel Ruiz, and Joke van de Plassche [HAR96]:

Expressiveness: Were certain things impossible to express? Were some things difficult to express?

Frequency of errors: What are the most common errors and what are the frequencies of those errors. Why those errors occur? How can they be avoided?

Redundancy: Was redundancy present in models? Is it possible to identify different types of redundancy? How can redundancies be avoided?

Locality of change: Do changes propagate through the models? If so, what are the causes, and can they be avoided?

Reusability: Do the models enable reusability?

Reliability: Do models enable consistency checks? If not, why and how can the inconsistencies be avoided?

Translatability: Are the models consistent and expressive enough foruse as an input to a simulation tool?

Compatibility: What is the distribution of results of the above criteria? Does the language favor any specific kind of simulation application domain?

3.7. Methodology Summary

The problems of using simulation tools with textual languages can be solved by means of graphical user interfaces. This research suggests the use of an object-oriented visual meta-language based on the general modeling pattern. The language is used to specify the components and relations of the modeling domain. The discussion in this chapter implies that the Unified Modeling Language cannot be used because of its insufficient support for instances and inconsistent behavioral diagrams. If the language is strictly designed to have a “transformability” property, a software tool can interpret the specifications and automatically provide the design environment. The language assumes the existence of behavioral descriptions of the components in the simulation tool, which is intended to run the scenario developed. Hence, behavioral descriptions are excluded. The design principles and success criteria for the language are also presented in this chapter.

4. FRAMEWORK AND LANGUAGE DEFINITION

This chapter presents the framework of the proposed solution and the definition of the Visual Language for Generic Modeling (VLGM). VLGM is intended to be simple enough to be implementable, but complex enough to represent any desired model. It can be extended to have more capabilities, and has the potential to become a powerful generic tool.

4.1. Framework

The system proposed involves three main steps where the semantics and structures may significantly differ. The first step is the specification of the components and relations required in the modeling environment, which is achieved through use of the visual language, VLGM. The second step involves using the automatically created modeling environment to design scenarios composed of instantiated components and relations. The third step is the invocation of the simulation to use the designed scenario.

VLGM consists of four data types and three relation types as shown in Figure 27 and Figure 28. Data types are special types of classes of object-oriented paradigm. They are abstraction mechanisms for a group of attributes representing data structures in the domain. As stated previously, the modeling process involves instantiation of components and relations in the work area. Therefore, VLGM, the first step in the framework, provides the structures to describe these components and relation types. The scenario environment, the second step, interprets VLGM specifications and provides instantiation mechanisms.

“Component Type” and “Relation Type” describe the components and relations in the domain. Typically, the components have connection points called “ports.” The relations connect ports. “Port Type” is used to describe ports in the domain. Once a port is specified, different components may have the same type of port or as many as required. The Relation Type specifies the type of an allowable connection between two Port Types. For example, in the digital circuitry design domain, components such as “and,” “or,” and “xor” gates have two input and one output port. The relation that connects these ports is simply a cable.

“Data Type” provides a grouping mechanism for attributes. If a data structure is shared between components, this data structure can be defined as a Data Type and the components may contain a copy of that structure. In the networking domain, as detailed in Chapter Six, a probability distribution is shared by both packet source and queue components, thus, it is defined as a Data Type.

Three types of relations are defined between the data types of VLGM as shown in Figure 28. Type extension is analogous to the inheritance relation of UML. In a type extension relation, the child type inherits the attributes of its ancestors. If the type extension occurs between component types, in addition to the attributes, the child type also inherits the ports of the ancestor. The composition relation is a strong coupling between types. All of the data types in VLGM can contain a Data Type and Component Types can contain Port Types. The Relation Type, as explained previously, specifies two Port Types that it can connect. In a VLGM diagram, this is represented as a Port Type Selection relation between a Relation Type and the Port Types to which it relates.

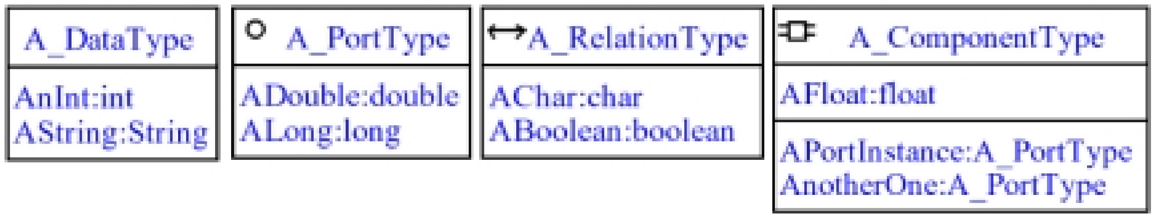


Figure 27. VLGM Data Types

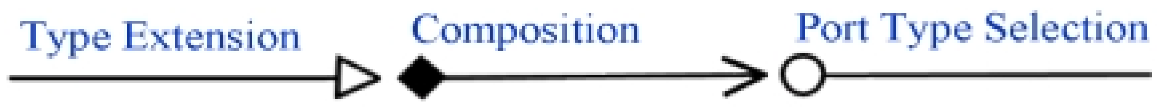


Figure 28. VLGM Relations

In the VLGM framework, two diagram types are used:

Library Diagrams: Before users can develop scenarios they must have libraries of pre-built core components. Library diagrams are the formal specification of these components and their relationships. This is where VLGM, the first step in the framework, is applied to specify the modeling environment. Definitions in these diagrams are considered to be a library. Dependencies between libraries may occur in large projects where some libraries may import and use others. In essence, libraries are grouping mechanisms for the structures defined inside.

Scenario Diagrams: This is the application of the second step of the framework. The components and relations from selected libraries are interpreted and the modeling environment is provided according to their specifications. Typically, a browser will help locate the types of components and relations available. The user will choose components from a list and instantiate them in the work area. Similarly they will select relation types and use them to connect relevant ports. These scenarios can be designated as components to be used in higher-level designs. For example, a library containing logic gates can be used to create a two-bit adder scenario, and by defining this scenario as a component, four two-bit adders can be used to design a four-bit adder and so on. This capability of the framework provides scalability.

4.2. Visual Meta-Language for Generic Modeling (VLGM)

Languages are generally described by two different aspects: syntax and semantics. Syntax refer to the rules for combining textual or graphical symbols to create valid sentences in the language and does not deal with the meaning of the sentence. Semantics, on the other hand, refers to the meaning of the valid sentences. Since VLGM is relatively small, the description provided in this section presents the syntax of the language, with only passing discussion of some semantic concepts.

4.2.1. Primitives

Primitive types are used as parts of complex structures of the language, namely Data Type, Port Type, Relation Type and Component Type. Primitive types are named and have data types. When displayed, the colon symbol is put between name and type. Table 6 demonstrates the primitive types, their applicable properties, and example usages.

Table 6. Primitive Types

Type	Required	Explanation	Range	Unit	Decimal	Example
Float	√	√	√	√	√	x:float
Double	√	√	√	√	√	y:double
Integer	√	√	√	√		age:int
Long	√	√	√	√		index:long
Char	√	√	√			selection:char
String	√	√	√			name:string
Boolean	√	√				isMale:boolean

The following properties of each type are used to enforce consistency of the values during parameterization:

Required: Indicates a boolean value (true/false). When set to true, parameterization of the type is mandatory.

Explanation: String value that explains the type.

Range: Specifies the value range of the type. A series of ranges can be defined for a primitive type such as ages between 12..35 and 45..55. Ranges specify alphabetical order for char types and list of selectable values for string types.

Unit: A string value applicable only for numeric types that explains the unit of the value. Used to resolve possible ambiguities such as type named “time” which may hold seconds or nanoseconds.

Decimals: An integer value standing for number of decimal digits, which is applicable for float and double types only.

4.2.1.1. Numeric Types

Following table shows the values that numeric types can hold.

Table 7. Numeric Types

Type	Size	Description (smallest and largest positive values)
Integer	32 bits	signed integer (-2.14e+9 --> 2.14e+9)
Long	64 bits	long signed integer (-9.22e+18 --> 9.22e+18)
Float	32 bits	floating-point number (1.402e-45 --> 3.402e+38)
Double	64 bits	double precision floating-point (4.94e-324 --> 1.79e+308)

4.2.1.2. Boolean Type

Boolean is a special data type that may hold the value “true” or “false.”

4.2.1.3. Char Type

The language uses the data type “char” to store a single character. The parameterization of char can be constrained by means of a predefined set of range values. Ranges specify the alphabetical order for char types. For example, four answer choices of a question can be specified by “a”..“d” and “A”.. “D.”

4.2.1.4. String Type

A string is a sequence of characters. Like char data type, parameterization of strings can be limited by means of a predefined set of range values. Ranges specify list of selectable string values for associated string type. For instance, the routing algorithm for a router can be specified with the range of “EIGRP,” “RGRP,” and “RIP.” The user selects one of the string values from the list.

4.2.1.5. Arrays

Array types can be defined to hold multiple elements of the same type. It is applicable to all primitive and complex types. There is no constraint on the number of dimensions an array may have. Each dimension is associated with minimum and maximum length values, which are used to force the user to instantiate each dimension between these values. Example:

MyIntegerArray[1,2][1,4]:integer

The array defined above is a two-dimensional array with specified minimum and maximum dimension values. This means the user may instantiate this array as 1x1, 1x2, 1x3, 1x4, 2x1, 2x2, 2x3, or 2x4 matrices.

4.2.2. Complex Data Structures

The complex data structures of the language are actually specialized classes in the object-oriented paradigm. Unlike classes in classic object-orientation, VLGM types do not have behaviors. They are basically a collection of primitive types or other complex types. Each complex type in the language has semantics and properties designed to achieve their functionality.

4.2.2.1. Data Type

Data Type is the basic complex structure that groups attributes. Data types might be *abstract*, which means this data type cannot be instantiated directly and will probably be used as an ancestor to other data types in a type extension relation. A Data Type instance is represented as a box with its name on the first line. Abstract data types are marked with a line connecting top and right sides on the corner of the box as a triangle. The box also contains the attributes of the data type as shown in Figure 29.

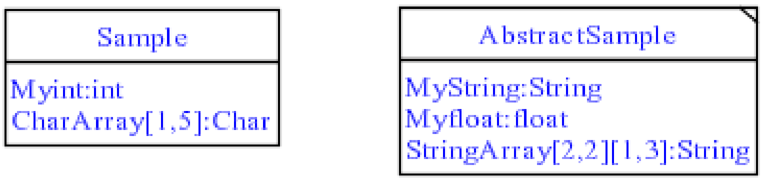


Figure 29. Data Type

4.2.2.2. Port Type

Port Types contain a set of attributes and are represented with a box marked with a circle in the top-left corner as shown in Figure 30. Port Types are used to define the connection points of Component Types. For example, if the designer is dealing with digital circuit design, they may need to define a common port type to be used with elements such as gates, microprocessors or decoders. These elements will have different numbers of the same type of connection point.

For visualization purposes, Port Types are associated with a port symbol property, which is used to show the port of the component during scenario development. Figure 31 shows the list of symbols.

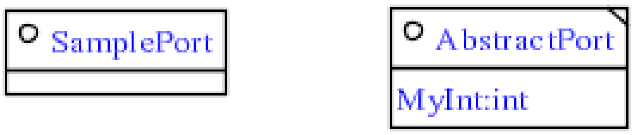


Figure 30. Port Type

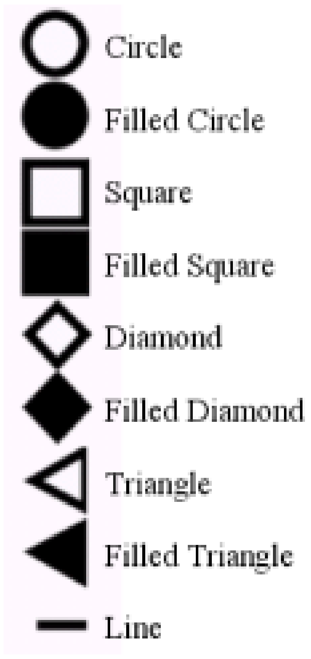


Figure 31. Port Symbols

4.2.2.3. Relation Type

Relation Types contain a set of attributes and are represented with a box marked with an arrowed line in the top-left corner as shown in Figure 32. Relation Types are used to define the relations that can connect ports of components together during scenario development. The Relation Type has two special properties that specify the possible incoming and outgoing ports to which the relation can connect. For example, in a digital circuitry design, the signal ports of the logic gates can be connected with a “cable” relation, which is specified to connect single bit signal ports. In the same system, bus ports cannot involve in a “cable” relation.

For visualization purposes, Relation Types are associated with head and tail arrow symbols and line properties, which specify the appearance of the relation during scenario development. Figure 33 shows the list of arrow symbols and line types.

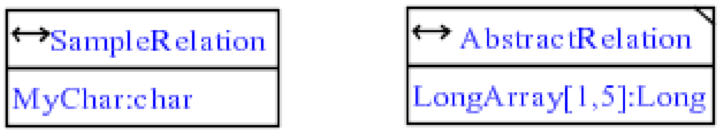


Figure 32. Relation Type

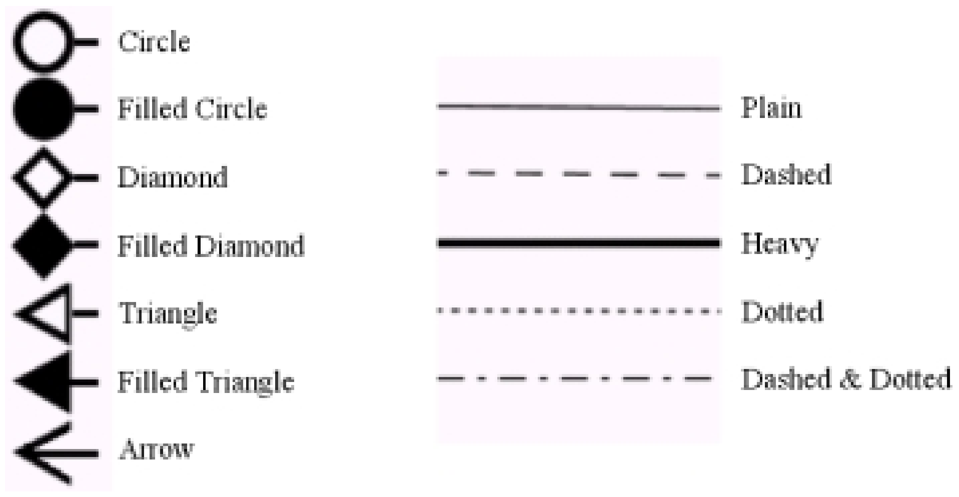


Figure 33. Arrow and Line Types

4.2.2.4. Component Type

Component Types contain a set of attributes and ports and are represented with a box marked with a box in the top-left corner as shown in Figure 34. In the scenario development phase, Component Types are presented either as an image (if associated with an image file) or a generic box with port symbols. Components are instantiated directly and connected to each other through their ports. If an image is assigned to a component type, the coordinates of the ports of the component are defined relative to the top-left corner of the image. Otherwise, the location of the port is selected as one of top, left, right or bottom sides of the box. As shown in Figure 34, the list of ports and list of attributes that a component contains are separated with a line.

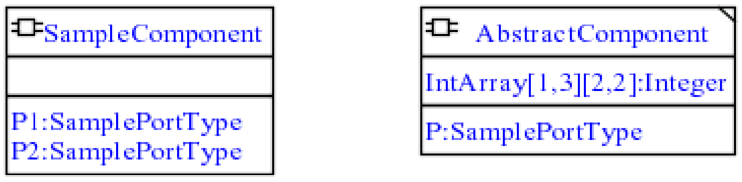


Figure 34. Component Type

During Port Type assignment to Component Types, cardinality rules for connections through these ports must be defined separately for each port. This design provides the flexibility needed to use the same types of ports with different cardinality restrictions. Cardinality rules specify the number of connections allowed through the port. Each port of a Component Type has FROM and TO cardinalities. The FROM cardinality specifies the number of allowed connections originating from that port and the TO cardinality specifies the number of allowed connections incoming to the port. As shown in the following examples, the cardinality specification notation uses “..” between minimum and maximum allowed connections and “n” to mean “any.” These cardinality constraints are enforced in the scenario design.

0..1	: Zero or one
n	: Any (can be zero)
1..n	: Between 1 and “n” (can’t be zero)

3	: exactly 3
2..5	: between 2 and 5

4.2.3. Relations

Three types of relations are defined between the complex types explained previously: Type Extension, Composition, and Port Type Selection.

4.2.3.1. Type Extension

Type extension is similar to the inheritance relation in object-orientation and is represented as a plain line with an empty triangle in the head pointing towards the ancestor type. All four complex types, Data Type, Port Type, Relation Type, and Component Type, can extend either Data Types or the same type as themselves. Multiple-extension, which means extending more than one type at the same time, is allowed. Since Data, Port, and Relation Types contain only attributes, type extension involves inheritance of these attributes by the child type. In type extension between Component Types, in addition to attributes, ports of the ancestor are also inherited by the child Component Type. Recursive type extension is not allowed.

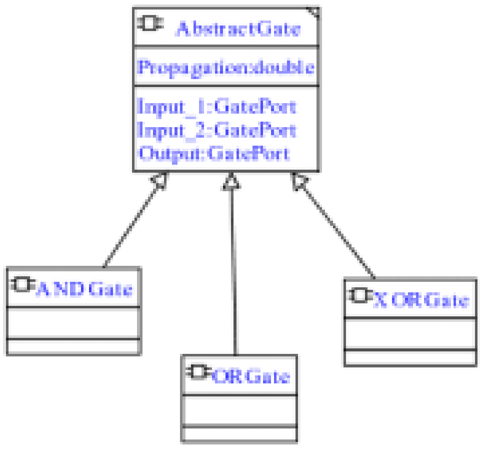


Figure 35. Type Extension

Unlike inheritance in object-orientation, attributes with the same name are not overridden in the child type. Instead, two different attributes with the same name are differentiated with "." notation (involving the library name) in the context of the child type. For example, the attribute named "Propagation" inherited by the "ANDGate" component type in Figure 35 is accessed with the long name "LogicElements.AbstractGate.Propagation" in the context of the "ANDGate." A detailed discussion of accessibility through the type hierarchy is provided in Section 4.2.4.

4.2.3.2. Composition

Composition in VLGM is analogous to composition as defined in UML. This relation can be defined from Components, Ports, Relations, and Data Types to other Data Types, which means that all of these complex structures can carry a Data Type as an attribute. It can also be defined from Component Type to Port Type.

Composition is different from the aggregation relation in UML, where the relation is actually a pointer to a data structure. The difference between aggregation and composition in UML is that in composition, when the owner is deleted, the aggregate is not allowed to exist. However, in aggregation the aggregate can still exist even if its owner does not. For example, in the case of composition, if a car is destroyed that means its engine, transmission, and other parts are also destroyed. In case of an aggregation, if a department is closed its employee can be transferred to other departments of the company (the employee is not destroyed!). For the purposes of VLGM, the aggregation relation is not relevant, since Data Types cannot be instantiated directly if they are not owned or inherited by Components, Ports, or Relation Types.

Composition between Component Types and Port Types are considered to be one-to-one relations, since each port of Component, even if they are the same type of port, carries its own semantics. Owner type side cardinality of a composition relation involving a Data Type is always considered to be one, and the other side may either be one or specified by array dimensions. This cardinality restriction on the composition is required for forward transformability and consistency of the language. Figure 36 shows an example of composition between components, ports and data types.

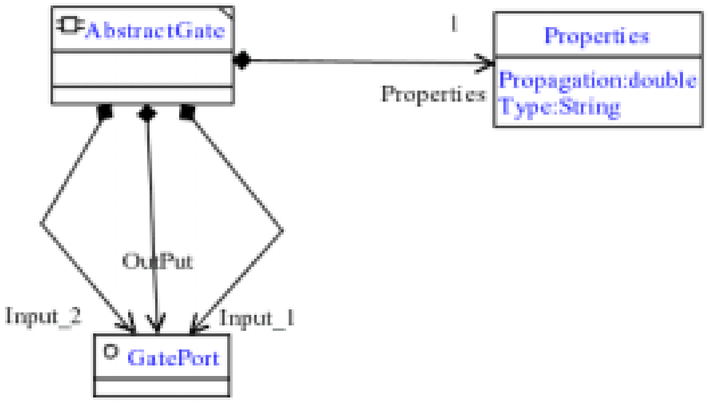


Figure 36. Composition

4.2.3.3. Port Type Selection

The Relation Type, as defined previously, specifies the type of an allowable connection between two Port Types. In a VLGM diagram, this is shown as a Port Type Selection relation between a Relation Type and the Port Types to which it relates. A Relation Type's FROM and TO Port Types are shown with this type of relation in Figure 37. The figure indicates that a Relation Type called "Line" can connect to the incoming and outgoing (TO and FROM) connections on Port Type "GatePort."

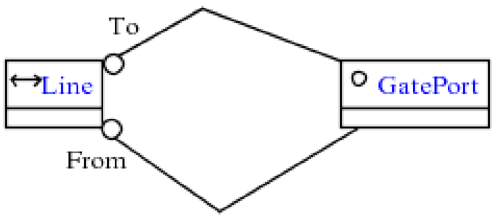


Figure 37. Port Type Selection

4.2.4. Accessibility Through The Type Hierarchy

Assume a type hierarchy as shown in Figure 38. Four Data Types, each located in different library, are defined as extending one another. Two composition relations occur involving Type_4, Type_3, and Type_1. Type_4 contains instances of Type_1 and Type_3 named T_1 and T_3 (role names in the composition relation) respectively. This diagram raises questions about how to access inherited attributes and whether attribute names can be overloaded (defined the same in more than one class).

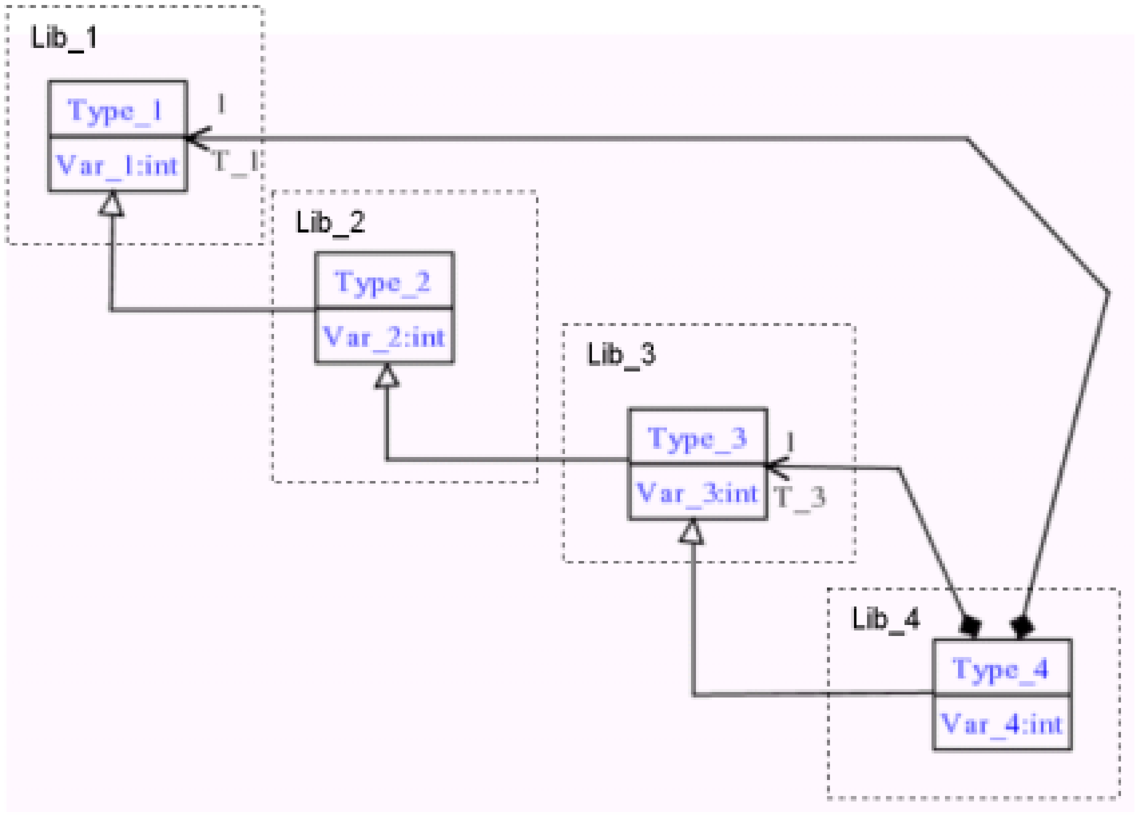


Figure 38. Accessibility Through Type Hierarchy

Accessibility in the hierarchy can be approached in two ways: using short or long naming schemas. In a short naming schema, every variable is accessed locally with the direct name. This kind of implementation does not allow inherited names to be used again. Another problem that might occur is that if Var_1 is directly accessed in Type_4 (see Figure 38), which Var_1 actually has been accessed is unknown: Var_1 related to T_1 instance, Var_1 related to T_3 instance (through extension) or the Var_1 inherited by Type_4 itself (through extension). A long naming schema, which avoids these problems, is applied in VLGM. In this schema, inherited attributes are accessed with their type names and type's library name with a "." notation.

Lib_3.Type_3.Lib_2.Type_2.Lib_1.Type_1.Var_1	(Inherited)
T_1.Var_1	(Composition)
T_3.Lib_2.Type_2.Lib_1.Type_1.Var_1	(Composition-Inherited)

4.2.5. Case Definitions

Case Definitions are used to allow or disallow the parameterization of a group of primitive or complex type attributes based on the values of other attributes of the complex type instances. Using the long name dot-notation, attributes of complex types may be referenced as a condition in case definitions. If the condition occurs, the attributes associated with the case definitions get activated and vice versa. This property of the language is quite useful if there are interdependencies between the attributes of the complex type.

A sample case definition is provided in Figure 39. Since the long name reference to the condition attribute is generally large, the conditions associated with the case groups are not designed to be visible in the type definition. In this example, the intention is to define probabilistic distributions. The string named "Type" actually has the following set of range values: "Normal" and "Binomial." A normal distribution requires "Mean" and "Variance," but a binomial distribution requires "p" (probability) and "n" (number of occurrences) values to be parameterized. So, when a normal distribution is selected, "variance" and "mean" values are required and when a binomial distribution is selected, "p" and "n" values are required.

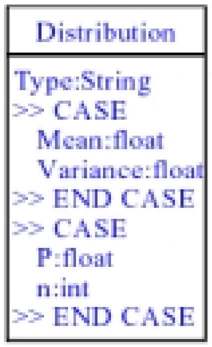


Figure 39. Case Definitions

4.3. Scenario Environment

This section discusses the tasks and important issues in the scenario development environment.

4.3.1. Generic Component Representation

If a Component Type is not associated with an image file, a generic box representation is used. Ports are located on the sides of the box according to their place and symbol specifications. Port names are also written next to them. In the center of the box, a component name given by the user and component type name (smaller and in parenthesis) are located as shown in Figure 40.

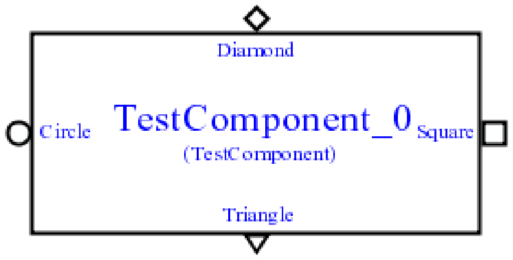


Figure 40. Generic Component Representation

4.3.2. Consistency Checking and Graphical Constraining

By design, the scenario development environment has the ability to do automated consistency checks. Table 8 lists the graphical constraining and consistency warnings that are implementable.

Table 8. Consistency Checks and Constraints

Where?	What?	How?
Attributes	Mandatory parameterization	IsRequired property
Attributes	Range enforcements	Range sets
Arrays	Array dimension enforcements	Minimum and maximum properties of each dimension
Relations	Limited to connect selected Port Types	FROM and TO port type selection
Ports	Connection cardinality limitations	Port's FROM and TO cardinality properties

4.3.3. Scenario Environment Scalability

Every attribute value and port of the component in the design can be marked, (or "mapped"), to be parameterized in a higher level of the hierarchical design. By mapping attributes and ports of the components in the scenario, the scenario itself can become a component. The new component will carry the mapped attributes and ports in it. This kind of hierarchical design provides a grouping mechanism, thus, giving scalability to the scenario development environment. An example of this kind of scaling is given in Chapter Six under the digital logic case study.

4.4. Summary

In this chapter, a detailed discussion of first two steps of the proposed framework is provided. The framework consists of three steps: the specification of the modeling domain, the scenario design, and the transformation. VLGM is introduced with its primitive types, complex types and relationships. A discussion of VLGM concepts is also provided. The accessibility of attributes through the type hierarchy is explained with an example. The implementable consistency checks and graphical constrains are listed. Finally, the process named “mapping” is introduced as a tool providing scalability to the scenario development environment.

5. IMPLEMENTATION

In order to demonstrate the capabilities of VLGM, a demonstration environment was created. This chapter discusses the design of this environment.

5.1. Design Architecture

This section provides guidelines to understand the design of the implementation. As stated previously, VLGM is intended to be simple enough to easily implement, but flexible enough to model any system. The belief is that the implementation architecture discussed in this section can be a basis for a larger-scale implementation.

The program has two major functions. One is used to develop VLGM designs that specify components and their relations. The other is used to develop scenarios from defined components and relations. Combined, these two areas provide the necessary environment to create simulation scenario models.

5.1.1. Packages

The program is divided into three packages according to functionality as shown in Figure 41. The parser package is responsible for classes of language abstractions, libraries that contain and maintain these abstractions, a library handler to maintain multiple libraries, and a parser for saving and retrieving of designs. The elements package performs visualization of the library elements, the forms that maintains their properties, and consistency of design. The GUI package organizes menus, toolbars, and manages user interaction in a hierarchical manner.

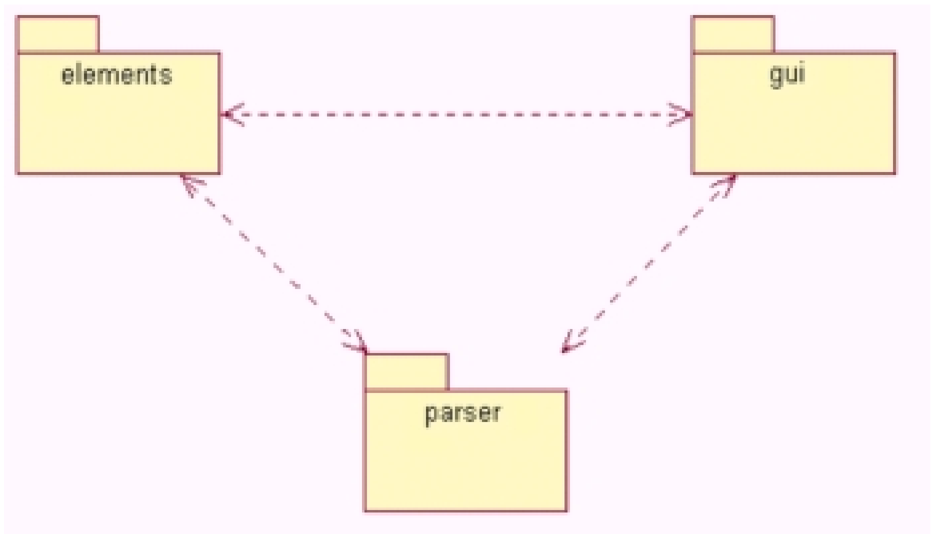


Figure 41. Packages

5.1.1.1. The Parser Package

The class diagram in Figure 42 is part of the parser package and forms the basis of the abstractions in the language. All design elements, with some exceptions, in VLGM inherit from the abstract class named "PrimitiveType." It organizes common attributes and methods for primitive and complex types defined in the language. Ranges for numeric types and char types are implemented as separate classes, allowing multiple range definitions. Ranges for string types are implemented as Java Vectors.

Complex types in VLGM compose groups of primitive types. The "PrimitiveCollection" abstract class, inheriting from "PrimitiveType," is designed to hold a group of primitives and handle common methods over collections of attributes. "ArrayType" and "CaseDefinitions" also contain a group of primitives, and thus they inherit from "PrimitiveCollection."

