UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

A METHODOLOGY FOR DEVELOPMENT OF DOMAIN SPECIFIC SIMULATION APPLICATIONS AND ENVIRONMENTS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

KITTI SETAVORAPHAN
Norman, Oklahoma
2009

A METHODOLOGY FOR DEVELOPMENT OF DOMAIN SPECIFIC SIMULATION APPLICATIONS AND ENVIRONMENTS

A DISSERTATION APPROVED FOR THE
SCHOOL OF INDUSTRIAL ENGINEERING

BY

_____
Dr. F. Hank Grant, Chair

_____
Dr. Scott A. Moses

_____
Dr. Shivakumar Raman

_____
Dr. Suleyman Karabuk

_____
Dr. S. Lakshmivarahan

I would like to dedicate this dissertation to A. Alan B. Pritsker, whose unique insights and methodology in modeling and simulation were inspirational to my advisor F. Hank Grant, who in turn passed his knowledge on to me. Professor Pritsker's thinking process has become a model for my own thinking process. His tremendous contributions to the field should not be forgotten, and future scholars should review his fundamental work in the area.

ACKNOWLEDGEMENTS

First, I would like to express my gratitude to Dr. F. Hank Grant for being my advisor and allowing me to work at the Center for the Study of Wireless Electromagnetic Compatibility, and for everything he has taught me during the last four years at the University of Oklahoma. He has inspired and motivated me to reach high standards for being not only a proficient industrial engineer but also a better man. I am greatly honored to be able to work with this outstanding professor whose wisdom and advice have endlessly been given to me. His name will be recognized every step on my path to success.

I am also grateful to all of my dissertation committee members: Dr. Scott A. Moses, Dr. Shivakumar Raman, Dr. Suleyman Karabuk, and Dr. S. Lakshmivarahan, for their advice, guidance, and encouragement throughout my research. They have always been there and served their best to push me to move forward.

I would like to thank the following faculty members and staff in the School of Industrial Engineering at the University of Oklahoma: Dr. Mary Court for offering me the opportunity to pursue my doctoral degree; Dr. Randa Shehab for supporting me in many aspects in every situation; Dr. Hillel Kumin for giving me important insights of being a good problem solver; Dr. B. Mustafa Pulat for training me to be a Lean/Six Sigma Green Belt industrial engineer; Dr. Chen Ling for being my advisor when I was the President of Thai Student Association, Dr. Yongpei Guan for showing me the gaps I needed to fill in; Dr. Kash Baker for offering me his help anytime I was confronting a problem; Cheryl Carney for helping me handle all funding and sponsorship materials; Jean Shingledecker

for helping me solve miscellaneous issues; and Amy Piper for facilitating me while being in this Ph.D. program.

I am thankful of Dr. Glenn Kuriger for training me to work at the Center for the Study of Wireless Electromagnetic Compatibility and being a great partner. My appreciation also goes to my good personal friends Dr. Kang-Hung Yang, Zhili Zhou, and Chris Poyner, who gave me friendship, care, and support throughout these academics years.

For this achievement, I will always remember Dr. Ming Zhou, my thesis advisor at Indiana State University, for training and encouraging me to breakthrough my limitations. I would like to remember Dr. Qun Zhang with gratitude for his technical assistance on my dissertation and his kindly supportive role that never let me down. And, it would not have been possible for me to come this far if it had not been for all of the guidance, support, and encouragement provided by Dr. Kittiphan Techakittiroj, my undergraduate advisor at Assumption University, Thailand. I indeed appreciate their superb backups and approachability.

I could happily study and live at the University of Oklahoma because I have had a group of amazing people who always supported and cheered me up. My special thanks go to: Tinky, Jana, Pong, and Sak (Jana's Restaurant); Mac and Pin (Panang Restaurant), Dear, Evert, Sukhum, Pook, Tak, Bumbim, Nong, and Kaka (Thai Raja Restaurant); Eke, Nong and Pundit (excellent neighborhood); Vee (mechanics); and Him (acquaintance). I am also grateful of James Vernon for his special support in improving my English and preparing me for entering real-world of industry.

I am infinitely grateful of my life's greatest blessings which are my adorable parents: Pricha and Suwanna; my loving sisters: Nopparatna and Juree; and also my heroic brothers: Sekson and Suntaya for their immense love and never-ending support. Without them, I would not be who I am and I am going to be.

I would especially like to thank my be-soul-loved sweetheart, Dr. Emma Asnachinda, for always being at my side in every moment of happiness and sadness. Her smiling and warm-hearted feeling has inspired me to endure all the highs and lows that occurred during my doctoral program. I would like to express my deepest appreciation to Kritsa Chindanon for being my ever-best friend, cheering me up, and keeping up my good spirits. He has made my study life in the United States unforgettable. It is also my pleasure to remember Sethaphon, Prakaikaew, Suthum, Tippy, Neung, Peung, Dew, Karin, and Mon for giving care and offering me their help whenever it was needed.

I would like to thank Lord Buddha for his wisdom enlightening my path to consciousness and peace; and my Guardians and Holy Spirits for embracing me with blessings and love.

Finally, as this is being, I see not only these letters typed on white space but also all the memories through these many years that are flashing back like thousands of movies showing on my computer screen. They are so many actors playing in my story, and it is difficult for me to mention everyone's names and thank them all individually. I would like to say this, *"You are apiece of my life, and I will never ever forget any of you. Thank you very much for everything you have supported and done for me. With all, I'll never walk alone."*

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In the modeling and simulation (M&S) arena, simulation developers have been exploring the concepts that facilitate modeling real world elements using appropriate simulation artifacts within the context of the domain of the application. However, there are some critical issues that distort their effectiveness and efficiency. The first issue is the quantity and quality of assumptions and constraints made during the M&S development, concerning the completeness of simulation models to represent reality. The second issue is the levels of model composability and simulation interoperability, affecting the possibility of data exchange and reusability. The third issue is development of an effective simulation-based environment such that the implementation of the concepts effectively implemented. Thus, this research study aims to develop a methodology that addresses these issues to improve the development of simulation models and the creation of simulation modeling environments particular to specific domains. Conceptual simulation modeling (CSM), model transformation, and domain specific simulation environment (DSSE) create the foundations for this methodology to bridge the gap between reality and simulation.

CHAPTER 1

Introduction

1-1. Overview of Research

In general practice, simulation modeling is performed as a development process focusing on design and experimentation of models on a computer. Often, the leading role in defining specifications and requirements of the development process is weighed on the side of the terms of simulation (e.g., languages, environments, and applications) rather than modeling (e.g., concepts, formalisms, and representations). This is because modeling is still viewed as more of an art than science, whereas simulation is considered as a solid framework – that puts the development at ease with controllability (i.e., a property of a system to be controlled by manipulating the initial state/inputs to the system to obtain the desired state/outputs over a time interval). However, in many cases, the simulation framework causes unnecessary constraints in representing the true characteristics and semantics of reality – which reduces maintainability/sustainability (i.e., a property of a system or its components/attributes to be reused or modified to adapt to a changed environment) of the simulation models.

In the Modeling and Simulation (M&S) arena, the balance between controllability and maintainability/sustainability is very crucial – in bridging the gap between reality and simulation – when conducting a simulation modeling study. To achieve the goal, a modeling framework needs to be independently developed as well as potentially mapped into the simulation framework. The main purpose is to model real world elements by using appropriate simulation artifacts effectively and efficiently. However, since the real

world elements/systems continued to grow in size and complexity, the need for better procedures and techniques for simulation modeling is more apparent. This research study, therefore, is focused on three critical issues that lay out the foundations for improving the future of M&S development.

The first issue is the quantity and quality of assumptions and constraints made during the M&S development, concerning the completeness of simulation models to represent reality. From a simulation perspective, this issue is focused on how well information and knowledge in reality are conceptualized and transformed into simulation modeling concepts. To facilitate the conceptualization and transformation of concepts from one domain to another, the approach of conceptual simulation modeling (CSM) is critical. CSM is also determined either as a mechanism capturing the structural and behavioral characteristics of a problem domain or as an interface providing knowledge representations for cross-domain communication – which results in creating a modeling framework for a problem domain.

It is important to have maintainability/sustainability in modeling and gain controllability in simulation. The modeling framework retrieved from CSM has become a key to success. This is because the modeling framework is not only a process for parsing the boundaries, requirements, and elements from reality to simulation but also a blueprint for specifying the structures and environments for a simulation framework corresponding to the problem domain. As a result, the potential of mapping between these frameworks exists – which leads to another agenda lying within the first issue.

The completeness of simulation models to represent reality is an ideal concept to bridge the gap between reality and simulation. The more positive the mapping, the more

complete the simulation models. It follows that the next step is to develop a domain specific simulation environment (DSSE). The DSSE can be viewed as an overlapping framework of the modeling and simulation aspects, supporting using simulation artifacts to model real world elements for such a specific problem domain. This also enables the satisfactory of both controllability and maintainability/sustainability for the simulation models. However, developing a DSSE from a scratch is indeed difficult, and probably leads to the lost in translation of concepts from the target domain to the simulation domain – which distorts its effectiveness and efficiency. It, thus, requires a documented guideline to structure a DSSE. In this research study, the CSM approach is applied to develop such documentation providing knowledge representations that describe the structural and behavioral characteristics of the problem domain in terms of both real world and simulation architecture and context. This aims not only to facilitate the development of DSSEs but also to resolve the first issue.

The second issue is the levels of model composability and simulation interoperability, affecting the possibility of reusability and data exchange of components. This issue is a consequence from retrieving a conceptual simulation model. Practically, the conceptual simulation model is unable to be implemented directly. This is because the conceptual simulation model provides documentation of the model characteristics but is not in an executable form. It, thus, still needs to be transformed from conceptual components into executable components.

The transformation of conceptual simulation models is the process of data exchange between the sources and targets, whose semantics are controlled by the levels of model composability (for conceptualization) and simulation interoperability (for

implementation). Failure in transferring true semantics of the conceptual simulation models to the implementation details costly affect the development of DSSEs, including general simulation modeling studies. The impact does not mean only errors in simulation functionalities but infeasibilities in reusing those for future simulation projects. Therefore, the conceptual simulation models need to be transformed into contextualized documentation, so that their semantics of structural and behavioral contents can be represented within a simulation context that is understandable and accessible by human and computer. This is to ensure that the simulation contents targeting for implementation are still specified within the modeling framework.

The third issue is the simulation-based environment that is the implementation of the concepts developed. This issue is also considered as a deterministic problem when having more than one choice of selection for mapping between conceptualization and simulation. In general, a simulation model can be built on either a generic (e.g., commercial software like Arena, Visual SLAM, etc.) or a specific (e.g., SNAP) simulation environment/host simulation language. They both contain and take advantages and disadvantages from each other. Moreover, the selection is also depended on an individual's experience and expertise in simulation modeling and those choices – which results in the expressiveness of use.

This issue inspires this research study to develop a methodology that facilitates the mapping of concepts between two domains for implementation – at minimum development cost. The methodology aims to develop a DSSE using a generic existing simulation environment/host simulation language. The core idea behind the methodology is that the individual can enforce his/her own modeling framework to match the

requirements of the simulation environment/language (aka. framework) it is plugged into – by normalizing them into a uniform representation. Obviously, most simulation environments/languages are developed based on object orientation, which is similar to the characteristics of the modeling framework developed by using CSM. Thus, an object is used in common for their representation level.

Based on the object-oriented approach, it allows the individual to exploit the aspects of an object to develop a simulation building block that represents a functionality corresponding to both reality and simulation. Simulation building blocks are then collected in libraries and linked together for testing simulation studies. Having a reasonable number of simulation building block libraries, after creating, editing, and reusing them for a period of time, the individual is able to establish his/her own DSSE on the existing simulation environment/language for resolving similar problems within the domain. Figure 1-1 illustrates the overview of using the methodology for the development of simulation building blocks to create a DSSE. The detailed explanation will be given in the following chapters of this dissertation.

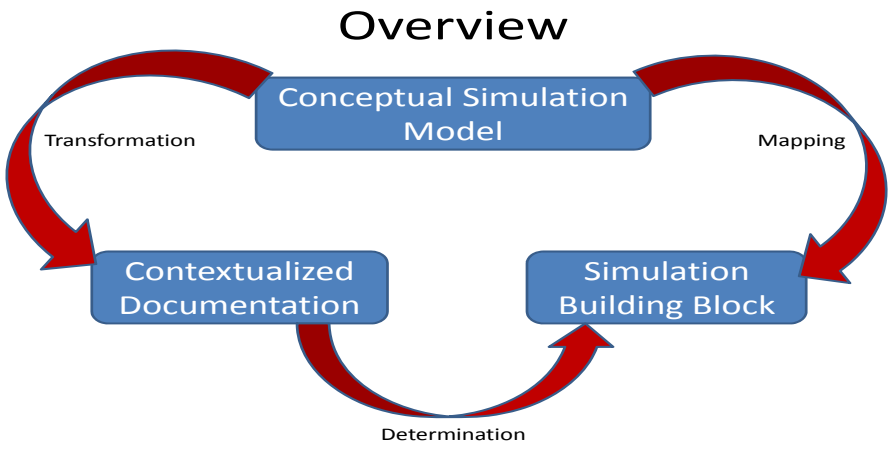

Figure 1 - 1: An overview of the methodology

1-2. Outline of the Dissertation

This dissertation is written under a hybrid format that consists of a collection of three stand-alone papers describing the methodologies critical for the entire research study and four standard-written chapters providing general knowledge. One of the papers has been published in a referee-reviewed conference, while the rest will be submitted to publications in this area. This dissertation is organized as follows. Chapter 2 provides a thorough review of related literature that is needed for understanding the core concepts of this dissertation. Chapter 3 presents the methodology of conceptual simulation modeling to structure a domain specific simulation environment. Chapter 4 presents the methodology of transforming a conceptual simulation model into contextualized documentation. Chapter 5 presents the methodology of mapping between conceptualization and simulation. Chapter 6 provides a case study to demonstrate the implementation of the methodologies developed in this dissertation onto a real world application. Chapter 7 presents the general conclusions that can be drawn from this research study and offers recommendations for further research.

CHAPTER 2

Literature Review

This chapter is designed to be a general overview to summarize the relevant literature that explains the basic concepts and approaches used throughout this research study – to layout a strong foundation for advanced studies (in the following chapters). In addition to Chapter 3 – 5, each chapter also includes a section for literature review to provide a background of study that leads to the development processes for each methodology.

2-1. Simulation

A system is defined as a collection of items that are joined together to characterize interaction or interdependence toward the accomplishment of study or interest (Banks and Carson 1984; Graham et al. 2000). Law (2007) states that *"most real-world systems are too complex to allow realistic models to be evaluated analytically, and these models must be studied by means of simulation."* Simulation is the process of designing a mathematical-logical model that represents a real-world system by imitating the system's characteristics, often over time, and experimenting with this model on a computer (Kelton et al. 2007; Pritsker and O'Reilly 1999). Essentially, computer simulation is seen as a reliable and effective decision-support tool that decision makers use to *"evaluate a system numerically and provide data to estimate the desired true characteristics of the system"* (Law 2007). This allows decision makers to assess a variety of what-if scenarios

to help enhance analysis of the entire system, without having to build, disrupting, and destroying the system (Manivannan 1998; Pritsker and O'Reilly 1999).

According to Law (2007), from simulation modeling world view, models of systems can be classified into three dimensions: static/dynamic; deterministic/stochastic; and continuous/discrete. A static simulation model represents a system at a particular time, while a dynamic simulation model represents a system involving over time. A simulation model is called deterministic when it does not contain any probabilistic (i.e., random) components. However, many systems are modeled as having at least some random input components, and these create stochastic simulation models. A discrete simulation occurs when the dependent variables change only at specified points in simulated time, referred to as event times, whereas in continuous simulation the dependent variables change continuously over simulated time. This dissertation is focused on a methodology that facilitates building dynamic, stochastic and discrete-event simulation models for real world systems.

In the literature, different approaches have been proposed to build simulation models focused on the main operational problems, for instance, queuing and bottleneck problems, resource allocation and scheduling techniques, equipment utilization, throughput, and operational efficiency in the domain systems. These simulation models can be developed from a sequence of operational processes, using different simulation languages (i.e., MODSIM II, SIMAN, and Visual SLAM) and programming languages (i.e., Visual Basic, C, and C++). Usually, simulation models cover both the physical resources (i.e., cranes and vehicles) and the components for control and strategies,

providing a testing environment for algorithms and systems evaluation (Hartmann 2004). Consequently, simulation projects can be carried out for a variety of specific purposes.

As discussed the literature, a simulation model must be reusable, flexible, and extendable to support rapid changes and improved level of detail concerning the operational behavior of the real world systems. To achieve that purpose, an object-oriented modeling approach is used to facilitate the development of simulation models. The structural and behavioral characteristics found in the system can be viewed as an object. Each object contains necessary features that support data abstraction, encapsulation (hiding information), inheritance, and dynamic binding (Rumbaugh et al. 1991; David 1996). These features provide modularity, composability, and reusability essential in developing complex systems and in particular simulation models. The object-oriented approach has been applied in modeling and simulating complex domain systems such as a general port container terminal (Yun and Choi 1999) and the intermodal exchange points in the transportation network (Mathew et al. 2005).

The object-oriented modeling approach has also been used as a concrete foundation for further development of simulation modeling. The standard programming languages, such as C++ and JAVA, provide a powerful framework that greatly facilitates the implementation of object-oriented design and modeling methodology and its capability for creating flexible, modular, and reusable simulation-related extensions. Healy and Kilgore (1998) introduce Silk^{TM}, a JAVA-based simulation, which represents a unique combination of process-oriented modeling constructs and the object-oriented features. Silk^{TM} provides the power and flexibility to program within industry standard development environments. In addition, extending the object-oriented modeling

capabilities with a standard programming language also offers the design capabilities for domain-specific simulation modeling (Ferayorni and Sarjoughian 2007).

2-2. Structure of the Research

The main focus of this research study is to develop a methodology that is capable of generalizing and accessing the structural and behavioral characteristics of the real world systems to support the development of reusable and sustainable simulation models. Conceptual modeling and domain specific simulation environments are determined as the key approaches to lay out the backbone structure for developing the methodology.

2-2-1. Conceptual Modeling

"The conceptual model is a non-software specific description of the simulation model that is to be developed, describing the objectives, inputs, outputs, content, assumptions and simplifications of the model" (Robinson 2004). Pace (2000) also defines a conceptual model as *"a simulation developer's way of translating modeling requirements… into a detailed design framework…, from which the software that will make up the simulation can be built."* Furthermore, a summary of some key facets of conceptual modeling and the definition of the definition of a conceptual model stated by Robinson (2006) are as follows:

- Conceptual modeling is about transforming a problem situation into model requirements to define what is going to be modeled and how;
- Conceptual modeling is iterative and repetitive throughout a modeling study;

- A conceptual model is a simplified representation of the real system;
- A conceptual model is independent of the model code or software, whereas model design includes both conceptual model and the design of the code; and
- The collaboration between the client (i.e., person for whom the model is being built) and the modeler is needed in conceptual modeling.

In brief, conceptual modeling is seen as an approach used to translate the concepts from the application domain into the simulation domain. This approach assists the model developers in capturing the structural and behavioral characteristics of the domain by developing logical and descriptive representations which create an interface representing cross-domain communication between the application domain and the simulation domain.

Zhou et al. (2006) state that *"conceptual modeling (CM) has been recognized as a critical step that directly affects the quality and efficiency of simulation projects. Good CM practice significantly reduces communication barriers, shorten project time, and improve the quality of simulation."* A conceptual model can be described by using knowledge representation notations such as semantic/logical graphs, where the nodes represent concepts (e.g., activities and states), and the arcs represent relationships among concepts (Cyre 1999; Zhou et al. 2004). Nonetheless, there are few methods/tools available to assist in the conceptual modeling phase.

Heavey and Ryan (2006) carry out a selective review of a number of current process modeling methods/tools and categorize those into: formal methods and descriptive methods. Formal methods, such as Petri Nets, Discrete Event System

Specification (DEVS), and State Charts, provide a formal basis and numerous software implementations of these methods. In contrast, descriptive methods have little formal basis and are primarily descriptive software implementations including IDEF3, Integrated Enterprise Modeling (IEM), CIMOSA, and UML State Charts. However, none of these methods/tools are advances sufficiently to support the development of model constructs for a domain specific simulation environment. This is because these current process modeling methods/tools can only implement domain conceptualization but not the simulation implementation.

Setavoraphan (2005) developed a simulation modeling tool based on object-oriented modeling approach and IDEF3 method. This tool represents both the process-oriented view of the target domain and modeling elements (e.g., attribute and operation aspects) required for the simulation implementation and to create a simulation modeling instance for a particular application domain. Further, each simulation modeling instance is able to deploy other object-oriented features such as polymorphism (e.g., methods and procedures), aggregation (e.g., a-part-of relationship for decomposition/specialization), and generalization (e.g., composition). These features provide not only the different levels of representations of the application domain but also the reusability and flexibility of the models, which are needed for developing model constructs used in a domain specific simulation environment. Thus, the simulation modeling instance plays as a key role in developing model constructs in this research study.

2-2-2. Domain Specific Simulation Environment

The same kind of questions exists in the characterization of the operations of a system within a particular domain, and decision makers must answer these over and over again. Verbraeck and Valentin (2002) observe that *"often, however, new simulation models are built for each question, if possible copying some parts of previous models. Structured reuse of simulation components is rarely seen."* In most generic discrete-event simulation environments, such as Arena, Promodel, and Automod, model developers must translate their domain specific requirements into the general modeling components such as queues and resources (Valentin and Verbraeck 2005). To facilitate the development of models in a certain domain, domain-specific simulation languages may be used to create simulation model development environments which provide model constructs that represent domain specific system elements which are familiar to the analyst.

"*The idea behind domain-specific modeling languages is their ability to define the relationships between concepts in a domain and specify key semantics and constraints associated with those domain concepts"* (Ferayorni and Sarjoughian 2007). The concepts in the domain come from the knowledge acquisition processes which can be literature review, domain expert interviews, and actual experience in the field of interest. The concepts are categorized into two groups: basic and special concepts (Zhou et al. 2004). Basic concepts are shared by all models of the domain and belong to the domain of simulation knowledge. Special concepts are used to define and describe the unique characteristics of different application systems, associating with particular domain knowledge. As well, the key semantics that provide modeling elements and the

constraints that set the boundaries in model are retrieved from the knowledge acquisition process and translated into the simulation concepts.

Later, these simulation concepts can be further developed into a domain specific simulation language – which generates a modeling tool for resolving specific and repeated problems. The modeling tool, at a certain mature level of reusability, sustainability, and efficiency, then becomes the core element of the architecture defining and specifying requirements and constraints for designing a domain specific simulation environment. A domain specific simulation language is as much concerned with programming, whereas a domain specific simulation environment is considered as much of a system that facilitates programming, running, and storing model constructs. Keep in mind that a simulation modeling language is the foundation of the development of a simulation environment, so understanding the characteristics of a simulation modeling language is indeed critical – prior to the development of a domain specific simulation environment. However, it is not such necessary to always develop a specific simulation environment to support the domain specific simulation language. This is because the language can be constructed in a simple programming-language environment such as FORTRAN or Visual BASIC.

The development of domain specific simulation languages/environments has been rare but instances have been seen over the years. Grant and Pritsker (1974) constructed the Electroplating Simulation Program (ESP) as a domain specific simulation modeling language for the evaluation of production, waste discharge and housekeeping aspects of existing electroplating processes and also for the evaluation of potential changes in those plating processes for improved pollution control. The Safeguards Network Analysis

Procedure (SNAP) is another example of a domain specific simulation language/environment used for evaluating the resistance of a fixed-site safeguards system to sabotage or theft (Miner and Grant 1978). However, due to not only the limitations of the domains themselves but also the requirements for advanced simulation skills, it seems difficult to draw model developers' interests to building a domain specific simulation language/environment – that is not only time consuming but perhaps one-time use.

There have been several panel discussions, e.g., the Winter Simulation Conferences, mentioning the use of domain specific simulation languages/environments as the next step for discrete event simulation research. According to Valentin and Verbraeck (2005), they conclude the advantages of applying the approach as follows:

- Problem owners have a better understanding of the simulation model because the concepts of the conceptual model can be recognized in the simulation;
- New simulation experiments are easy to generate;
- The simulation model is easier to validate because only the applicability of the model constructs needs to be checked and not the inner-workings; and
- The simulation model needs less instances of model constructs, with improved overview and model management.

These advantages support and encourage modern model developers to develop simulation models using model constructs that represent domain specific system elements, which allow them to carry out many simulation projects for common but complicated domains. For example, a domain specific simulation environment for the Automatic Guided

Vehicles (AGVs) system used between the airport Schiphol and the flower auction Aalsmeer in the Netherlands has been developed to test an advanced control system called TRACES, for the design of the terminals, the control mechanisms, and the AGVs (Heijden et al. 2002). For airport terminal modeling, a domain specific simulation environment can also be built to provide model constructs for different simulation studies (Verbraeck and Valentin 2002).

The dissertation of Saanen (2004) also shows that a domain specific simulation environment facilitates the model developer in instantiating and parameterizing particular types of elements related to container terminal operations, such as container cranes, for the simulation model instead of developing the detailed behavior of each crane. It is clearly seen that the simulation modeling elements can be reused and implemented for other specific projects under the certain domain. This helps the simulation model developer or user shorten time consuming and reduce constraints for building a new simulation model.

Although *"domain specific simulation environments are often incomplete, hard to maintain, and model developers need to overcome initial low trust for these environments"* (Valentin and Verbraeck 2005), the capabilities of domain specific simulation modeling language to provide environments of reusability and faster model development and experimentation are still crucial. To address these advantages, domain specific simulation environments need to match a set of requirements suggested by Valentin and Verbraeck (2005), as follows:

a) Requirements for domain specific simulation environments:

- *Usable within several simulation studies*; A domain specific simulation environment should not be used only in one case, but it should be suitable for easily developing simulation models for several simulation studies.
- *Usable at different levels of abstraction, or detail*; A domain specific simulation environment should provide several model constructs available to represent one system element at different levels of abstraction by representing different complexities.
- *Clearly define the scope of applicability of the domain specific environment so the user knows when to use and when not to use*; Adjusting the existing model constructs or developing new constructs might be needed when the domain specific simulation environment is not suited for use for a certain problem.
- *Easily extendable with new model constructs*; New model constructs can be added to a domain specific simulation environment to represent system elements at a different level of abstraction/detail.
- *Support material to gain trust*; Sufficient support material, e.g., a user manual, online documentation, etc. should be available to show the users of the domain specific simulation environment how system elements are to be applied in developing model details.
- *Additional analysis tools or instruments to support understanding of the outcomes of simulation models*; Output analysis tools should be provided to enable model developers to analyze and observe the outcome of their

simulation model that represents their system effective. These should include statistical analysis as well as graphical analysis tools.

- *Ability to easily build simulation models that are understandable for problem owners and show valid behavior*; A domain specific simulation environment should consist of sufficient model constructs that can represent the various system elements in a way understandable and easily used by problem owner.

b) Requirements for model constructs in domain specific simulation environments;

- *Follow basic rules of systems thinking and software engineering*; The development of model constructs of domain specific simulation environments should follow the concepts of decomposition and design of interfaces.
- *User interface for parameterization in terminology of problem owner and problem domain*; The user interfaces should contain terms that the problem owner is used to and allow him/her to set parameters of the model construct via the user interfaces.
- *Not too much functionality in one model construct*; Performance indicators, parameters, and functionalities of a model construct should be set appropriately and not be overly complex.
- *Performance indicators that make sense to problem owners*; The model constructs should provide performance indicators that reflect the interests

of the problem owners rather than just default statistics – to enable them to trust the model and its outcomes.

- *Model constructs separated for physical (the existing or planned system) and control system (the logical elements which have no physical existence and also control model execution details) elements*; This separation makes the model constructs easier to use and more flexible.
- *Generate errors and warnings for model developers during model development*; This support is to give guidance if a model developer is doing things that are not entitled or matching with the model constructs. The model development process using domain specific environments will automatically generate documentation that defines the model and makes it easy to expand later, perhaps by other developers.

c) Requirements for supporting the design of domain specific simulation environments;

- *Support developers of domain specific simulation environments*; The design methodology should not make the process in building a domain specific simulation environment unnecessarily difficult for the developers.
- *Provide insight into the complexity of the domain for problem owners and future model developers*; Knowledge acquisition is needed for developers to receive input from a problem owner to develop model constructs that can be used in several simulation studies and represent system elements valid and understandable.