Since arrays have no limits on dimensions, a separate class for arrays named "ArrayDimensions" is defined. "ArrayType" also contains a "sample" primitive. This sample holds the type that is defined as an array. It allows arrays of complex types. "CaseDefinitions" may have a group of cases where each case is linked to an attribute, which may be a complex type.

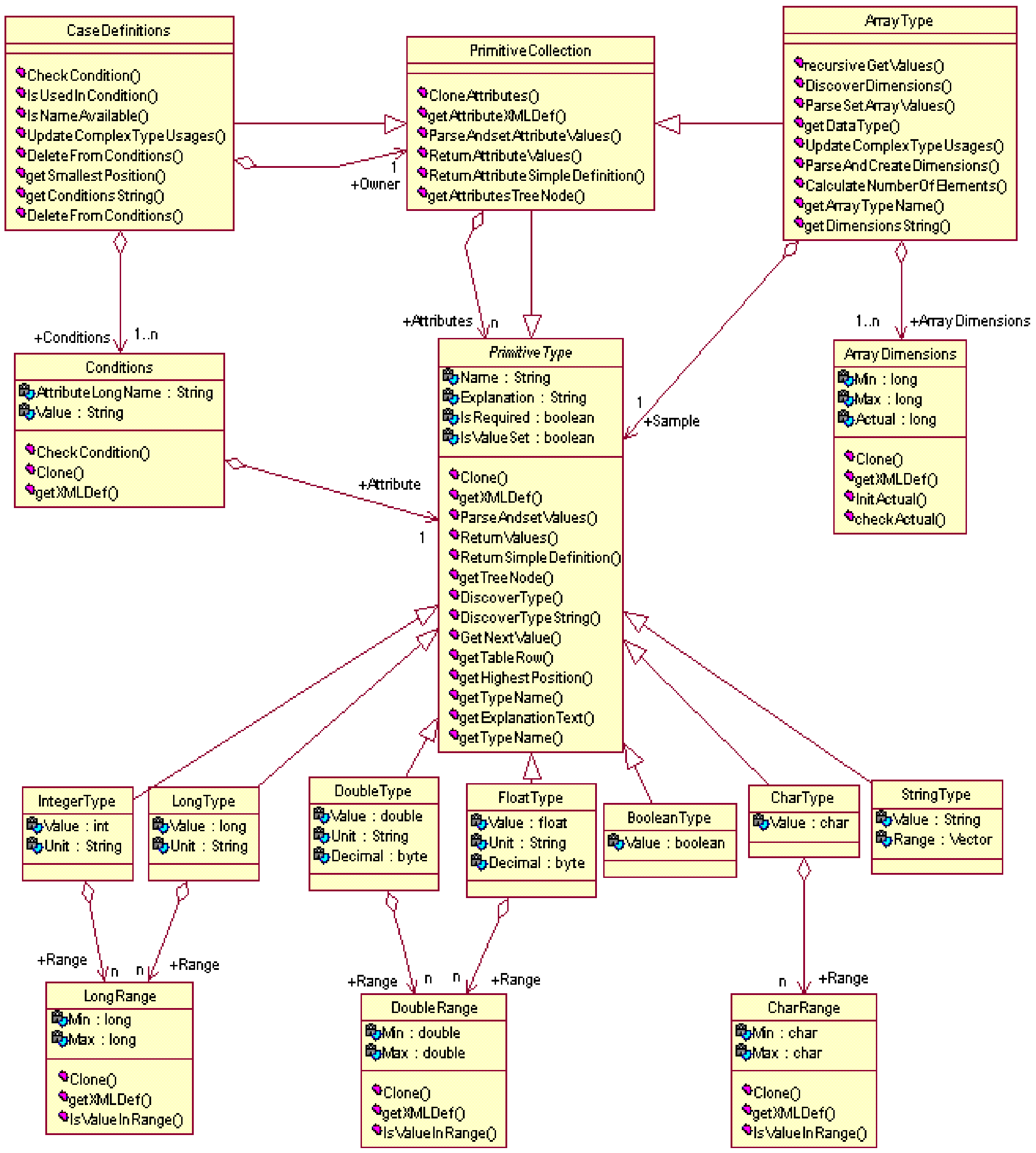


Figure 42. Class Diagram of Primitive Types

Figure 43 shows the complex types defined in the same package. “ComplexDataType” inherits from “PrimitiveCollection” and type extensions are implemented with the

"Extensions" class that has a link back to "ComplexDataType." Port, Component, and Relation types are implemented in separate classes that inherit from "ComplexDataType." In addition to the attributes, Components have a collection of ports. "RelationType" is linked to two ports that it connects and every port has a list of incoming and outgoing relations involved.

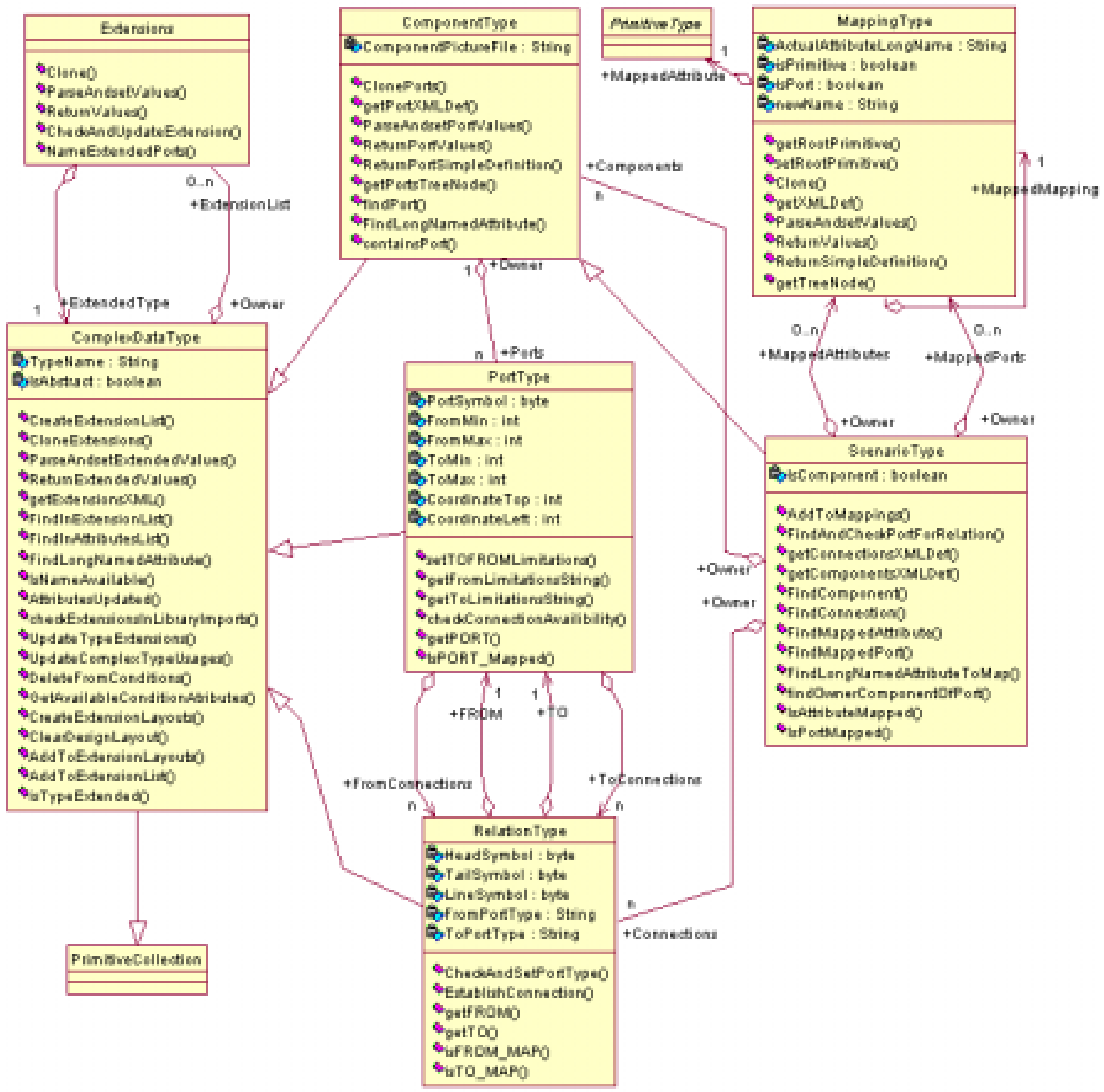


Figure 43. Class Diagram of Complex Types

A scenario is composed of instantiated components and relations. As stated previously, a scenario might be defined as a component and used in a hierarchical manner, therefore, it inherits from “ComponentType” and contains a list of attributes and ports that are mapped to be parameterized or connected at the higher level. When a scenario is used as a component in another scenario, its mapped attributes or ports might be mapped again to a higher level.

As shown in Figure 44, a library is composed of type definitions and a singleton class named “LibraryHandler” which manages multiple libraries. Both of these classes work with the “LibraryParser” that implements the required interface and uses the Xerces XML Parser.

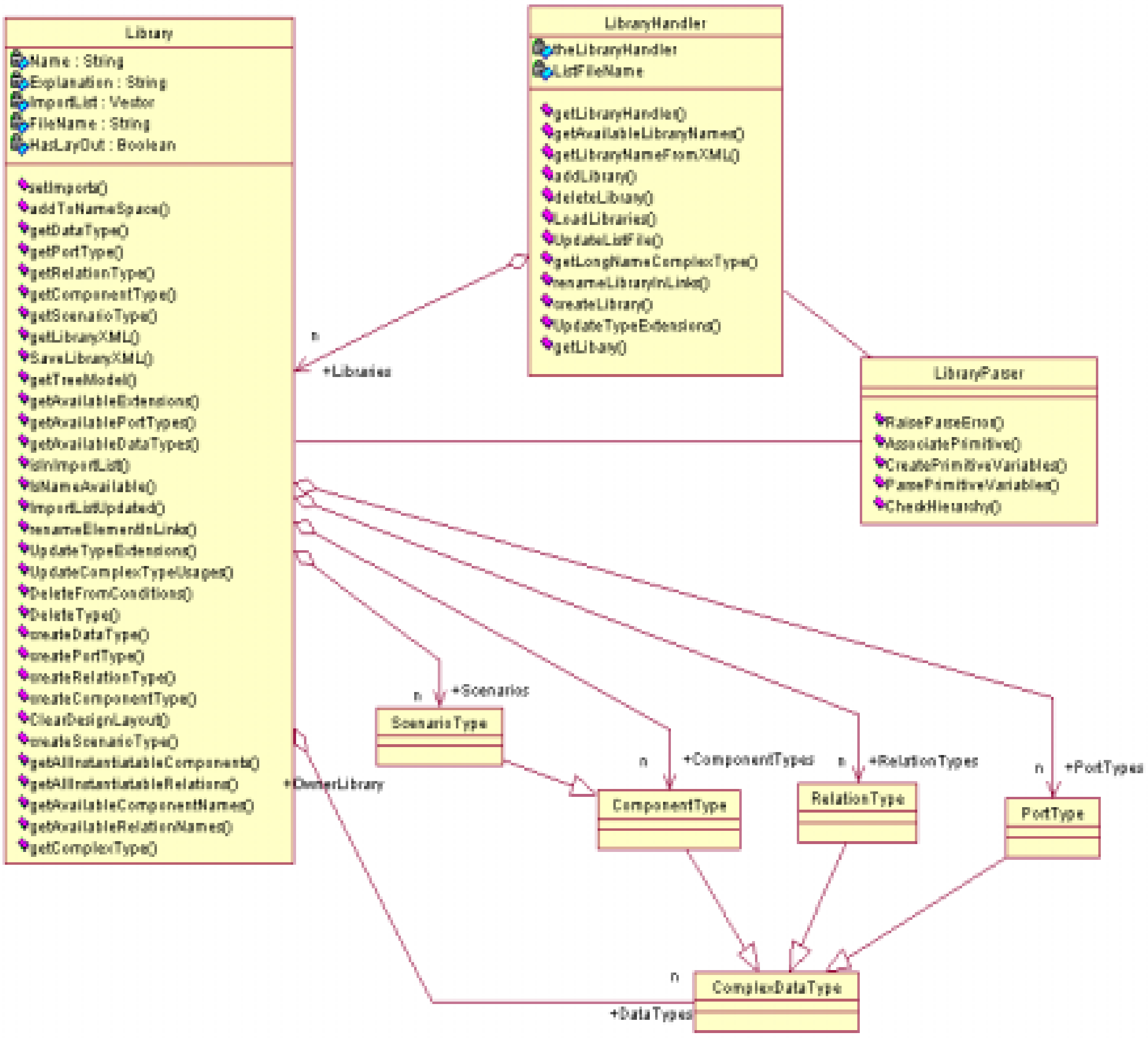


Figure 44. Class Diagram of Library Management

The type hierarchy shown in Figure 42, Figure 43, and Figure 44 lack design layout information. The program should be able to save and retrieve coordinates associated with elements and relations as they occur in the workspace. A design is generally composed of nodes and relations. In order to handle layouts, two classes are defined (see Figure 45), one for nodes and one for relations. "NodeDesignLayout" keeps track of the top-left corner and other special visualization requirements of a node. "RelationDesignLayout" is able handle coordinates of multiple points.

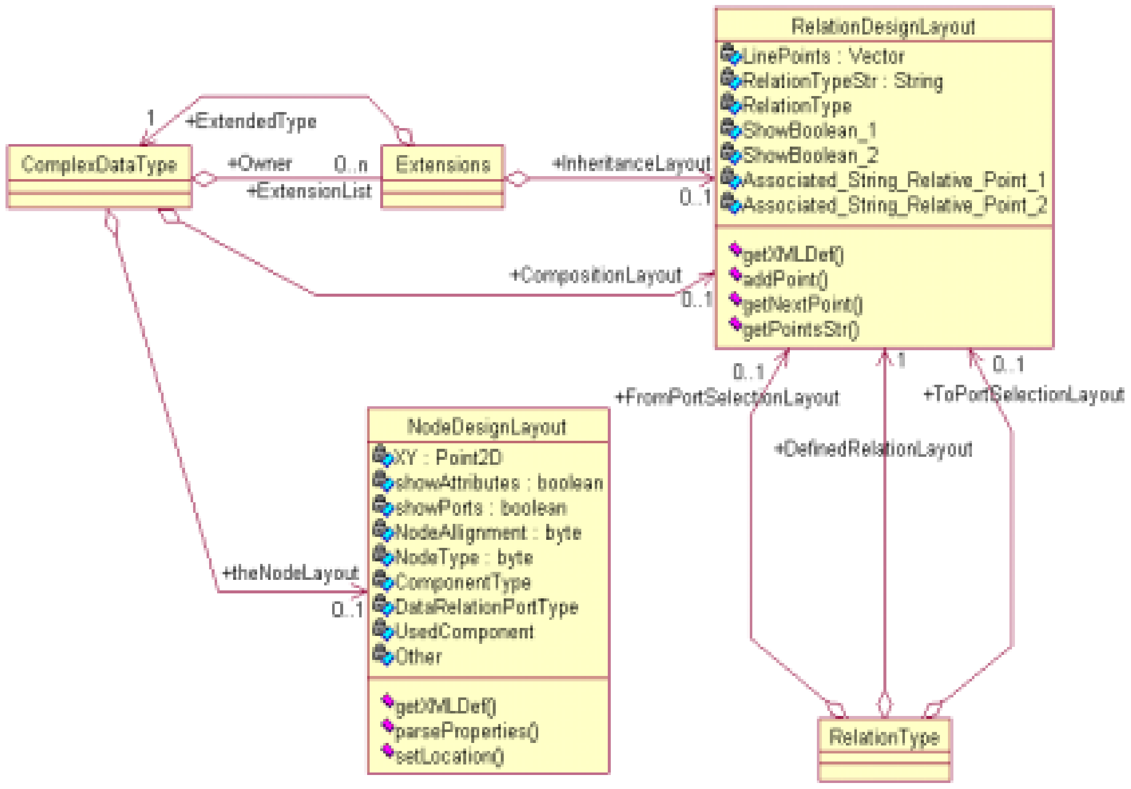


Figure 45. Class Diagram of Design Layouts

5.1.1.2. The Elements Package

The elements package is responsible for visualization of the design elements and applying user actions to the design. "GUI_Abstract_Structure" is an abstract class that organizes nodes and relations of a graph. It contains a library, where the node and relation representations are maintained as a type hierarchy. As stated earlier, the program has two similar functional areas. As shown in Figure 46, "GUI_Library" (for VLGM design) and

“GUI_Scenario” classes (for scenario design) inherit from “GUI_Abstract_Structure.” This design decision saves considerable effort in coding.

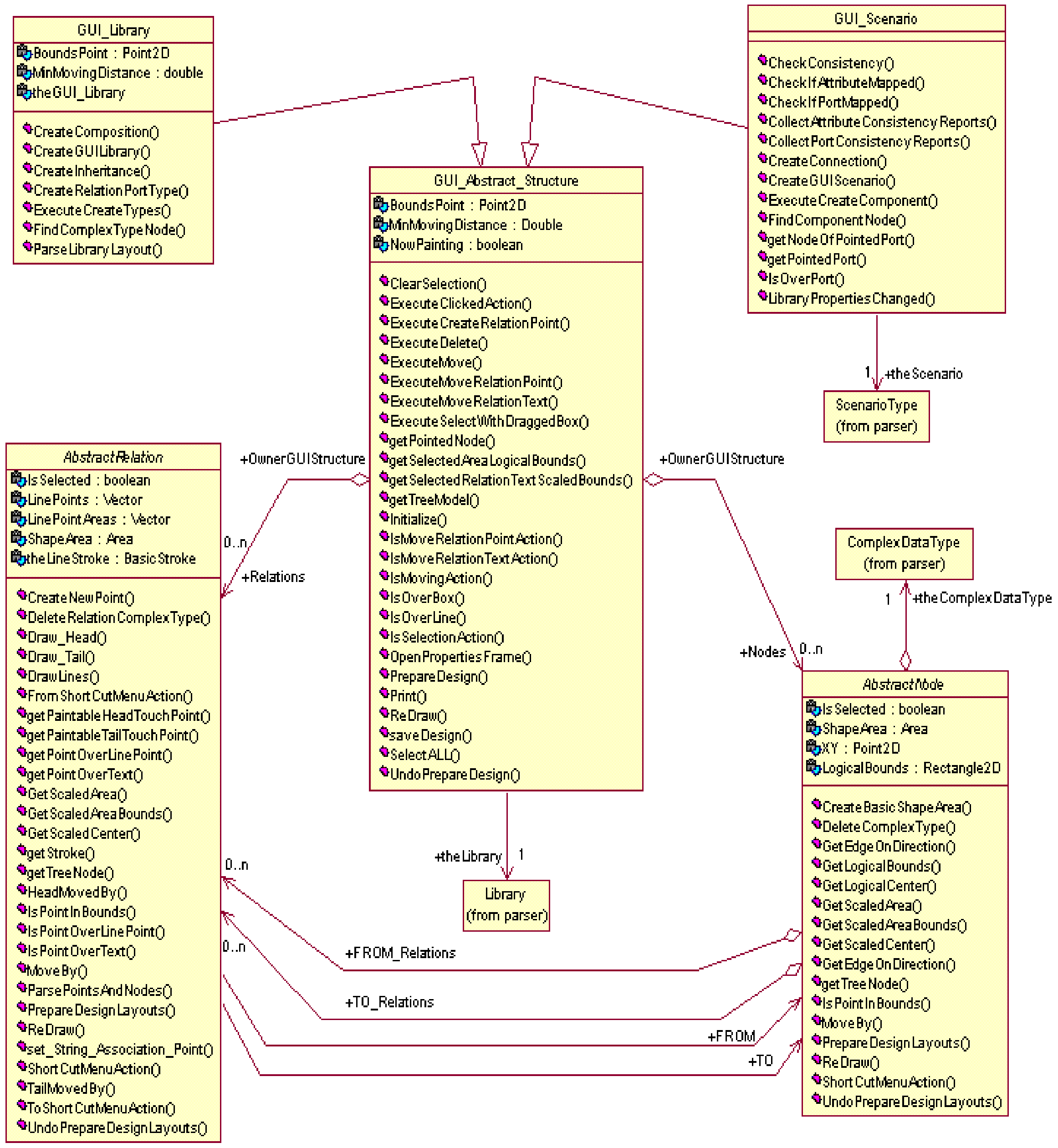


Figure 46. Class Diagram of Visualized Design

Class diagrams in Figure 47 and Figure 48 show the hierarchical design for each possible node and relation. As seen in Figure 47, an abstract class named “AbstractLibraryNode” inheriting from “AbstractNode” forms the basis of nodes for components, ports, relations and data types in the VLGM design. In the scenario development environment, the nodes are instances of user specified components, thus a single class named "Node_UserDefined" manages visualization of nodes. To implement the visualization of the ports of the components "Associated_Port" class is used.

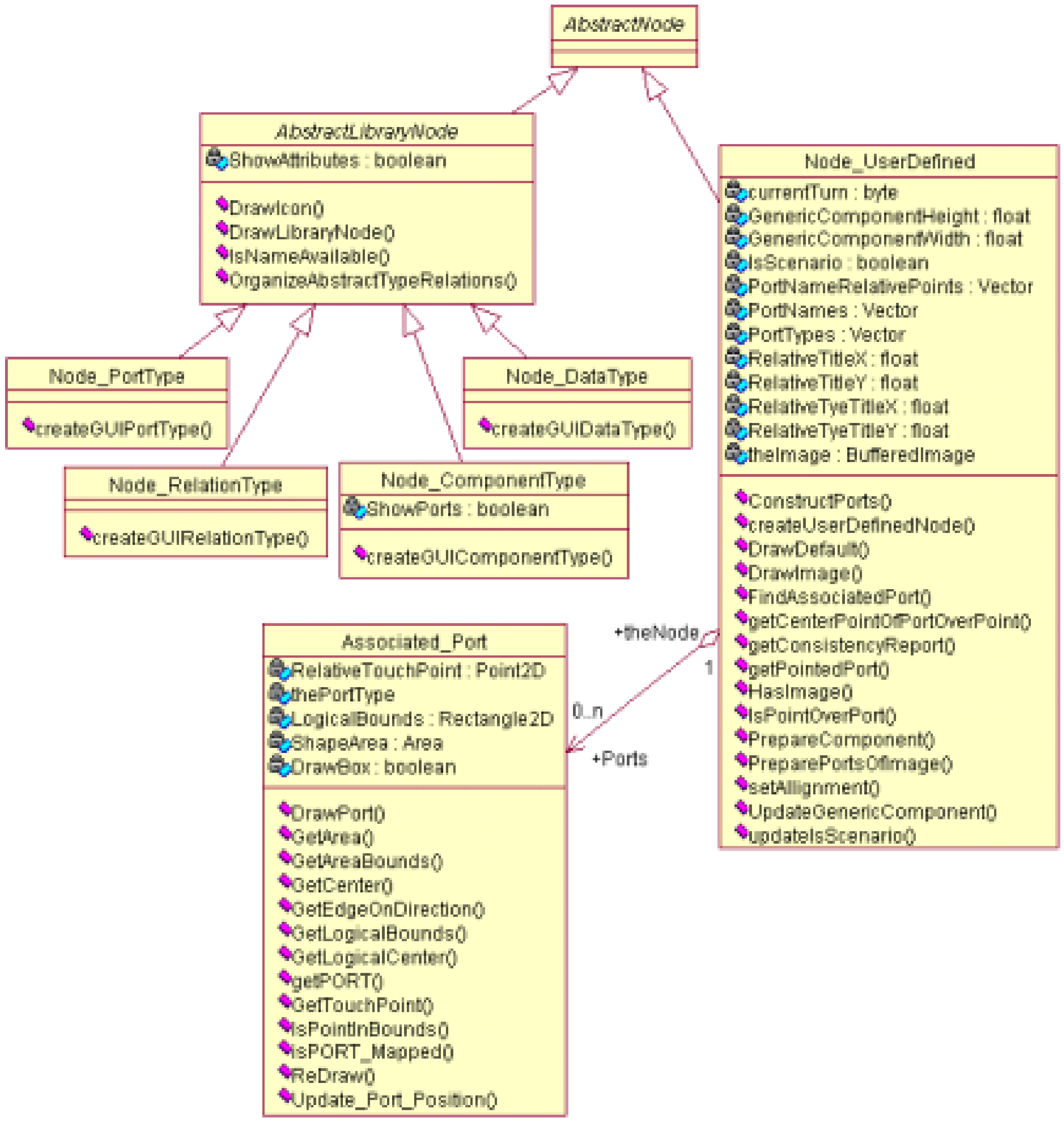


Figure 47. Class Diagram of Nodes

As shown in Figure 48, an abstract class named "AbstractLibraryRelation" inheriting from "AbstractRelation" forms the basis of relationships in the VLGM design. "Relation_UserDefined" manages visualization of relations in the scenario development environment. The composition and port type selection relations have strings associated to the head or tail of the visual representations. The visualization of these strings is managed by the class named “Associated_String.”

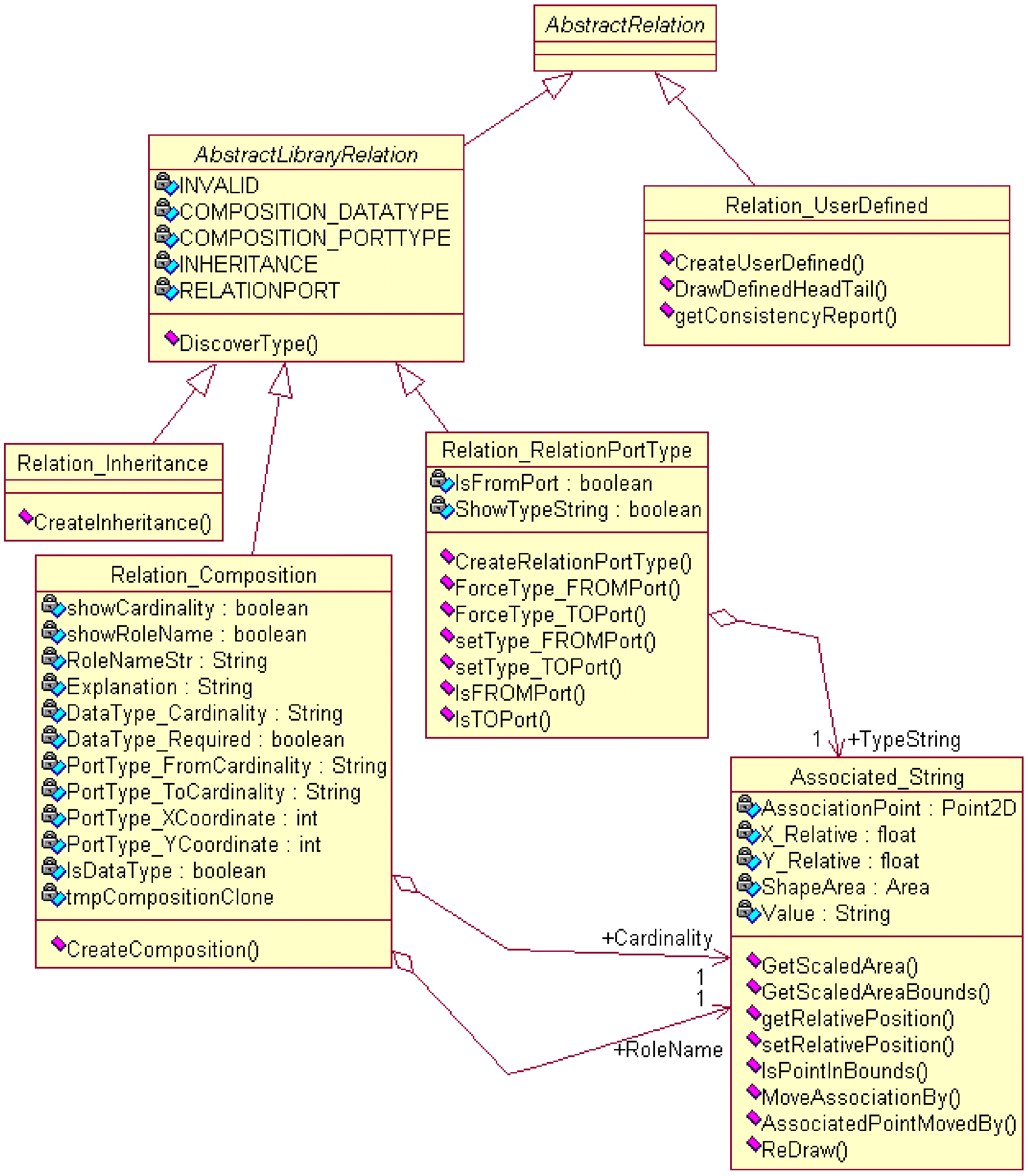


Figure 48. Class Diagram of Relations

5.1.1.3. The GUI Package

The GUI Package consists of main user interface elements such as the main window, tool bar, menu bar, and work area. Because of the two design areas mentioned earlier, the “AbstractWorkArea” abstract class is defined to contain the same functionality for both design panels. As shown in Figure 49, “LibraryWorkPANEL” (for VLGM design) and “ScenarioWorkPANEL” classes (for scenario design) inherit from “AbstractWorkArea.” They take mouse actions and implement design interaction by working with the “GUI_Library” and “GUI_Scenario” classes in the elements package.

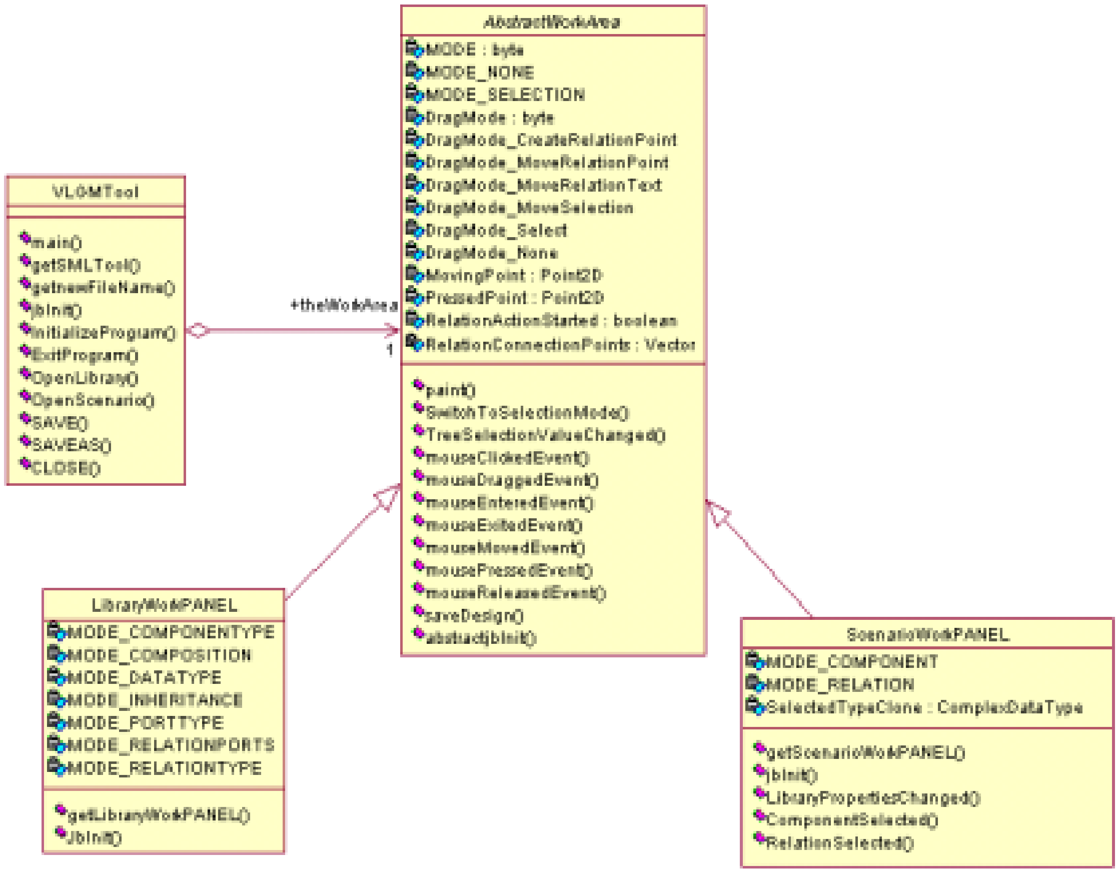


Figure 49. Class Diagram of Main Design Area

Figure 50 shows the relations between the remaining GUI elements of the program. “AbstractWorkArea” contains a “WorkPanel” which, in turn, contains a “WorkSpace.” The design elements are drawn by the “WorkSpace” class, which extends Java’s “JComponent” class. “WorkSpace” overrides JComponent's “paintComponent()” method, which is automatically called by the JVM in case of a refresh in the GUI, and manually triggered by the program as a result of user design actions. “WorkPanel” implements zooming and scrolling of the user’s design by means of the “TransformationManagement” class. Printing is also implemented in "WorkSpace" class.