- *Provide insight in required data/information/system knowledge*; All kinds of required domain knowledge should be provided by the problem owners to developer in descriptive details. Metadata should be included in the model development to enable self-documentation and future expansion.
- *Provide definition and overview of deliverables*; Deliverables of the developer of the domain specific simulation environment during the development process will enable trust and understanding between the problem owners and model developers in the design of the domain specific simulation environment.

d) Guidelines for use of domain specific simulation environment.

- *Make sure that all steps of a simulation study are performed*; It is important to perform all process steps (i.e., formulate problem, specify model, build model, simulate model, and use model) of a full simulation study for it to be valid.
- *Pay attention to trust of model developers in the domain specific simulation environment*; Ensure that the model constructs of the domain specific simulation environment match with the problem within the domain.
- *Evaluate the selection of model constructs*; Specifications should be provided to the model developer with insight how to appropriately select model constructs corresponding to different levels of abstraction/detail.

The requirements stated above can be used to define reference architecture for developing domain specific simulation environments, which facilitate the model developers to define minimum configurations to design and use model constructs.

This research seeks to provide a methodology for building domain specific simulation environments. The goal is to use this methodology to build simulation applications that are self documenting, easy to expand, and easy to use by users that go beyond the model builder, as simulation application is today. To extend the research discussed above and to accomplish this goal, the development processes of building domain specific simulation environments have been organized into three steps: conceptualization, documentation, and translation. Conceptualization involves defining the elements of the domain important in models and their relationships. Documentation provides the detail necessary to actually build a simulator. Translation is concerned with linking the details developed in documentation to the components of an existing modeling language such that the domain specific environment can be realized. The goals are twofold: to easily build a "one time" use application that is well documented, and, to build a reusable modeling environment that can solve future design problems as they arise.

2-3. References

1. Banks, J. and J. S. Carson (1984). *Discrete-Event System Simulation*, Englewood Cliffs, NJ: Prentice-Hall, Inc.
2. Cyre, W. R. 1999. Conceptual Modeling and Simulation. In *Proceedings of the 1999 IEEE International Conference on Computer Design*, 293-296.

3. David, R.C.H. (1996). *Object-Oriented Analysis and Simulation*, 1st ed., Addison Wesley Longman.
4. Ferrayorni, A.E., and H. S. Sarjoughian (2007). Domain driven simulation modeling for software design, *SCSC*, 297-304.
5. Graham, D.W., Cassady, C.R., Bowden, R.O., and LeMay, S.A. (2000). Modeling intermodal transportation systems: establishing a common language, *The Transportation Law Journal*, 27, 55-68.
6. Grant, F.H. and Pritsker, A.A.B. (1974). Models of cadmium electroplating processes. *NSF(RANN) GRANT GI-35106*, Purdue Univesity.
7. Hartmann, S. (2004). Generating scenarios for simulation and optimization of container terminal logistics, *OR Spectrum*, 26, 171-192.
8. Healy, K.J. and Kilgore, R.A. (1998). Introduction to SILK™ and JAVA-based simulation, *Proceedings of the 1998 Winter Simulation Conference*, 327-334.
9. Heavey, C. and J. Ryan. 2006. Process modeling support for the conceptual modeling phase of a simulation project. In *Proceedings of the 2006 Winter Simulation Conference*, ed. L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto. 801-808. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
10. Heijden, M.C. van der, van Harten, A., Ebben, M.J.R., Saanen, Y.A., Valentin, E.C., and Verbraeck, A. (2002). Using simulation to design an automated underground system for transporting freight around Schiphol Airport, *Interfaces*, 32 (4), 1-19.

11. Kelton, W.D., Sadowski, R.P., and Sturrock, D.T. (2007). *Simulation with Arena*, 4th ed., Boston: McGraw Hill.

12. Law, A.M. (2007). *Simulation Modeling and Analysis*, 4th ed., Singaport: McGraw Hill International.

13. Manivannan, M.S. (1998). Simulation of logistics and transportation systems. In J. Banks (Ed.), *Handbook of simulation, principles, methodology, advances, applications, and practice* (pp. 571 – 602). New York: John Wiley & Sons, Inc.

14. Mathew, R., Leathrum, J.F. Jr., Mazumdar, S., Frith, T., and Joines, J. (2005). An object-oriented architecture for the simulation of networks of cargo terminal operations, *JDMS*, 2 (2), 101-116.

15. Miner, R.J. and Grant, F.H. III. (1978). *User's guide for SNAP*. Pritsker & Associates, Inc.

16. Pace, D. K. 2000. Ideas about simulation conceptual model development. *Johns Hopkins APL Technical Digest* 21: 327-336.

17. Pritsker, A.A. and O'Reilly, J.J. (1999). *Simulation with Visual SLAM and AweSim*. 2nd ed., New York: John Wiley & Sons.

18. Robinson, S. 2004. Simulation: The practice of model development and use. Wiley.

19. Robinson, S. 2006. Conceptual modeling for simulation: Issues and research requirements. In *Proceedings of the 2006 Winter Simulation Conference*, ed. L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto. 792-800. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

20. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991) *Object-oriented modeling and design*, New York: Prentice Hall.
21. Saanen, Y.A., (2004). *An approach for designing robotized marine container terminals*, Doctoral Dissertation, Delft University of Technology, Netherlands.
22. Setavoraphan, K. 2005. Conceptual simulation modeling for regional distribution center systems, Thesis, Indiana State University, Indiana, USA.
23. Valentin, E.C. and Verbraeck, A. (2005). Requirements for domain specific discrete event simulation environments, *Proceedings of the 2005 Winter Simulation Conference*, 654-663.
24. Verbraeck, A. and Valentin, E. (2002). Simulation building blocks for airport terminal modeling, *Proceedings of the 2002 Winter Simulation Conference*, 1199-1206.
25. Yun, W.Y. and Choi, Y.S. (1999). A simulation model for container-terminal operation analysis using an object-oriented approach, *International Journal of Production Economics*, 59, 221-230.
26. Zhou, M., Son Y.J., and Chen, Z. (2004). Knowledge representation for conceptual simulation modeling, *Proceedings of the 2004 Winter Simulation Conference*. 450-458.
27. Zhou. M., Q. Zhang, and Z. Chen. 2006. What can be done to automate conceptual simulation modeling?. In *Proceedings of the 2006 Winter Simulation Conference*, ed. L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto. 809-814. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

CHAPTER 3

Conceptual Simulation Modeling: The Structure of Domain Specific Simulation Environment

"Reproduced with automatic permission from [Setavoraphan, K. and Grant, F. H. (2008) Conceptual simulation modeling: The structure of domain specific simulation environment from The Winter Simulation Conference 2008, 975-986]. It has been modified somewhat to reflect current advances in this research."

Abstract

This chapter focuses on the development of a conceptual simulation modeling tool that can be used to structure a domain specific simulation environment. This approach can be used to structure either the development of a single-application model or an environment appropriate for building several models in a specific domain.

The issues in Software Engineering and Knowledge Engineering such as object-oriented concepts and knowledge representations are addressed to identify and analyze modeling frameworks and patterns of a specific problem domain. Thus, its structural and behavioral characteristics can be conceptualized and described in terms of simulation architecture and context. Moreover, symbols, notations, and diagrams are developed as a communication tool that creates a blueprint to be seen and recognized by both domain experts and simulation developers, which lead to the effectiveness and efficiency in the simulation development of any specific domains.

3-1. Introduction

In the past ten years, there have been several panel discussions at, e.g., the Winter Simulation Conferences (Zhou, Son, and Chen 2004; Heavey and Ryan 2006; Robinson

2006a), the OR Society Simulation Workshop (Robinson 2006b, Wang and Brooks 2006), and the BETADE Workshop (Verbraeck and Dahanayake 2002), which acknowledge the use of conceptual modeling (CM) approach and domain specific simulation environment (DSSE) approach as a critical step to improve the quality and efficiency of discrete event simulation research studies/projects. The literature mainly states that good practice of these two approaches significantly reduce communication barriers, organize model structure, shorten project time, and improve simulation development processes (Vreede, Verbraeck, and Eijck 2003; Valentin and Verbraeck 2005; Zhou, Zhang, and Chen 2006). Although their advantaged are addressed and supported in the same direction by several simulation studies, CM and DSSE still have so far received little attention from simulation developers because CM is viewed as more of an art than science (Brooks 2006), while DSSE is lack of trust of those (Valentin and Verbraeck 2005).

Numerous articles of, for example, Cyre (1999); Deursen, Klint, and Visser (2000); Pace (2000); Yilmaz and Oren (2004); Valentin and Verbraeck (2005); and Robinson (2006a, 2006b), propose ideas on definitions, requirements, limitations, and methods for the development of CM and DSSE to overcome the struggles in those simulation developers' mind. However, most of them are still reluctant to apply CM and DSSE approach to develop their simulation projects. This is because only a few number of literature demonstrate how to transform and develop those concepts into a standard method/tool that can be used to capture and describe elements required for both CM and DSSE. A research study by Teeuw and van den Berg (1997), for instance, introduces the conceptual framework as developed in their testbed project by using symbols and

notations to describe a system's behaviors, relations, and entities. A conceptual model can also be built by using knowledge representation notations such as semantic/logical graphs, where the nodes represent concepts, and the arcs represent relationships among concepts (Cyre 1999; Zhou, Son, and Chen 2004). Furthermore, a selective review of a number of current modeling methods/tools carried out by Heavey and Ryan (2006) shows that simulation developers have become more aware of using standard methods/tools such as Petri Nets, DEVS, IDEF3, and UML, to develop their own conceptual models. As well, simulation building block terminology is proposed by a research team, BETADE, at Delft University of Technology, The Netherlands, in 2001 to provide a standard methodology for the DSSE development (Verbraeck and Dahanayake 2002), instead of relying on old-fashioned programming. It can be said that the trend of the CM and DSSE research studies is moving forward to acquiring more sophisticated, universal, and user-friendly methods/tools to serve both CM and DSSE requirements effectively and efficiently. However, none of the available methods/tools exists to satisfy this demand.

One of the critical reasons is that both CM and DSSE are viewed from different perspectives that not only isolate them into two distinct disciplines but also eliminate an opportunity for their collaborative modeling and representation formalisms in developing simulation projects. The fact that the foundations of modeling concepts and processes for CM and DSSE are similar allows them to overlap in some aspects (see Valentin and Verbraeck 2002; Verbraeck and Valentin 2002; Vreede, Verbraeck, and Eijck 2003; Zhou, Setavoraphan, and Chen 2005). The concepts developed by CM processes are transformed into the logical and structural components for DSSE, whereas the result of the implementation of those in DSSE becomes a feedback mechanism that provides a

better understanding of both the problem owners to improve their conceptual models for better DSSE (see Figure 3-1). This iterative CM and DSSE development process is performed until DSSE generates a complete standard set of the specifications and patterns – that can be transformed into basic building blocks. Then these building blocks are integrated to form a stand-alone simulation template, which is capable of representing systems as a domain specific simulation model for the simulation builders as well as a domain specific conceptual model for the domain experts. Consequently, the simulation template is delivered as the basic component to develop (commercial) simulation software and a knowledge-based simulation system.

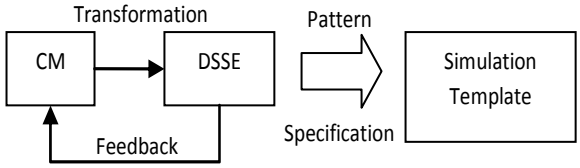

Figure 3 - 1: The relationship between CM and DSSE

The main idea of this research study is to focus CM concepts and techniques to further its potentials in characterizing the general behavioral and structural characteristics of a specific problem domain to generate a model that contains processes, elements, controls and requirements for simulation. This is generally referred to as conceptual simulation modeling (CSM). Thus, the CM approach is determined to be the backbone of the development of a CSM tool that can be used to structure a DSSE for discrete-event simulation modeling problems. Section 3-2 briefly describes the key concepts within Software Engineering (SE) and Knowledge Engineering (KE) that comprise the baseline foundations of CSM development. The concepts are formalized into different layers and representations to construct standard symbols, notations, and diagrams to be used in

CSM, which is illustrated in Section 3-3, including an example for illustration. Finally, conclusions and further research ideas are given in Section 3-4.

3-2. Key Concepts

The simulation development process is a kind of problem-solving process that determines a context, environment, and boundary of a real-world problem domain to be developed and used for experimentation. CSM plays a critical role as a specialized tool to facilitate the understanding of the problem, support communication between domain experts and simulation developers, and represent the knowledge needed by the simulation system to simulate/solve the problem. CSM also uses the underlying convergent concepts used to develop conceptual models from both SE and KE, which are: first, object-oriented concepts from the discipline SE; and second, knowledge representations (or "levels" in some literature) from the discipline KE (see more details about the CM methods in Dieste et al. 2001). However, CSM requires more advanced approaches to access, formalize, and use these concepts to overcome the barriers and drawbacks during constructing and transforming a conceptual simulation model. These are decomposition and composition approaches.

3-2-1. Decomposition Approach

The significant problem found in applying object-oriented concepts and defining knowledge representations is how to determine and represent the concepts derived from both application knowledge and simulation knowledge (see Zhou, Son, and Chen 2004) at an appropriate abstract level to satisfy the efficiency of CSM. The determination of the

level of abstraction is strongly influenced by the objectives of the design or the questions needed to be answered (Benjamin et al. 1993). Nevertheless, no single abstract model is sufficient to be expressed at different levels of precision and to attack specific problems (Booch, Jacobson, and Rumbaugh 1999).

Decomposition is a crucial approach used to handle complexity and represent the behavioral and structural characteristics of the target problem domain at an appropriate level of detail (Zhou, Setavoraphan, and Chen 2005). Moreover, it is the paramount idea of object-oriented concepts (Meyer 1997), which is used to formalize modeling frameworks for CSM due to its inherent support for abstraction-centric, reusable, and adaptable design (Zhou, Zhang, and Chen 2006). Using abstraction, aggregation, and specialization aspects, the object orientation provides decomposition to the simulation developers to capture descriptions at varying abstraction levels and integrate all those sub domains into a comprehensive behavioral description for the problem domain.