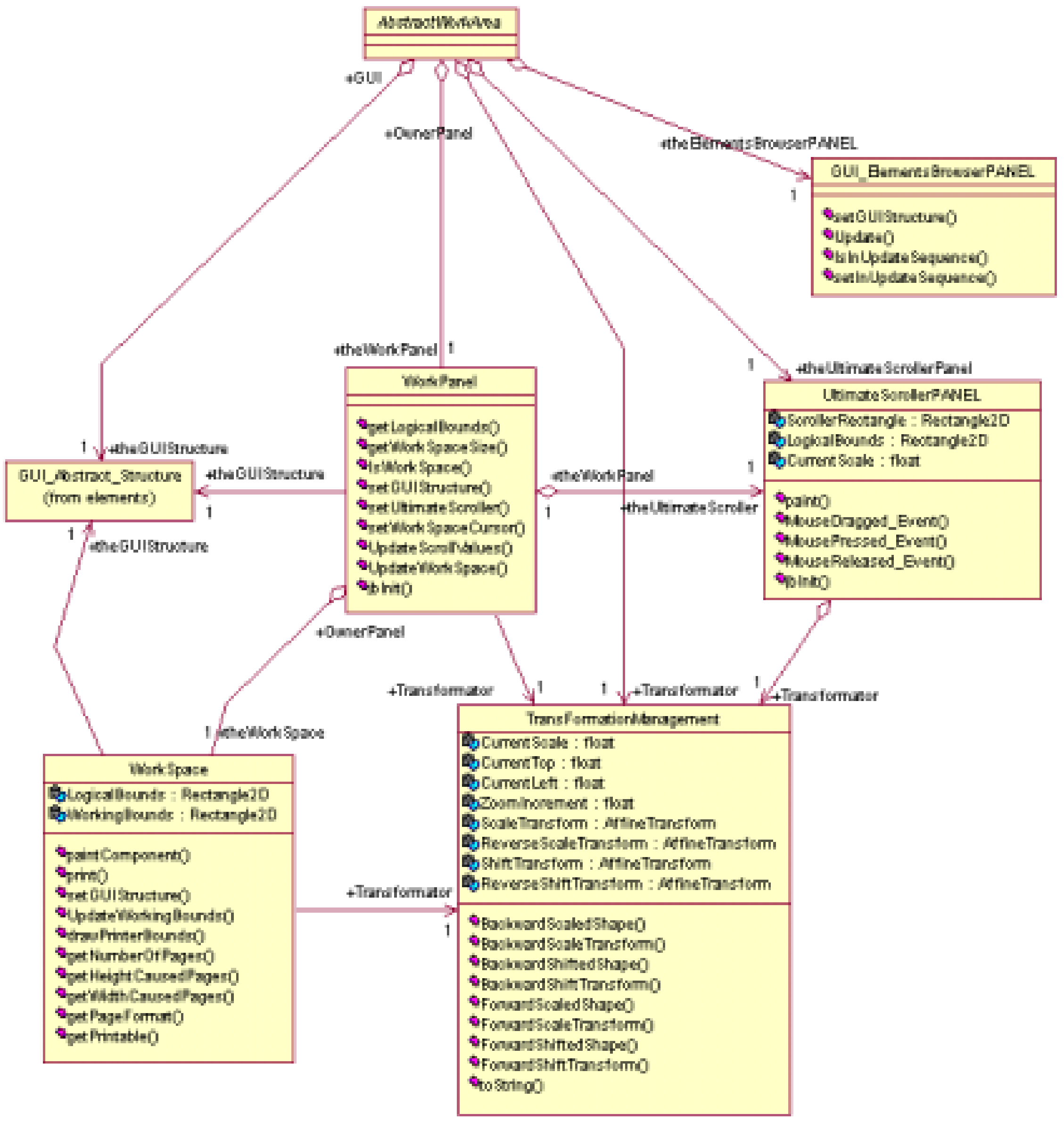


Figure 50. Class Diagram of GUI Elements

5.1.2. XML Format

XML (eXtensible Markup Language) is a markup language for documents containing structured information. XML is a set of tags and declarations, similar to HTML. Unlike HTML, however, the tags in XML are not fixed and users are free to develop their own tags. With this capability, it is a meta-language for describing markup languages [HRO99]. The data in an XML document is structured, which makes it easy to parse, handle, and share with others. Saving the design as an XML document is achieved by a hierarchical walk through the elements of the library. Parsing the XML definitions back is done by means of the XML parser package provided by the Apache Software Foundation [APA01].

The document in Table 9 demonstrates a sample library with a single component and relation. Elements in the design are hierarchically located inside the "Library" tag, analogous to the type hierarchy explained in Section 5.1.1.1, The Parser Package. The name of the library (LibraryName) may be different than the name of the document. A tag is defined for each type (Data Type, Port Type, Relation Type, and Component Type) in the VLGM. Properties of these types become attributes to the tag. When a defined type is used, a new tag with dot notation is defined. As seen in the sample, two defined port types are used in the Component Type named "Objective_Node," with tags defined by dot notation indicating the library in which the port definition can be found. The layout of the design elements is also included as an attribute to the tag with which it's associated. The details and semantics of this example are explained as a case study in Chapter Six. The full definitions of XML tags and attributes used by the system can be found in Appendix A.

Table 9. Sample XML Document

```
<LIBRARY LibraryName="DECISION_ANALYSIS" Explanation="This library
    implements decision tree">
  <PortType TypeName="ParentsPort" Explanation="Connects this objective to
      its parent objective" NodeLayout="91.0,214.0,true" Symbol="Triangle" />
  <PortType TypeName="ChildrenPort" Explanation="Connects the objective
      to its sub-objectives" NodeLayout="319.0,214.0,true" Symbol="Square" />
  <RelationType TypeName="Connection" Explanation="Connects objectives to
      its sub objectives" HeadSymbol="None" TailSymbol="None"
      LineSymbol="Double"
      FromPortType="DECISION_ANALYSIS.ParentsPort"
      ToPortType="DECISION_ANALYSIS.ChildrenPort"
      FromPortRelationLayout="-46.0,14.0,true"
      ToPortRelationLayout="21.0,16.0,true" NodeLayout="197.0,70.0,true" />
- <ComponentType TypeName="Objective_Node" Explanation="" PictureFile=""
      NodeLayout="195.0,412.0,true,true">
  - <Attributes>
      <String Explanation="This field is used as title in the MsExcel
          Worksheet if applicable" Required="True">Caption</String>
      <String Explanation="This field contains detailed information about
          the objective">Explanation</String>
      <Float Explanation="Percentage value between 0 and 100"
          Unit="Percentagevalue"
          Range="0.0..100.0">LocalWeight</Float>
    </Attributes>
  - <Ports>
      <DECISION_ANALYSIS.ParentsPort Explanation="" From="1"
          To="0" Left="-3" Top="-3" RelationLayout="10.0,18.0,true"
          Value=",">SuperObjective</DECISION_ANALYSIS.ParentsPort
          >
      <DECISION_ANALYSIS.ChildrenPort Explanation="" From="0"
          To="n" Left="-4" Top="-4" RelationLayout="3.0,19.0,true"
          Value=",">SubObjectives</DECISION_ANALYSIS.ChildrenPort
          >
    </Ports>
  </ComponentType>
</LIBRARY>
```

5.1.3. Values in Complex Structures

When a scenario is constructed by instantiation of components and relations, the associated attributes are parameterized. These values are saved in the XML document and integrated as a “Value” attribute of relevant tags. String and char values are put in single quotation marks, but numeric and Boolean values are used directly. All values defined in the “Value” attribute of a XML definition must be delimited by double quotation marks. Extension and composition dependencies between complex types make it difficult to save into a single XML attribute. To solve the problem, a set notation is used. Table 10 shows how the set notation is used for complex types.

For example, “**{{{{_},{_}},{{_},{_}},{{_},{_}}},{{{_},{_}},{{_},{_}},{{_},{_}}}}**” would indicate a 2x3x2 array. Based on the type definition, these notations might occur recursively, one inside another.

Table 10. Set Notation for Parameters

Type	Notation
Array attribute	Each value and dimension is put inside **{ }** and equal level dimensions are separated with a comma.
Case	Attribute values are put in **{ }** and separated with commas if the case condition is currently true.
Data Type, Port Type, Relation Type	Attribute values are put in **{ }** and separated with commas. If it extends other types, the following schema is used: **{{Extended Attributes},{Owned Attributes}}**
Component Type	**{{AttributeValues},{PortValues}}** schema is used. In case of extension, the following schema is used: **{{{Extended AttributeValues},{Extended PortValues}}, {{Owned AttributeValues},{Owned PortValues}}}**
Scenario Type	Values of components and relations are separated with commas. The following schema is used: **{{Component Values},{Relation Values}, {Mapped AttributeValues},{Mapped PortValues}}**

5.2. The Application User Interface

As stated earlier, the application is composed of two design areas, thus, two different file types are applicable: Library and Scenario. These are separated as seen on the screen-shots of the tool bar in Figure 51.

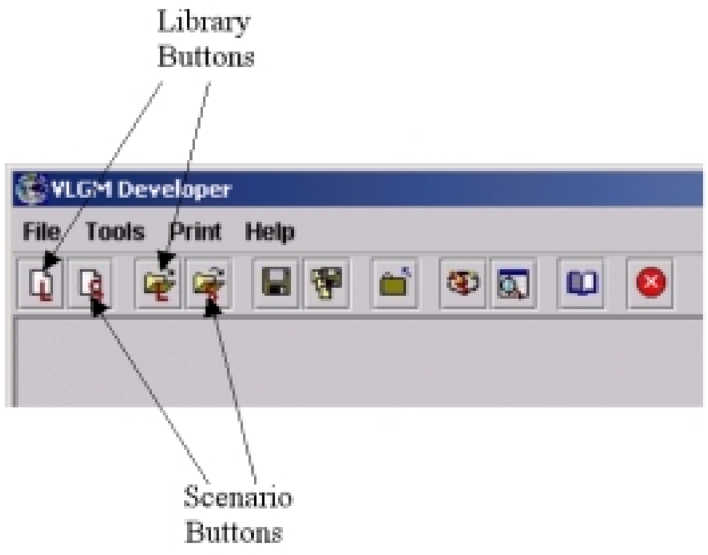


Figure 51. Accessing Design Areas

5.2.1. VLGM Design

Figure 52 shows the VLGM design area. Four types of elements (Data, Port, Relation and Component Types) and three types of relations (Composition, inheritance and port type selection) can be chosen from the tool bar on the left. The elements and relations can be browsed using the tree, and the design can be scrolled easily by means of the box-scroller. Zoom in/out buttons and the print button are put in the design area. The program is able to print the design across multiple pages and the dashed lines in the design indicate the page boundaries. The properties of the elements or relations in the work area can be edited by means of shortcut menus that appear when they are right-clicked upon.

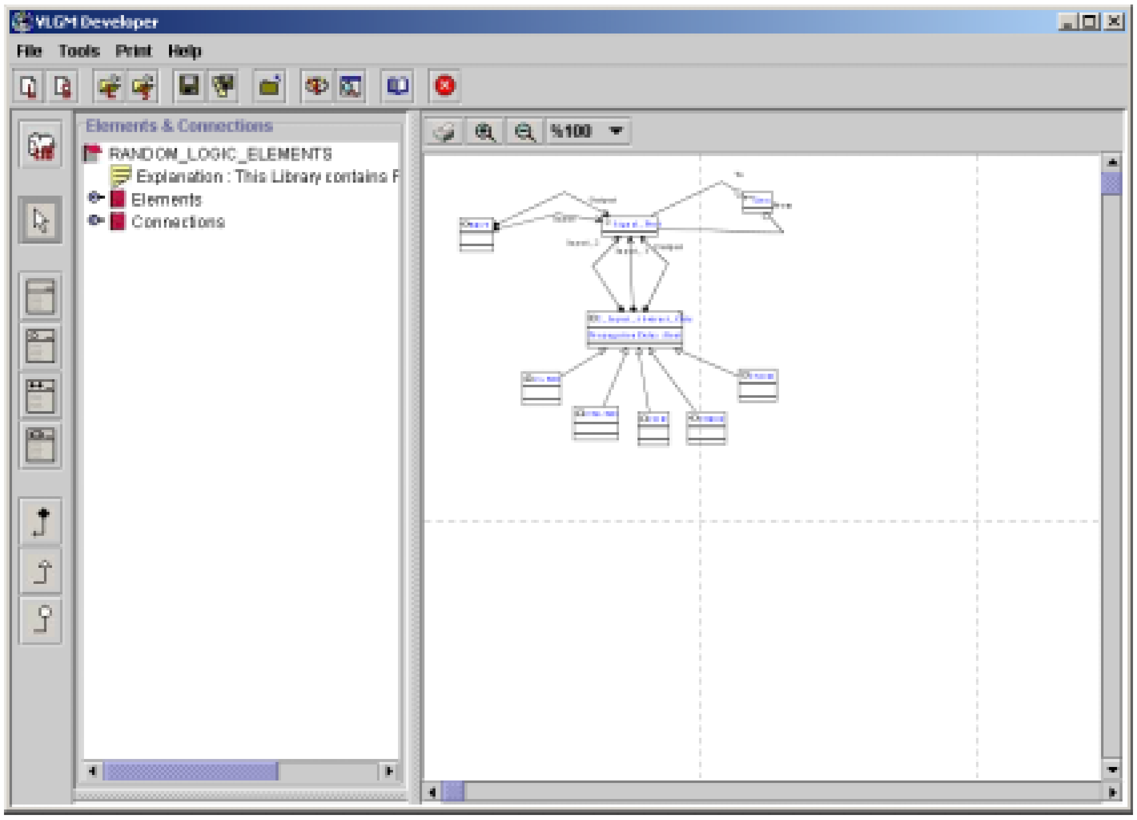


Figure 52. VLGM Design Area

5.2.2. Scenario Design

In order to work in the scenario development environment, relevant libraries should be loaded. The user may then create a new scenario design. As seen in Figure 53, the user may select components or relations from the libraries using the toolbar on the left. Right-clicking over the components or relations will open a short-cut menu through which actions can be performed (see Figure 54).

The consistency check over the design is triggered by the button with check icon. It opens the form listing the inconsistencies as seen in Figure 55. This process involves inspection of the component and relations in the scenario. The list will contain the mandatory attributes that are not parameterized and the ports that are not connected according to their cardinality constraints. Table 8 in Section 4.3.2. lists the graphical constraining and consistency warnings that are implemented.

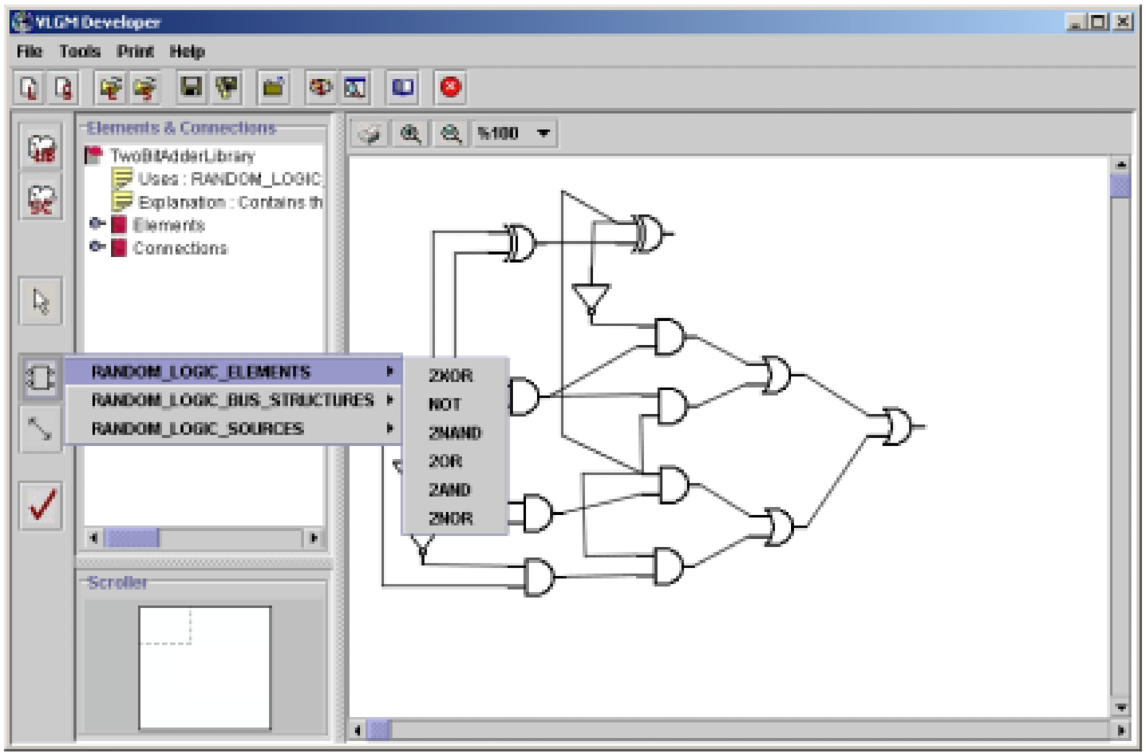


Figure 53. Scenario Design Area

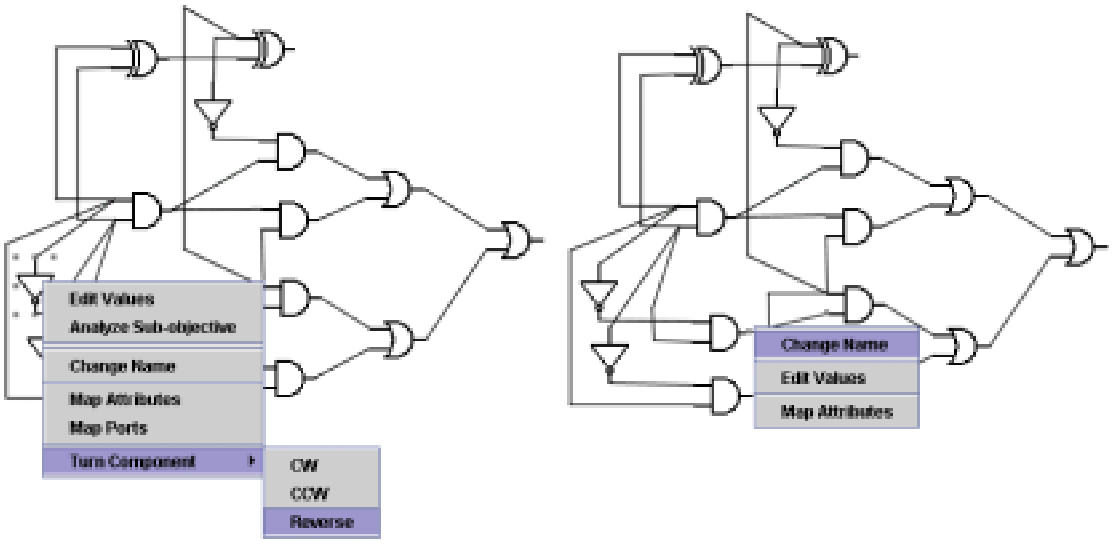
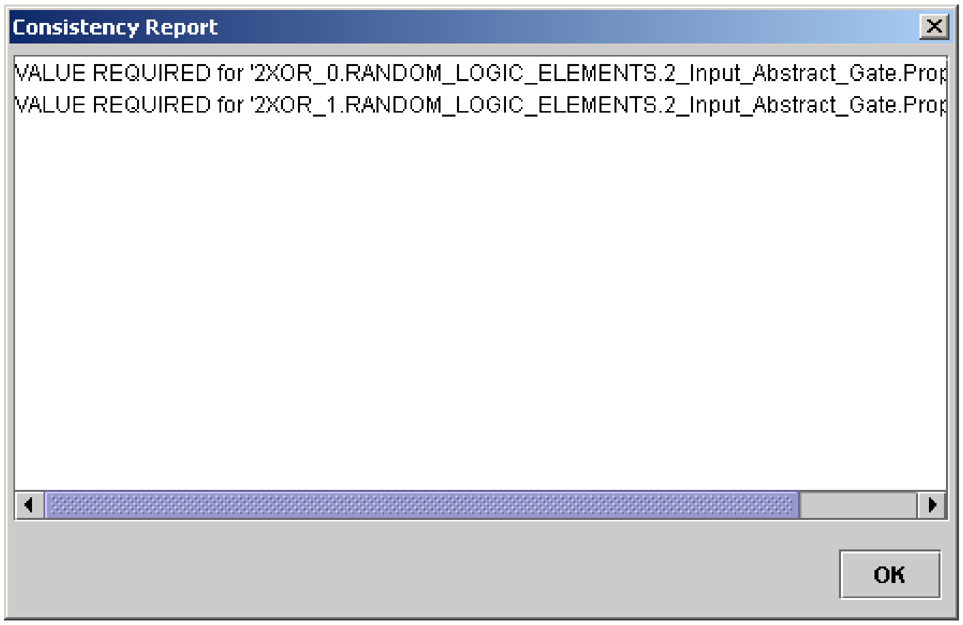


Figure 54. Component and Relation Short-Cut Menu



```
VALUE REQUIRED for '2XOR_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Prop
VALUE REQUIRED for '2XOR_1.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Prop
```

Figure 55. Consistency Report

6. LANGUAGE EVALUATION WITH CASE STUDIES

In this chapter, four case studies for different application domains are presented. They each emphasize a different aspect of the language. The first case study covers the domain of digital circuitry modeling. In this study, a 16-Bit adder unit is built hierarchically using random logic elements. The second case study contains a queueing model that can be used to analyze the behavior of a computer network. The third case study demonstrates a system to support decision-makers with an analysis process. Finally, the fourth shows how VLGM can be used to design a combat scenario for use in military simulations. The full XML documents created by the case studies can be found in Appendix C.

6.1. Digital Circuitry

This case study emphasizes the scalability of the language system. Logic elements are defined as a VLGM library and then used to develop a scenario that implements a two-bit adder. Four instances of this scenario are then used as components to develop a four-bit adder and four instances of a four-bit adder are used to create a 16-bit adder.

6.1.1. VLGM Library Diagrams

To achieve the required functionality, three libraries are required: one for logic gates, one for bus structures, and one for sources (a 1,0 generator). Figure 56 defines the logic elements. “Signal_Port” is a port type for the connection points of the logic elements. The relation type named “Line” is able to connect two ports of type “Signal_Port.” The “NOT” gate has one input and one output port, where the “2_Input_Abstract_Gate” has two inputs and one output. The “2_Input_Abstract_Gate” also has an attribute, a float named

“PropagationDelay,” associated with it. The “2_Input_Abstract_Gate” is defined as an abstract component so it cannot be instantiated. The other gates, “2AND,” “2OR,” “2NAND,” “2NOR,” and “2XOR” extends “2_Input_Abstract_Gate.” Therefore, they all have two inputs, one output, and a “PropagationDelay” attribute.

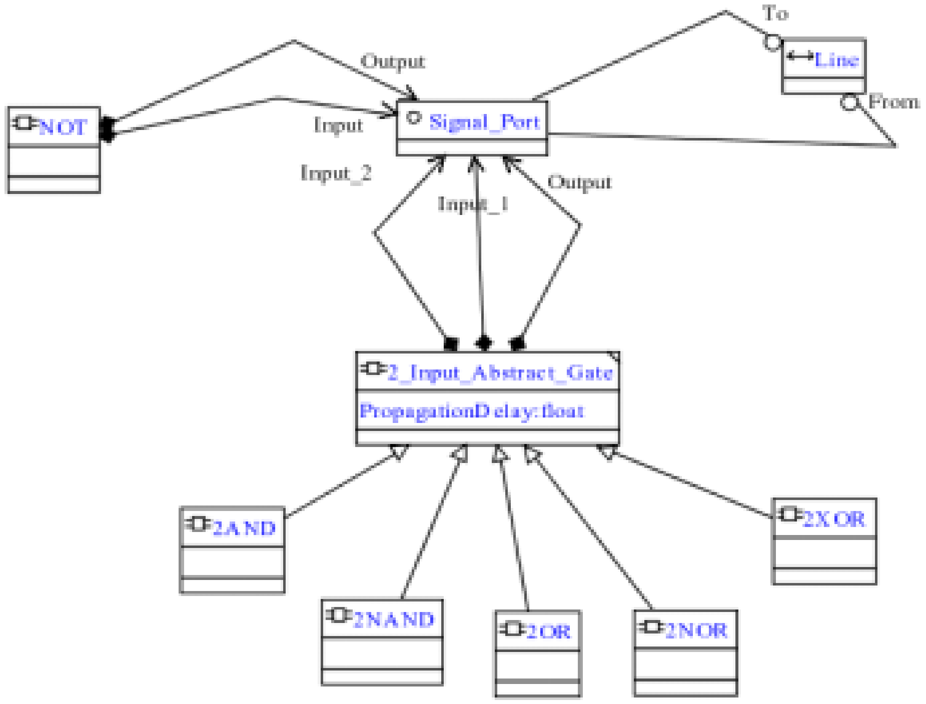


Figure 56. Logic Elements

Though not visible in Figure 56, the “PropagationDelay” attribute is limited to a range of 0.0-200.0, must be parameterized (user must supply a value), and has a time unit of nanoseconds. Each gate is associated with an image file, and port coordinates on the images are provided. For the “Line” relation, the line type is set to “Plain” and head and tail arrows are set to “None.” The forms associated with these specifications can be found in Appendix B.

Figure 57 contains components for bus conversion, and defines a relation for representing a bus line. For the “16BitBus” relation, the line type is set to “Double,” and head and tail arrows are set to “None.” It can connect two ports of type “16BitBusPort.” Each of the components named “1TO16” and “16TO1” has 16 ports of type “Signal_Port” and one port of type “16BitBusPort”.

Figure 58 shows the last VLGM design for this case study. It contains the signal generators. They all have single output port and image files associated with them. “Square_Wave” has a mandatory float attribute named “freq” with a unit type of MHz.

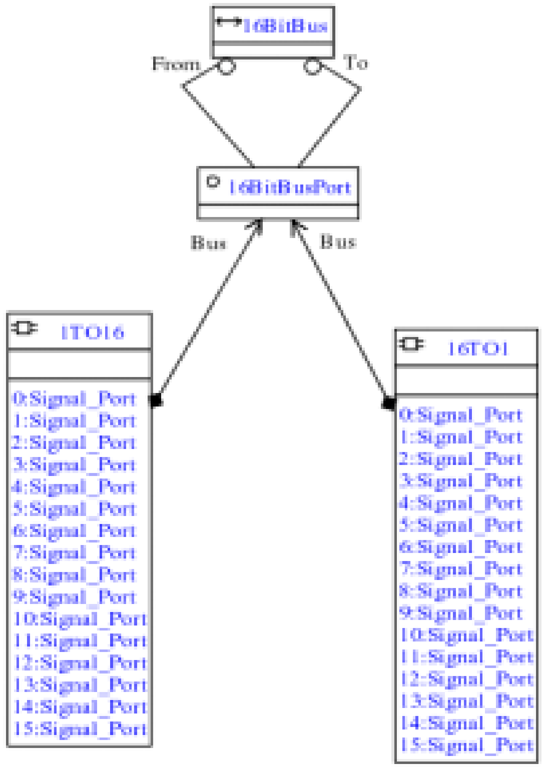


Figure 57. Bus Structures

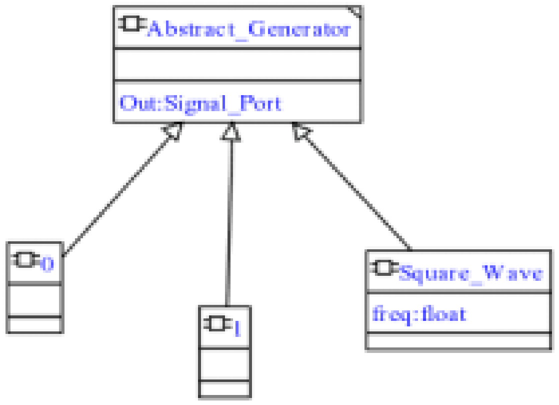


Figure 58. Signal Generators

6.1.2. Scenario Design

Using the libraries defined in the previous section, a series of scenarios were developed. First, a two-bit adder was built from the formulas derived from the truth table shown in Table 11.

Figure 59 shows the design of the adder as developed using the demonstration implementation tool. The ports marked in the diagram are mapped. When this design is used as a component in higher levels, the component shows only these five ports as attributes. These ports are named as “X,” “Y,” “CarryIN,” “Sum,” and “CarryOUT.” Mapping of a port or an attribute is done through the short cut menus associated with components and relations. The sample forms that are used to map ports and attributes are included in Appendix B.

Table 11. Two-Bit Adder Truth Table and Formulas

X	Y	CarryIN	Sum	CarryOUT
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$Sum = (X \otimes Y) \otimes CarryIN$$

$$\begin{aligned} CarryOUT = {} & (X \wedge Y \wedge CarryIN) \vee (X \wedge Y \wedge \neg CarryIN) \vee \\ & (\neg X \wedge Y \wedge CarryIN) \vee (X \wedge \neg Y \wedge CarryIN) \end{aligned}$$

The two-bit adder, once designated as a component, can be used in a new scenario design. Figure 60 uses four two-bit adders to build a four-bit adder. Carryout ports are connected to carry in ports from least through most significant bits. The remaining ports are mapped with the following new names “X_0,” “Y_0,” “SUM_0,” “X_1,” “Y_1,” “SUM_1,” “X_2,” “Y_2,” “SUM_2,” “X_3,” “Y_3,” “SUM_3,” “CarryIN,” and “CarryOUT.”

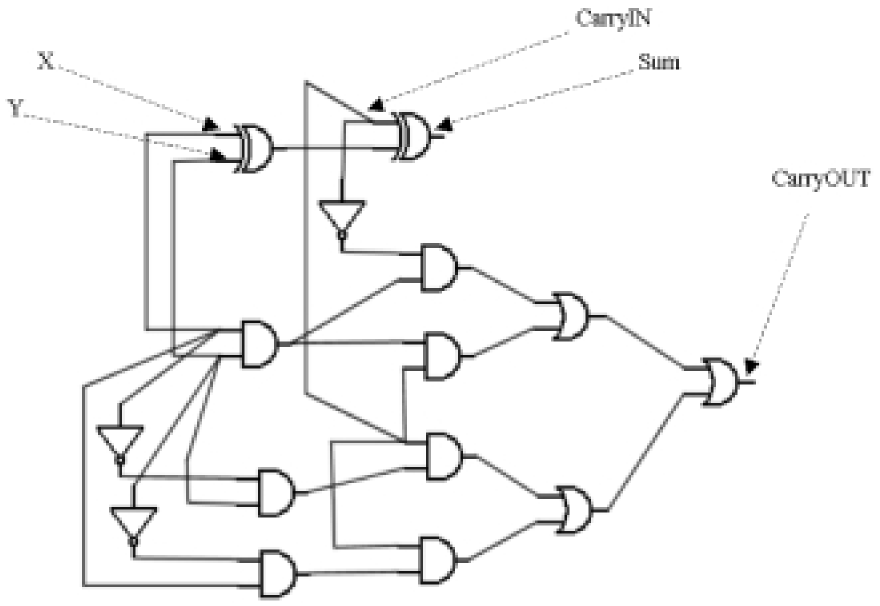


Figure 59. Two-Bit Adder

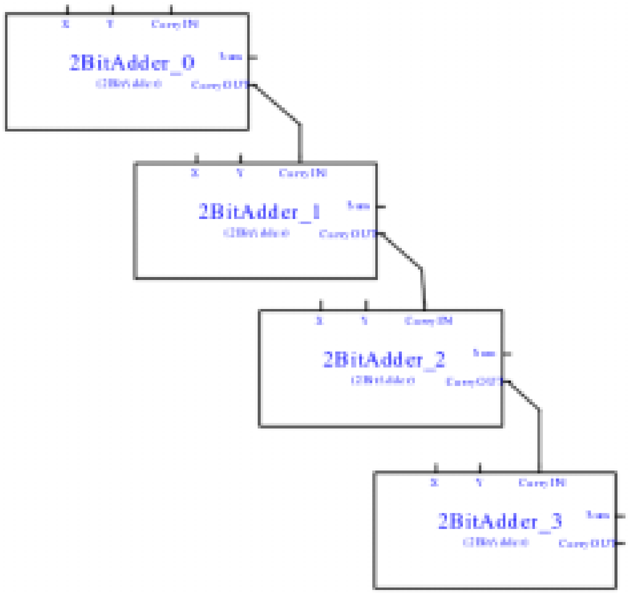


Figure 60. Four-Bit Adder

Using a similar process, a 16-bit adder can be constructed as shown in Figure 61. In Figure 62, the 16-bit adder is used with bus conversion elements and a source producing “0” for carry in input of the least significant bit. Bus connections and the carry out port are mapped, and the design is wrapped again as a component. Finally, the 16-bit full adder with bus connections shown in Figure 63 is ready to be used in any future design.

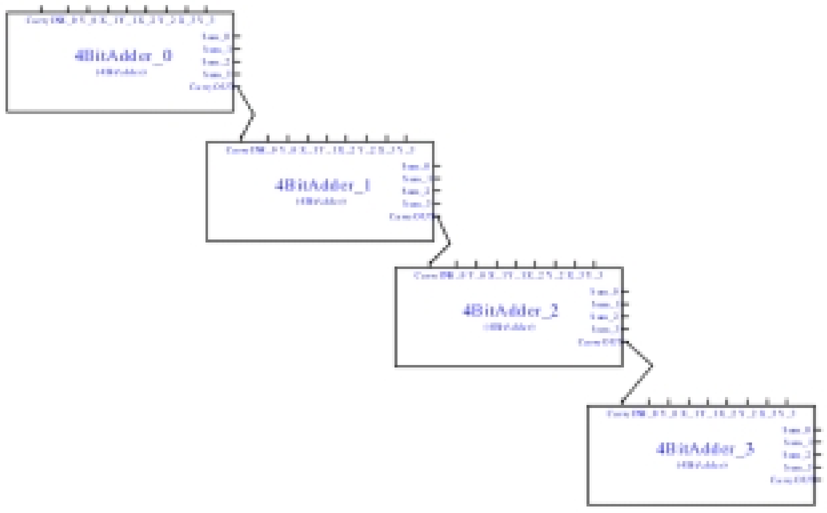


Figure 61. 16-Bit Adder

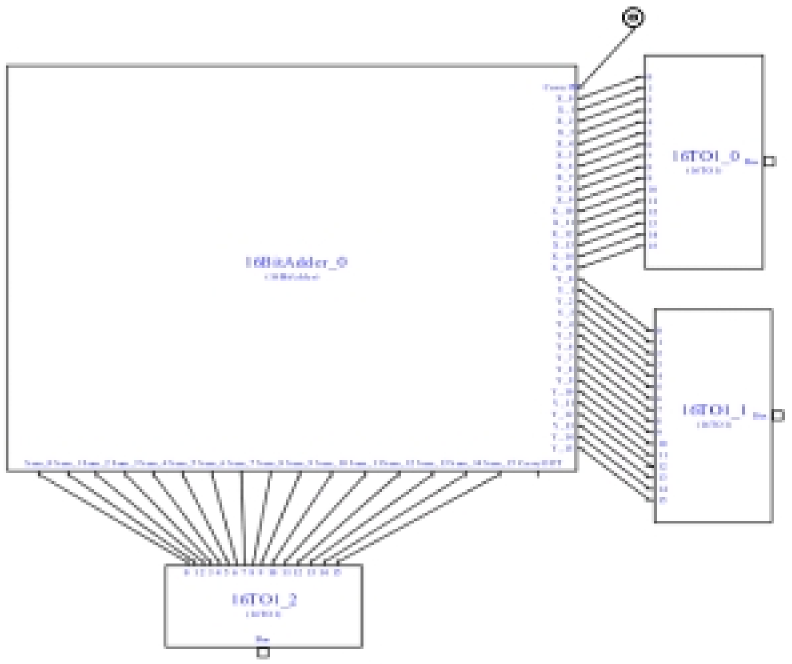


Figure 62. 16-Bit Adder with Bus Connections

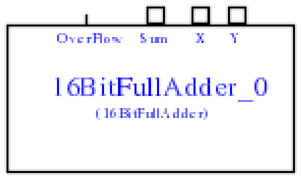


Figure 63. 16-Bit Full Adder

6.1.3. Results

Grouping and abstraction mechanisms proved to be the fundamental tools for any engineering process to handle large-scale complex problems. Two kinds of grouping mechanisms are provided with the language system. First, elements can be grouped under different libraries and imported into scenarios that need them. Second, the scenario environment has the ability to designate a scenario as a component and instantiate copies of it in higher-level designs. These two abstraction tools make VLGM very effective at modeling large systems. Both grouping mechanisms were used effectively in the case study leading to the conclusion that the language has excellent scalability.

However, importing libraries into other libraries can introduce interlibrary dependencies. These dependencies are invisible to the user, making it difficult to keep track of them as the number of related libraries increase. To solve this problem and make VLGM more complete, “Library Relation Diagrams” showing library dependencies may be needed as an extension to the language. A sample notation for library relation diagrams is shown in Figure 64.

The interdependencies also affect the locality of change. If a core library or a scenario were changed, the library or scenarios that are using it might fall into an inconsistent state, resulting in invalid models. Therefore, changes in design propagate through higher levels and the analyst should be aware of that.

The library definitions are already reusable; once a component or a relation type is defined it may be reused in scenarios as much as required. Scenario wrapping also allows the reusability of scenarios. As demonstrated in the case study with adders, VLGM strongly supports reuse.

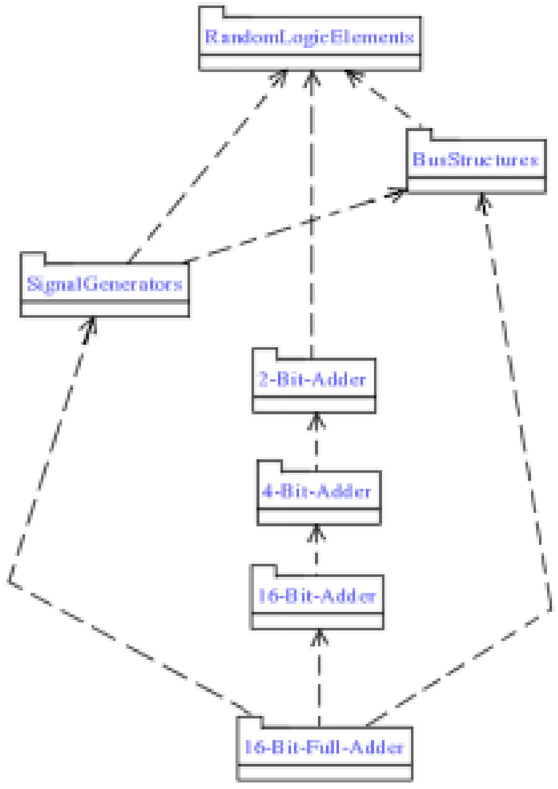


Figure 64. Notation for Inter-library Relation Diagrams

6.2. Networking

A computer network is a collection of computers, servers, and other components connected with some topology that allows the easy flow of data and use of resources between one another. Typically, finite capacity resources are shared and demands upon resources are managed as a queueing system. Communication lines between sub-networks, as one of the main shared resources, are often subject to analysis. In many cases, since the network architecture is complex, analytical solutions are not possible. Therefore, simulation models based on queueing theory are used in analysis.

In this case study, a library is defined for modeling queue theory elements. Unlike the previous case study, this design has a more complex attribute structure. The intent is to show that various types of relations and complex attributes can be handled without causing inconsistencies.

6.2.1. VLGM Library Diagrams

Figure 65 shows a VLGM library containing the basic elements needed to build queue models. Three types of relations and five types of components are available. The “Source” component type is able to produce user-defined packets at the rate specified by a distribution attribute. The distribution of the target addresses of the packets is specified by the “AddressDistribution” attribute. Target addresses lay between zero and “NumberofAddresses” minus one. The parameters associated with the distribution function depend on the distribution type and are defined using the case-structure of the language. The “Queue” component type services the incoming packets with a service rate specified by a distribution function. The “Sink’ component destroys incoming packets. The “Link” and “DelayedLink” relations are able to connect ports of “Source,” “Queue,” or “Sink” components from input to output ports.

In order to model duplex connections between network nodes, the “DuplexPaidLink” relation type is defined on “DuplexPort.” The “Transceiver” component is able to transmit the data coming into its input over a duplex port, receive the data from the duplex port and pass it to its output port.

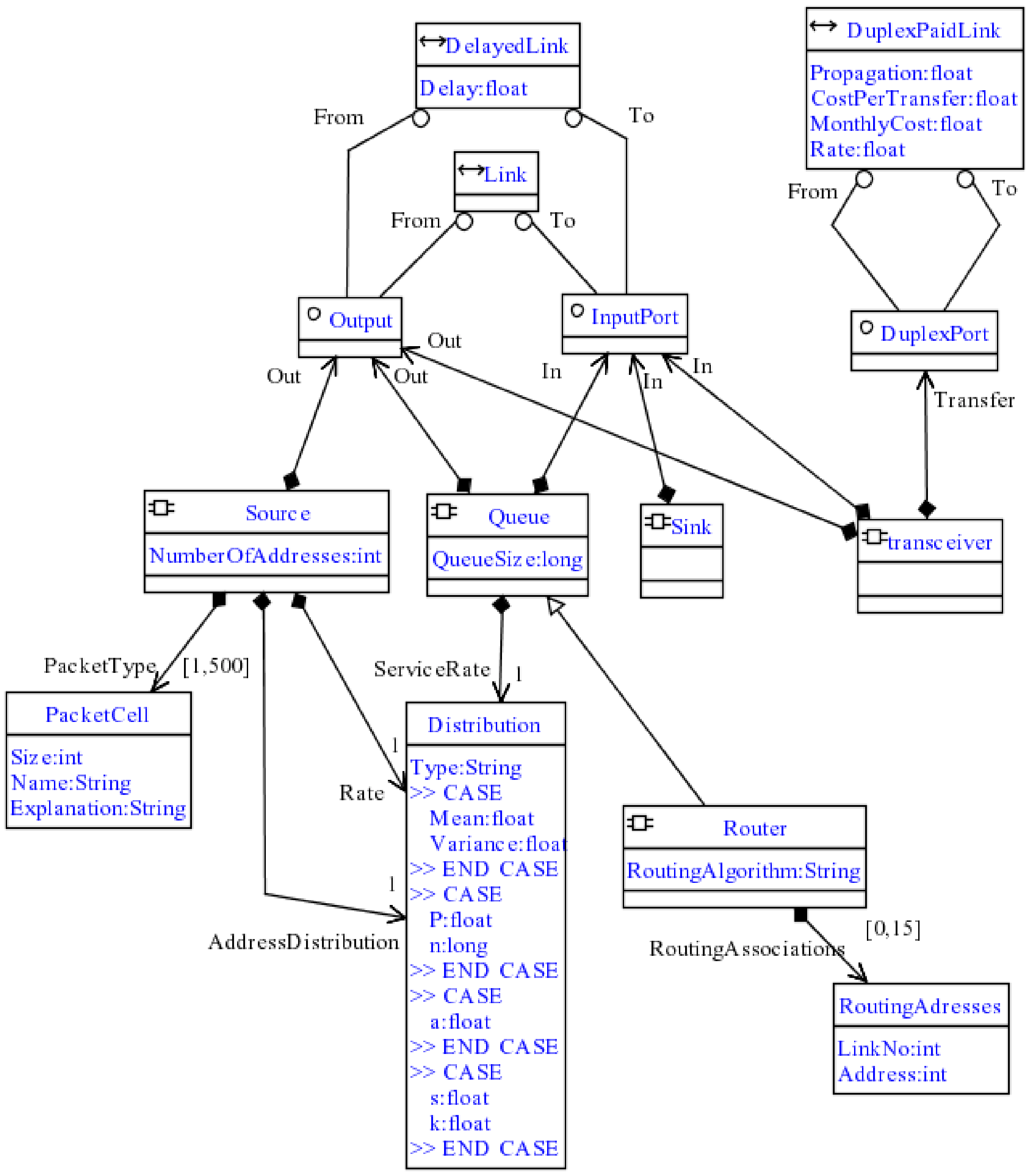


Figure 65. Queue Elements Library

Routing between multiple nodes can be modeled by the “Router” component, which extends the “Queue,” and thus has a queue capacity and service rate. The “RoutingAlgorithm” attribute of the “Router” component can be set to one of “EIGRP,” “RGRP,” or “RIP” string values.

6.2.2. Scenario Design

The scenario in Figure 66 models a node in a network system. It consists of most basic elements. A source produces packets with a distribution and hands them to the queue. Packets are processes by queue and transferred by the transceiver. The sink discards packets incoming through the transceiver. The model in Figure 67 stands for a hub that routes incoming packets based on their addresses. Both hub and node models are designated as components and transceiver ports are mapped. The network scenario in Figure 68 is modeled using the hub and node scenarios.

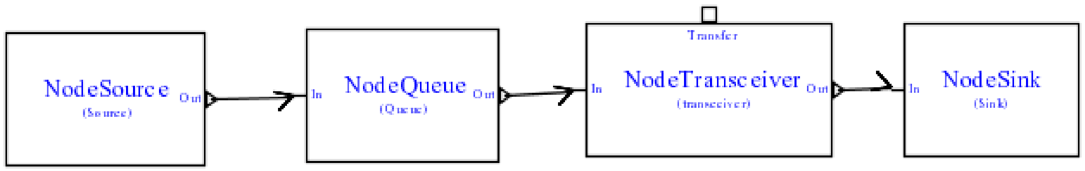


Figure 66. Node Model

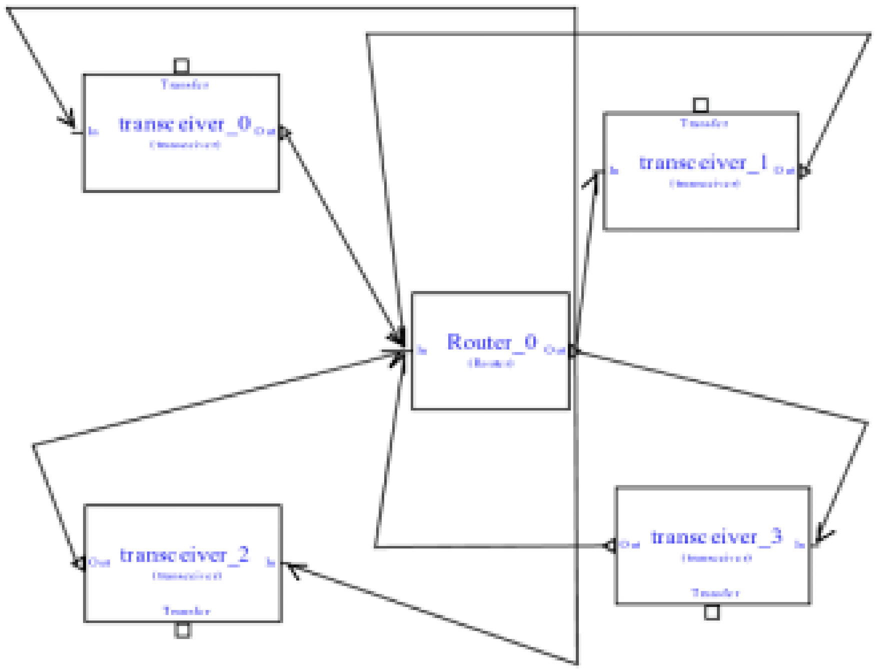


Figure 67. Hub Model

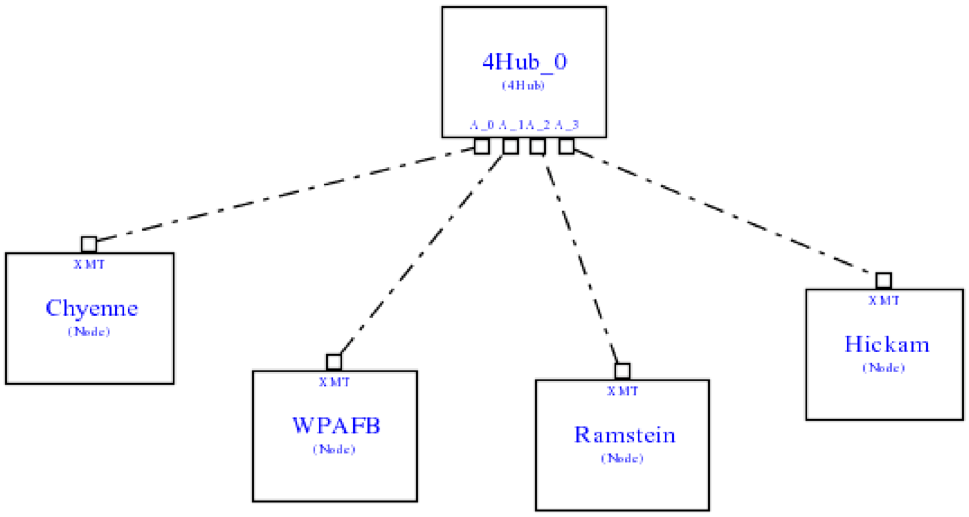


Figure 68. AF Network Model

6.2.3. Results

In the previous case study on digital circuit design, all the components involved had static behavior and were not subject to conditional behavior changes. Network modeling, however, is more difficult when compared to the digital circuit design. In a typical network, the routing process is one of the basic aspects of the system, and it must be modeled properly to assure the validity of the analysis models. Since routing is a behavior, it is difficult to capture its properties completely and efficiently using a static model. As seen in the design diagrams, the hub model is not as intuitive as the node model. In this situation, in order to capture the behavior of the system, the “RoutingAlgorithm” attribute is used. This attribute can be set to one of “EIGRP,” “RGRP,” or “RIP.” These three types of behaviors are assumed to exist in the actual simulation tool. During the scenario design phase, the user is asked to select the behavior of the router. This is an excellent example demonstrating how to capture the behavior of a system using static structures.

6.3. Sensitivity Analysis

This case study examines the transformability of the language system in the domain of decision support. A decision making processes involve the identification of a main objective and construction of a decision tree. Typically, the main objective is divided into subobjectives. Each subobjective may also be divided into their subobjectives and so on. A library containing a single type of component and relation specifies the decision tree model. Nodes on the decision tree are associated with weighting values between 0-100, where the total local weight values of all siblings of a node should be equal to 100. Using weight values, calculations are applied over a decision tree to derive conclusions to support decision-makers. The implementation of these calculations requires interpretation of scenarios, which is the third step of the suggested framework explained in Chapter Four. This case study provides the evidence that designed scenarios can be analyzed and interpreted to achieve calculations. In other words, the scenarios can be transformed into desired formats.

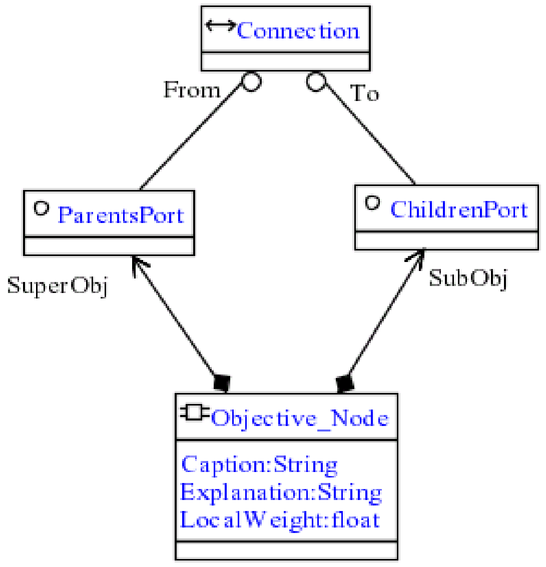


Figure 69. Decision Tree Elements

6.3.1. VLGM Library Diagrams

Figure 69 is a VLGM diagram sufficient in scope to model the required functionality. The “Objective_Node” component has two ports to connect it to its super and sub objectives. The “Connection” relation connects these ports. The cardinality values of the ports are designed to limit the model to a tree, where a node has only one parent but may have any number of child nodes.

6.3.2. Scenario Design

Using the library in Figure 69, a decision tree with the main objective “buy the best car” was built. As seen in Figure 70, the original generic component representation is not sufficient for the visual requirements of this modeling domain. For the digital circuitry case study, it was not a problem that the propagation values for logic elements in the design were not visible. But for a decision tree, each box should contain the caption and the local weight value so it is clear to the user how the analysis will be performed.

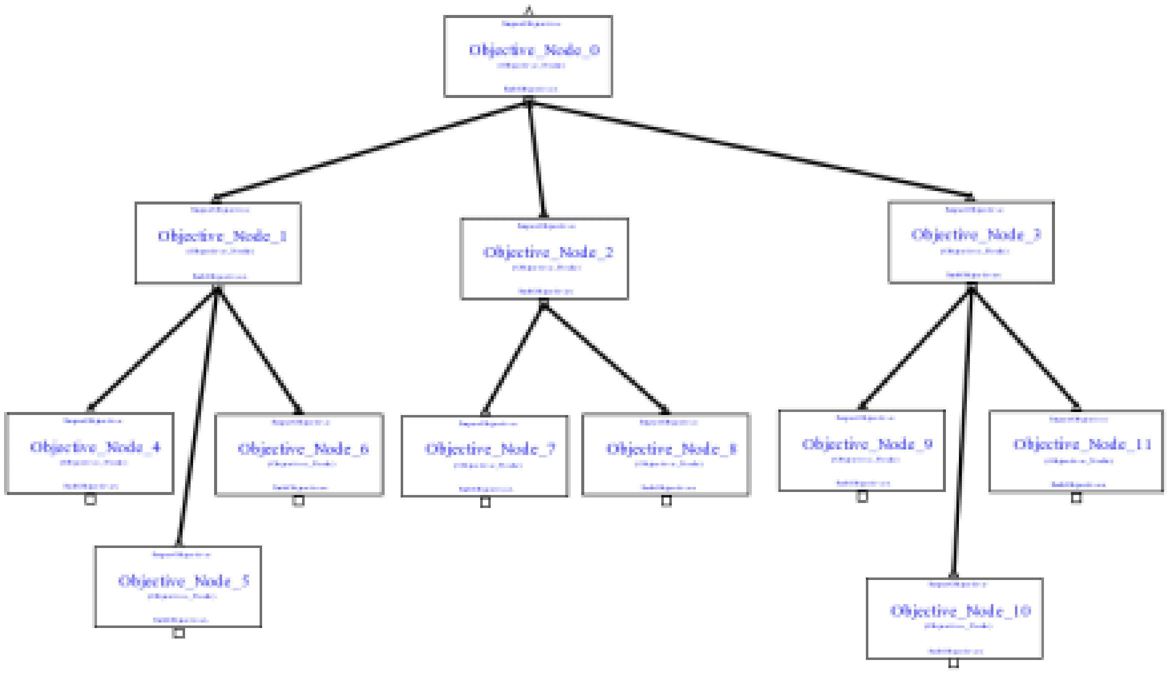


Figure 70. Decision Tree for Best Car

For demonstration purposes, a copy of the program was altered to include a caption and weight in the node representation. The model shown in Figure 71 is easier to interpret and develop by a designer.

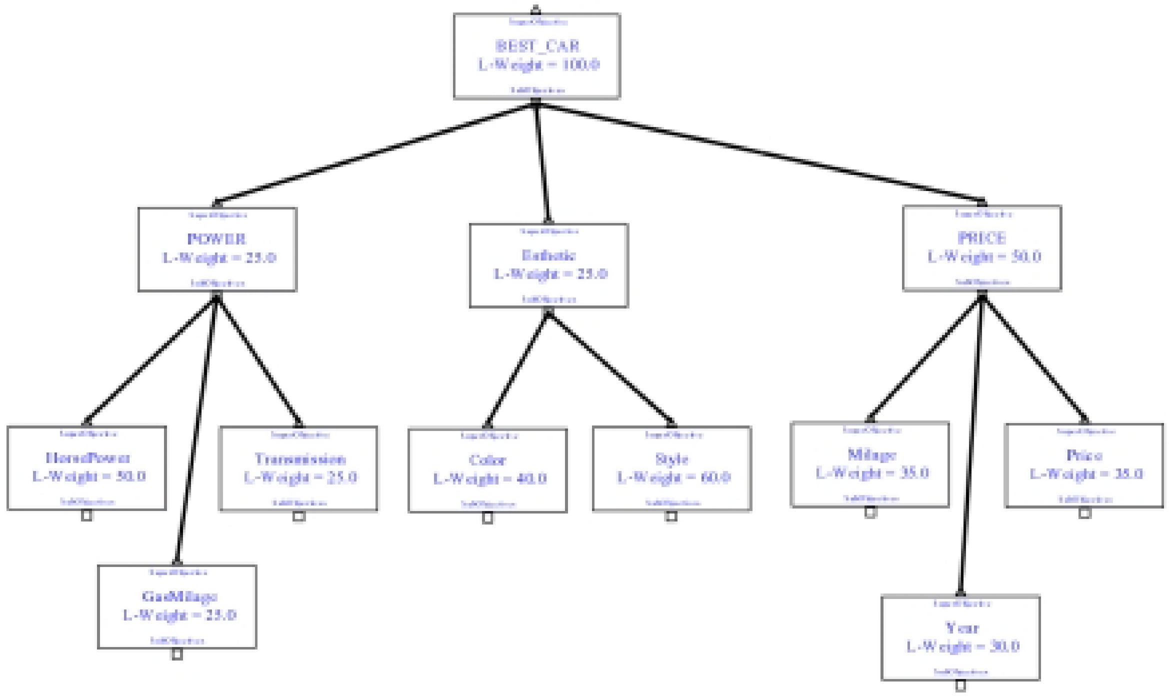


Figure 71. Best Car Model with Adjusted Visualization

6.3.3. Interpretation of Scenario

Several different kinds of calculations and analysis are applicable over this kind of decision tree. For the purposes of this research, the last step in the decision-making process, sensitivity analysis, is implemented. The sensitivity analysis process helps determine how the outcome of a quantitative analysis depends on its inputs.

6.3.4. Sensitivity Analysis Method

The leaf nodes of a decision tree are called the “attributes of alternatives.” The global weight value for each attribute is calculated by the multiplication of local weight values of nodes on its path. A set of global weight values for attributes is obtained. Each multiplication must be normalized to a scale of 0-100 and the summation of the global weights of the attributes will always be equal to 100.

For the sample tree, the set of global weight values shown in Table 12 is obtained.

Table 12. Attribute Global Weights

Horse Power	Gas Mileage	Transmission	Color	Style	Mileage	Year	Price
12.5	6.25	6.25	10	15	17.5	15	17.5

Assume that the attributes of three alternative cars are evaluated as shown in Table 13.

Table 13. Alternative Attribute Evaluations

Alternatives	HP	GM	Trans.	Color	Style	Mileage	Year	Price	$\sum_i w_i e_i$
Ford	90	60	80	80	95	80	80	80	82.25
Honda	80	70	80	100	90	70	90	100	86.125
Hyundai	85	80	80	65	80	95	95	90	85.75

The right most column in Table 13 shows the total scores of the alternatives by taking into account the global weights of attributes in Table 12. To calculate this column, first, each global weight value and related evaluation values are multiplied. In other words, each row in Table 13 is multiplied by Table 12. Then, the multiplications in each row are added together. It's formulated as , where 'w_i' $\sum_i w_i e_i / 100$ and 'e_i' denote global weights and evaluations, respectively. Then, it is easy to decide which car is better just by comparing the total scores.

Sensitivity analysis involves changing the weight factors of a single node from 0 to 100 in increments. The process explained previously is re-applied after each iteration. As the weight factor of the selected node is changed, the total of global weights of the selected node’s siblings are adjusted so as to remain at 100. This case study asks the decision-maker for the proportional values of the siblings so the weights can be adjusted. This new analysis method has been proposed by Yucel Riza Kahraman [KAH02]. For details of this and similar sensitivity analysis methods refer to Kahraman’s research [KAH02].

For each alternative, this method will result in 101 different evaluation values that can be visualized on a graph. The graph will reveal how results vary by weight change in the selected sub objective.

6.3.5. Implementation

To implement the sensitivity analysis process, extra coding was required to walk through the nodes in the design, locate the root and leaves of the tree, find siblings of a given node, and perform calculations. The implementation was based on the type hierarchy explained in Chapter Five. As shown in Figure 72, a menu item was added to the short cut menu for the user to start the process. Once the process is triggered, the program finds the siblings and asks the user to provide proportion values. The total of the proportions should be 100 to preserve the consistency of the evaluation.

After the user provides the required values as shown in Figure 73, the algorithm iterates and saves the captions and the global weights of the attributes (multipliers) into a text file. The user can open an Excel worksheet that has fields for alternatives and macros to read the multipliers and prepare the graphs as shown in Figure 74 Figure 75.

Figure 75 shows the scores of alternatives as the importance of mileage increases from left to right. This graph reveals that Honda scores higher if the importance of the mileage is low, but Hyundai scores higher if the importance of the mileage is high.