The central idea of decomposition is to breakdown the complexity of a problem domain into less complex sub domains by eliminating irrelevant details and highlighting the important behavioral and structural characteristics (Hofmann 2004). The frames of reference of these sub domains can be extended or modified to satisfy the objectives of the design (Lee and Wyner 2003). This allows each sub domain to interact with a set of other sub domains to provide complete representation and enable the modeling of the domain (Davis 2001). For further benefits and criteria of decomposition, see the works by Davis (2001) and Hofmann (2004).

To avoid the tendency of characterizing CSM as “more of an art than science”, the constraints of decomposition need to be specified to manage abstraction of the

domain at hand. A proceeding paper (Zhou, Setavoraphan, and Chen 2005) proposes a set of mathematics notations to describe the functions and constraints of two types of decomposition: serial decomposition and parallel decomposition. In this chapter, a process-oriented view is used to define a problem domain as a set of sequenced processes in a generic level, which can be decomposed into (multi) sub lower-level processes, controlled by constraints. These constraints are addressed here in narrative description instead of mathematics notation.

First, serial decomposition must satisfy the following constraints:

- A top level process must be decomposed into sub processes in order to form a serial-sequence order, and each sub process' input and output specified must be available when executed; and
- The set of sub processes must be a partition of its higher-level process, completely dividing the functionality of the higher-level process; and
- Precedence relation is required among the sub processes; and
- The attributes defined for the sub processes and the aggregation of these sub processes must be consistent with the attributes defined for the decomposed process; and
- The input and output external to the set of sub processes must match the original input and output associated with its higher-level process; and
- The total process time is a sum of sub process times.

Second, parallel decomposition mostly follows the constraints defined in serial decomposition. The difference is that parallel decomposition requires Boolean logical

operators, for example, AND, OR, and XOR, to support the functionalities of logical branching out (e.g., deterministic branching or probabilistic branching) from the predecessor of the original process. These logical operators allow decomposition to specify several alternative combinations of causes and effects to extend the consequences of the original process (Bell, Snooke, and Price 2005). As a result of decomposition, the simulation developers are able to capture a set of sequential processes within the domain, corresponding to the simulation requirements to create a conceptual simulation model at the appropriate levels of detail.

3-2-2. Composition Approach

Another encountered problem is that most of products (outcomes) from CSM fail to be reused in new simulation applications. Reusability of models, modules, or elements is a challenge not only at abstraction level (conceptual simulation models) but also at implementation level (domain specific simulation environments). The failure of capturing and explicitly representing specifications of constraints, objectives, features, and the semantics of components at the conceptual level generates an incompatible framework of those within the domain specific simulation environment, reducing the reusability of model constructs. On the other hand, the incompleteness of encapsulating (modularizing) and inheriting data (e.g., objects and processes) of the model constructs creates the loss of the model functionalities and contexts at the implementation level, affecting the trust of simulation developers in the conceptual simulation models, which reduces their reusability. Thus, an approach is needed to support model reusability for these two levels.

Composability is described as an approach with compositional mechanisms that provides "*the ability to compose models/modules across a variety of application domains, levels or resolution and time scales*" (Kasputis and Ng 2000), plus "*the capability to select and assemble simulation components in various combinations into simulation systems to satisfy user requirements*" (Petty and Weisel 2003). Though, the current capability in composability is limited (Kasputis and Ng 2000) due to the complexity of the selection of components in the context of simulation (Winnell and Ladbrok 2003) and is determined to be an NP-hard problem. Still the simulation developers can apply this approach to design a frame of reference for the possible compositions to increase the possibility of model for reuse in any environment.

In general, there are two types of composability: syntactic composability and semantic composability, used to represent the modeling formalism for the selection of components (Petty and Weisel 2003). First, syntactic composability requires compatible implementation details which include timing mechanisms and interface specifications for all possible compositions. Second, semantic composability requires a meaningful/valid composition. "*Both syntactic and semantic composability are necessary for simulation composability*" (Bartholet et al. 2004) in terms of the development of the interfaces and the component internals within the defined simulation framework.

In addition, composability can be conducted in two dimensions which are referred to as the horizontal and vertical dimension (Page and Opper 1999). In the horizontal dimension, the components are applied in terms of peer-to-peer integration with respect to the scope of the model by justifying a level of modeling abstraction with respect to a set of modeling objectives, which is fundamentally hard to do correctly. This is because

"*the presence of multiple models and multiple levels of abstraction increases the difficulty*", which has been referred to as the multiresolution modeling problem (Page and Opper 1999). On the other hand, composability in the vertical dimension facilitates a level of modeling abstraction through aggregation/disaggregation, which may in turn not provide the best or even a valid solution. It can be seen that the vertical composability is more flexible to facilitate the composition of decomposed components to create a model corresponding to the specific requirements. Therefore, composability in the vertical dimension is mainly applied in this study to avoid complexity, though, it may compensate with the loss of validity.

Butler (1998) identifies three crucial components: assembly, extension, and parameterization, as follows:

- Assembly: connecting existing modeling components in possibly unique ways through a common environment;
- Extension: modifying or extending the original functionality of an existing model component through either function override or selective feature activation/deactivation; and
- Parameterization: changing parameters which control the operational and behavioral characteristics in an existing model component.

He also states the design requirements for composability to shape the technical and operational approach in his work. Moreover, a number of research studies have been conducted to investigate modeling formalism, context, dependency, and framework for

model reuse (Yilmaz and Oren 2004; Spiegel, Reynolds and Brogan 2005; Sarjoughian and Huang 2005) to improve and facilitate model composability.

The results from these studies support not only the techniques of model composability but also the impact of model composability choices in a variety of degrees of model composition, limitation, and complexity. The idea behind these results shows that the concepts, theories, and techniques of model composability consist of abstraction, hierarchy (aggregation/disaggregation), and encapsulation that belong to the object-oriented aspects (Sarjoughian and Huang 2005). Use of these aspects is crucial in developing a framework that provides standardized patterns to define the scopes of design and development of model components and representations for CSM and DSSE. It must be kept in mind that as long as a set of the model components and representations are a pattern-based development within the framework, the reusability of the conceptual simulation models and the model constructs in DSSE is more flexible and more meaningful when conducting a new simulation project. Moreover, it needs to make sure that the composition of the model components and representations must be tested in the level of CSM prior to implement those in DSSE to avoid the conflicts of functionalities between these two levels.

3-3. Illustration of a CSM prototype

3-3-1. Background of Study

In the previous section, the importance of the decomposition and composition approach is illustrated by a means of the application and control to the use of the key concepts: the object orientation and knowledge representation, in the development of a

CSM tool. Less attention applied in the management of modeling complexity (levels of detail) and the arrangement of modeling compatibility (levels of selection) results in ineffectiveness and inefficiency of the overall modeling structure and context. Most of the simulation developers know the basic object-oriented concepts described in many publications (e.g., Rumbaugh et al. 1991, Coleman et al. 1993), but few of them recognize the methods of formalism of these concepts to develop robust and reusable knowledge representations as modeling frameworks for simulation (see Zhou, Zhang, and Chen 2006). It has been found out that there are many generic (standard) methods/tools that are available to support CM (e.g., IDEF3, DEVS, Petri Nets, and UML), but they fail to accomplish bidirectional transference of concepts and information between application domain and simulation domain. As a result, most of the time these methods/tools simply create difficulties in the CSM and DSSE construction and translation rather than to achieve the simulation-template's goal.

It can be said that there is a need for a defined simulation modeling framework that facilitates not only domain conceptualization but also simulation implementation. A thesis (Setavoraphan 2005) illustrates a CSM tool, called "Simulation Modeling UOB" (SMU), used to formalize concepts into a simulation modeling framework. This tool is developed from the transformation of knowledge representations in a platform of process descriptions derived from IDEF3 method, collaborating with the object-oriented approach. Each instance in SMU employs both process-oriented and component-based view to represent the processes lying within the target problem domain and the simulation modeling elements (e.g., entities, attributes, and functions) satisfying the simulation requirements. Furthermore, it is able to apply the object-oriented features to

facilitate modeling decomposition and composition. Having the capability to formalize concepts at different levels of detail and to generate robust and reusable modeling frameworks, SMU has been applied to develop conceptual simulation models for a variety of application domains such as warehousing operations (Setavoraphan 2005) and inland waterway lockage operations (Setavoraphan and Grant 2008). However, the current capability of SMU is focused on delivering a detailed modeling framework that provides both static and dynamic representations required for structuring DSSE.

Some examples of simulation projects developed under the DSSE approach include:

- Electroplating Simulation Program, ESP (Grant and Pritsker 1974) by using a programming language;
- Safeguards Network Analysis Procedure, SNAP (Miner and Grant 1978) by developing a network language;
- Airport Terminal Modeling of Amsterdam Airport Schiphol (Verbraeck and Valentin 2002) and the Robotized Marine Container Terminals (Saanen 2004) by using simulation building blocks.

Accordingly from above, it can be said that every single DSSE development fundamentally consists of static modeling components (e.g., physical layouts) and dynamic modeling components (e.g., entities). These fundamental concepts need to be integrated into the CSM tool for better mapping and transforming concepts prior to develop a simulation modeling framework. A research study by Iba, Matsuzawa, and Aoyama (2004) emphasizes on the Model Driven Development created based on Model

Driven Architecture and Executable UML to use high-level modeling languages to enhance the capability of CM in representing the overall behavioral and structural characteristics of a domain, including their interactions, from both static and dynamic views. Their project development supports the idea of improving SMU, the existing CSM tool, by integrating its original concepts with UML to cover its limitations and to be used in this study.

3-3-2. General Structure

The main purpose of this chapter is to deliver a concrete idea that integrates and formalizes the concepts mentioned in the preceding sections by illustrating a CSM prototype temporarily named as "Integrated Simulation Acknowledge Procedure" (ISAP). ISAP is a tool for capturing the concepts in a specific problem domain and transforming them into a set of descriptive processes, static and dynamic modeling components, interactions, and rules/algorithms which are defined within a simulation modeling framework. The framework created by ISAP consists of three layers: the initialization layer (IL), the process layer (PL), and the termination layer (TL) (see Figure 3-2). First, IL provides initial information about the simulation experiment to be performed (e.g., number of simulation runs, number of attributes/variables, and time to begin/end simulation). Second, PL describes the behavioral and structural characteristics of the problem domain and simulation domain. Third, TL sets the procedures of terminating simulation and printing out a simulation output report. Each of these layers consists of a group of ISAP symbols, notations, and diagrams which are arranged to

define and represent modeling structures, elements, and relationships. Within the limited space of this published paper, only the process layer is discussed.

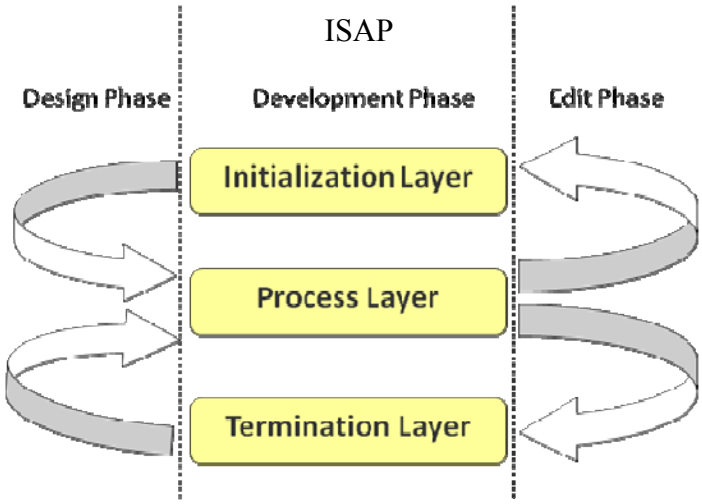

Figure 3 - 2: Three layers in ISAP with three phases

The construction of ISAP is based on the modeling and simulation process (Pritsker and O'Reilly 1999) and adapted into three phases: the design phase, the development phase, and the edit phase (also see Figure 3-2). First, the design phase is to formulate problem and specify model for PL according to the design objectives in IL and TL. Second, the development phase is to build models individually for each layer. Third, the edit phase is to test models and use their feedbacks to correct errors found in these layers, and also this phase needs modification in IL and TL to satisfy the new requirements for PL. Moreover, the construction of PL is divided into two subsystems: static modeling subsystem and dynamic modeling subsystem. Both of them require the use of symbols, notations, and diagrams for robust and reusable representations. An example of an inventory system of a large discount house (Pritsker and O' Reilly 1999) is used to illustrate the construction of these two subsystems in PL.

3-3-3. Demonstration

To illustrate these concepts described above, consider a large discount house that is planning to install a periodic review-reorder point inventory system to control its in-house inventory of a particular radio. This system is able to manage backorders in the case where customers demand the radio when it is not in stock. 80 percent will go to another discount house to find it, determined as lost sales, whereas the other 20 percent will be put on the backorder list and wait for the next shipment arrival. The inventory status is reviewed every four weeks to decide if an order should be placed. The company policy is to order up to the stock control level of 72 radios whenever the inventory position, consisting of the radios in stock plus the radios on order minus the radios on backorder, is found to be less than or equal to the reorder point of 18 radios. The procurement lead time requires constantly three weeks.

3-3-3-1. Static Modeling Subsystem

The first step is to specify the physical characteristics in the target problem domain. It can be seen that the inventory system consists of an actual (in-house) inventory subsystem and a virtual (periodic review-reorder) inventory subsystem. ISAP provides symbols and notations that represent different three static components: BUILD, SPACE, and CROSS. A BUILD component is used to identify a point in a system where some physical objects are moved through or changed their states. A SPACE component is used to identify an area in the system through which physical objects may pass or temporarily stay. A CROSS component is used to identify locations in the system which is the physical objects engaged with multi cross-domain subsystems. In this example,

only BUILD components are used to represent the actual inventory subsystem and the virtual inventory subsystem, where the flows and transition states of e.g., demands and order-signals, take place, shown in Figure 3-3. Each BUILD component is defined with its identical component label that is connected to its dynamic modeling subsystem containing the logical process flows and parameters needed. The connection is made through "@" and followed by a specified dynamic modeling subsystem label (DMSL). An arrow is used to indicate a precedence of movement that may occur in only one direction between two physical components, which means there exists one or more interchanges or flows of objects and information between the components. One of the obvious benefits of having static (physical) components for CSM is a top-view perspective that shows the core structures and the focused frames of the domain, which can be further developed either as apiece (decomposition) or as a whole (composition) within the defined domain structural boundaries.