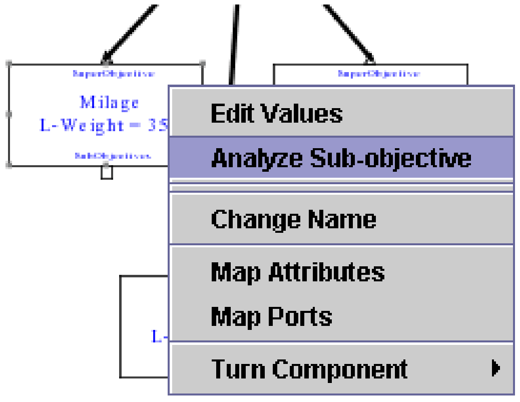
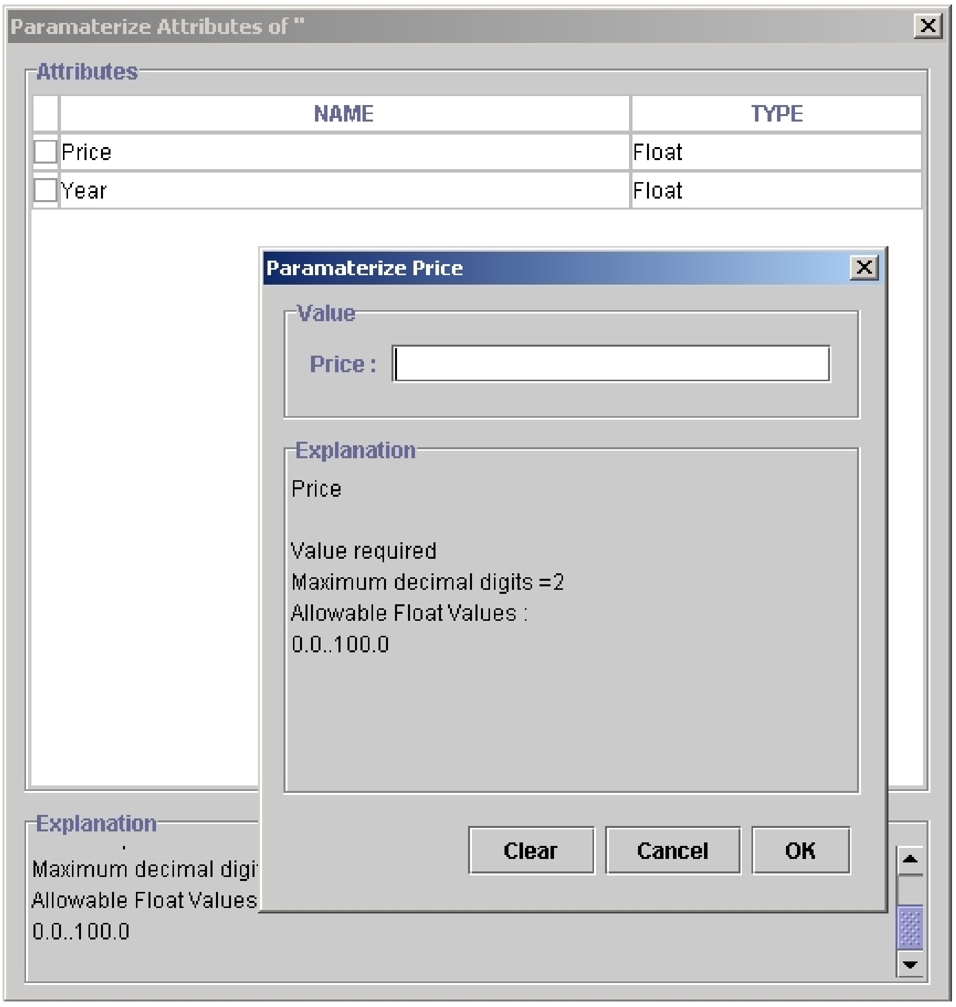


Figure 72. Adjusted Component Short-Cut Menu



	NAME	TYPE
	Price	Float
	Year	Float

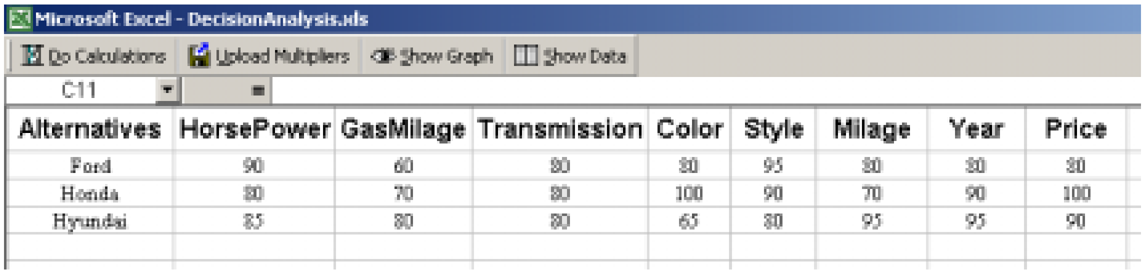
Price :

Price

Value required
Maximum decimal digits =2
Allowable Float Values :
0.0..100.0

Maximum decimal digi
Allowable Float Values
0.0..100.0

Figure 73. Proportional Values for Siblings



Microsoft Excel - DecisionAnalysis.xls

Do Calculations | Upload Multipliers | Show Graph | Show Data

C11

Alternatives	HorsePower	GasMilage	Transmission	Color	Style	Milage	Year	Price
Ford	90	60	80	80	95	80	80	80
Honda	80	70	80	100	90	70	90	100
Hyundai	85	80	80	65	80	95	95	90

Figure 74. Alternative Attribute Evaluation

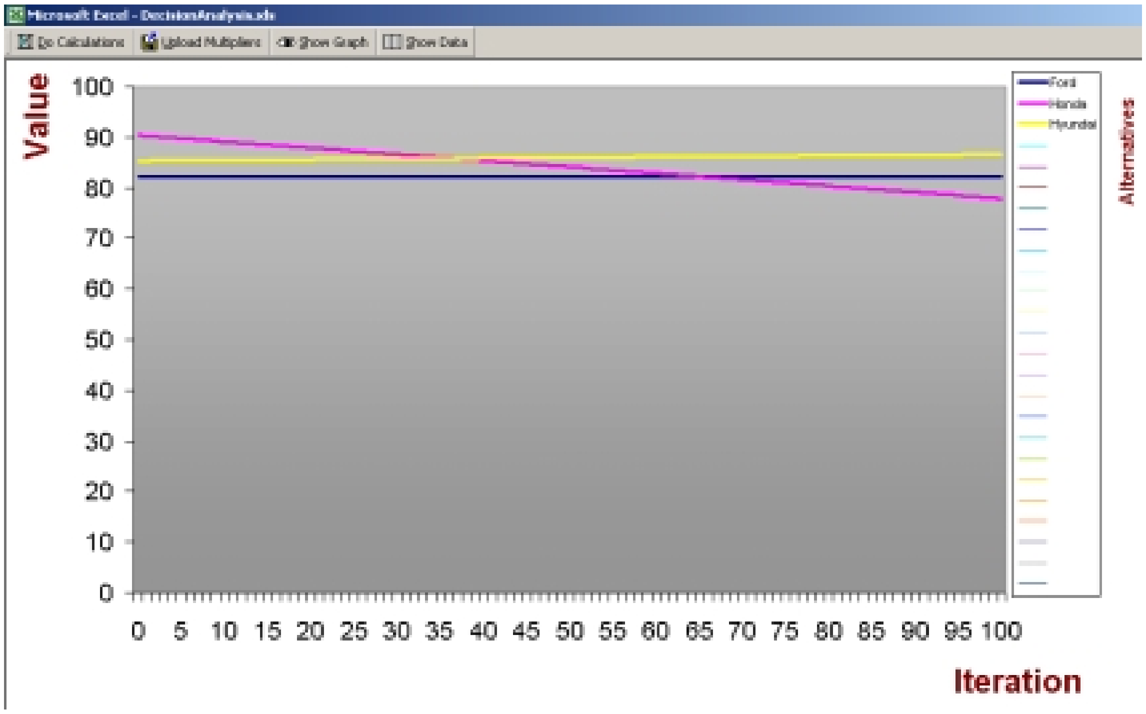


Figure 75. Sensitivity Analysis Graph

6.3.6. Results

This study revealed that the transformation of scenario diagrams is feasible. Transforming scenarios into a new format or applying calculations over a design requires a good understanding of two concepts: First, the underlying structure of the language implementation as described in Chapter Five must be understood and second, the structure of the libraries which define the components, relations, and other data structures involved must be understood.

This study also reveals a visualization problem. The generic box structure for Component Types may be improved to hold values of desired (marked) attributes associated with the component as shown in Figure 76. This can be implemented by adding a boolean property to the definition of primitive types. Refer to Section 4.2.1 for the properties of primitive types. This addition to the language would improve its use of space.

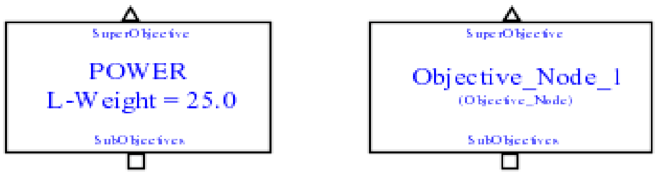


Figure 76. Attribute Value Visibility

6.4. Mission Planning

This case study demonstrates whether the language is suitable for simulation systems that the DoD employs. In a typical combat simulation, missions, weapons, and tactics are modeled. These models are executed in the simulation and results are analyzed. The descriptions of players in these models are complex and detailed.

In order to simplify modeling, some abstractions can be made. For example, the user does not change the detailed description of an F-16 frequently. Therefore, some attributes might be suppressed and relevant attributes and relations can be abstracted to ease the modeling process. This case study explores the applicability of VLGM the domain of combat simulations.

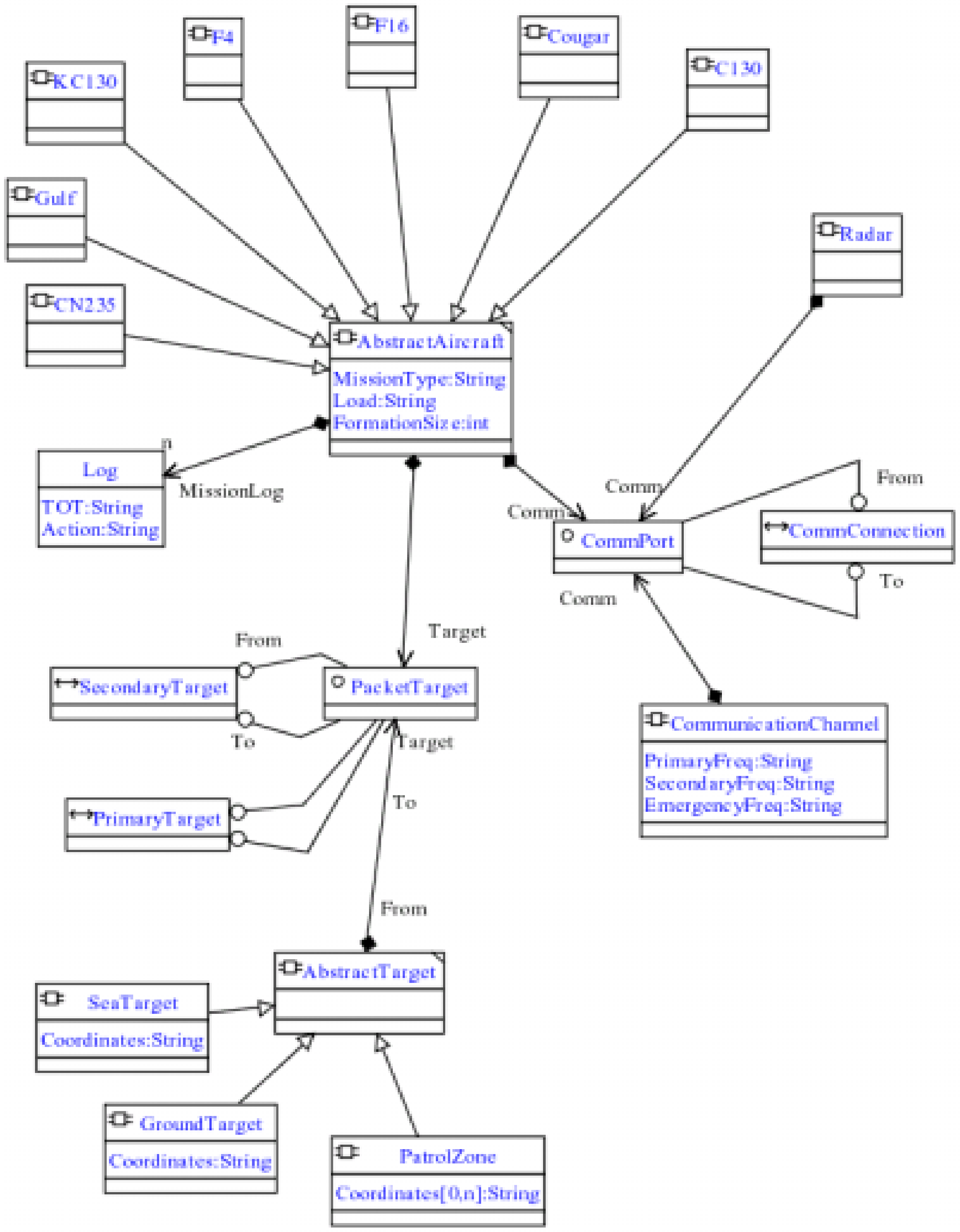


Figure 77. Mission Planning Library

6.4.1. VLGM Library Diagrams

The VLGM diagram in Figure 77 contains the basic elements for mission planning. It specifies different kinds of aircraft and targets, a radar, and a communication channel. An aircraft can be connected to targets through “PacketTarget” ports either with the “PrimaryTarget” or the “SecondaryTarget” ports. The radar and aircraft can be connected to communication channels. Each of the components defined in the library are associated with an image file.

6.4.2. Scenario Design

Figure 78 shows a simple scenario. F-16 and F-4 formations are given primary and secondary sea and ground targets respectively. The Gulf flies an air command center mission in the same zone as the search and rescue aircraft, CN235. Radar control is provided and all aircraft use the same communication channels.

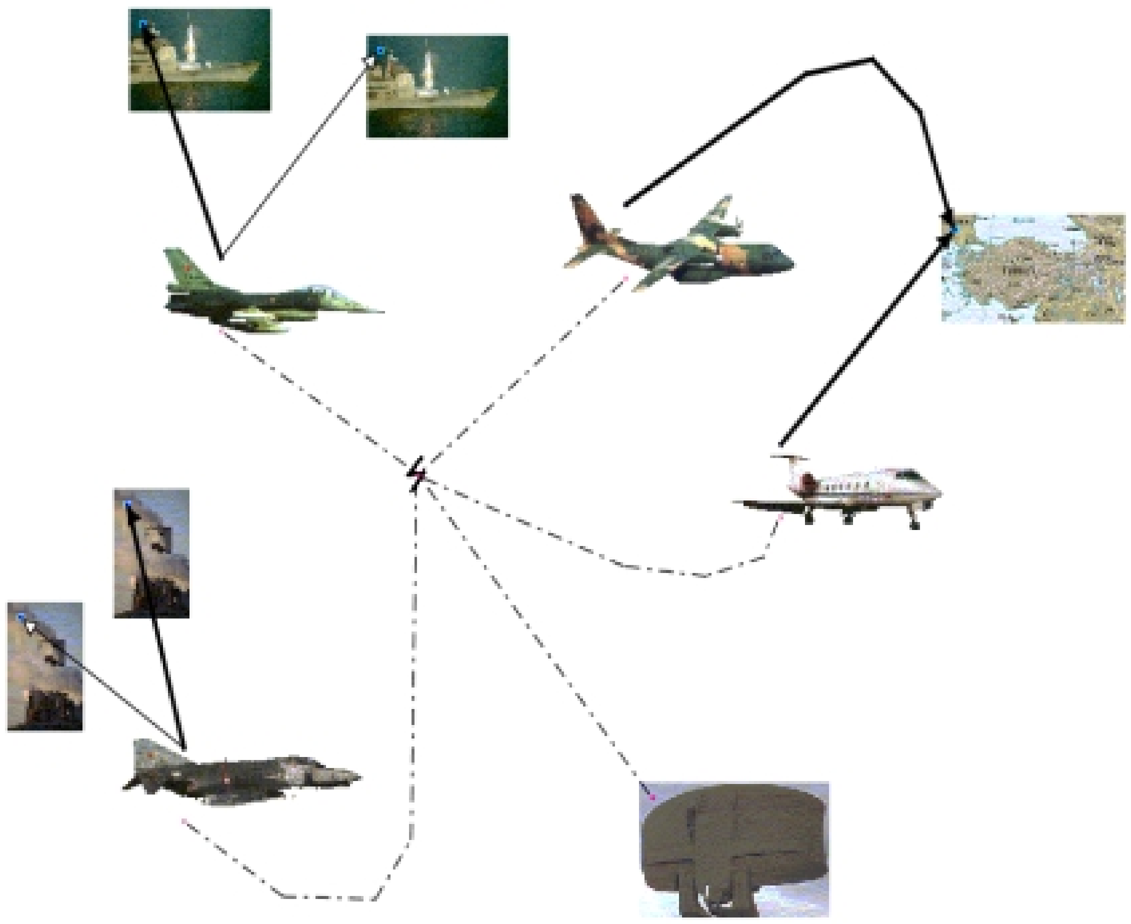


Figure 78. Mission Scenario

6.4.3. Results

The study shows that working with images provides a modeling environment that is much more intuitive. This capability of the language resolves visualization problems for many modeling domains. This kind of pictorial representations is especially valuable for modeling domains working with real life objects such as the combat simulation systems that the DoD employs. Although the player-oriented data structures of most DoD simulation systems are very complex, some abstractions can be made to simplify the design. Unnecessary details can be hard-coded and only frequently changed attributes can be included in VLGM library designs. As the case study on sensitivity analysis showed, conversion of scenarios to other file formats may be required. An algorithm for combat simulations that converts the scenarios developed in demonstration tool into textual definitions for desired simulation tool is applicable.

6.5. Summary of Case Studies

In this chapter, four case studies are demonstrated. The first case study about digital circuitry modeling showed the scalability of VLGM by hierarchical design. It also suggests that interdependencies between libraries can be visualized by library diagrams. The second case study on network modeling shows an example of behavioral specification by static structures. The third case study demonstrates the transformability of VLGM designs by an implementation of a sensitivity analysis method. Finally, the fourth shows how VLGM can be used to design a combat scenario for use in combat simulations.

7. CONCLUSIONS

7.1. Evaluation

This section presents a discussion on the usability of VLGM based on the success criteria suggested in Chapter Three.

7.1.1. Expressiveness

The language has the power of the object-oriented paradigm, which makes it possible to model complex structures efficiently by means of its tools such as extension, composition, and instantiation. But the language makes an assumption that limits its domain to modeling static aspects of systems. It does not attempt to model behavior. As the case study in network domain demonstrated, it is difficult to model the behavior of a system using VLGM. The designer can always make assumptions about the presence of certain kinds of behaviors, and the behavior of a component in the system can be parameterized from a list of possible behaviors like the parameterization of an attribute. As presented, however, it can be concluded that behavior of components is difficult to express with VLGM.

7.1.2. Frequency of errors

The most frequent error that occurred in the design process was caused by interdependencies between libraries. When a design in a lower-level library is changed, it affects the higher-level models and sometimes invalidates them. This kind of error is difficult to avoid and requires experience with the language and object-oriented concepts. In the scenario development environment, the graphical constraining and consistency check capability provided by the VLGM definition disallows most types of errors that might occur.

7.1.3. Redundancy

It is believed that the existence of redundancy in a design depends mainly on the designer's experience with object-oriented concepts and the modeling domain in question. A lack of perspective over the modeling domain might cause redundancy in models. Unavoidable redundancy was not encountered in any of the studies. VLMG is very simple with four types of elements and three types of relations. This simplicity reduces the chance of redundancy in designs.

7.1.4. Locality of change

As explained in the case study on digital circuitry design, if a library or scenario is changed, the libraries or scenarios using it may fall into an inconsistent state, resulting in invalid models. Changes in designs propagate through higher levels and the analysts should be aware of that.

7.1.5. Reusability

The VLGM library definitions are already reusable by design. Once a relation or a component is defined, it may be reused in scenario diagrams through instantiation. The language's tool allows wrapping a scenario for use in higher levels of the design process, which enables reusability of scenarios. Therefore, reusability of both library elements and scenarios is assured.

7.1.6. Reliability

Consistency checking of VLGM library diagrams can be achieved by checking the design with the language specification. For scenarios, the library definitions allow consistency checking. The accuracy of this consistency check depends on the user’s specifications in the library diagrams. For example, if the user defines a float attribute for “probability” but does not set its range to be between 0 and 1, the consistency check of its parameterization won’t produce an error if it has been set out of range. In general, however, reliability is enhanced by the simple, yet flexible rules of VLGM.

7.1.7. Translatability

VLGM library diagrams are interpreted, and the components and relations specified can be used in scenario diagrams. The implementation of the software tool as part of this research is itself evidence for the translatability of library diagrams. Although it might be difficult for domains with complex data structures, translatability of the scenario diagrams is feasible as presented in the case study on sensitivity analysis.

7.1.8. Compatibility

The language is compatible with most types of modeling domains. Some limitations occur on domains where the user needs to define the behavior of the components or the relations involved. It can be concluded that the language is fully compatible with those domains where the static layout of components and the parameterization of attributes are the most essential design tasks. As shown in the case studies on mission planning and digital circuitry, working with images increases usability and compatibility with a closer match to real life representations of the components involved.

7.2. Future Study

This research assembles ideas from various disciplines such as simulation, visualization, modeling, language theory, and software engineering. The study is open to developments and new ideas in these disciplines.

VLGM is designed to be as small as possible to show the applicability of the proposed solution. As a meta-language, it holds the potential to become a very generic tool for the modeling community. However, it lacks behavior modeling. The extension of the language with behavior modeling would make it even more powerful – if different kinds of behavior modeling approaches are integrated.

Although this generic approach may solve the problems of textual simulation systems that the DoD employs, more research is required on the component-relation structure of these systems. Once the VLGM libraries and conversion algorithms are developed, the designed scenarios might be converted to the desired simulation tool. If the same VLGM libraries are used and different conversion algorithms (for each different simulation tool) are developed a scenario design may be converted to desired combat simulation tool. This approach may enable interoperability between combat simulation tools and crosscheck of simulation results. Another study may focus on integration of a map background in the scenario development environment, so that, location parameters (coordinates) of components may be automatically set by position of the component over the map.

VLGM might also be viewed as a specific form of UML. A further effort might focus on establishing common grounds between VLGM and UML. Extension of UML with the functionality proposed in this research would be very useful for the simulation community.

7.3. Summary

This research started with the problems of using simulation tools with textual languages. The solution for these problems is graphical user interfaces that allow the user to model the system as a graph that consists of components and relations. The types of components and relations depend on the domain of interest. However, a meta-language that will be used to specify the components and relations can be designed. If the language is designed to have the “transformability” property, the specifications made with that language can be interpreted by a software tool automatically. As a result the tool will be able to provide a modeling environment for the specified domain.

This kind of approach not only solves the problems of simulation tools, which lack user interfaces, but also provides a generic user interface for any kind of modeling domain. This study surveyed the applicability of this idea. A visual language named “Visual meta-Language for Generic Modeling” was designed and implemented as a software tool. The software tool proved the transformability of the language and showed the feasibility of the component-relation modeling approach.

BIBLIOGRAPHY

[AND98]	Andries, Marc, Gregor Engels, Jan Rekers, "How to Represent a Visual Specification," *Visual Language Theory* (Book after Workshop on Theory of Visual Languages -TVL '96 edited by Kim Marriott and Bernd Meyer, pages 245-259, Spring-Verlag New York, 1998, USA.
[APA01]	The Apache Software Foundation, *Xerces version 1.3.0*, 2001, Online Document, http://xml.apache.org/xerces-j/index.html
[BAN96]	Banks, Jerry, Randall R. Gibson, "Getting Started in Simulation Modeling," IIE Solutions, Vol 28, pages 34-39, November 1996, USA.
[BAN97]	Banks, Jerry, Randall R. Gibson, "Simulation Modeling: Some Programming Required," IIE Solutions, Vol 29, pages 26-31, February 1997, USA.
[BAN98]	Banks, Jerry, Randall R. Gibson, "Simulation Evolution," IIE Solutions, Vol 30, pages 26-30, November 1998, USA.
[BAR00]	Bargiela, Andrzej, "Strategic Directions in Simulation and Modeling," Conference of Professors and Heads of Computing, UK Computing Research Strategy CPHC Meeting, Manchester, 6-7 January 2000.
[BOC97A]	Bock, Conrad, James Odell, "A More Complete Model of Relations and Their Implementation," Journal of Object-oriented Programming, Vol 10, pages 38-40, June 1997, USA.
[BOC97B]	Bock, Conrad, James Odell, "A More Complete Model of Relations and Their Implementation: Mappings," Journal of Object-oriented Programming, Vol 10, pages 28-30, October 1997, USA.
[BOC98A]	Bock, Conrad, James Odell, "A More Complete Model of Relations and Their Implementation: Roles," Journal of Object-oriented Programming, Vol 11, pages 51-54, May 1998, USA.
[BOC98B]	Bock, Conrad, James Odell, "A More Complete Model of Relations and Their Implementation: Aggregation," Journal of Object-oriented Programming, Vol 11, pages 68-70, September 1998, USA.
[BOC99A]	Bock, Conrad, "Three Kinds of Behavior Models," Journal of Object-oriented Programming, Vol 12, pages 36-39, July/August 1999, USA.
[BOC99B]	Bock, Conrad, "Unified Behavior Models," Journal of Object-oriented Programming, Vol 12, pages 65-68, September 1999, USA.

[BOC00] Bock, Conrad, “A More Object-Oriented State Machine,” Journal of Object-oriented Programming, Vol 12, pages 36-38, January 2000, USA.

[BOS94] Bossel, Hartmut, *Modeling and Simulation*, A.K. Peters Ltd, 1994,USA.

[CAR99] Card, Stuart K., Jock D.Mackinlay, Ben Shneiderman, *Readings In Information Visualization*, *Using Vision To Think*, Morgan Kaufman Publishers Inc, 1999, San Francisco, California

[CHE76] Chen, Peter Pin- Shan, “The Entity-Relationship Model – Toward a Unified View of Data,” ACM Transactions on Database Systems, Vol 1, pages 9-36, March 1976.

[CLA99] Clarke, Philip, “Simulation Concert Party,” IEE Review, Vol 45, pages 82-84, March 1999, USA.

[CRE98] Creighton, Oliver, “Questions, Options, and Criteria: Elements of Design Space Analysis,” Design Rational Seminar Slides, July 23rd 1998, Online Document, http://wwwbruegge.in.tum.de/people/creighto/stud/DR98/QOC

[FOW99] Fowler, Martin, Kendall Scott, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley, 1999, USA.

[GAM95] Gamma, Eric, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley Longman, 1995, Massachusetts.

[GOU99] Goucem, Ali, “Multi-Domain Modeling and Simulation,” IEE Review, Vol 45, pages 85-87, March 1999, USA.

[GRE96] Green, T.R.G., M. Petre, “Usability Analysis of Visual Programming Environments: A ‘Cognitive Dimensions’ Framework,” Journal of Visual Languages and Computing, Academic Press Limited, Vol 7, pages 131-174, No 2, 1996.

[HAR96] Harmelen, Frank van, Manfred Aben, Fidel Ruiz, Joke van de Plassche, “Evaluating a Formal KBS Specification Language,” IEEE expert, Vol 11, pages 56-62, February 1996.

[HEN97] Henderson-Sellers B., D. Firesmith, I.M. Graham, “The Benefits of Common Object Modeling Notation,” Journal of Object-Oriented Programming, Vol 10, pages 28-34, September 1997, USA.

[HRO99] Harold, Elliotte Rusty, *XML Bible*, 1999, IDG Books Worldwide, CA, USA.

[JAC99] Jacobson, Ivar, Grady Booch, James Rumbaugh, “The Unified Process,” IEEE Software, Vol 16, pages 96-102, May/June ’99, USA.

[JAG00] Jager, D., “Generating Tools From Graph-Based Specifications,” Information and Software Technology, Vol-42, pages 129-139, No 2, 2000, Spring-Verlag New York, USA

[KAH02] Kahraman, Yucel Riza, “Robust Sensitivity Analysis For Multi-Attribute Deterministic Hierarchical Value Models,” MS Thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, AFIT/GOR/ENS/02-10, March 2002, Unpublished Document

[KAO97] Kao, Diana, Norman P. Archer, “Abstraction in Conceptual Modeling,” International Journal of Human Computer Studies, Vol 12, pages 125-150, January 1997, USA.

[MAR98A] Marriot, Kim, Bernd Meyer, *Visual Language Theory*, Articles of Workshop on Theory of Visual Languages (TVL ’96), 1998, Spring-Verlag New York, USA.

[MAR98B] Marriot, Kim, Bernd Meyer, Kent B. Wittenburg, “A Survey of Visual Language Specification and Recognition,” *Visual Language Theory* (Book after Workshop on Theory of Visual Languages - TVL ‘96 edited by Kim Marriott and Bernd Meyer, pages 5-85, Spring-Verlag New York, 1998, USA.

[MOD00] Modelica Association, *Modelica Specification Version 1.4*, 2000, Online Document, http://www.modelica.org/documents/ModelicaSpec14.pdf

[MUL97] Muller, Pierre Alain, *Instant UML*, Wrox Press Ltd, 1997, UK.

[NAR98] Narayan, N.Hari, Roland Hubscher, “Visual Language Theory: Towards a Human-Computer Interaction Perspective,” *Visual Language Theory* (Book after Workshop on Theory of Visual Languages -TVL ‘96 edited by Kim Marriott and Bernd Meyer, pages 86-128, Spring-Verlag New York, 1998, USA.

[OMG01] Object Management Group (OMG), *UML 1.4 Specification*, 2001, Online Document, http://www.omg.org/technology/documents/formal/uml.htm

[PAI00] Paige, R.F., J.S. Ostroff, P.J. Brooke, “Principles For Modeling Language Design,” Information and Software Technology, Vol 42, pages 665-675, July 2000, UK.

[RUM91] Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, 1991, USA.

[RUM99] Rumbaugh, James, Ivar Jacobson, Grady Booch, *The Unified Modeling Language Reference Manual*, Addison Wesley Longman, 1999, Massachusetts.

[SCA96] Scaife, Mike, Yvonne Rogers, “External Cognition: How Do Graphical Representations Work?,” International Journal of Human-Computer Studies, Vol 45, pages 185-213, August 1996, United Kingdom

[SHU96] Shum, Simon J. Buckingam, “Representing Hard-to-Formalize, Contextualised, Multidisciplinary, Organizational Knowledge,” Workshop on Knowledge Media for Improving Organizational Expertise, 1st International Conference on Practical Aspects of Knowledge Management, Basel, Switzerland, 30-31 October 1996, http://ksi.cpsc.ucalgary.ca/AIKM97/sbs/sbs-paper2.html

[SHU97] Shum, Simon J. Buckingam, Allan Maclean, Victoria M. E. Bellotti, Nick V. Hammond, “Graphical Argumentation and Design Cognition,” Human Computer Interaction, Lawrence Erlbaum Associates, Vol 12, pages 267-300, No 3, 1997.

[STE99] Stevens, Perdita, Rob Pooley, *Using UML*, Addison Wesley Longman, 1999, Massachusetts

[TAY99] Tayfun, Avni, “Simulation Modeling Primer,” IIE Solutions, Vol 31 No 9, pages 38-41, September 1999, USA.

[WEB96] *Webster’s New Universal Unabridged Dictionary*, Barnes & Noble Inc., Random House Value Publishing Inc., 1996, USA.