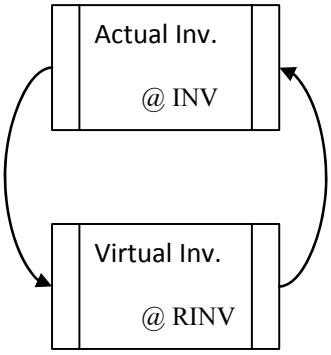

Figure 3 - 3: BUILD components for inventory system

3-3-3-2. Dynamic Modeling Subsystem

The next step is to describe the dynamics of the domain in terms of application knowledge and simulation knowledge, determined as the core of the ISAP development process. Each dynamic modeling subsystem can be view as a document folder that has its own label (DMSL), sub-folder(s) (Ref#), and page number ($ #). Each page is divided into three sections: the SMU section, the relation section, and the sequence-diagram section. The first section follows the major structure described in Setavoraphan (2005), shown in Figure 3-4, whereas the rest of the sections apply the symbols, notations, and diagrams which are adapted from the UML modeling approach (Booch, Jacobson, and Rumbaugh 1999).

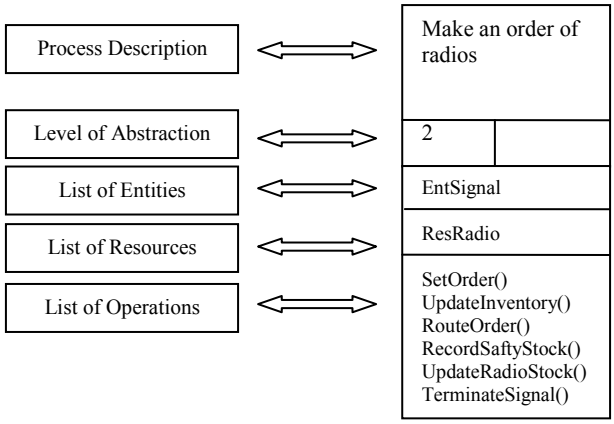

Figure 3 - 4: General structure and an SMU example

Each SMU is used to represent as an intimate simulation (block) module that moves the entities through the process or change the entities' transition states; calls the resources required for the process; and executes the operations to complete the process. As a module, an SMU can be decomposed into two or more sub SMUs to cover the detailed levels of the process. For example, SMU *Make An Order of Radios* (Figure 3-4)

can be decomposed into SMU *Prepare An Order* and SMU *Make A Transshipment*, shown in Figure 3-5. At lower levels of decomposition, the reference number for level of abstraction of a child SMU (X) consists of three distinct numbers separated by periods. The first number is the last number in the reference number of X's parent SMU. The second number is the number assigned to the particular decomposition of the parent SMU in which X occurs (Note: Numbers are assigned to a set of decompositions and SMUs in order of different creations/points of view). Finally, the third number is an actual X's SMU reference number. The relationship between the parent SMU and child SMUs is determined as a-part-of relationship or aggregation in which SMUs representing the entities, resources, and operations of some processes are associated with an SMU representing the entire assembly of processes. Thus, each decomposition must be taken carefully to avoid the loss of details and the incompleteness of the process. As well, the composition of the existing SMUs into a new SMU requires standard/common parameters to reduce the invalidity of the model functionalities, which is similar to the methods used in the object-oriented programming. Suppose that the SMUs in Figure 3-5 are individual SMUS. To compose these two SMUs into one, a crucial requirement is to make sure that they assess the same entities, utilize the same resources, and execute the operations with the same attributes and variables. Also, the flow of entities and operations must be logical sequences.

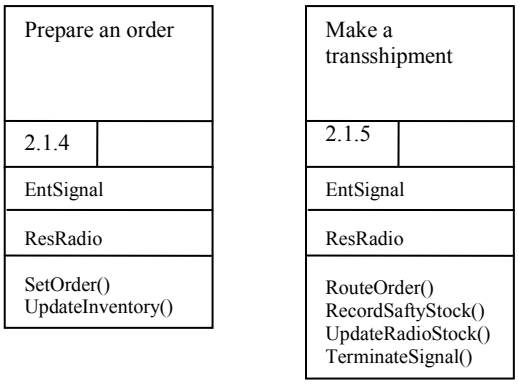

Figure 3 - 5: A decomposition of SMU *Make An Order of Radios*

The relation section provides information of the conditions and decisions for branching, preceding, and interacting between two SMUs. Figure 3-6 gives some examples of notations.

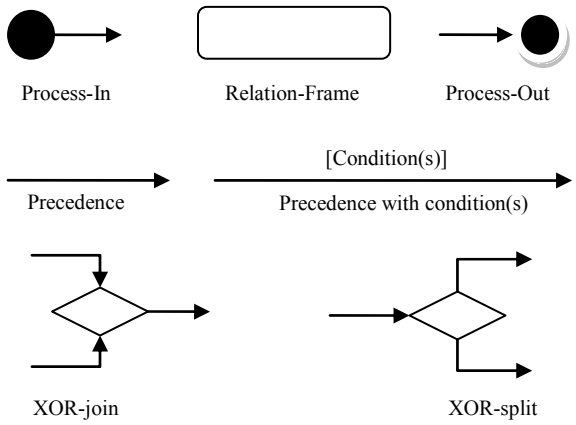

Figure 3 - 6: Some examples of notations for relations

In the Relation-Frame, the precedence and logical relationships that tie SMUs (see Fig. 3-8) represent the flow-paths for the entities, including the conditions that create the alternative flow-paths. This relation-view provides the simulation developers the

conceptual foresee of the entity flow in the sub-system, which supports the verification of logic associated with SMUs.

Finally, the sequence-diagram section shows a series of messages exchanged by a selected set of objects in SMUs, with an emphasis on the chronological course of communication between SMUs – which is used to indicate the status and the responding sequences from taking an action (operation) of the objects related to SMUs. Some crucial notations are shown in Figure 3-7.

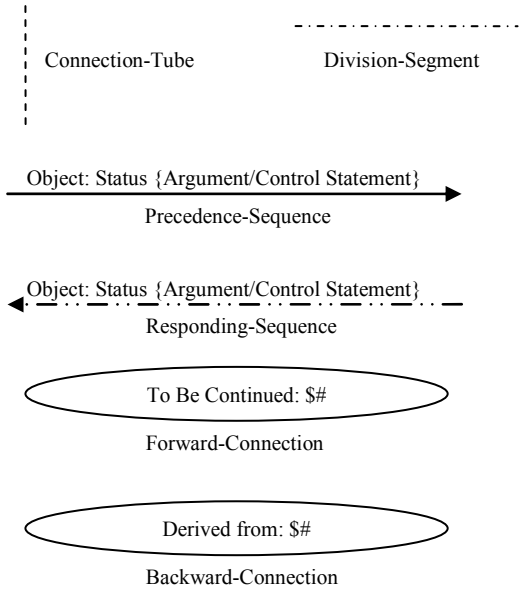

Figure 3 - 7: Notations for the sequence-diagram section

The sequence diagram, on the other hand, can be determined as a conceptual simulation that illustrates a brief simulation run. It includes (see Fig. 3-8 for a better understanding) both Begin and End runs, initial set-up for variables, entities and their flows, resource utilization, variable changes, activities, and time sequences (divided by division-segment and prioritized activity orders). This diagram is also used to pre-check whether or not individual or a set of SMUs have sufficient parameters (e.g.,

entity/resource types, variables, and operations) to fulfill the simulation requirements prior to further DSSE development.

The later step is to provide the descriptions of the objects and operations used in these sections in a tabular form (table). Each table gives not only an object's generic information (e.g., name, type, description, and associated parameters) but also its extension (e.g., event state, rules, and algorithms) if needed. There is no specific regulation in designing a table of description. The design is depended on the demand and detailed level of information.

(Note: due to the size of the tables and figures, they are partially shown in the appendices section for DMSL: RINV as an example)

The final step is to revise every section and connect them together by using Connection-Tube. This line contains data given in each SMU and passes them throughout its length. Thus, the simulation developers are able to keep track of every action and transaction state of the objects, by following the lines (Top-to-Bottom or Bottom-to-Top relation) and other associated notations (Left-to-Right or Right-to-Left relation), which helps support their conceptual thinking. It is seen that the logic behind the development of the ISAP process layer is to access a domain from a very generic component to sub-components with different detailed levels and to maintain the completeness of encapsulation and inheritance of component data for the component reusability. This means that ISAP well deploys the decomposition approach to remove the complexity of conceptual thinking as well as the composition approach to extend the scope of conceptual thinking.

The results of the connection and association of these SMUs, notations, and descriptions are transformed into a network statement. Here is the network statement of DMSL: RINV, as shown below.

Ref# 0:

```
1   SetReorderPoint, Reorder point;
2   CreateSignal, Arrival rate, Time of first arrival, Max # of demands;
3   CheckInventory, Resource#, File resource#, Inventory position;
4   Condition, INV_POS <= REORDER_PT;
5   SetOrder, Order quantity;
6   UpdateInventory, Inventory position;
7   RouteOrder, Lead time;
8   RecordSafetyStock, File#, Resource#, File resource#, Number of radios;
9   UpdateRadioStock, Resource#, File resource#, Number of radios;
10  TerminateSignal, Max# of signals;
11  Condition, INV_POS > REORDER_PT;
12  TerminateSignal, Max# of signals;
```

Each line of the network statement contains sequential-order numbers and operation names with their parameters. Using a network statement is a basic idea found in the structure of commercial simulation software such as Arena© (Kelton, Sadowski, and Sadowski 2002) and AweSim© (Pritsker and O'Reilly 1999) to create simulation modeling frameworks for the construction and control of simulation modules. Therefore, a simulation modeling framework defined by the network statement and by other aspects through the ISAP development process can be used to generate appropriate simulation modules for the DSSE development.

3-4. Conclusions

The specialization of the CM concepts and techniques is taken as the main idea of this research study to improve the CSM approach. This is because CSM has been seen as

a critical approach that is used to shorten gaps of communication between the domain experts and the simulation developers and to reduce difficulties of transformation of the concepts between two different domains of knowledge. However, CSM has been largely ignored, especially when conducted the development of DSSE.

ISAP is a prototype that is developed based on the conceptual modeling approach under the SE and KE disciplines to support the development of a conceptual simulation model. Moreover, ISAP is designed to match with the structural and behavioral characteristics of the DSSE development process. Simulation developers, thus, can apply ISAP to generate robust and reusable simulation modeling frameworks that can be used as blueprints giving designs and instructions for the specific simulation development projects.

Nevertheless, there is still more room for improvement for this ISAP prototype to fulfill other simulation requirements, for example, dynamic parameter assignment, random distributed data generation, database, and simulation-module interface. For this study, the ISAP prototype is expected by the authors that it is able to encourage simulation developers to enhance the current capability of the available modeling methods/tools to take simulation development to the state-of-art level.

3-5. Acknowledgements

The authors would like to thank the Center for Systems Modeling and Simulation at Indiana State University, Kang-Hung Yang, and Zhilli Zhou (Computational Optimization and Logistics Lab, the University of Oklahoma) for their help and cooperation.

3-6. Appendices

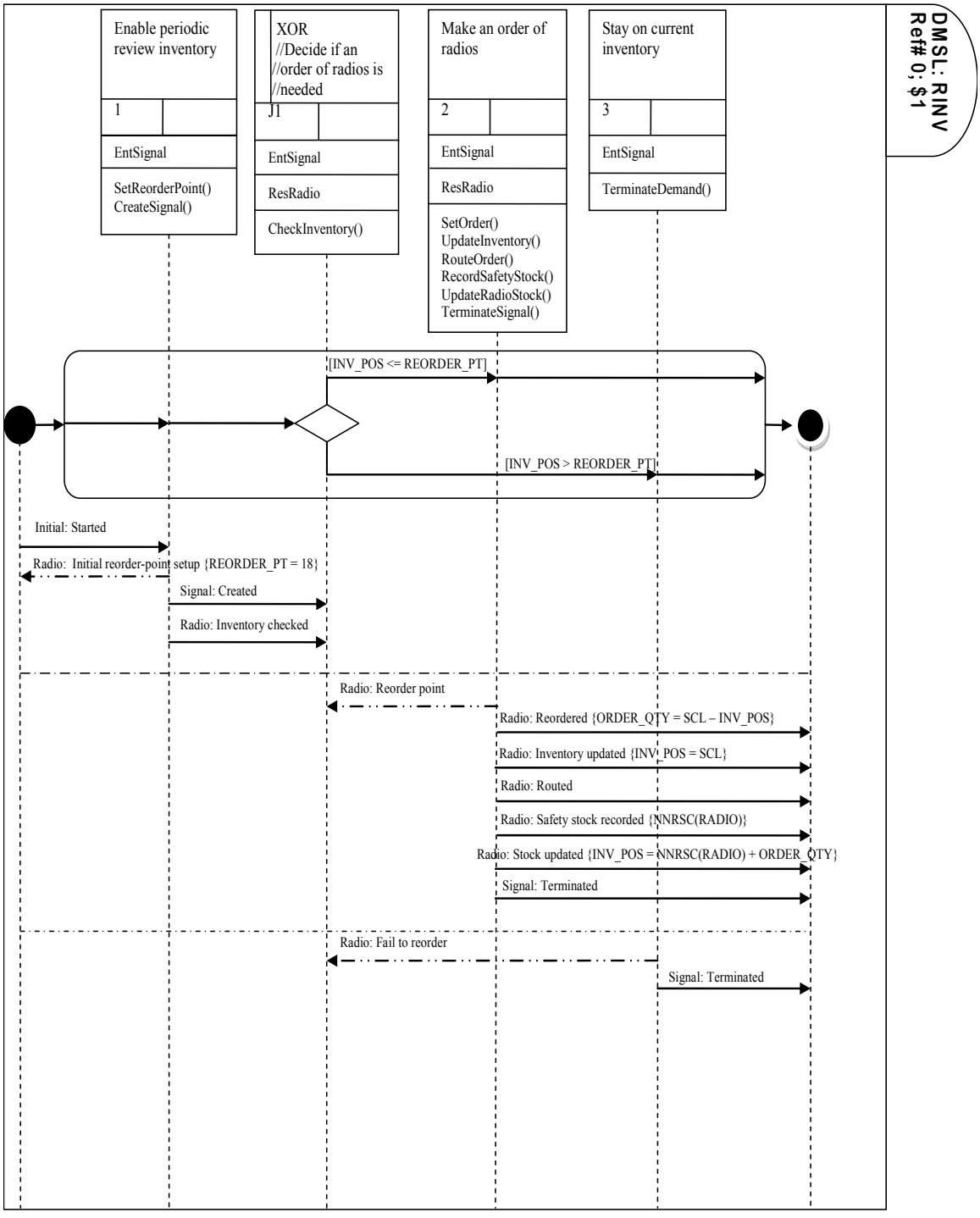

Figure 3 - 8: An example of DMSL: RINV

Table 3 - 1: Description of objects for DMSL: RINV.

Object Name	Type	Description	Parameters
EntSignal	Entity	This object represents a signal entity to enable the periodic review-reorder inventory system.	: Arrival rate
ResRadio	Resource	This object represents radio resources that can be altered corresponding to the inventory status.	: Resource# : Resource capacity : Queue# : Queue capacity

Table 3 - 2: Description of operations for DMSL: RINV.

Operation Name	Actor	Description	Attributes	Global Variables
CheckInventory()	ResRadio	An action is to determine if radio is available to satisfy a customer demand.	N/A	: Resource# : File resource# : Inventory position
CreateSignal()	EntSignal	A signal entity is created to the systems.	: Arrival rate : Time of first arrival : Max# of signal entities	
SetReorderPoint()	N/A	An action is to set a reorder point for the inventory system.	N/A	: Reorder point
SetOrder()	N/A	An action is to set a quantity of order	N/A	: Order quantity
RecordSafetyStock()	ResRadio	Number of radios are available at that time of arrival of shipment.	N/A	: File# : Resource# : File resource# : Number of radios
RouteOrder()	ResRadio	A quantity of order is transport to the discount house's inventory with a lead time	N/A	: Lead time
UpdateInventory()	N/A	Number of radios in the inventory are updated	N/A	: Inventory position
UpdateRadioStock()	ResRadio	Number of radios are updated.	N/A	: Resource# : File resource# : Resource capacity
TerminateSignal()	EntSignal	Each signal entity is terminated.	N/A	: Max# of signals

Table 3 - 3: Description of variables for DMSL: RINV.

Variable	Equivalence	Description
INV_POS	Inventory position	It contains the overall number of radios derived from both sub-systems.
NNRS(RADIO)	Number of radios	It shows the exact number of radios at the physical inventory.
ORDER_QTY	Order quantity	It indicates the number of radios per an order.
REORDER_PT	Reorder point	It sets the minimum number of radios in the physical inventory for reorder.
SCL	Stock control level	It limits the maximum number of radios in the physical inventory.

3-7. References

1. Bartholet, R. G., D. C. Brogan, P. F. Jr. Reynolds, and J. C. Carnahan. 2004. In search of the philosopher's stone: Simulation composability versus component-based software design, In *Proceedings of the Fall 2004 Simulation Interoperability Workshop*, Orlando, Florida.
2. Bell, J., N. Snooke, and C. Price. 2005. Functional decomposition for interpretation of model based simulation. In *Proceedings of the 19th International Workshop of Qualitative Reasoning*, 192-198.
3. Benjamin, P. C., M. Blinn, F. Fillion, and R. J. Mayer. 1993. Intelligent support for simulation modeling: a description-driven approach. In *Proceedings of the 1993 Summer Computer Simulation Conference*.
4. Booch, G., I. Jacobson, and J. Rumbaugh. 1999. *The Unified Modeling Language user guide*. Massachusetts:Addison-Wesley.
5. Brooks, R. J. 2006. Some thoughts on conceptual modeling: Performance, complexity and simplification. In *Proceedings of the 2006 OR Society Simulation Workshop*.

6. Butler, B. 1998. Simulation composability for JSIMS, In *Proceedings of the 2nd International Workshop on Distributed Interactive Simulation and Real-Time Applications*, 4-14.
7. Coleman, D., P. Arnold, S. Bodorff, C. Dollin, and H. Gilchrist. 1993. *Object oriented development: The fusion method*. Prentice Hall.
8. Cyre, W. R. 1999. Conceptual Modeling and Simulation. In *Proceedings of the 1999 IEEE International Conference on Computer Design*, 293-296.
9. Davis, W. J. 2001. Distributed simulation and control: The foundations. In *Proceedings of the 2001 Winter Simulation Conference*, ed. B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer. 187-198. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
10. Deursen, A. V., P. Klint, and J. Visser. 2000. Domain-specific languages: An annotated bibliography. Available via <http://homepages.cwi.nl/~arie/papers/dslbib/> [accessed February 14, 2008].
11. Dieste, O., N. Juristo, A. M. Moreno, and J. Pazos. 2001. Conceptual modeling in software engineering and knowledge engineering: Concepts, Techniques and trends. *Handbook of Software Engineering & Knowledge Engineering* 1: 733-766. New Jersey: World Scientific.
12. Grant, F. H. and A. A. B. Pritsker. 1974. Models of cadmium electroplating processes. *NSF (RANN) GRANT GI-35106*, Purdue University.
13. Heavey, C. and J. Ryan. 2006. Process modeling support for the conceptual modeling phase of a simulation project. In *Proceedings of the 2006 Winter Simulation Conference*, ed. L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D.

M. Nicol, and R. M. Fujimoto. 801-808. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

14. Hofmann, M. A. 2004. Criteria for decomposing systems into components in modeling and simulation: Lessons learned with military simulations. *Simulation* 80: 357-365.
15. Iba, T., Y. Matsuzawa, and N. Aoyama. 2004. From conceptual models to simulation models: Model Driven Development of Agent-Based simulations. In *Proceedings of the 9th Workshop on Economics and Heterogeneous Interacting Agents*. 1-12 Kyoto, Japan.
16. Kasputis, S. and H. C. Ng. 2000. Composable simulations. In *Proceedings of the 2000 Winter Simulation Conference*, ed. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick. 1577-1584. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
17. Kelton, W. D., R. P. Sadowski, and D. A. Sadowski. 2002. *Simulation with Arena*. 2nd ed. New Yok: McGraw Hill.
18. Lee, J. and G. M. Wyner. 2003. Defining specialization for data flow diagrams. *Information Systems* 28: 651-671.
19. Meyer, B. 1997. *Object-oriented software construction*. 2nd ed. London: Prentice Hall.
20. Miner, R. J. and F. H. Grant. 1978. *User's guide for SNAP*. Pritsker & Associates, Inc.
21. Pace, D. K. 2000. Ideas about simulation conceptual model development. *Johns Hopkins APL Technical Digest* 21: 327-336.

22. Page, E. H. and J. M. Opper. 1999. Observations on the complexity of composable simulation. In *Proceedings of the 1999 Winter Simulation Conference*, ed. P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans. 553-560. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
23. Petty, M. D. and E. W. Weisel. 2003. A composability lexicon, In *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, Orlando, Florida.
24. Pritsker, A. A. B. and J. J. O'Reilly. 1999. *Simulation with Visual SLAM and AweSim*. 2nd ed. New York: John Wiley & Sons.
25. Robinson, S. 2006a. Conceptual modeling for simulation: Issues and research requirements. In *Proceedings of the 2006 Winter Simulation Conference*, ed. L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto. 792-800. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
26. Robinson, S. 2006b. Issues in conceptual modeling for simulation: Setting a research agenda. In *Proceedings of the 2006 OR Society Simulation Workshop*.
27. Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. 1991. *Object-oriented modeling and design*. New Jersey: Prentice Hall.
28. Saanen, Y. A. 2004. *An approach for designing robotized marine container terminals*, Doctoral Dissertation, Delft University of Technology, Netherlands.
29. Sarjoughian, H. and D. Huang. 2005. A multi-formalism modeling composability framework: Agent and discrete-event models. In *Proceedings of the 9th IEEE*

International Symposium on Distributed Simulation and Real Time Applications. 249-256.

30. Setavoraphan, K. 2005. Conceptual simulation modeling for regional distribution center systems, Thesis, Indiana State University, Indiana, USA.
31. Setavoraphan, K. and F. H. Grant. 2008. *Simulation model for lockage operation*, contributed to the INFORMS Southwest Regional Conference, Texas A&M University, Texas, USA.
32. Spiegel, M., P. F. Reynolds, and D. C. Brogan. 2005. A case study of model context for simulation composability and reusability, In *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
33. Teeuw, W. B. and H. van den Berg. 1997. *On the quality of conceptual models*. Available on <http://osm7.cs.byu.edu/ER97/workshop4 /tvdb.html> [accessed February 14, 2008].
34. Valentin, E. C. and A. Verbraeck. 2002. Guidelines for designing simulation building blocks. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yucesan, C. –H. Chen, J. L. Snowdon, and J. M. Charnes. 563-571. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
35. Valentin, E. C. and A. Verbraeck. 2005. Requirements for domain specific discrete event simulation environments. In *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A.

Joines. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

36. Verbraeck, A. and A. Dahanayake. 2002. *Building blocks for Effective Telematics Appliation Development and Evaluation* (BETADE), Delft University of Technology, Netherlands.
37. Verbraeck, A. and E. Valentin. 2002. Simulation building blocks for airport terminal modeling. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yucesan, C. –H. Chen, J. L. Snowdon, and J. M. Charnes. 563-571. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
38. Vreede, G. J., A. Verbraeck, and D. T. T. V. Eikck. 2003. Integrating the conceptualization and simulation of business processes: A modeling method and arena template. *SIMULATION* 79: 43-55.
39. Wang, W. and R. J. Brooks. 2006. Improving the understanding of conceptual modeling. In *Proceedings of the 2006 OR Society Simulation Workshop*.
40. Winnell, A. and J. Ladbrook. 2003. Towards composable simulation: Supporting the design of engine assembly lines. In Proceedings of the 17th European Simulation Multiconference, 431-436.
41. Yilmaz, L. and T. I. Oren. 2004. A conceptual model for reusable simulations within a model –simulator-context framework. In *Proceedings of the Conference on Conceptual Modeling and Simulation*, Part of the 13M – International Mediterranean Modeling Multiconference. 235-241.
42. Zhou, M., Y. J. Son, and Z. Chen. 2004. Knowledge representation for conceptual simulation modeling. In *Proceedings of the 2004 Winter Simulation Conference*,

ed. R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters. 450-458. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

43. Zhou, M., K. Setavoraphan, and Z. Chen. 2005. Conceptual simulation modeling of warehousing operations. In *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines. 1621-1626. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

44. Zhou. M., Q. Zhang, and Z. Chen. 2006. What can be done to automate conceptual simulation modeling?. In *Proceedings of the 2006 Winter Simulation Conference*, ed. L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto. 809-814. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

3-8. Additional Works

The above material in Chapter 3 is entirely derived from the paper submitted to the Winter Simulation Conference 2008. To expand on this and provide results of additional research, we provide the following supplementary explanations for: 1) the initialization layer (IL); 2) the termination layer (TL); and 3) the process layer (PL), respectively. Moreover, a new example is given to support these extras. The purpose is to provide the simulation developers a better understanding in applying a conceptual simulation modeling tool such as ISAP in their simulation projects more effectively and efficiently.

3-8-1. New Demonstration

To illustrate an expanded level of detail concerning the concepts of ISAP, consider a basic lockage operation that is generally found in most of the U.S. inland waterways transportation systems. This example is more complicated so that the readers can recognize how important each layer in ISAP is and how to utilize those available symbols, notations, and diagrams to transform their conceptualization into knowledge representations.

There appears to require a link between smaller regional ports and oversea ports, delivering containers for deep-sea vessels. This is because there are some areas that the deep-sea vessels are unable to access due to their capacity and size. Inland waterway transportation is used as alternative to transport containers into the hinterland on rivers, using inland barges. However, there are some locations where significant changes in

water levels occur, which requires those vessels to use one or more locks to carry them up or down from one pool to the next.

According to the papers by the U.S. Army Corps of Engineers and Bandy (1987, 1988, 1991, 1996), the basic procedures for a lockage to service vessels traveling either upstream or downstream are the same (see Figure 3-9 for an overview from www.navlocks-hpo.usace.army.mil).

Figure 3 - 9: An example of navigation lock system

First, the water level inside the lock has to be the same as where vessels are located before allowing the vessels to enter the lock. The underwater valves at both the upstream and downstream gates are used to control the water elevation inside the lock by using the advantages of the water-level differences and gravity to fill water in or drain water out of the lock. It takes about 10-15 minutes to fill or empty a lock chamber, which depends on the size of valve opening and the total change of elevation for the water inside the lock. Second, the entrance gate is opened, while the gate on the other end of the lock is closed to maintain the proper water level. The vessels sail into the lock. Third, when the entrance gate is closed, the underwater valves at the other end of the lock are

opened to allow the water elevation inside the lock to eventually equalize with that on the other side of the exit gate. Fourth, the vessels sail out when the exit gate is opened to continue their trips.

Another issue related to the lock system is to fully utilize the lock's capacity. Because each lock has specific usable dimensions (width times length) that produce limited space inside its chamber, the number of vessels that will fit in the lock plays a role in operating a lockage. The size and shape of the vessels are taken into consideration in terms of accessing a lockage and dealing with safety issues. Actually, traffic on the waterway navigation systems consists of two main types of vessels: boats and barge tows. Boats can be determined as e.g., passenger, fishing, or government owned vessels. A barge tow consists of a tow boat and a set of barges (from one to sixteen or more barges) that carry a variety of products, such as coal, sand, grain, and chemicals, and containers. An average-size lock can handle twenty smaller boats, while only eight barges followed by a tow boat within a 3x3 configuration can fit the lock. A barge tow having nine or more barges or overall dimensions exceeding the usable dimensions of the lock cannot fit into the lock for a single lockage. Therefore, two main types of lockage operations associated with barge tows: a single lockage and a double lockage; are discussed, as the main focus of this research.