[WIN98] Wind River Systems, *pLUG&SIM Information*, 1998, Web Page, http://www.windriver.com/products/html/plug_sim_ds.html

APPENDIX A. XML TAGS AND ATTRIBUTES OF THE FILE FORMAT

Tag	Property	Type	Restrictions	Example
Library	LibraryName	String	No special characters	<LIBRARY LibraryName="QUEUE" Explanation="This Library contains my elements" Uses="Lib1,Lib2" Layout="True"> </LIBRARY>
	Explanation	String	-	
	Uses	String	Predefined Library Name	
	Layout	Boolean	"True," "False"	
DataType	TypeName	String	No special characters, Name space limitations	<DataType TypeName="Distribution" Explanation="defines parameters of a distribution" NodeLayout="258.0,426.0,true"> </DataType>
	Explanation	String	-	
	Abstract	Boolean	"True," "False"	
	NodeLayout	String	Visibility parameters depending on implementation	
DataType Instance	Name (WhiteSpace)	String	No special characters, Name space limitations	<QUEUE.Distribution Explanation="Production Rate" Required="True" RelationLayout="-40.0, 0.0,-9.0,-28.0,true,true" Value=",{'Normal',{150.0,15.0},,,}"> Rate </QUEUE.Distribution>
	Explanation	String	-	
	Required	Boolean	"True," "False"	
	Value	String	As shown in Chapter-5	
	RelationLayout	String	Composition visibility parameters depending on implementation	
ComponentType	TypeName	String	No special characters, Name space limitations	<ComponentType TypeName="Source" Explanation="Produces " PictureFile="" NodeLayout="95.0,301.0,true,true"> </ComponentType>
	Explanation	String	-	
	Abstract	Boolean	"True," "False"	
	NodeLayout	String	Visibility parameters depending on implementation	
	PictureFile	String	Full file path of image for component	
ComponentType Instance	Name (WhiteSpace)	String	No special characters, Name space limitations	<QUEUE.Source NodeLayout="39.0,59.0" NodeAlignment="NORMAL" Value=",{{{,{1024,'Data','gchfsh'}}},,{'Normal',{150.0,15.0},,,},,{'Normal',{150.0,15.0},,,},3},{,}"> Source_0 </QUEUE.Source>
	Value	String	As shown in Chapter-5	
	NodeAlignment	String	"NORMAL,""CW,""CCW,""REVERSE"	
	NodeLayout	String	Visibility parameters depending on implementation	
PortType	TypeName	String	No special characters, Name space limitations	<PortType TypeName="InputPort" Explanation="Input Port Definition" Abstract="False" NodeLayout="355.0,185.0,true" Symbol="Line"> </PortType>
	Explanation	String	-	
	Abstract	Boolean	"True," "False"	
	NodeLayout	String	Visibility parameters depending on implementation	
	Symbol	Constant	"Default," "Circle,"	

			“FilledCircle,” ”FilledSquare,” ”Line,” “Square,” “Diamond,” “FilledDiamond,” “Triangle,” “FilledTriangle,” None”	
<u>PortType Instance</u>	Name (WhiteSpace)	String	No special characters, Name space limitations	<QUEUE.Output Explanation="Output port of source" From="n" To="n" Left="-2" Top="-2" RelationLayout="-42.0, 15.0,true" Value=","> Out </QUEUE.Output>
	Explanation	String	-	
	Value	String	As shown in Chapter-5	
	From	String	“0..1,””1..n,””n” // n or number	
	To	String	“0..1,””1..n,””n” // n or number	
	Top	Integer	Bitmap/box size -1 Left; -2 Right; -3 Top; -4 Bottom	
	Left	Integer	Bitmap/box size -1 Left; -2 Right; -3 Top; -4 Bottom	
	RelationLayout	String	Composition visibility parameters depending on implementation	
RelationType	TypeName	String	No special characters, Name space limitations	<RelationType TypeName="Link" Explanation="Link without Delay" HeadSymbol="Arrow" TailSymbol="None" LineSymbol="Plain" FromPortType="QUEUE.Output" ToPortType="QUEUE.Input Port" FromPortRelationLayout="-44.0,9.0,true" ToPortRelationLayout="18.0,9.0,true" NodeLayout="288.0,101.0,true"> </RelationType>
	Explanation	String	-	
	Abstract	Boolean	“True,” “False”	
	NodeLayout	String	Visibility parameters depending on implementation	
	HeadSymbol	Byte	“Default,” “None,” “Arrow,” “Triangle,” “FilledTriangle,” “Diamond,” “FilledDiamond,” “Circle,” “FilledCircle”	
	TailSymbol	Byte	Same as Head Symbol	
	LineSymbol	Byte	“Default,” “Plain,” “Dashed,” “Double,” ”Dot,””DashDot”	
	FromPortRelationLayout	String	PortTypeSelection relation visibility parameters depending on implementation	
	ToPortRelationLayout	String	PortTypeSelection relation visibility parameters depending on implementation	
	FromPortType	String	Name Space Restrictions	
	ToPortType	String	Name Space Restrictions	
<u>RelationType Instance</u>	Name (WhiteSpace)	String	No special characters, Name space limitations	<QUEUE.Link Explanation="Link without Delay" From="Source_0.Out" To="Queue_0.In" Value=","
	Value	String	As shown in Chapter-5	
	From	String	Name Space Restrictions	
	To	String	Name Space Restrictions	

	RelationLayout	String	Arbitrary relation visibility parameters in scenario depending on implementation	RelationLayout=""> Link_0 </QUEUE.Link>
ScenarioType	TypeName	String	No special characters, Name space limitations	<ScenarioType TypeName="Node" Explanation="" Component="True"> </ScenarioType>
	Explanation	String	-	
	Component	Boolean	“True,” “False”	
<u>ScenarioType Instance</u>	Name (WhiteSpace)	String	No special characters, Name space limitations	<NodeModel.Node NodeLayout="74.0,168.0" NodeAlignment="NORMAL " Value="{{,{{{,{1024,'Data','gchfsh'}}},,{'Normal',{150.0,15.0},,,},,{'Normal',{150.0,15.0},,,},3},{,},,{,{'Normal',{150.0,15.0},,,},1500},{,,,},,,{,},,,{,,,,,}},{,,,,,},,{,}}"> Chyenne </NodeModel.Node>
	Value	String	As shown in Chapter-5	
	NodeAlignment	String	“NORMAL,”“CW,””CCW,””REVERSE”	
	NodeLayout	String	Visibility parameters depending on implementation	
Int	Name (WhiteSpace)	String	No special characters, Name space limitations	<Int Explanation="An int" Unit="" Required="True" Value="15.0" Range="0..12,15,18..25"> SampleInt </Int>
	Explanation	String	-	
	Required	Boolean	“True,” “False”	
	Unit	String	-	
	Value	Integer	Range Limitations	
	Range	String	-	
Long	Name (WhiteSpace)	String	No special characters, Name space limitations	<Long Explanation="A Long Array" Unit="" Required="True" Value="" Range="0..12,15,18..25"> LongArray[1,3][3,5] </Long>
	Explanation	String	-	
	Required	Boolean	“True,” “False”	
	Unit	String	-	
	Value	Long	Range Limitations	
	Range	String	-	
Float	Name (WhiteSpace)	String	No special characters, Name space limitations	<Float Explanation="Variance of Normal distribution" Unit="" Decimal="5" Required="True" Value="15.0" Range="0..200"> Variance </Float>
	Explanation	String	-	
	Required	Boolean	“True,” “False”	
	Unit	String	-	
	Decimal	Byte	-	
	Value	Float	Range Limitations	
	Range	String	-	
Double	Name (WhiteSpace)	String	No special characters, Name space limitations	<Double Explanation="Variance of Normal distribution" Unit="" Decimal="5" Required="True" Value="15.0" Range="0..200"> Variance </Double>
	Explanation	String	-	
	Required	Boolean	“True,” “False”	
	Unit	String	-	
	Decimal	Byte	-	
	Value	Double	Range Limitations	
	Range	String	-	
Boolean	Name (WhiteSpace)	String	No special characters, Name space limitations	<Boolean Explanation="A Boolean Array" Required="True"> BoolArray[1,n] </Boolean>
	Explanation	String	-	
	Required	Boolean	“True,” “False”	
	Value	Boolean	“True,” “False”	

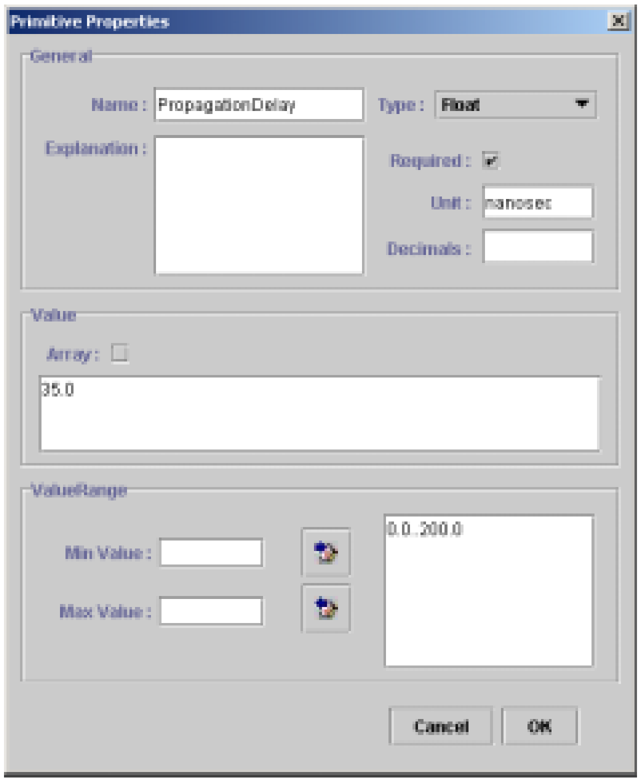
Char	Name (WhiteSpace)	String	No special characters, Name space limitations	<Char Explanation="A Char"
				Required="True"
				Range="'a'..'k','o','r'..'w'"
				Value="a'">
				MyChar
				</String>

	Explanation	String	-	
	Required	Boolean	“True,” “False”	
	Value	Char	Range Limitations	
	Range	String	--	

String	Name (WhiteSpace)	String	No special characters, Name space limitations	<String Explanation="Type of Distribution"
				Required="True"
				Range="'Normal','Poisson','Binomial','Pareto'"
				Value="'Normal'">
				Type
				</String>

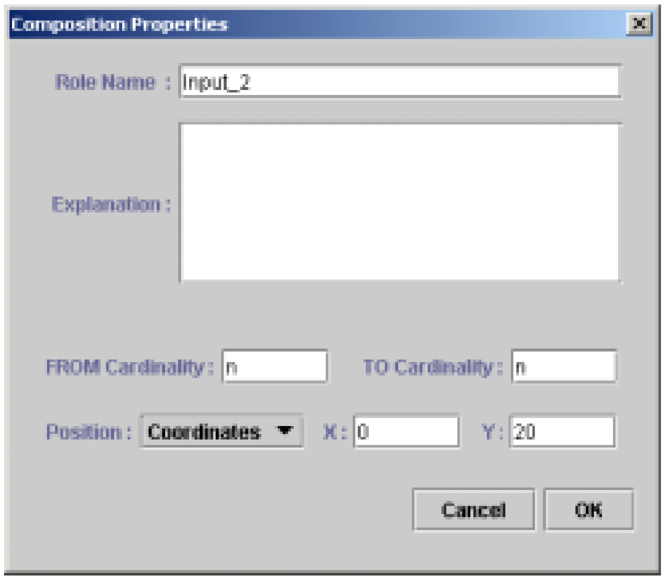
	Explanation	String	-	
	Required	Boolean	“True,” “False”	
	Value	String	Range Limitations	
	Range	String	-	
Case	Conditions	String	Name space limitations	<Case Conditions="Type='Poisson'" > </Case>
Attributes	-			<Attributes></Attributes>
Ports	-			<Ports></Ports>
Interface	-			<Interface></Interface>
AttributeMaps	Mapping (WhiteSpace)	String	Name space limitations	<AttributeMaps> transceiver_0.MyChar= NewNamedChar </AttributeMaps>
PortMaps	Mapping (WhiteSpace)	String	Name space limitations	<PortMaps> transceiver_0.Transfer= XMT </PortMaps>
Components	-			<Components> </Components>
Connections	-			<Connections> </Connections>
Extension	ExtendedTypeName	String	Name space limitations	<Extension ExtendedTypeName="QUEUE.Queue" RelationLayout=""> </Extension>
	RelationLayout	String	Extension relation visibility parameters depending on implementation	

APPENDIX B. SCREEN SHOTS



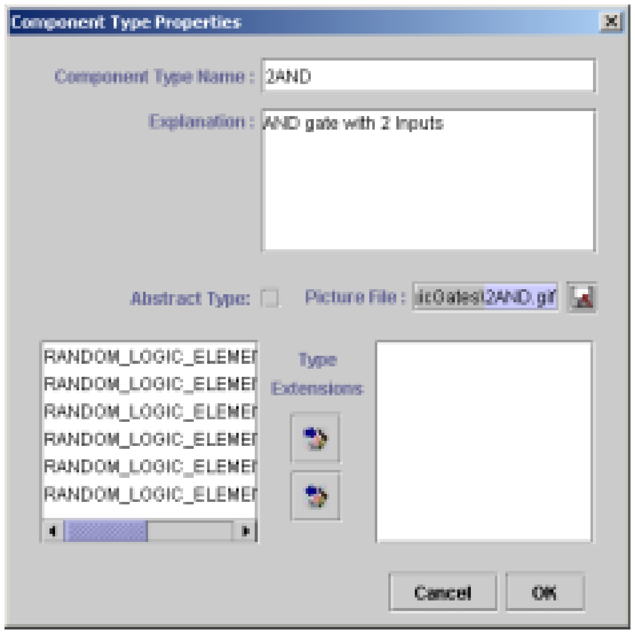
Name : PropagationDelay
Type : Float
Explanation :
Required : ✓
Unit : nanosec
Decimals :
Array : ☐
35.0
Min Value :
Max Value :
0.0..200.0

Figure 79. Primitive Properties



Role Name : Input_2
Explanation :
FROM Cardinality : n
TO Cardinality : n
Position : Coordinates
X : 0
Y : 20

Figure 80. Composition Cardinality


Component Type Properties

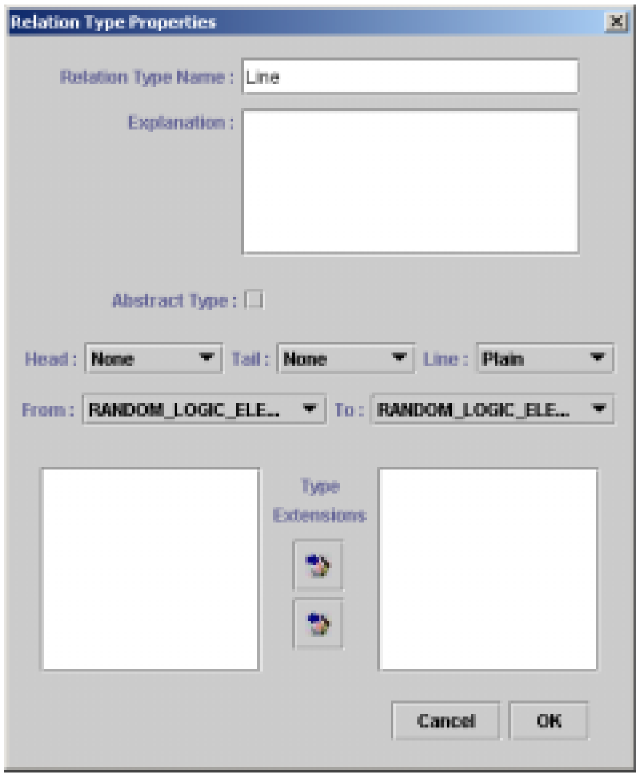
Component Type Name : 2AND

Explanation : AND gate with 2 inputs

Abstract Type: ☐ Picture File : ic\Gates\2AND.gif

RANDOM_LOGIC_ELEMEI
RANDOM_LOGIC_ELEMEI
RANDOM_LOGIC_ELEMEI
RANDOM_LOGIC_ELEMEI
RANDOM_LOGIC_ELEMEI
RANDOM_LOGIC_ELEMEI

Figure 81. Setting Component Image


Relation Type Properties

Relation Type Name : Line

Explanation :

Abstract Type : ☐

Head : None Tail : None Line : Plain

From : RANDOM_LOGIC_ELE... To : RANDOM_LOGIC_ELE...

Figure 82. Relation Type Properties

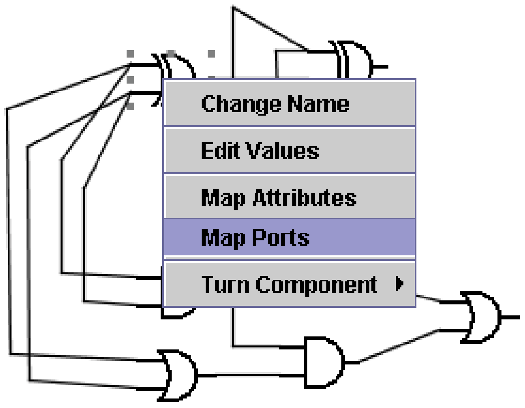
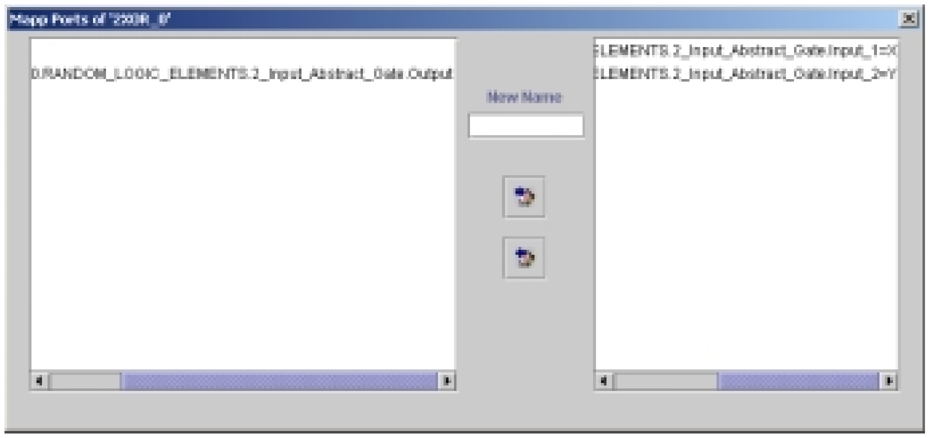


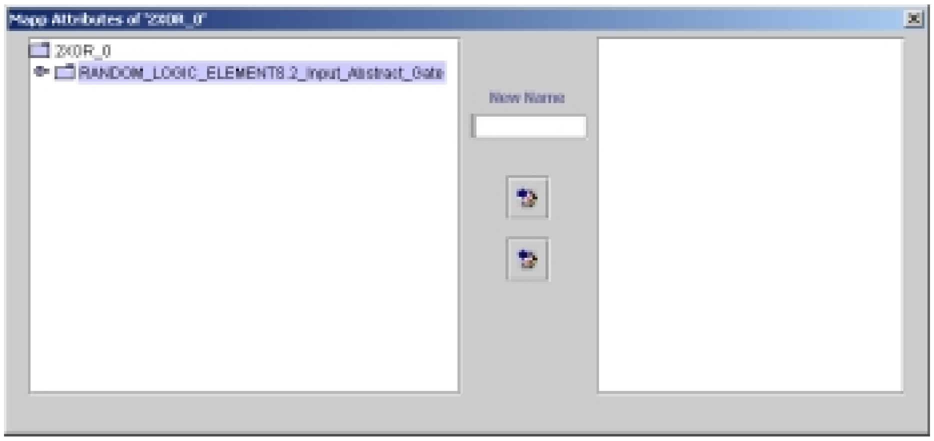
Figure 83. Component Pop-Up Menu


Mapp Ports of '2XOR_0'

0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Output

ELEMENTS.2_Input_Abstract_Gate.Input_1=X
ELEMENTS.2_Input_Abstract_Gate.Input_2=Y

Figure 84. Mapping Ports


Mapp Attributes of '2XOR_0'

2XOR_0
RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate

Figure 85. Mapping Attributes

APPENDIX C. XML DOCUMENTS OF CASE STUDIES

Section 1: Digital Circuitry

a) Random Logic Elements

```
- <LIBRARY LibraryName="RANDOM_LOGIC_ELEMENTS" Explanation="This Library contains
      Random Logic Elements">
    <PortType TypeName="Signal_Port" Explanation=""
        NodeLayout="297.8704833984375,103.53668212890625,true" Symbol="Line" />
    <RelationType TypeName="Line" Explanation="" HeadSymbol="None" TailSymbol="None"
        LineSymbol="Plain" FromPortType="RANDOM_LOGIC_ELEMENTS.Signal_Port"
        ToPortType="RANDOM_LOGIC_ELEMENTS.Signal_Port"
        FromPortRelationLayout="602.75927734375,132.11111450195312,12.0,8.361122131347656,true"
        ToPortRelationLayout="498.0,45.0,-11.0,-14.0,true"
        NodeLayout="533.6858520507812,63.786407470703125,true" />
  - <ComponentType TypeName="2_Input_Abstract_Gate" Explanation="Abstract gate definition with
        delay time, 2 input and 1 output ports" Abstract="True" PictureFile=""
        NodeLayout="272.5833740234375,265.39776611328125,true,true">
    - <Attributes>
        <Float Explanation="" Unit="nanosec" Required="True" Range="0.0..200.0"
            Value="35.0">PropagationDelay</Float>
      </Attributes>
    - <Ports>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="0"
            Top="5" RelationLayout="-21.0,35.0,true"
            Value=",">Input_1</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="0"
            Top="20" RelationLayout="283.0,188.0,-86.85232543945312,14.84228515625,true"
            Value=",">Input_2</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="42"
            Top="13" RelationLayout="409.0,183.0,27.0,21.0,true"
            Value=",">Output</RANDOM_LOGIC_ELEMENTS.Signal_Port>
      </Ports>
    </ComponentType>
  - <ComponentType TypeName="2AND" Explanation="AND gate with 2 Inputs"
        PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\DigitalSIM\LogicGates\2AND.gi
        f" NodeLayout="165.58334350585938,366.2310791015625,true,true">
      <Extension ExtendedTypeName="RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate"
          RelationLayout="" />
      <Attributes />
      <Ports />
    </ComponentType>
  - <ComponentType TypeName="2NAND" Explanation="NAND gate with 2 Inputs"
        PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\DigitalSIM\LogicGates\2NAND.
        gif" NodeLayout="251.5833740234375,425.2310791015625,true,true">
      <Extension ExtendedTypeName="RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate"
          RelationLayout="" />
      <Attributes />
      <Ports />
  </ComponentType>
- <ComponentType TypeName="2OR" Explanation="OR gate with 2 Inputs"
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\DigitalSIM\LogicGates\2OR.gif"
      NodeLayout="357.7545166015625,433.8416748046875,true,true">
    <Extension ExtendedTypeName="RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate"
        RelationLayout="" />
    <Attributes />
    <Ports />
  </ComponentType>
- <ComponentType TypeName="2NOR" Explanation="NOR gate with 2 Inputs"
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\DigitalSIM\LogicGates\2NOR.gi
      f" NodeLayout="442.8994140625,432.6951904296875,true,true">
    <Extension ExtendedTypeName="RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate"
        RelationLayout="" />
    <Attributes />
    <Ports />
  </ComponentType>
- <ComponentType TypeName="2XOR" Explanation="XOR with 2 input"
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\DigitalSIM\LogicGates\2XOR.gi
      f" NodeLayout="527.0,360.0,true,true">
    <Extension ExtendedTypeName="RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate"
        RelationLayout="" />
    <Attributes />
    <Ports />
  </ComponentType>
- <ComponentType TypeName="NOT" Explanation="Negation gate"
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\DigitalSIM\LogicGates\NOT.gif"
      NodeLayout="59.0,107.0,true,true">
    <Attributes />
  - <Ports>
      <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="0"
          Top="13" RelationLayout="224.0,101.0,-49.0,11.0,true"
          Value=",">Input</RANDOM_LOGIC_ELEMENTS.Signal_Port>
      <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="42"
          Top="13" RelationLayout="234.0,64.0,-33.0,-20.0,true"
          Value=",">Output</RANDOM_LOGIC_ELEMENTS.Signal_Port>
    </Ports>
  </ComponentType>
</LIBRARY>
```

b) Bus Structures

```
- <LIBRARY LibraryName="RANDOM_LOGIC_BUS_STRUCTURES" Explanation="Contains elements
      required to poll single lines into busses" Uses="RANDOM_LOGIC_ELEMENTS">
  <PortType TypeName="16BitBusPort" Explanation="" NodeLayout="180.0,142.0,true"
      Symbol="Square" />
  <RelationType TypeName="16BitBus" Explanation="" HeadSymbol="None" TailSymbol="None"
      LineSymbol="Double" FromPortType="RANDOM_LOGIC_BUS_STRUCTURES.16BitBusPort"
      ToPortType="RANDOM_LOGIC_BUS_STRUCTURES.16BitBusPort"
      FromPortRelationLayout="170.0,86.0,-46.0,8.0,true"
      ToPortRelationLayout="283.0,88.0,21.0,7.0,true" NodeLayout="189.0,32.0,true" />
- <ComponentType TypeName="1TO16" Explanation="Transforms 1 bus input into 16 bits"
      PictureFile="" NodeLayout="59.0,241.0,true,true">
    <Attributes />
  - <Ports>
```

```
        <RANDOM_LOGIC_BUS_STRUCTURES.16BitBusPort Explanation="" From="n" To="n"
            Left="-1" Top="-1" RelationLayout="-41.0,20.0,true"
            Value=",">Bus</RANDOM_LOGIC_BUS_STRUCTURES.16BitBusPort>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">0</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">1</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">2</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">3</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">4</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">5</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">6</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">7</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">8</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">9</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">10</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">11</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">12</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">13</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">14</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-2" Top="-
            2" Value=",">15</RANDOM_LOGIC_ELEMENTS.Signal_Port>
      </Ports>
    </ComponentType>
  - <ComponentType TypeName="16TO1" Explanation="Transforms 16 bits input into a bus"
        PictureFile="" NodeLayout="305.0,252.0,true,true">
      <Attributes />
    - <Ports>
        <RANDOM_LOGIC_BUS_STRUCTURES.16BitBusPort Explanation="" From="n" To="n"
            Left="-2" Top="-2" RelationLayout="20.0,19.0,true"
            Value=",">Bus</RANDOM_LOGIC_BUS_STRUCTURES.16BitBusPort>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" Left="-1" Top="-1"
            Value=",">0</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">1</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">2</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">3</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">4</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">5</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">6</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">7</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">8</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">9</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">10</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">11</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">12</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">13</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">14</RANDOM_LOGIC_ELEMENTS.Signal_Port>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="-1" Top="-
            1" Value=",">15</RANDOM_LOGIC_ELEMENTS.Signal_Port>
      </Ports>
    </ComponentType>
  </LIBRARY>
```

c) Sources

```
- <LIBRARY LibraryName="RANDOM_LOGIC_SOURCES" Explanation="This library contains source
      elements for digital circuit design"
      Uses="RANDOM_LOGIC_ELEMENTS,RANDOM_LOGIC_BUS_STRUCTURES">
  - <ComponentType TypeName="Abstract_Generator" Explanation="" Abstract="True" PictureFile=""
        NodeLayout="158.0,26.0,true,true">
      <Attributes />
    - <Ports>
        <RANDOM_LOGIC_ELEMENTS.Signal_Port Explanation="" From="n" To="n" Left="20"
            Top="9" Value=",">Out</RANDOM_LOGIC_ELEMENTS.Signal_Port>
      </Ports>
    </ComponentType>
  - <ComponentType TypeName="0" Explanation="Produces logical 0"
        PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\DigitalSIM\LogicGates\0_Gener
        ator.gif" NodeLayout="97.0,169.0,true,true">
      <Extension ExtendedTypeName="RANDOM_LOGIC_SOURCES.Abstract_Generator"
          RelationLayout="" />
      <Attributes />
      <Ports />
    </ComponentType>
  - <ComponentType TypeName="1" Explanation="Generates logical 1"
        PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\DigitalSIM\LogicGates\1_Gener
        ator.gif" NodeLayout="207.0,208.0,true,true">
      <Extension ExtendedTypeName="RANDOM_LOGIC_SOURCES.Abstract_Generator"
          RelationLayout="" />
      <Attributes />
      <Ports />
    </ComponentType>
  - <ComponentType TypeName="Square_Wave" Explanation="Generates square signal with given
        period"
        PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\DigitalSIM\LogicGates\Square_
        Wave_Generator.gif" NodeLayout="303.0,174.0,true,true">
      <Extension ExtendedTypeName="RANDOM_LOGIC_SOURCES.Abstract_Generator"
          RelationLayout="" />
    - <Attributes>
        <Float Explanation="frequancy of the square wave" Unit="MHz" Required="True">freq</Float>
      </Attributes>
      <Ports />
    </ComponentType>
  - <ComponentType TypeName="Number_Generator" Explanation="Produces Numbers on the data
        output shifting with the given frequency"
        PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\DigitalSIM\LogicGates\Number
        _Generator.gif" NodeLayout="142.0,351.0,true,true">
    - <Attributes>
        <Float Explanation="frequency of changing to next number in the list" Unit="MHz"
            Required="True">freq</Float>
        <Int Explanation="Numbers list to be produced" Unit="" Required="True"
            Range="0..65365">RepetingNumbers[1,n]</Int>
      </Attributes>
    - <Ports>
        <RANDOM_LOGIC_BUS_STRUCTURES.16BitBusPort Explanation="" From="n" Left="29"
            Top="14" Value=",">Data</RANDOM_LOGIC_BUS_STRUCTURES.16BitBusPort>
      </Ports>
    </ComponentType>
  </LIBRARY>
```

d) Two-Bit Adder

```
- <LIBRARY LibraryName="TwoBitAdderLibrary" Explanation="Contains the scenario of two bit adder
      composed of random logic elements"
      Uses="RANDOM_LOGIC_ELEMENTS,RANDOM_LOGIC_BUS_STRUCTURES,RANDOM_LO
      GIC_SOURCES">
  - <ScenarioType TypeName="2BitAdder" Explanation="" Component="True" PictureFile="">
    - <Components>
        <RANDOM_LOGIC_ELEMENTS.2XOR NodeLayout="102.0,50.0" NodeAlignment="NORMAL"
            Value="{{,{35.0},{,,,,,}}},,">2XOR_0</RANDOM_LOGIC_ELEMENTS.2XOR>
        <RANDOM_LOGIC_ELEMENTS.2AND NodeLayout="105.0,163.0"
            NodeAlignment="NORMAL"
            Value="{{,{35.0},{,,,,,}}},,">2AND_0</RANDOM_LOGIC_ELEMENTS.2AND>
        <RANDOM_LOGIC_ELEMENTS.2XOR NodeLayout="196.0,43.0" NodeAlignment="NORMAL"
            Value="{{,{35.0},{,,,,,}}},,">2XOR_1</RANDOM_LOGIC_ELEMENTS.2XOR>
        <RANDOM_LOGIC_ELEMENTS.NOT NodeLayout="156.0,89.0" NodeAlignment="CW"
            Value=",,{,,,}">NOT_0</RANDOM_LOGIC_ELEMENTS.NOT>
        <RANDOM_LOGIC_ELEMENTS.2AND NodeLayout="212.0,119.0"
            NodeAlignment="NORMAL"
            Value="{{,{35.0},{,,,,,}}},,">2AND_1</RANDOM_LOGIC_ELEMENTS.2AND>
        <RANDOM_LOGIC_ELEMENTS.2AND NodeLayout="215.0,228.0"
            NodeAlignment="NORMAL"
            Value="{{,{35.0},{,,,,,}}},,">2AND_2</RANDOM_LOGIC_ELEMENTS.2AND>
        <RANDOM_LOGIC_ELEMENTS.2OR NodeLayout="292.0,147.0" NodeAlignment="NORMAL"
            Value="{{,{35.0},{,,,,,}}},,">2OR_1</RANDOM_LOGIC_ELEMENTS.2OR>
        <RANDOM_LOGIC_ELEMENTS.NOT NodeLayout="24.0,219.0" NodeAlignment="CW"
            Value=",,{,,,}">NOT_1</RANDOM_LOGIC_ELEMENTS.NOT>
        <RANDOM_LOGIC_ELEMENTS.2AND NodeLayout="115.0,249.0"
            NodeAlignment="NORMAL"
            Value="{{,{35.0},{,,,,,}}},,">2AND_3</RANDOM_LOGIC_ELEMENTS.2AND>
        <RANDOM_LOGIC_ELEMENTS.NOT NodeLayout="32.0,266.0" NodeAlignment="CW"
            Value=",,{,,,}">NOT_2</RANDOM_LOGIC_ELEMENTS.NOT>
        <RANDOM_LOGIC_ELEMENTS.2AND NodeLayout="116.0,296.0"
            NodeAlignment="NORMAL"
            Value="{{,{35.0},{,,,,,}}},,">2AND_4</RANDOM_LOGIC_ELEMENTS.2AND>
    <RANDOM_LOGIC_ELEMENTS.2OR NodeLayout="381.0,185.0" NodeAlignment="NORMAL"
        Value="{{,{35.0},{,,,,,}}},,">2OR_0</RANDOM_LOGIC_ELEMENTS.2OR>
    <RANDOM_LOGIC_ELEMENTS.2AND NodeLayout="211.0,288.0"
        NodeAlignment="NORMAL"
        Value="{{,{35.0},{,,,,,}}},,">2AND_5</RANDOM_LOGIC_ELEMENTS.2AND>
    <RANDOM_LOGIC_ELEMENTS.2AND NodeLayout="214.0,170.0"
        NodeAlignment="NORMAL"
        Value="{{,{35.0},{,,,,,}}},,">2AND_6</RANDOM_LOGIC_ELEMENTS.2AND>
    <RANDOM_LOGIC_ELEMENTS.2OR NodeLayout="294.0,259.0" NodeAlignment="NORMAL"
        Value="{{,{35.0},{,,,,,}}},,">2OR_2</RANDOM_LOGIC_ELEMENTS.2OR>
  </Components>
- <Connections>
    <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
        From="2XOR_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1"
        To="2AND_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1"
        Value=","
        RelationLayout="61.0,55.0,61.0,168.0">Line_0</RANDOM_LOGIC_ELEMENTS.Line>
    <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
        From="2XOR_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2"
        To="2AND_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2"
        Value=","
        RelationLayout="77.0,71.0,77.0,183.0">Line_1</RANDOM_LOGIC_ELEMENTS.Line>
    <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
        From="2XOR_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Output"
        To="2XOR_1.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2"
        Value="," RelationLayout="">Line_2</RANDOM_LOGIC_ELEMENTS.Line>
    <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
        From="2XOR_1.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1"
        To="NOT_0.Input" Value=","
        RelationLayout="179.0,48.0">Line_5</RANDOM_LOGIC_ELEMENTS.Line>
    <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="NOT_0.Output"
        To="2AND_1.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1"
        Value="," RelationLayout="">Line_6</RANDOM_LOGIC_ELEMENTS.Line>
    <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
        From="2AND_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Output"
        To="2AND_1.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2"
        Value="," RelationLayout="">Line_7</RANDOM_LOGIC_ELEMENTS.Line>
    <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
        From="2XOR_1.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1"
        To="2AND_2.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1"
        Value=","
        RelationLayout="156.0,25.0,156.0,205.0">Line_9</RANDOM_LOGIC_ELEMENTS.Line>
    <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
        From="2AND_1.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Output"
        To="2OR_1.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1" Value=","
        RelationLayout="">Line_10</RANDOM_LOGIC_ELEMENTS.Line>
    <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
        From="2AND_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1"
        To="NOT_1.Input" Value=","
        RelationLayout="">Line_3</RANDOM_LOGIC_ELEMENTS.Line>
    <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="NOT_1.Output"
        To="2AND_3.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1"
        Value="," RelationLayout="">Line_4</RANDOM_LOGIC_ELEMENTS.Line>
    <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
        From="2AND_3.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Output"
        To="2AND_2.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2"
        Value="," RelationLayout="">Line_8</RANDOM_LOGIC_ELEMENTS.Line>
    <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
        From="2AND_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2"
```

```
          To="2AND_3.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2"
          Value=","
          RelationLayout="85.0,244.0,86.0,268.0">Line_12</RANDOM_LOGIC_ELEMENTS.Line>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
          From="2AND_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2"
          To="NOT_2.Input" Value=","
          RelationLayout="">Line_13</RANDOM_LOGIC_ELEMENTS.Line>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="NOT_2.Output"
          To="2AND_4.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1"
          Value="," RelationLayout="">Line_14</RANDOM_LOGIC_ELEMENTS.Line>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
          From="2AND_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1"
          To="2AND_4.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2"
          Value=","
          RelationLayout="24.0,201.0,24.0,316.0">Line_15</RANDOM_LOGIC_ELEMENTS.Line>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
          From="2AND_2.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1"
          To="2AND_5.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1"
          Value=","
          RelationLayout="171.0,233.0,173.0,294.0">Line_16</RANDOM_LOGIC_ELEMENTS.Line>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
          From="2AND_4.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Output"
          To="2AND_5.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2"
          Value="," RelationLayout="">Line_17</RANDOM_LOGIC_ELEMENTS.Line>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
          From="2OR_1.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Output"
          To="2OR_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1" Value=","
          RelationLayout="">Line_19</RANDOM_LOGIC_ELEMENTS.Line>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
          From="2AND_2.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Output"
          To="2OR_2.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1" Value=","
          RelationLayout="">Line_11</RANDOM_LOGIC_ELEMENTS.Line>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
          From="2AND_5.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Output"
          To="2OR_2.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2" Value=","
          RelationLayout="">Line_20</RANDOM_LOGIC_ELEMENTS.Line>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
          From="2OR_2.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Output"
          To="2OR_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2" Value=","
          RelationLayout="">Line_18</RANDOM_LOGIC_ELEMENTS.Line>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
          From="2AND_2.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1"
          To="2AND_6.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2"
          Value="," RelationLayout="">Line_21</RANDOM_LOGIC_ELEMENTS.Line>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
          From="2AND_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Output"
          To="2AND_6.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1"
          Value="," RelationLayout="">Line_22</RANDOM_LOGIC_ELEMENTS.Line>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation=""
          From="2AND_6.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Output"
          To="2OR_1.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2" Value=","
          RelationLayout="">Line_23</RANDOM_LOGIC_ELEMENTS.Line>
   </Connections>
 - <Interface>
      <PortMaps>2XOR_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1=X,
          2XOR_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_2=Y,
          2XOR_1.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Input_1=CarryIN,
          2XOR_1.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Output=Sum,
         2OR_0.RANDOM_LOGIC_ELEMENTS.2_Input_Abstract_Gate.Output=CarryOUT</Po
         rtMaps>
      </Interface>
   </ScenarioType>
</LIBRARY>
```