For a single lockage operation, there are two scenarios to be considered at the lock. First, the barge tows have overall dimensions less than the usable dimensions of the lock, which means they can fit into the lock as a single batch. Second, the barge tows having an overall length exceeding the usable length of the lock require reconfiguration. External force provided by electric wrenches that are located on shore both upstream and

downstream of the lock is typically needed. A double lockage operation is required to support the barge tows that consist of nine or more barges followed by the tow boat. They are too big to fit into the lock for a single lockage. To operate a double lockage, these barge tows move through the lock in the following manner (Bandy, 1996): a) The barge tow is moved into the lock and the front barges are tied to the side of the lock. b) The barge tow is separated and reconfigured in such a way that each of the two parts will fit into the lock lengthwise. c) The back part of the barge tow is moved out of the lock by the tow boat. d) The lockage for the front part is completed. e) An electric wrench is used to pull the front part of the barge tow out of the lock. f) The water level inside the lock is returned to the initial level. g) The tow boat moves the back part of the barge tow into the lock. h) The second lockage is completed for the back part of the barge tow. i) The back part of the barge tow moves into position behind the front part. j) The barge tow is tied back together. k) The barge tow continues the journey. It can be noticed that the vessels waiting on the other side are not allowed to enter the lock when the first lockage is completed – due to the safety concerns.

A lock on an inland waterway is determined a time-consuming operational system that delay the transshipment of containers or products, which may affect not only a point on the river but also the entire navigation network systems. A simulation model is then made to model and simulate the current system to obtain statistical results for performance analysis such as time in system, resource utilization, and waiting time. However, in a long-term development process, this simulation model is possibly obsolete sooner or later – due to conditional changes in the system (e.g., parameters and processes). Furthermore, building a simulation model by using either programming

languages or generic simulation environments is considered as a reluctant job. It is not only a time and cost consuming activity but also a recurrent process. It, thus, is very influential for the development of a domain specific simulation environment (DSSE) that provides reusable/editable model constructs, including supportive libraries. All these availabilities are particularly designed to match the requirements of the specific domain so that a variety of simulation projects can be produced with a minimum of time, cost, and recurrent development processes.

As previously described, ISAP supports the simulation developers to structure a simulation modeling framework by creating three different layers, following processes under each construction phase. This demonstration, however, is not exhibited in detailed steps-by-steps. It is aimed to illustrate how to use ISAP to create a blueprint providing layouts and guidelines for designing control agent and logic/process agent – representing basic mechanisms found in most generic simulation environments.

First, the control agent is required to define experimental conditions in terms of initial parameters and output options for the simulation, corresponding to its modeling objectives and constraints. This creates a frame of reference that manages and controls model compatibility and interoperability of model constructs to be on the same levels/meanings through the entire simulation environment. The control agent can be portrayed in IL and TL. Second, the logic/process agent is required to support a flow of entities through a network of interconnected processes that depicts the operation of the system under study. This creates a simulation framework that sets relationships, transitions (routing and branching points), and sequences of the network to be within a significant boundary. The logic/process agent can be portrayed in PL.

3-8-2. Initialization Layer (IL)

Each simulation run requires an initial setup. It is necessary to define experimental conditions to layout a frame of reference that any model constructs can access and retrieve their execution references (e.g., run length and number of runs) or initial references (e.g., queue populations and queuing priorities). Initialization is taken into the first-time simulation run as a startup to detect any errors and to verify/validate simulation. This leads to model improvement and correctness. Moreover, initialization is considered a part of design of experiments for simulation, which can adversely influence the estimators of steady-state performance (Pritsker and O'Reilly 1999). Therefore, initialization is viewed as a process to design and define a set of parameterized references whose settings can be modified per experimentation.

In general, there are standard references regularly required for a basic startup or setup; for example, number of runs to make, run length, beginning/ending time of a run, and number of attributes/variables. However, the more complicated a simulation model is, the more parameterized references for initialization are needed. The extension list of initialization may include queuing priorities, initial variable values, and random number seeds – requiring more careful attention in defining their statements. This is because a run of simulation is controlled as a whole piece rather than as individuals. It, thus, is crucial to avoid any conflicts that may affect model compatibility and interoperability levels from using parameters (including attributes and variables) through interconnected model constructs.

To list and define parameterized references for initialization, not only are their physical names and values needed but also are implicit and explicit relations of those

references' statement required. What the implicit relation means is the relevance of elements defined within a statement of reference. For instance, "queuing priority" reference requires file number of the target queue, priority rule to rank entities (e.g., FIFO or LIFO), and expression to evaluate to determine ranking of entities. Its statement must be able to clearly describe relations that the priority rule is selected to determine the value in the specified expression to rank entities put in the target file number. On the other hand, the explicit relation indicates the meaning (interoperability) of the physical names used among references. For example, if "File#" in the statement of "queuing priority" reference means the file number of a queue, it must specifically be used for this meaning throughout other references. Having well-defined statements of initialization, therefore, is essential to avoid any errors in simulation runs and experiments.

ISAP provides a method/tool identically exploited in the initialization layer (IL) to conceptualize and list necessary parameterized references of initialization for simulation. In addition, a frame of reference, including statements and relations, can be visually represented and easily viewed as either individuals or all. A graphical representation created from a combination of Fishbone Diagram and UML Object Diagram has been developed under the scopes of the ISAP prototype. The ideas behind the combination of these two diagrams are:

- The Fishbone Diagram is a problem-analysis tool providing a systematic (conceptual thinking) way in generating information, classifying information types (e.g., cause-effect, topic-detail, or problem-solution), and prioritizing importance of information. It is helpful for listing references demanded for initialization.

- The UML Object Diagram is a graphical representation for the static (structural) view of the system using objects, attributes, and relations. It is useful for clarifying and visualizing a reference's statement within limited space.

An example of IL diagram designed for this problem is given in Figure 3-10.

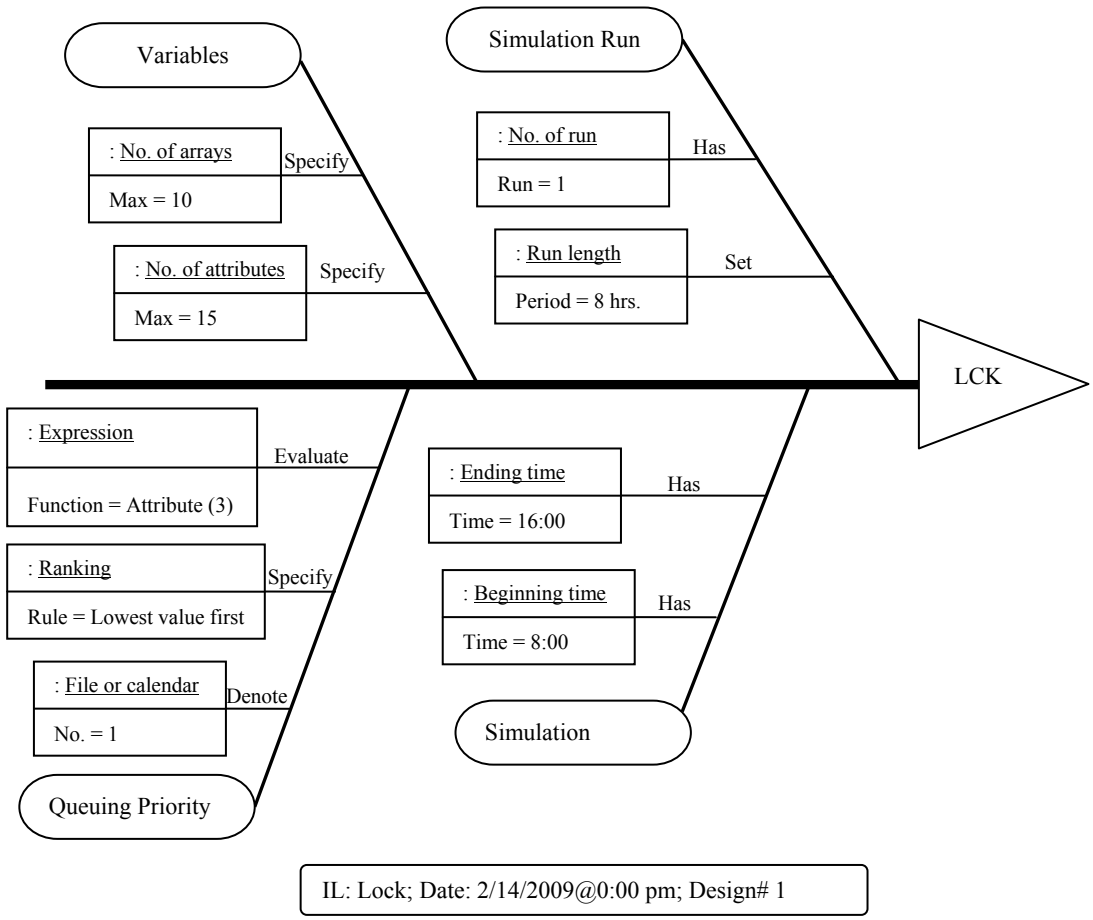

Figure 3 - 10: A diagram representing initialization of Lock

The IL diagram has a triangle at the right hand side, where the initialization to be executed is written. LCK, in this case, is a destination assigned to the process layer (IL) where these references are initiated. The main body of the diagram is a horizontal line from which branches the references, represented as bones. These bones are drawn towards the left-hand side of the main body and are each labeled with the reference

names. Off each of the large bones, there may be one or more smaller bones attaching with item boxes. Each item box contains an element name and its attribute value, and combines together with other item boxes (if available) to make a statement for the reference name. Above each smaller bone, a relation between the reference name and the item box is defined – using a word of "transitive verb". As a result, the entire section of a big-bone branch (including smaller bones) can be translated into the reference's statement.

Furthermore, a box under the IL diagram is available for recording information of the diagram construction, which needs IL name, date/time, and design number. The purpose is to keep track modification taken place at a time – to make a collection of designs. It leads to reduce repetitiveness of defining initialization for not only simulation models but also experimental designs. Afterward, the collection of designs will be transformed into a standard initialization format available in DSSE.

3-8-3. Termination Layer (TL)

A system has been viewed as a black box where selected inputs are processed to generate desired outputs. In general, to decrease the degrees of being a black box, the primary focus is on the development and improvement of the system's processes. However, defining parameters used in these processes are critical. Some parameters drive the processes to function, whereas some force the processes to produce outputs. Often, appropriate outputs are hardly achieved because of a lack of well-defined parameters. Consequently, a frame of reference for outputs must be specified into a standard pattern. It aims to eliminate irrelevant and unnecessary parameters that may cause any distortion

in either processes or outputs. This concept can be applied to the development of simulation by a means of determination of parameterization.

Parameterization of the projected outputs can be portrayed in ISAP's termination layer (TL) by using a tabular-cell pattern. It simply forms a table for description of data fields for parameterization, as shown in Table 3-4. The first data field (1) contains a projected output's label such as TIS (time in system), LUT (lock utilization), and WAT (waiting time). Then, the later data field numbers (e.g., 2, 3, 4, and so on) include parameters associated with each projected output. For example, to obtain a value of TIS, these following parameters: Arrival time of entity and Departure time of entity must be given.

Table 3 - 4: Description of data fields for parameterization

	Fields*					
1	2	3	4	5	6	7
TIS	Arrival time of entity	Departure time of entity				
LUT	Lock number	Lock busy time	Lock idle time	Total simulation run		
WAT	Queue number	Arrival time of entity at queue	Departure time of entity from queue			
TER	Maximum entities	Time limit				

TL is also aimed to design a pattern for termination of simulation. Basically, simulation is terminated when it reaches a specified number of maximum entities, time limit, and customized condition. It needs to be noted here that a value set for time limit in TL can be the same or different from one defined for run length in IL. This is because

they both are used upon different basis. Run length is initiated to identify a timeframe for a simulation run, whereas time limit is physically set to end a simulation run on a purpose. Termination (TER) is given at the bottom of the table.

Moreover, it is able to advance the usage of this table by assigning attributes, variables, or default values '{}' that practically match those in a simulation language or application to these parameters. Table 3-5 illustrates an example of data fields that provide parameters with their assigned attributes/variables/defaults expected to be used in Visual SLAM and AweSim network models (see Pritsker and O'Reilly 1999).

Table 3 - 5: Description of data fields for parameterization with assignments

	Fields*					
1	2	3	4	5	6	7
TIS	Arrival time of entity ATRIB[1]	Departure time of entity TNOW				
LUT	Lock number	Lock busy time {0}	Lock idle time {0}	Total simulation run {∞}		
WAT	Queue number {0}	Arrival time of entity at queue ATRIB[2]	Departure time of entity from queue TNOW			
TER	Maximum entities {∞}	Time limit {∞}				

Although the table above is unable to be applied directly into DSSE, it creates a good structural view laying out a core design where parameterization and termination are well determined.

3-8-4. Process Layer (PL)

When a draft design of the control agent: IL and TL, has been done, the next step is to input all the frames of reference into a simulation framework being created in the process layer (PL). To be more efficient and effective in the development of DSSE's logic/process agent, the construction of PL is divided into two modeling subsystems: static modeling subsystem and dynamic modeling subsystem. Both of them require the use of symbols, notations, and diagrams for robust and reusable representations of physical and behavioral characteristics – related to the processes of the target domain.

3-8-4-1. Static Modeling Subsystem

The first step is to specify the physical characteristics in the target problem domain. Along a navigation system, there may exist one or more locks to operate, space for waiting areas, and ports for operational transition – which need to be laid out within a framework of simulation. ISAP provides symbols and notations that represent different three static components: BUILD, SPACE, and CROSS.

A BUILD component is used to identify a point in a system where some physical objects are moved through or changed their states. In this demonstration, each lock is a point on the river that barge tows travel through and their statuses change (e.g., batched or unbatched) depending on conditions. As well, the states of resources at the lock (e.g., lockage and wrench) alter (e.g., idle or busy) at a point of time. BUILD components are used to represent locks and to identify their located points within the framework – that reflects the reality.