e) Four-Bit Adder

```
- <LIBRARY LibraryName="FourBitAdderLibrary" Explanation=""
      Uses="RANDOM_LOGIC_ELEMENTS,TwoBitAdderLibrary">
  - <ScenarioType TypeName="4BitAdder" Explanation="" Component="True" PictureFile="">
    - <Components>
        <TwoBitAdderLibrary.2BitAdder NodeLayout="130.0,75.0" NodeAlignment="NORMAL"
            Value="{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.
            0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,
            {{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,,}
            }">2BitAdder_0</TwoBitAdderLibrary.2BitAdder>
        <TwoBitAdderLibrary.2BitAdder NodeLayout="210.0,168.0" NodeAlignment="NORMAL"
            Value="{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.
            0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,
            {{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,,}
            }">2BitAdder_1</TwoBitAdderLibrary.2BitAdder>
        <TwoBitAdderLibrary.2BitAdder NodeLayout="287.0,260.0" NodeAlignment="NORMAL"
            Value="{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.
            0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,
            {{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,,}
            }">2BitAdder_2</TwoBitAdderLibrary.2BitAdder>
        <TwoBitAdderLibrary.2BitAdder NodeLayout="358.0,361.0" NodeAlignment="NORMAL"
            Value="{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.
            0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,
            {{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,,}
            }">2BitAdder_3</TwoBitAdderLibrary.2BitAdder>
      </Components>
    - <Connections>
        <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="2BitAdder_0.CarryOUT"
            To="2BitAdder_1.CarryIN" Value=","
            RelationLayout="312.0,144.0">Line_0</RANDOM_LOGIC_ELEMENTS.Line>
        <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="2BitAdder_1.CarryOUT"
            To="2BitAdder_2.CarryIN" Value=","
            RelationLayout="387.0,234.0">Line_1</RANDOM_LOGIC_ELEMENTS.Line>
        <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="2BitAdder_2.CarryOUT"
            To="2BitAdder_3.CarryIN" Value=","
            RelationLayout="460.0,322.0">Line_2</RANDOM_LOGIC_ELEMENTS.Line>
      </Connections>
    - <Interface>
        <PortMaps>2BitAdder_0.CarryIN=CarryIN, 2BitAdder_0.X=X_0, 2BitAdder_0.Y=Y_0,
            2BitAdder_0.Sum=Sum_0, 2BitAdder_1.X=X_1, 2BitAdder_1.Y=Y_1,
            2BitAdder_1.Sum=Sum_1, 2BitAdder_2.X=X_2, 2BitAdder_2.Y=Y_2,
            2BitAdder_2.Sum=Sum_2, 2BitAdder_3.X=X_3, 2BitAdder_3.Y=Y_3,
            2BitAdder_3.Sum=Sum_3, 2BitAdder_3.CarryOUT=CarryOUT</PortMaps>
      </Interface>
   </ScenarioType>
</LIBRARY>
```

f) 16-Bit Adder

```
- <LIBRARY LibraryName="SixteenBitAdderLibrary" Explanation=""
     Uses="RANDOM_LOGIC_ELEMENTS,TwoBitAdderLibrary,FourBitAdderLibrary">
  - <ScenarioType TypeName="16BitAdder" Explanation="" Component="True" PictureFile="">
    - <Components>
        <FourBitAdderLibrary.4BitAdder NodeLayout="18.0,35.0" NodeAlignment="NORMAL"
          Value="{{{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{3
          5.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}}
          ,,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,
          ,}},{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,
          ,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{3
          5.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,}},{{{{
          ,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{
          {,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,
          ,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,}},{{{{,{35.0},
          {,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0
          },{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{
          {,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,}}},{,,,,,},,{,,,,,,,,,,,,,,
          ,,,,,,,,,,}}">4BitAdder_0</FourBitAdderLibrary.4BitAdder>
        <FourBitAdderLibrary.4BitAdder NodeLayout="176.0,138.0" NodeAlignment="NORMAL"
          Value="{{{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{3
          5.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}}
          ,,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,
          ,}},{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,
          ,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{3
          5.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,}},{{{{
          ,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{
          {,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,
          ,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,}},{{{{,{35.0},
          {,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0
          },{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{
          {,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,}}},{,,,,,},,{,,,,,,,,,,,,,,
          ,,,,,,,,,,}}">4BitAdder_1</FourBitAdderLibrary.4BitAdder>
        <FourBitAdderLibrary.4BitAdder NodeLayout="325.0,238.0" NodeAlignment="NORMAL"
          Value="{{{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{3
          5.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}}
          ,,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,
          ,}},{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,
          ,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{3
          5.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,}},{{{{
          ,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{
          {,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,
          ,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,}},{{{{,{35.0},
          {,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0
          },{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{
          {,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,}}},{,,,,,},,{,,,,,,,,,,,,,,
          ,,,,,,,,,,}}">4BitAdder_2</FourBitAdderLibrary.4BitAdder>
        <FourBitAdderLibrary.4BitAdder NodeLayout="477.0,348.0" NodeAlignment="NORMAL"
          Value="{{{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{3
          5.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}}
          ,,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,
          ,}},{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,
          ,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{3
          5.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,}},{{{{
          ,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{
          {,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,
          ,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,}},{{{{,{35.0},
          {,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0
          },{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{
```

```
        {,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,,}}},{,,,,,},,{,,,,,,,,,,,,,,,,
        ,,,,,,,,,,}}">4BitAdder_3</FourBitAdderLibrary.4BitAdder>
    </Components>
  - <Connections>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="4BitAdder_0.CarryOUT"
        To="4BitAdder_1.CarryIN" Value=","
        RelationLayout="213.0,116.0">Line_0</RANDOM_LOGIC_ELEMENTS.Line>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="4BitAdder_1.CarryOUT"
        To="4BitAdder_2.CarryIN" Value=","
        RelationLayout="368.0,219.0">Line_1</RANDOM_LOGIC_ELEMENTS.Line>
      <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="4BitAdder_2.CarryOUT"
        To="4BitAdder_3.CarryIN" Value=","
        RelationLayout="528.0,316.0">Line_2</RANDOM_LOGIC_ELEMENTS.Line>
    </Connections>
  - <Interface>
      <PortMaps>4BitAdder_0.CarryIN=CarryIN, 4BitAdder_0.X_0=X_0, 4BitAdder_0.X_1=X_1,
        4BitAdder_0.X_2=X_2, 4BitAdder_0.X_3=X_3, 4BitAdder_1.X_0=X_4,
        4BitAdder_1.X_1=X_5, 4BitAdder_1.X_2=X_6, 4BitAdder_1.X_3=X_7,
        4BitAdder_2.X_0=X_8, 4BitAdder_2.X_1=X_9, 4BitAdder_2.X_2=X_10,
        4BitAdder_2.X_3=X_11, 4BitAdder_3.X_0=X_12, 4BitAdder_3.X_1=X_13,
        4BitAdder_3.X_2=X_14, 4BitAdder_3.X_3=X_15, 4BitAdder_0.Y_0=Y_0,
        4BitAdder_0.Y_1=Y_1, 4BitAdder_0.Y_2=Y_2, 4BitAdder_0.Y_3=Y_3,
        4BitAdder_1.Y_0=Y_4, 4BitAdder_1.Y_1=Y_5, 4BitAdder_1.Y_2=Y_6,
        4BitAdder_1.Y_3=Y_7, 4BitAdder_2.Y_0=Y_8, 4BitAdder_2.Y_1=Y_9,
        4BitAdder_2.Y_2=Y_10, 4BitAdder_2.Y_3=Y_11, 4BitAdder_3.Y_0=Y_12,
        4BitAdder_3.Y_1=Y_13, 4BitAdder_3.Y_2=Y_14, 4BitAdder_3.Y_3=Y_15,
        4BitAdder_0.Sum_0=Sum_0, 4BitAdder_0.Sum_1=Sum_1, 4BitAdder_0.Sum_2=Sum_2,
        4BitAdder_0.Sum_3=Sum_3, 4BitAdder_1.Sum_0=Sum_4, 4BitAdder_1.Sum_1=Sum_5,
        4BitAdder_1.Sum_2=Sum_6, 4BitAdder_1.Sum_3=Sum_7, 4BitAdder_2.Sum_0=Sum_8,
        4BitAdder_2.Sum_1=Sum_9, 4BitAdder_2.Sum_2=Sum_10, 4BitAdder_2.Sum_3=Sum_11,
        4BitAdder_3.Sum_0=Sum_12, 4BitAdder_3.Sum_1=Sum_13,
        4BitAdder_3.Sum_2=Sum_14, 4BitAdder_3.Sum_3=Sum_15,
        4BitAdder_3.CarryOUT=CarryOUT</PortMaps>
    </Interface>
  </ScenarioType>
</LIBRARY>
```

g) 16-Bit Full Adder

```
- <LIBRARY LibraryName="SixteenBitFullAdderLibrary" Explanation=""
    Uses="RANDOM_LOGIC_ELEMENTS,RANDOM_LOGIC_SOURCES,TwoBitAdderLibrary,Fo
    urBitAdderLibrary,SixteenBitAdderLibrary,RANDOM_LOGIC_BUS_STRUCTURES">
  - <ScenarioType TypeName="16BitFullAdder" Explanation="" Component="True" PictureFile="">
    - <Components>
        <SixteenBitAdderLibrary.16BitAdder NodeLayout="17.0,80.0" NodeAlignment="CW"
          Value="{{{{{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,
          {35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,
          }}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,
          ,,,,,}},{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},
          {,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,
          {35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,,}},{
          {{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}}
          ,,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},
          {,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,,}},{{{{,{35.
          0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{3
          5.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}}
          ,,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,},,{,,,,,,,,,}}},{,,,,,},,{,,,,,,,,,,
```

,,,,,,,,,,,,,,}},{{{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{
{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,
,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,},,{,,
,,,,,,,}},{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0
},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{
{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,},,{,,,,,,,,,}},
{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}
},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0}
,{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,},,{,,,,,,,,,}},{{{{,{35
.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{3
5.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}}
,,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,},,{,,,,,,,,,}}},{,,,,,},,{,,,,,,,,,,
,,,,,,,,,,,,,,}},{{{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{
{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,
,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,},,{,,
,,,,,,,}},{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0
},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{
{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,},,{,,,,,,,,,}},
{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}
},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0}
,{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,},,{,,,,,,,,,}},{{{{,{35
.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{3
5.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}}
,,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,},,{,,,,,,,,,}}},{,,,,,},,{,,,,,,,,,,
,,,,,,,,,,,,,,}},{{{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{
{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,
,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,},,{,,
,,,,,,,}},{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0
},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{
{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,},,{,,,,,,,,,}},
{{{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}
},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0}
,{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,},,{,,,,,,,,,}},{{{{,{35
.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{3
5.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,,,{,,,},{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}}
,,,{{,{35.0},{,,,,,}}},,,{{,{35.0},{,,,,,}}},,},{,,},,{,,,,,,,,,}}},{,,,,,},,{,,,,,,,,,,
,,,,,,,,,,,,,,}}},{,,,,,},,{,,}}">16BitAdd
er_0</SixteenBitAdderLibrary.16BitAdder>
<RANDOM_LOGIC_SOURCES.0 NodeLayout="520.0,31.0" NodeAlignment="CW"
Value="{{,,{,}}},,">0_0</RANDOM_LOGIC_SOURCES.0>
<RANDOM_LOGIC_BUS_STRUCTURES.16TO1 NodeLayout="540.0,71.0"
NodeAlignment="NORMAL"
Value=",,{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,}">16TO1_0</RANDOM_LOGIC_BUS_STRUCTURES.16TO
1>
<RANDOM_LOGIC_BUS_STRUCTURES.16TO1 NodeLayout="548.0,296.0"
NodeAlignment="NORMAL"
Value=",,{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,}">16TO1_1</RANDOM_LOGIC_BUS_STRUCTURES.16TO
1>
<RANDOM_LOGIC_BUS_STRUCTURES.16TO1 NodeLayout="147.0,523.0"
NodeAlignment="CW"
Value=",,{,,,,,,,,,,,,,,,,,,,,,,,,,,,,,}">16TO1_2</RANDOM_LOGIC_BUS_STRUCTURES.16TO
1>
</Components>
- <Connections>
<RANDOM_LOGIC_ELEMENTS.Line Explanation=""
From="0_0.RANDOM_LOGIC_SOURCES.Abstract_Generator.Out"
To="16BitAdder_0.CarryIN" Value=","
RelationLayout="">Line_16</RANDOM_LOGIC_ELEMENTS.Line>

```
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_0"
    To="16TO1_0.0" Value=","
    RelationLayout="">Line_0</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_1"
    To="16TO1_0.1" Value=","
    RelationLayout="">Line_1</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_2"
    To="16TO1_0.2" Value=","
    RelationLayout="">Line_2</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_3"
    To="16TO1_0.3" Value=","
    RelationLayout="">Line_3</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_4"
    To="16TO1_0.4" Value=","
    RelationLayout="">Line_4</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_5"
    To="16TO1_0.5" Value=","
    RelationLayout="">Line_5</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_6"
    To="16TO1_0.6" Value=","
    RelationLayout="">Line_6</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_7"
    To="16TO1_0.7" Value=","
    RelationLayout="">Line_7</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_8"
    To="16TO1_0.8" Value=","
    RelationLayout="">Line_8</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16TO1_0.9"
    To="16BitAdder_0.X_9" Value=","
    RelationLayout="">Line_9</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_10"
    To="16TO1_0.10" Value=","
    RelationLayout="">Line_10</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_11"
    To="16TO1_0.11" Value=","
    RelationLayout="">Line_11</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_12"
    To="16TO1_0.12" Value=","
    RelationLayout="">Line_12</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_13"
    To="16TO1_0.13" Value=","
    RelationLayout="">Line_13</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_14"
    To="16TO1_0.14" Value=","
    RelationLayout="">Line_14</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.X_15"
    To="16TO1_0.15" Value=","
    RelationLayout="">Line_15</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_0"
    To="16TO1_1.0" Value=","
    RelationLayout="">Line_17</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_1"
    To="16TO1_1.1" Value=","
    RelationLayout="">Line_18</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_2"
    To="16TO1_1.2" Value=","
    RelationLayout="">Line_19</RANDOM_LOGIC_ELEMENTS.Line>
```

```
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_3"
      To="16TO1_1.3" Value=","
      RelationLayout="">Line_20</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_4"
      To="16TO1_1.4" Value=","
      RelationLayout="">Line_21</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_5"
      To="16TO1_1.5" Value=","
      RelationLayout="">Line_22</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_6"
      To="16TO1_1.6" Value=","
      RelationLayout="">Line_23</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_7"
      To="16TO1_1.7" Value=","
      RelationLayout="">Line_24</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_8"
      To="16TO1_1.8" Value=","
      RelationLayout="">Line_25</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_9"
      To="16TO1_1.9" Value=","
      RelationLayout="">Line_26</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_10"
      To="16TO1_1.10" Value=","
      RelationLayout="">Line_27</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_11"
      To="16TO1_1.11" Value=","
      RelationLayout="">Line_28</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_12"
      To="16TO1_1.12" Value=","
      RelationLayout="547.0,436.0">Line_29</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_13"
      To="16TO1_1.13" Value=","
      RelationLayout="">Line_30</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_14"
      To="16TO1_1.14" Value=","
      RelationLayout="">Line_31</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Y_15"
      To="16TO1_1.15" Value=","
      RelationLayout="">Line_32</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_0"
      To="16TO1_2.0" Value=","
      RelationLayout="">Line_33</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_1"
      To="16TO1_2.1" Value=","
      RelationLayout="">Line_34</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_2"
      To="16TO1_2.2" Value=","
      RelationLayout="">Line_35</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_3"
      To="16TO1_2.3" Value=","
      RelationLayout="">Line_36</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_4"
      To="16TO1_2.4" Value=","
      RelationLayout="">Line_37</RANDOM_LOGIC_ELEMENTS.Line>
<RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_5"
      To="16TO1_2.5" Value=","
      RelationLayout="">Line_38</RANDOM_LOGIC_ELEMENTS.Line>
```

```
        <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_6"
            To="16TO1_2.6" Value=","
            RelationLayout="">Line_39</RANDOM_LOGIC_ELEMENTS.Line>
        <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_7"
            To="16TO1_2.7" Value=","
            RelationLayout="">Line_40</RANDOM_LOGIC_ELEMENTS.Line>
        <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_8"
            To="16TO1_2.8" Value=","
            RelationLayout="">Line_41</RANDOM_LOGIC_ELEMENTS.Line>
        <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_9"
            To="16TO1_2.9" Value=","
            RelationLayout="">Line_42</RANDOM_LOGIC_ELEMENTS.Line>
        <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_10"
            To="16TO1_2.10" Value=","
            RelationLayout="">Line_43</RANDOM_LOGIC_ELEMENTS.Line>
        <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_11"
            To="16TO1_2.11" Value=","
            RelationLayout="">Line_44</RANDOM_LOGIC_ELEMENTS.Line>
        <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_12"
            To="16TO1_2.12" Value=","
            RelationLayout="">Line_45</RANDOM_LOGIC_ELEMENTS.Line>
        <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_13"
            To="16TO1_2.13" Value=","
            RelationLayout="">Line_46</RANDOM_LOGIC_ELEMENTS.Line>
        <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_14"
            To="16TO1_2.14" Value=","
            RelationLayout="">Line_47</RANDOM_LOGIC_ELEMENTS.Line>
        <RANDOM_LOGIC_ELEMENTS.Line Explanation="" From="16BitAdder_0.Sum_15"
            To="16TO1_2.15" Value=","
            RelationLayout="">Line_48</RANDOM_LOGIC_ELEMENTS.Line>
      </Connections>
    - <Interface>
        <PortMaps>16BitAdder_0.CarryOUT=OverFlow, 16TO1_2.Bus=Sum, 16TO1_0.Bus=X,
            16TO1_1.Bus=Y</PortMaps>
      </Interface>
    </ScenarioType>
  </LIBRARY>
```

Section 2: Network

a) Queue Elements

```
- <LIBRARY LibraryName="QUEUE" Explanation="Example Queue">
  - <DataType TypeName="Distribution" Explanation="defines parameters of a distribution"
        NodeLayout="258.0,426.0,true">
      <String Explanation="Type of Distribution" Required="True"
          Range="'Normal','Poisson','Binomial','Pareto'" Value="'Normal'">Type</String>
    - <Case Conditions="Type='Normal'">
        <Float Explanation="Mean paramater for normal distribution" Unit="" Decimal="5"
            Required="True" Value="150.0">Mean</Float>
        <Float Explanation="Variance of Normal distribution" Unit="" Decimal="5" Required="True"
            Value="15.0">Variance</Float>
      </Case>
    - <Case Conditions="Type='Binomial'">
```

```
        <Float Explanation="Probability" Unit="" Decimal="5" Required="True"
            Range="0.0..1.0">P</Float>
        <Long Explanation="Number of trials" Unit="" Required="True">n</Long>
      </Case>
    - <Case Conditions="Type='Poisson'">
        <Float Explanation="The average number of occurrences of the Poisson process" Unit=""
            Decimal="5" Required="True">a</Float>
      </Case>
    - <Case Conditions="Type='Pareto'">
        <Float Explanation="Shape parameter for Pareto distribution" Unit="" Decimal="5"
            Required="True">s</Float>
        <Float Explanation="Location parameter for Pareto distribution" Unit="" Decimal="5"
            Required="True">k</Float>
      </Case>
    </DataType>
  - <DataType TypeName="PacketCell" Explanation="Defines a cell of a packet with name size and
        explanation." NodeLayout="9.0,420.0,true">
      <Int Explanation="Size of cell inside the packet definition" Unit="bits" Required="True">Size</Int>
      <String Explanation="Name of the cell">Name</String>
      <String Explanation="An explanation about the properties of cell">Explanation</String>
    </DataType>
  - <DataType TypeName="RoutingAdresses" Explanation="" NodeLayout="524.0,592.0,true">
      <Int Explanation="" Unit="" Required="True">LinkNo</Int>
      <Int Explanation="" Unit="" Required="True">Address</Int>
    </DataType>
    <PortType TypeName="InputPort" Explanation="Input Port Definition"
        NodeLayout="355.0,185.0,true" Symbol="Line" />
    <PortType TypeName="Output" Explanation="Output Port Definition" NodeLayout="191.0,187.0,true"
        Symbol="Triangle" />
    <PortType TypeName="DuplexPort" Explanation="" NodeLayout="535.0,195.0,true" Symbol="Square"
        />
    <RelationType TypeName="Link" Explanation="Link without Delay" HeadSymbol="Arrow"
        TailSymbol="None" LineSymbol="Plain" FromPortType="QUEUE.Output"
        ToPortType="QUEUE.InputPort" FromPortRelationLayout="-44.0,9.0,true"
        ToPortRelationLayout="18.0,9.0,true" NodeLayout="288.0,101.0,true" />
  - <RelationType TypeName="DelayedLink" Explanation="Link with Delay" HeadSymbol="Arrow"
        TailSymbol="None" LineSymbol="Double" FromPortType="QUEUE.Output"
        ToPortType="QUEUE.InputPort" FromPortRelationLayout="222.0,100.0,-64.0,9.0,true"
        ToPortRelationLayout="395.0,93.0,37.0,9.0,true" NodeLayout="264.0,25.0,true">
      <Float Explanation="Delay time in seconds" Unit="seconds" Decimal="5"
          Required="True">Delay</Float>
    </RelationType>
  - <RelationType TypeName="DuplexPaidLink" Explanation="A link with a cost value for usage"
        HeadSymbol="None" TailSymbol="None" LineSymbol="DashDot"
        FromPortType="QUEUE.DuplexPort" ToPortType="QUEUE.DuplexPort"
        FromPortRelationLayout="523.0,144.0,-51.0,29.0,true"
        ToPortRelationLayout="627.0,144.0,32.0,30.0,true" NodeLayout="507.0,16.0,true">
      <Float Explanation="The time for a single bit to be transfered betweeb two connection points,
          depending on the type and length of link" Unit="seconds" Decimal="4" Required="True"
          Range="0.0..6000.0">Propagation</Float>
      <Float Explanation="The money paid for each byte of data transfer" Unit="cents per byte"
          Decimal="5" Range="0.0..1000.0">CostPerTransfer</Float>
      <Float Explanation="Money paid monthly for a link independent from the usage" Unit="Dollars"
          Decimal="2" Required="True" Range="0.0..1000000.0">MonthlyCost</Float>
      <Float Explanation="Transfer rate of the link" Unit="bits/sec" Required="True">Rate</Float>
    </RelationType>
  - <ComponentType TypeName="Source" Explanation="Produces" PictureFile=""
        NodeLayout="95.0,301.0,true,true">
    - <Attributes>
```

```
        <QUEUE.PacketCell Explanation="Defines bit level attributes of the packet to be produced"
            Required="True" RelationLayout="-62.844696044921875,-10.0,23.0,-
            10.0,true,true">PacketType[1,500]</QUEUE.PacketCell>
        <QUEUE.Distribution Explanation="Production Rate" Required="True" RelationLayout="-
            40.0,0.0,-9.0,-28.0,true,true"
            Value=",{'Normal',{150.0,15.0},,,}">Rate</QUEUE.Distribution>
        <QUEUE.Distribution Explanation="" Required="True" RelationLayout="170.0,539.0,-122.0,18.0,-
            11.0,-16.0,true,true"
            Value=",{'Normal',{150.0,15.0},,,}">AddressDistribution</QUEUE.Distribution>
        <Int Explanation="" Unit="" Required="True">NumberOfAddresses</Int>
    </Attributes>
  - <Ports>
        <QUEUE.Output Explanation="Output port of source" From="n" To="n" Left="-2" Top="-2"
            RelationLayout="-42.0,15.0,true" Value=",">Out</QUEUE.Output>
    </Ports>
  </ComponentType>
- <ComponentType TypeName="Queue" Explanation="Services" PictureFile=""
      NodeLayout="271.0,303.0,true,true">
  - <Attributes>
        <QUEUE.Distribution Explanation="Service rate for process" Required="True" RelationLayout="-
            74.0,-14.0,13.0,-11.0,true,true"
            Value=",{'Normal',{150.0,15.0},,,}">ServiceRate</QUEUE.Distribution>
        <Long Explanation="Number of bits the queue can hold" Unit="# bytes"
            Required="True">QueueSize</Long>
    </Attributes>
  - <Ports>
        <QUEUE.InputPort Explanation="Input port of queue" From="n" To="n" Left="-1" Top="-1"
            RelationLayout="-39.0,14.0,true" Value=",">In</QUEUE.InputPort>
        <QUEUE.Output Explanation="Output port of queue" From="n" To="n" Left="-2" Top="-2"
            RelationLayout="14.0,15.0,true" Value=",">Out</QUEUE.Output>
    </Ports>
  </ComponentType>
- <ComponentType TypeName="Sink" Explanation="" PictureFile=""
      NodeLayout="404.3414306640625,309.0939025878906,true,true">
    <Attributes />
  - <Ports>
        <QUEUE.InputPort Explanation="Input Port of Sink" From="n" To="n" Left="-1" Top="-1"
            RelationLayout="7.0,20.0,true" Value=",">In</QUEUE.InputPort>
    </Ports>
  </ComponentType>
- <ComponentType TypeName="transceiver" Explanation="" PictureFile=""
      NodeLayout="539.0,318.0,true,true">
    <Attributes />
  - <Ports>
        <QUEUE.InputPort Explanation="" From="n" To="n" Left="-1" Top="-1"
            RelationLayout="19.0,11.0,true" Value=",">In</QUEUE.InputPort>
        <QUEUE.Output Explanation="" From="n" To="n" Left="-2" Top="-2"
            RelationLayout="16.0,0.0,true" Value=",">Out</QUEUE.Output>
        <QUEUE.DuplexPort Explanation="" From="0..1" To="0..1" Left="-3" Top="-3"
            RelationLayout="23.0,21.0,true" Value=",">Transfer</QUEUE.DuplexPort>
    </Ports>
  </ComponentType>
- <ComponentType TypeName="Router" Explanation="" PictureFile=""
      NodeLayout="393.0,487.0,true,true">
    <Extension ExtendedTypeName="QUEUE.Queue" RelationLayout="" />
  - <Attributes>
        <QUEUE.RoutingAdresses Explanation="" Required="True" RelationLayout="-130.0,-15.0,4.0,-
            27.0,true,true">RoutingAssociations[0,15]</QUEUE.RoutingAdresses>
        <String Explanation="" Required="True"
            Range="'EIGRP','RGRP','RIP'">RoutingAlgorithm</String>
      </Attributes>
      <Ports />
    </ComponentType>
  </LIBRARY>
```

b) Node Model

```
- <LIBRARY LibraryName="NodeModel" Explanation="" Uses="QUEUE">
  - <ScenarioType TypeName="Node" Explanation="" Component="True" PictureFile="">
    - <Components>
        <QUEUE.Source NodeLayout="39.0,59.0" NodeAlignment="NORMAL"
            Value=",{{{,{1024,'Data','gchfsh'}}},,{'Normal',{150.0,15.0},,,},,{'Normal',{150.0,15.0},,,},3},
            {,}">Source_0</QUEUE.Source>
        <QUEUE.Queue NodeLayout="190.0,58.0" NodeAlignment="NORMAL"
            Value=",{,{'Normal',{150.0,15.0},,,},1500},{,,,}">Queue_0</QUEUE.Queue>
        <QUEUE.Sink NodeLayout="518.0,57.0" NodeAlignment="NORMAL"
            Value=",,{,}">Sink_0</QUEUE.Sink>
        <QUEUE.transceiver NodeLayout="334.0,59.0" NodeAlignment="NORMAL"
            Value=",,{,,,,,}">transceiver_0</QUEUE.transceiver>
      </Components>
    - <Connections>
        <QUEUE.Link Explanation="Link without Delay" From="Source_0.Out" To="Queue_0.In"
            Value="," RelationLayout="">Link_0</QUEUE.Link>
        <QUEUE.Link Explanation="Link without Delay" From="Queue_0.Out" To="transceiver_0.In"
            Value="," RelationLayout="">Link_1</QUEUE.Link>
        <QUEUE.Link Explanation="Link without Delay" From="transceiver_0.Out" To="Sink_0.In"
            Value="," RelationLayout="">Link_2</QUEUE.Link>
      </Connections>
    - <Interface>
        <PortMaps>transceiver_0.Transfer=XMT</PortMaps>
      </Interface>
    </ScenarioType>
  </LIBRARY>
```

c) Hub Model

```
- <LIBRARY LibraryName="HubLibrary" Explanation="" Uses="QUEUE">
  - <ScenarioType TypeName="4Hub" Explanation="" Component="True" PictureFile="">
    - <Components>
        <QUEUE.Router NodeLayout="295.0,191.0" NodeAlignment="NORMAL"
            Value="{{,{,{'Normal',{150.0,15.0},,,},1500},{,,,}}},{,'RIP'},">Router_0</QUEUE.Router>
        <QUEUE.transceiver NodeLayout="89.0,114.0" NodeAlignment="NORMAL"
            Value=",,{,,,,,}">transceiver_0</QUEUE.transceiver>
        <QUEUE.transceiver NodeLayout="442.0,91.0" NodeAlignment="NORMAL"
            Value=",,{,,,,,}">transceiver_1</QUEUE.transceiver>
        <QUEUE.transceiver NodeLayout="84.0,290.0" NodeAlignment="NORMAL"
            Value=",,{,,,,,}">transceiver_2</QUEUE.transceiver>
        <QUEUE.transceiver NodeLayout="453.0,308.0" NodeAlignment="NORMAL"
            Value=",,{,,,,,}">transceiver_3</QUEUE.transceiver>
      </Components>
    - <Connections>
        <QUEUE.Link Explanation="Link without Delay" From="transceiver_0.Out"
            To="Router_0.QUEUE.Queue.In" Value="," RelationLayout="">Link_0</QUEUE.Link>
      <QUEUE.Link Explanation="Link without Delay" From="transceiver_2.Out"
          To="Router_0.QUEUE.Queue.In" Value="," RelationLayout="">Link_1</QUEUE.Link>
      <QUEUE.Link Explanation="Link without Delay" From="Router_0.QUEUE.Queue.Out"
          To="transceiver_1.In" Value="," RelationLayout="">Link_2</QUEUE.Link>
      <QUEUE.Link Explanation="Link without Delay" From="Router_0.QUEUE.Queue.Out"
          To="transceiver_3.In" Value="," RelationLayout="">Link_3</QUEUE.Link>
      <QUEUE.Link Explanation="Link without Delay" From="transceiver_1.Out"
          To="Router_0.QUEUE.Queue.In" Value=","
          RelationLayout="588.0,56.0,265.0,57.0">Link_4</QUEUE.Link>
      <QUEUE.Link Explanation="Link without Delay" From="transceiver_3.Out"
          To="Router_0.QUEUE.Queue.In" Value=","
          RelationLayout="590.0,420.0,281.0,420.0">Link_5</QUEUE.Link>
      <QUEUE.Link Explanation="Link without Delay" From="Router_0.QUEUE.Queue.Out"
          To="transceiver_0.In" Value=","
          RelationLayout="394.0,89.0,35.0,89.0">Link_6</QUEUE.Link>
      <QUEUE.Link Explanation="Link without Delay" From="Router_0.QUEUE.Queue.Out"
          To="transceiver_2.In" Value=","
          RelationLayout="395.0,398.0,43.0,398.0">Link_7</QUEUE.Link>
    </Connections>
  - <Interface>
      <PortMaps>transceiver_0.Transfer=0, transceiver_1.Transfer=1, transceiver_2.Transfer=2,
          transceiver_3.Transfer=3</PortMaps>
    </Interface>
  </ScenarioType>
</LIBRARY>
```

d) Network of Air Force Bases

```
- <LIBRARY LibraryName="NetworkLibrary" Explanation="" Uses="QUEUE,NodeModel,HubLibrary">
  - <ScenarioType TypeName="4Bases" Explanation="" PictureFile="">
    - <Components>
        <HubLibrary.4Hub NodeLayout="225.0,13.0" NodeAlignment="REVERSE"
            Value="{{{{,{,{'Normal',{150.0,15.0},,,},1500},{,,,}}},{,'RIP'},,,,{,,,,,},,,{,,,,,},,,{,,,,,},,,{,,,,,}},{,,,,,
            ,,,,,,,,,,},,{,,,,,,,}}">4Hub_0</HubLibrary.4Hub>
        <NodeModel.Node NodeLayout="74.0,168.0" NodeAlignment="NORMAL"
            Value="{{,{{{,{1024,'Data','gchfsh'}}},,{'Normal',{150.0,15.0},,,},,{'Normal',{150.0,15.0},,,},3
            },{,},,{,{'Normal',{150.0,15.0},,,},1500},{,,,},,,{,},,,{,,,,,}},{,,,,,},,{,}}">Chyenne</NodeModel.No
            de>
        <NodeModel.Node NodeLayout="179.0,168.0" NodeAlignment="NORMAL"
            Value="{{,{{{,{1024,'Data','gchfsh'}}},,{'Normal',{150.0,15.0},,,},,{'Normal',{150.0,15.0},,,},3
            },{,},,{,{'Normal',{150.0,15.0},,,},1500},{,,,},,,{,},,,{,,,,,}},{,,,,,},,{,}}">Hickam</NodeModel.No
            de>
        <NodeModel.Node NodeLayout="389.0,167.0" NodeAlignment="NORMAL"
            Value="{{,{{{,{1024,'Data','gchfsh'}}},,{'Normal',{150.0,15.0},,,},,{'Normal',{150.0,15.0},,,},3
            },{,},,{,{'Normal',{150.0,15.0},,,},1500},{,,,},,,{,},,,{,,,,,}},{,,,,,},,{,}}">Ramstein</NodeModel.N
            ode>
        <NodeModel.Node NodeLayout="283.0,168.0" NodeAlignment="NORMAL"
            Value="{{,{{{,{1024,'Data','gchfsh'}}},,{'Normal',{150.0,15.0},,,},,{'Normal',{150.0,15.0},,,},3
            },{,},,{,{'Normal',{150.0,15.0},,,},1500},{,,,},,,{,},,,{,,,,,}},{,,,,,},,{,}}">WPAFB</NodeModel.No
            de>
      </Components>
    - <Connections>
        <QUEUE.DuplexPaidLink Explanation="A link with a cost value for usage"
            From="Chyenne.XMT" To="4Hub_0.0" Value=",{,,,}"
            RelationLayout="">DuplexPaidLink_0</QUEUE.DuplexPaidLink>
        <QUEUE.DuplexPaidLink Explanation="A link with a cost value for usage"
            From="Hickam.XMT" To="4Hub_0.1" Value=",{,,,}"
            RelationLayout="">DuplexPaidLink_1</QUEUE.DuplexPaidLink>
        <QUEUE.DuplexPaidLink Explanation="A link with a cost value for usage"
            From="WPAFB.XMT" To="4Hub_0.2" Value=",{,,,}"
            RelationLayout="">DuplexPaidLink_2</QUEUE.DuplexPaidLink>
        <QUEUE.DuplexPaidLink Explanation="A link with a cost value for usage"
            From="Ramstein.XMT" To="4Hub_0.3" Value=",{,,,}"
            RelationLayout="">DuplexPaidLink_3</QUEUE.DuplexPaidLink>
      </Connections>
    </ScenarioType>
  </LIBRARY>
```

Section 3: Sensitivity Analysis

a) Decision Tree Elements

```
- <LIBRARY LibraryName="DECISION_ANALYSIS" Explanation="This library implements decision
      tree">
    <PortType TypeName="ParentsPort" Explanation="Connects this objective to its parent objective"
        NodeLayout="101.0,170.0,true" Symbol="Triangle" />
    <PortType TypeName="ChildrenPort" Explanation="Connects the objective to its sub-objectives"
        NodeLayout="280.0,167.0,true" Symbol="Square" />
    <RelationType TypeName="Connection" Explanation="Connects objectives to its sub objectives"
        HeadSymbol="None" TailSymbol="None" LineSymbol="Double"
        FromPortType="DECISION_ANALYSIS.ParentsPort"
        ToPortType="DECISION_ANALYSIS.ChildrenPort" FromPortRelationLayout="-50.0,14.0,true"
        ToPortRelationLayout="21.0,16.0,true" NodeLayout="197.0,70.0,true" />
  - <ComponentType TypeName="Objective_Node" Explanation="" PictureFile=""
        NodeLayout="183.0,280.0,true,true">
    - <Attributes>
        <String Explanation="This field is used as title in the MsExcel Worksheet if applicable"
            Required="True">Caption</String>
        <String Explanation="This field contains detailed information about the
            objective">Explanation</String>
        <Float Explanation="Percentage value between 0 and 100" Unit="Percentagevalue"
            Range="0.0..100.0">LocalWeight</Float>
      </Attributes>
    - <Ports>
        <DECISION_ANALYSIS.ChildrenPort Explanation="" From="0" To="n" Left="-4" Top="-4"
            RelationLayout="3.0,19.0,true" Value=",">SubObj</DECISION_ANALYSIS.ChildrenPort>
        <DECISION_ANALYSIS.ParentsPort Explanation="" From="1" To="0" Left="-3" Top="-3"
            RelationLayout="-64.0,20.0,true"
            Value=",">SuperObj</DECISION_ANALYSIS.ParentsPort>
      </Ports>
    </ComponentType>
  </LIBRARY>
```

b) Best Car Sample

```
- <LIBRARY LibraryName="BestCarLibrary" Explanation="" Uses="DECISION_ANALYSIS">
  - <ScenarioType TypeName="BestCar" Explanation="" PictureFile="">
    - <Components>
    <DECISION_ANALYSIS.Objective_Node NodeLayout="405.0,63.0"
        NodeAlignment="NORMAL"
        Value=",{'BestCar',,100.0},{,,,}">Objective_Node_0</DECISION_ANALYSIS.Objective_No
        de>
    <DECISION_ANALYSIS.Objective_Node NodeLayout="139.5,228.0"
        NodeAlignment="NORMAL"
        Value=",{'Power',,25.0},{,,,}">Objective_Node_1</DECISION_ANALYSIS.Objective_Node>
    <DECISION_ANALYSIS.Objective_Node NodeLayout="464.0,221.5"
        NodeAlignment="NORMAL"
        Value=",{'Estetic',,25.0},{,,,}">Objective_Node_2</DECISION_ANALYSIS.Objective_Node
        >
    <DECISION_ANALYSIS.Objective_Node NodeLayout="897.0,230.5"
        NodeAlignment="NORMAL"
        Value=",{'Price',,50.0},{,,,}">Objective_Node_3</DECISION_ANALYSIS.Objective_Node>
    <DECISION_ANALYSIS.Objective_Node NodeLayout="34.0,384.0"
        NodeAlignment="NORMAL"
        Value=",{'HorsePower',,50.0},{,,,}">Objective_Node_4</DECISION_ANALYSIS.Objective_
        Node>
    <DECISION_ANALYSIS.Objective_Node NodeLayout="120.0,484.0"
        NodeAlignment="NORMAL"
        Value=",{'GasMilage',,25.0},{,,,}">Objective_Node_5</DECISION_ANALYSIS.Objective_N
        ode>
    <DECISION_ANALYSIS.Objective_Node NodeLayout="218.0,385.5"
        NodeAlignment="NORMAL"
        Value=",{'Transmission',,25.0},{,,,}">Objective_Node_6</DECISION_ANALYSIS.Objective
        _Node>
    <DECISION_ANALYSIS.Objective_Node NodeLayout="390.0,372.0"
        NodeAlignment="NORMAL"
        Value=",{'Color',,40.0},{,,,}">Objective_Node_7</DECISION_ANALYSIS.Objective_Node>
    <DECISION_ANALYSIS.Objective_Node NodeLayout="555.0,370.5"
        NodeAlignment="NORMAL"
        Value=",{'Style',,60.0},{,,,}">Objective_Node_8</DECISION_ANALYSIS.Objective_Node>
    <DECISION_ANALYSIS.Objective_Node NodeLayout="780.0,369.0"
        NodeAlignment="NORMAL"
        Value=",{'Milage',,35.0},{,,,}">Objective_Node_9</DECISION_ANALYSIS.Objective_Node
        >
    <DECISION_ANALYSIS.Objective_Node NodeLayout="873.0,493.5"
        NodeAlignment="NORMAL"
        Value=",{'Year',,30.0},{,,,}">Objective_Node_10</DECISION_ANALYSIS.Objective_Node>
    <DECISION_ANALYSIS.Objective_Node NodeLayout="996.0,369.0"
        NodeAlignment="NORMAL"
        Value=",{'Price',,35.0},{,,,}">Objective_Node_11</DECISION_ANALYSIS.Objective_Node>
  </Components>
- <Connections>
    <DECISION_ANALYSIS.Connection Explanation="Connects objectives to its sub objectives"
        From="Objective_Node_1.SuperObjective" To="Objective_Node_0.SubObjectives"
        Value="," RelationLayout="">Connection_0</DECISION_ANALYSIS.Connection>
    <DECISION_ANALYSIS.Connection Explanation="Connects objectives to its sub objectives"
        From="Objective_Node_2.SuperObjective" To="Objective_Node_0.SubObjectives"
        Value="," RelationLayout="">Connection_1</DECISION_ANALYSIS.Connection>
    <DECISION_ANALYSIS.Connection Explanation="Connects objectives to its sub objectives"
        From="Objective_Node_3.SuperObjective" To="Objective_Node_0.SubObjectives"
        Value="," RelationLayout="">Connection_2</DECISION_ANALYSIS.Connection>
    <DECISION_ANALYSIS.Connection Explanation="Connects objectives to its sub objectives"
        From="Objective_Node_4.SuperObjective" To="Objective_Node_1.SubObjectives"
        Value="," RelationLayout="">Connection_3</DECISION_ANALYSIS.Connection>
    <DECISION_ANALYSIS.Connection Explanation="Connects objectives to its sub objectives"
        From="Objective_Node_5.SuperObjective" To="Objective_Node_1.SubObjectives"
        Value="," RelationLayout="">Connection_4</DECISION_ANALYSIS.Connection>
        <DECISION_ANALYSIS.Connection Explanation="Connects objectives to its sub objectives"
            From="Objective_Node_6.SuperObjective" To="Objective_Node_1.SubObjectives"
            Value="," RelationLayout="">Connection_5</DECISION_ANALYSIS.Connection>
        <DECISION_ANALYSIS.Connection Explanation="Connects objectives to its sub objectives"
            From="Objective_Node_7.SuperObjective" To="Objective_Node_2.SubObjectives"
            Value="," RelationLayout="">Connection_6</DECISION_ANALYSIS.Connection>
        <DECISION_ANALYSIS.Connection Explanation="Connects objectives to its sub objectives"
            From="Objective_Node_8.SuperObjective" To="Objective_Node_2.SubObjectives"
            Value="," RelationLayout="">Connection_7</DECISION_ANALYSIS.Connection>
        <DECISION_ANALYSIS.Connection Explanation="Connects objectives to its sub objectives"
            From="Objective_Node_9.SuperObjective" To="Objective_Node_3.SubObjectives"
            Value="," RelationLayout="">Connection_8</DECISION_ANALYSIS.Connection>
        <DECISION_ANALYSIS.Connection Explanation="Connects objectives to its sub objectives"
            From="Objective_Node_10.SuperObjective" To="Objective_Node_3.SubObjectives"
            Value="," RelationLayout="">Connection_9</DECISION_ANALYSIS.Connection>
        <DECISION_ANALYSIS.Connection Explanation="Connects objectives to its sub objectives"
            From="Objective_Node_11.SuperObjective" To="Objective_Node_3.SubObjectives"
            Value="," RelationLayout="">Connection_10</DECISION_ANALYSIS.Connection>
      </Connections>
    </ScenarioType>
  </LIBRARY>
```

Section 4: Mission Planning

a) Mission Elements

```
- <LIBRARY LibraryName="MilitaryLibrary" Explanation="Contains mission planing elements">
  - <DataType TypeName="Log" Explanation="" NodeLayout="33.0,316.5,true">
      <String Explanation="">TOT</String>
      <String Explanation="">Action</String>
    </DataType>
    <PortType TypeName="PacketTarget" Explanation="" NodeLayout="226.0,463.0,true" Symbol="Line"
        />
    <PortType TypeName="CommPort" Explanation="" NodeLayout="381.0,364.0,true" Symbol="None" />
    <RelationType TypeName="PrimaryTarget" Explanation="" HeadSymbol="FilledTriangle"
        TailSymbol="None" LineSymbol="Double" FromPortType="MilitaryLibrary.PacketTarget"
        ToPortType="MilitaryLibrary.PacketTarget"
        FromPortRelationLayout="215.0,588.0,106.0,52.0,true" ToPortRelationLayout="212.0,556.0,114.0,-
        2.0,true" NodeLayout="52.0,552.0,true" />
    <RelationType TypeName="SecondaryTarget" Explanation="" HeadSymbol="Triangle"
        TailSymbol="None" LineSymbol="Plain" FromPortType="MilitaryLibrary.PacketTarget"
        ToPortType="MilitaryLibrary.PacketTarget" FromPortRelationLayout="224.0,454.0,3.0,-
        17.0,true" ToPortRelationLayout="210.0,510.0,0.0,20.0,true" NodeLayout="42.0,463.0,true" />
    <RelationType TypeName="CommConnection" Explanation="" HeadSymbol="None"
        TailSymbol="None" LineSymbol="DashDot" FromPortType="MilitaryLibrary.CommPort"
        ToPortType="MilitaryLibrary.CommPort" FromPortRelationLayout="587.0,323.0,14.0,-17.0,true"
        ToPortRelationLayout="586.0,429.0,17.0,16.0,true" NodeLayout="521.0,357.0,true" />
  - <ComponentType TypeName="AbstractAircraft" Explanation="" Abstract="True" PictureFile=""
        NodeLayout="230.0,229.0,true,true">
    - <Attributes>
        <MilitaryLibrary.Log Explanation="" RelationLayout="15.0,14.0,3.0,-
            19.0,true,true">MissionLog[0,n]</MilitaryLibrary.Log>
        <String Explanation="" Required="True" Range="'Search And Rescue','Air Drop','Air
              Refuel','Patrol','Command
              Center','Reconnaissance','SEAD','Target'">MissionType</String>
        <String Explanation="" Required="True">Load</String>
        <Int Explanation="" Unit="" Required="True" Value="1">FormationSize</Int>
    </Attributes>
  - <Ports>
        <MilitaryLibrary.PacketTarget Explanation="" From="2" To="0" Left="50" Top="10"
              RelationLayout="20.0,-19.0,true" Value=",">Target</MilitaryLibrary.PacketTarget>
        <MilitaryLibrary.CommPort Explanation="" From="n" To="n" Left="50" Top="60"
              RelationLayout="-50.0,-1.0,true" Value=",">Comm</MilitaryLibrary.CommPort>
    </Ports>
</ComponentType>
- <ComponentType TypeName="CN235" Explanation=""
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\MissionPlanning\PlanePics\CN2
      35.gif" NodeLayout="25.0,204.0,true,true">
    <Extension ExtendedTypeName="MilitaryLibrary.AbstractAircraft" RelationLayout="" />
    <Attributes />
    <Ports />
</ComponentType>
- <ComponentType TypeName="C130" Explanation=""
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\MissionPlanning\PlanePics\C130
      .gif" NodeLayout="471.0,44.0,true,true">
    <Extension ExtendedTypeName="MilitaryLibrary.AbstractAircraft" RelationLayout="" />
    <Attributes />
    <Ports />
</ComponentType>
- <ComponentType TypeName="Cougar" Explanation=""
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\MissionPlanning\PlanePics\Coug
      er.gif" NodeLayout="358.0,22.0,true,true">
    <Extension ExtendedTypeName="MilitaryLibrary.AbstractAircraft" RelationLayout="" />
    <Attributes />
    <Ports />
</ComponentType>
- <ComponentType TypeName="Gulf" Explanation=""
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\MissionPlanning\PlanePics\Gulf.
      gif" NodeLayout="12.0,132.0,true,true">
    <Extension ExtendedTypeName="MilitaryLibrary.AbstractAircraft" RelationLayout="" />
    <Attributes />
    <Ports />
</ComponentType>
- <ComponentType TypeName="KC130" Explanation=""
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\MissionPlanning\PlanePics\KC1
      30.gif" NodeLayout="25.0,53.0,true,true">
    <Extension ExtendedTypeName="MilitaryLibrary.AbstractAircraft" RelationLayout="" />
    <Attributes />
    <Ports />
</ComponentType>
- <ComponentType TypeName="F4" Explanation=""
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\MissionPlanning\PlanePics\F4.gi
      f" NodeLayout="132.0,23.0,true,true">
    <Extension ExtendedTypeName="MilitaryLibrary.AbstractAircraft" RelationLayout="" />
    <Attributes />
    <Ports />
</ComponentType>
- <ComponentType TypeName="F16" Explanation=""
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\MissionPlanning\PlanePics\F16.
      gif" NodeLayout="242.0,15.0,true,true">
    <Extension ExtendedTypeName="MilitaryLibrary.AbstractAircraft" RelationLayout="" />
    <Attributes />
    <Ports />
  </ComponentType>
- <ComponentType TypeName="CommunicationChannel" Explanation=""
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\MissionPlanning\PlanePics\Com
      mChannel.gif" NodeLayout="440.5,488.0,true,true">
  - <Attributes>
      <String Explanation="" Required="True">PrimaryFreq</String>
      <String Explanation="" Required="True">SecondaryFreq</String>
      <String Explanation="" Required="True">EmergencyFreq</String>
    </Attributes>
  - <Ports>
      <MilitaryLibrary.CommPort Explanation="" From="n" To="n" Left="15" Top="15"
          RelationLayout="-51.0,21.0,true" Value=",">Comm</MilitaryLibrary.CommPort>
    </Ports>
  </ComponentType>
- <ComponentType TypeName="AbstractTarget" Explanation="" Abstract="True" PictureFile=""
      NodeLayout="193.0,656.0,true,true">
    <Attributes />
  - <Ports>
      <MilitaryLibrary.PacketTarget Explanation="" From="0" To="n" Left="10" Top="10"
          RelationLayout="3.0,19.0,true" Value=",">Target</MilitaryLibrary.PacketTarget>
    </Ports>
  </ComponentType>
- <ComponentType TypeName="GroundTarget" Explanation=""
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\MissionPlanning\PlanePics\Grou
      ndTarget.gif" NodeLayout="78.0,759.0,true,true">
    <Extension ExtendedTypeName="MilitaryLibrary.AbstractTarget" RelationLayout="" />
  - <Attributes>
      <String Explanation="" Required="True">Coordinates</String>
    </Attributes>
    <Ports />
  </ComponentType>
- <ComponentType TypeName="SeaTarget" Explanation=""
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\MissionPlanning\PlanePics\SeaT
      arget.gif" NodeLayout="32.0,677.0,true,true">
    <Extension ExtendedTypeName="MilitaryLibrary.AbstractTarget" RelationLayout="" />
  - <Attributes>
      <String Explanation="" Required="True">Coordinates</String>
    </Attributes>
    <Ports />
  </ComponentType>
- <ComponentType TypeName="PatrolZone" Explanation=""
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\MissionPlanning\PlanePics\Patr
      olZone.gif" NodeLayout="231.0,781.0,true,true">
    <Extension ExtendedTypeName="MilitaryLibrary.AbstractTarget" RelationLayout="" />
  - <Attributes>
      <String Explanation="">Coordinates[0,n]</String>
    </Attributes>
    <Ports />
  </ComponentType>
- <ComponentType TypeName="Radar" Explanation=""
      PictureFile="D:\Hakan\Thesis\MYDOCS\Chapters\CaseStudies\MissionPlanning\PlanePics\rada
      r.gif" NodeLayout="556.0,156.0,true,true">
    <Attributes />
  - <Ports>
      <MilitaryLibrary.CommPort Explanation="" From="n" To="n" Left="10" Top="10"
          RelationLayout="-23.0,-18.0,true" Value=",">Comm</MilitaryLibrary.CommPort>
    </Ports>
    </ComponentType>
  </LIBRARY>
```

b) Sample Mission

```
- <LIBRARY LibraryName="MissionLibrary" Explanation="" Uses="MilitaryLibrary">
  - <ScenarioType TypeName="MissionImpossible" Explanation="" PictureFile="">
    - <Components>
        <MilitaryLibrary.Gulf NodeLayout="569.0,307.5" NodeAlignment="NORMAL"
           Value="{{,{,'Command Center','Standart',1},{,,,}}},,">Gulf_0</MilitaryLibrary.Gulf>
        <MilitaryLibrary.F16 NodeLayout="133.0,180.0" NodeAlignment="NORMAL"
           Value="{{,{,'Target','2xAim8',4},{,,,}}},,">F16_0</MilitaryLibrary.F16>
        <MilitaryLibrary.F4 NodeLayout="104.0,514.0" NodeAlignment="NORMAL"
           Value="{{,{,'Target','Std',4},{,,,}}},,">F4_0</MilitaryLibrary.F4>
        <MilitaryLibrary.GroundTarget NodeLayout="100.0,348.0" NodeAlignment="NORMAL"
           Value="{{,,{,}}},{'4545N4545W'},">GroundTarget_0</MilitaryLibrary.GroundTarget>
        <MilitaryLibrary.PatrolZone NodeLayout="744.0,161.5" NodeAlignment="NORMAL"
           Value="{{,,{,}}},{{{'3545N2525E'},{'3600N2535E'},{'3400N2500E'}}},">PatrolZone_0</Milit
           aryLibrary.PatrolZone>
        <MilitaryLibrary.SeaTarget NodeLayout="112.0,21.5" NodeAlignment="NORMAL"
           Value="{{,,{,}}},{'2525N5643E'},">SeaTarget_0</MilitaryLibrary.SeaTarget>
        <MilitaryLibrary.SeaTarget NodeLayout="297.0,39.5" NodeAlignment="NORMAL"
           Value="{{,,{,}}},{},">SeaTarget_1</MilitaryLibrary.SeaTarget>
        <MilitaryLibrary.GroundTarget NodeLayout="17.0,425.0" NodeAlignment="NORMAL"
           Value="{{,,{,}}},{},">GroundTarget_1</MilitaryLibrary.GroundTarget>
        <MilitaryLibrary.CommunicationChannel NodeLayout="322.0,323.5" NodeAlignment="NORMAL"
           Value=",{'121.5','118.0',},{,}">CommunicationChannel_0</MilitaryLibrary.CommunicationC
           hannel>
        <MilitaryLibrary.CN235 NodeLayout="448.0,144.5" NodeAlignment="NORMAL"
           Value="{{,{,,'Standart',1},{,,,}}},,">CN235_0</MilitaryLibrary.CN235>
        <MilitaryLibrary.Radar NodeLayout="509.0,548.0" NodeAlignment="NORMAL"
           Value=",,{,}">Radar_0</MilitaryLibrary.Radar>
      </Components>
    - <Connections>
        <MilitaryLibrary.PrimaryTarget Explanation=""
           From="Gulf_0.MilitaryLibrary.AbstractAircraft.Target"
           To="PatrolZone_0.MilitaryLibrary.AbstractTarget.Target" Value=","
           RelationLayout="">PrimaryTarget_0</MilitaryLibrary.PrimaryTarget>
        <MilitaryLibrary.PrimaryTarget Explanation=""
           From="F16_0.MilitaryLibrary.AbstractAircraft.Target"
           To="SeaTarget_0.MilitaryLibrary.AbstractTarget.Target" Value=","
           RelationLayout="">PrimaryTarget_1</MilitaryLibrary.PrimaryTarget>
        <MilitaryLibrary.PrimaryTarget Explanation=""
           From="F4_0.MilitaryLibrary.AbstractAircraft.Target"
           To="GroundTarget_0.MilitaryLibrary.AbstractTarget.Target" Value=","
           RelationLayout="">PrimaryTarget_2</MilitaryLibrary.PrimaryTarget>
        <MilitaryLibrary.SecondaryTarget Explanation=""
           From="F16_0.MilitaryLibrary.AbstractAircraft.Target"
           To="SeaTarget_1.MilitaryLibrary.AbstractTarget.Target" Value=","
           RelationLayout="">SecondaryTarget_0</MilitaryLibrary.SecondaryTarget>
        <MilitaryLibrary.SecondaryTarget Explanation=""
           From="F4_0.MilitaryLibrary.AbstractAircraft.Target"
           To="GroundTarget_1.MilitaryLibrary.AbstractTarget.Target" Value=","
           RelationLayout="">SecondaryTarget_1</MilitaryLibrary.SecondaryTarget>
        <MilitaryLibrary.CommConnection Explanation=""
           From="F16_0.MilitaryLibrary.AbstractAircraft.Comm"
```

```
            To="CommunicationChannel_0.Comm" Value=","
            RelationLayout="">CommConnection_0</MilitaryLibrary.CommConnection>
        <MilitaryLibrary.CommConnection Explanation=""
            From="Gulf_0.MilitaryLibrary.AbstractAircraft.Comm"
            To="CommunicationChannel_0.Comm" Value=","
            RelationLayout="605.0,394.0,560.0,407.0,496.0,400.0">CommConnection_1</MilitaryLibrary
            .CommConnection>
        <MilitaryLibrary.CommConnection Explanation=""
            From="F4_0.MilitaryLibrary.AbstractAircraft.Comm"
            To="CommunicationChannel_0.Comm" Value=","
            RelationLayout="231.0,625.0,303.0,627.0,331.0,586.0">CommConnection_2</MilitaryLibrary
            .CommConnection>
        <MilitaryLibrary.PrimaryTarget Explanation=""
            From="CN235_0.MilitaryLibrary.AbstractAircraft.Target"
            To="PatrolZone_0.MilitaryLibrary.AbstractTarget.Target" Value=","
            RelationLayout="638.0,65.5,690.0,55.5,727.5,91.5">PrimaryTarget_3</MilitaryLibrary.Prima
            ryTarget>
        <MilitaryLibrary.CommConnection Explanation=""
            From="CN235_0.MilitaryLibrary.AbstractAircraft.Comm"
            To="CommunicationChannel_0.Comm" Value=","
            RelationLayout="">CommConnection_3</MilitaryLibrary.CommConnection>
        <MilitaryLibrary.CommConnection Explanation="" From="Radar_0.Comm"
            To="CommunicationChannel_0.Comm" Value=","
            RelationLayout="">CommConnection_4</MilitaryLibrary.CommConnection>
      </Connections>
    </ScenarioType>
</LIBRARY>
```

Vita

First Lieutenant Hakan Canli was born in April 1974 in Izmir, Turkey. He graduated from Maltepe Military High School in 1992, and received his Bachelor of Science Degree in Computer Engineering in 1996, from the Turkish Air Force Academy located in Istanbul. He completed the pilot training course in March 1998. He began his military service career as a CN-235 pilot at 2nd Tactical Air Force, Diyarbakir, Turkey. In August 2000, he started at the Graduate School of Engineering and Management at the Air Force Institute of Technology at Wright-Patterson AFB, OH, to work toward a Masters of Science Degree in Computer Engineering. He is a member of Tau Beta Pi and Eta Kappa Nu, the National Engineering Honor Societies.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. REPORT DATE *(DD-MM-YYYY)* 10-03-2002	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From – To)* Mar 2001 – Mar 2002

4. TITLE AND SUBTITLE A VISUAL META-LANGUAGE FOR GENERIC MODELING	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Canli, Hakan, First Lieutenant, TUAF	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Building 640 WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCE/ENG/02M-1
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/SNZW AFMC Attn: Mike R. Foster Bldg 620 S1D34, 2241 Avionics Circle WPAFB OH 45433-7303 Comm: (937) 656-4464 DSN: 986-4464 x 3002 Email: mike.foster@wpafb.af.mil	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/SNZW
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research examines the usefulness of a visual meta-language (VLGM – Visual Language for Generic Modeling) developed for the specification of components and relations in a modeling domain. The language is designed to allow software tools to interpret specifications and automatically provide modeling environments.

VLGM makes use of the object-orientated software engineering methodology. It defines four types of special classes and three types of relations between them. Data types and primitive types are allocated with several attributes to provide restrictions and enable consistency checks over models.

As part of this research a software tool was designed. The tool provides a workspace for creating VLGM specifications. It interprets VLGM designs and provides a generic modeling environment. An XML document format is used as a persistence mechanism to promote reusability and sharing. Four case studies from different modeling domains are used to explore the applicability of the idea.

15. SUBJECT TERMS
VLGM, Visual Languages, Modeling, Object-Orientation, Simulation, UML, XML Document

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT** UU	**18. NUMBER OF PAGES** 186	**19a. NAME OF RESPONSIBLE PERSON** Major Karl S. Mathias
REPORT U	**ABSTRACT** U	**c. THIS PAGE** U			**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565, ext 4280

Standard Form 298 (Rev: 8-98)
Prescribed by ANSI Std. Z39-18