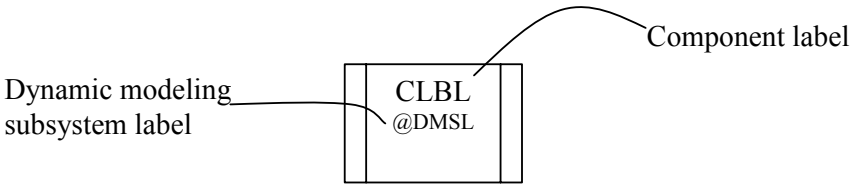

Figure 3 - 11: A symbol for BUILD

A SPACE component is used to identify an area in the system through which physical objects may pass or temporarily stay. It is seen that before entering a lock, barge tows are required to wait in a reserved area for a permission signal from a lockage controller. Thus, SPACE components are placed beside BUILD components to display waiting areas for barge tows.

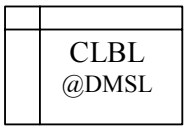

Figure 3 - 12: A symbol for SPACE

A CROSS component is used to identify locations in the system where the physical objects engaged with multi cross-domain systems. A cross-domain system is a system that possibly exploits similar parameterization, shares routing paths, or responds to consequences related with one or more different systems. A port, for example, is considered a cross-domain system because it engages with barge tows that travel from/to other locks or ports (determined as different systems) located on the same navigation system. Thus, to simulate this navigation system including a number of locks and ports, it is necessary to share information among systems (or cross-domain communication) to control interoperability of parameterization, layout design (e.g., routing), and dynamics

of the system (e.g., sequences of entity flow). The idea of using a CROSS component is to provide a better understanding in a mechanism of passing and returning parameters and entities within and between modularity (e.g., submodels) – a key development of simulation building blocks – which is discussed later in Chapter 5.

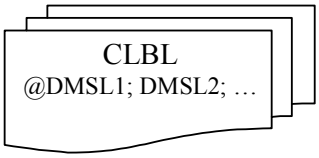

Figure 3 - 13: A symbol of CROSS

Each static component is defined with its identical component label that is connected to its dynamic modeling subsystem containing the logical process flows and parameters needed. The connection is made through "@" and followed by a specified dynamic modeling subsystem label (DMSL). There are two types of connections between static components: adjacency and precedence. Adjacency is used to indicate that movement may occur in either direction between two components. The symbol of adjacency is a line (Figure 3-14).

Figure 3 - 14: An adjacency line

An arrow, as shown in Figure 3-15, is used to indicate a precedence of movement that may occur in only one direction between two physical components. Movement may occur only in the direction of the arrow head. These connections show that there exist one or more interchanges or flows of objects and information between the components.

Figure 3 - 15: A precedence line

Figure 3-16 illustrates a physical layout on a navigation system, giving locations of ports, locks, waiting areas, and sections on the Mississippi River.

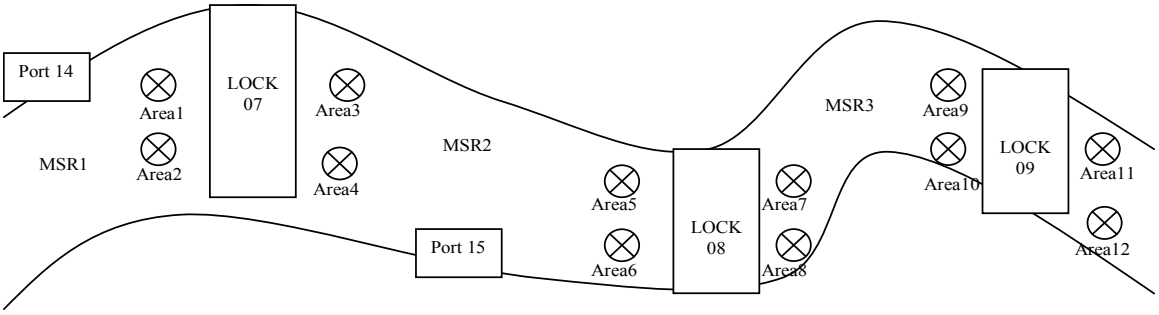

Figure 3 - 16: An assumed physical layout on the Mississippi River

These physical locations can be transformed into static symbolized components, which are connected together as a network, as shown in Figure 3-17.

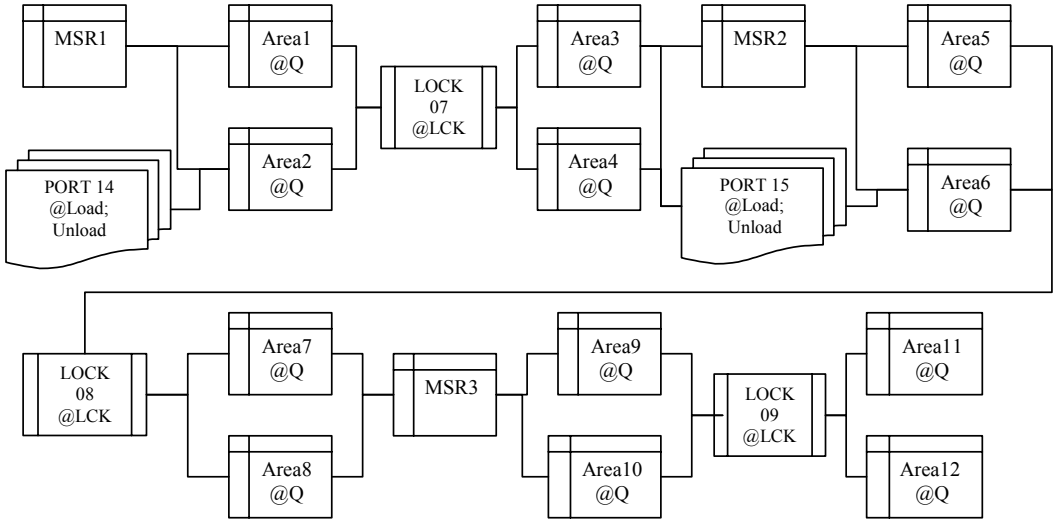

Figure 3 - 17: A network of static symbolized components

3-8-4-2 Dynamic Modeling Subsystem

The concepts of dynamic modeling subsystem have been already explained in depth in the paper. For this supplementary section, the entire set of the PL representations for the lockage operation is given in the following figures. It is necessary for the simulation developers to understand how to create each section, how to decompose an SMU, how to combine three sections, and what the entire page conceptually represents. Following the figures, tables of descriptions of the objects and operations are provided. There have been some changes in these tables to facilitate translating and mapping the conceptual simulation model into a domain specific simulation environment. For instance, operations are categorized into three different types: basic, extended, and specific, supporting determination of levels of configuration to fit levels of availabilities of parameters and functions in the target simulation programming or application.

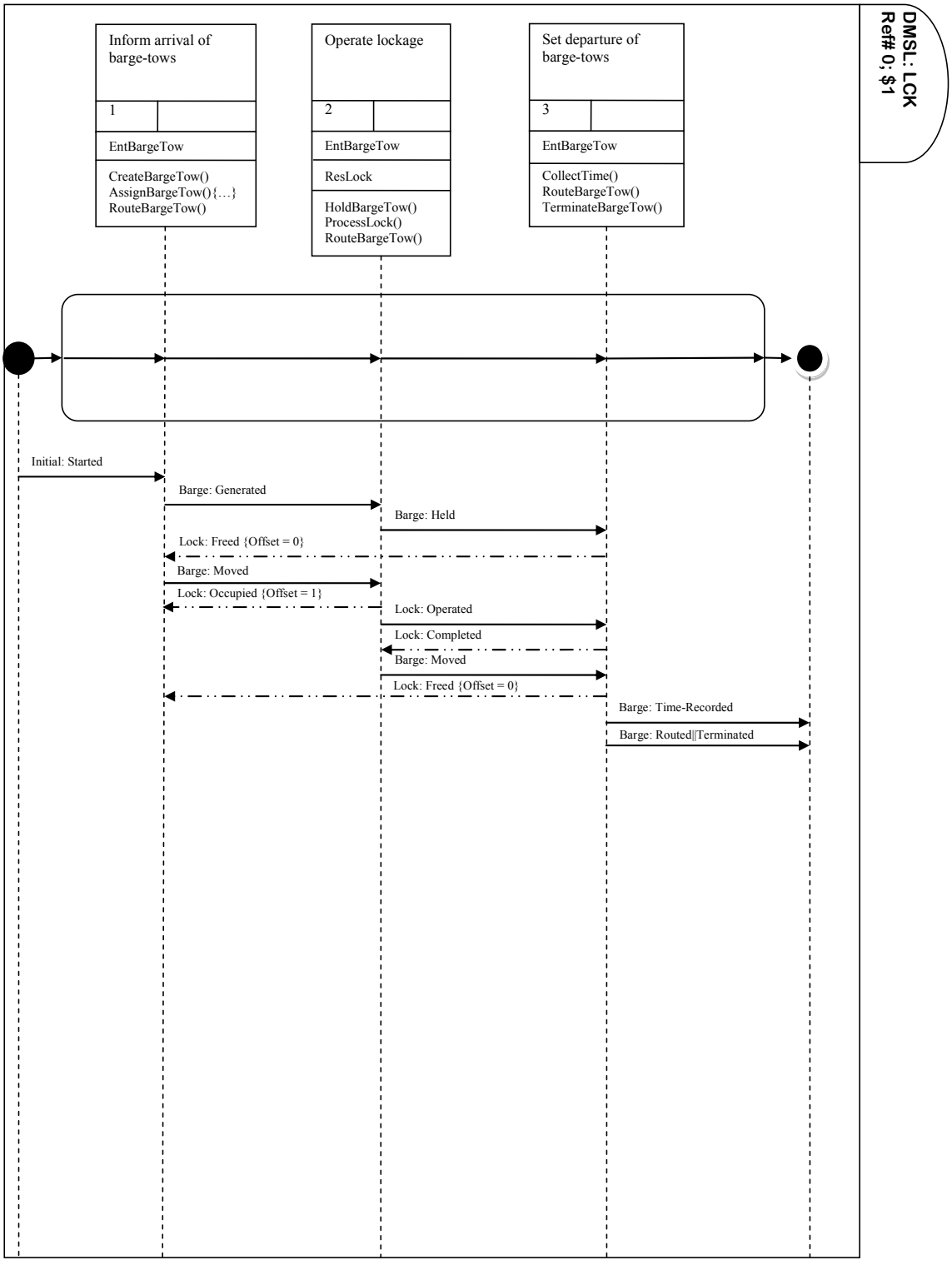

Figure 3 - 18: DMSL: LCK; sub-folder# 0; page# 1

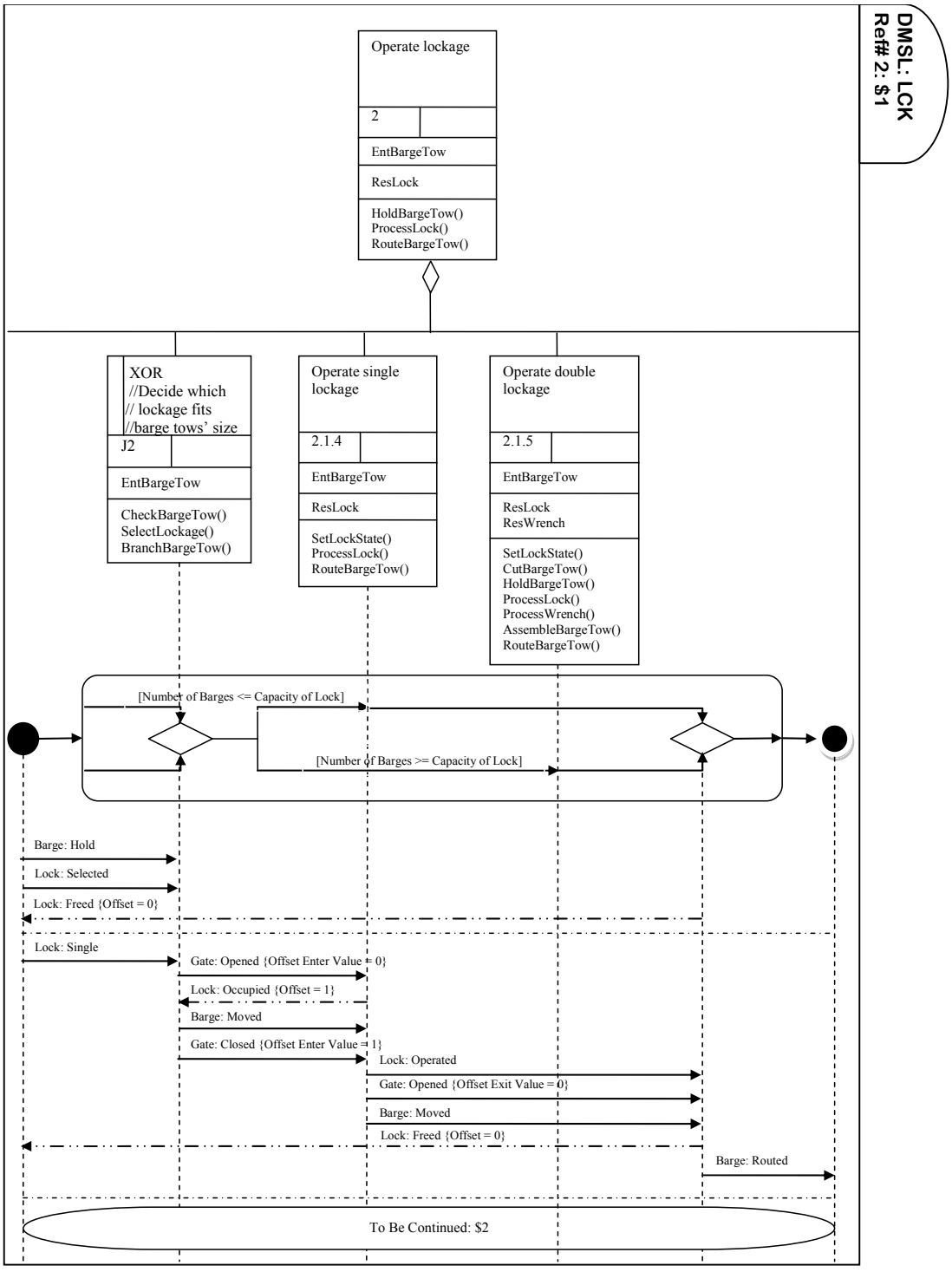

Figure 3 - 19: DMSL: LCK; sub-folder# 2; page# 1

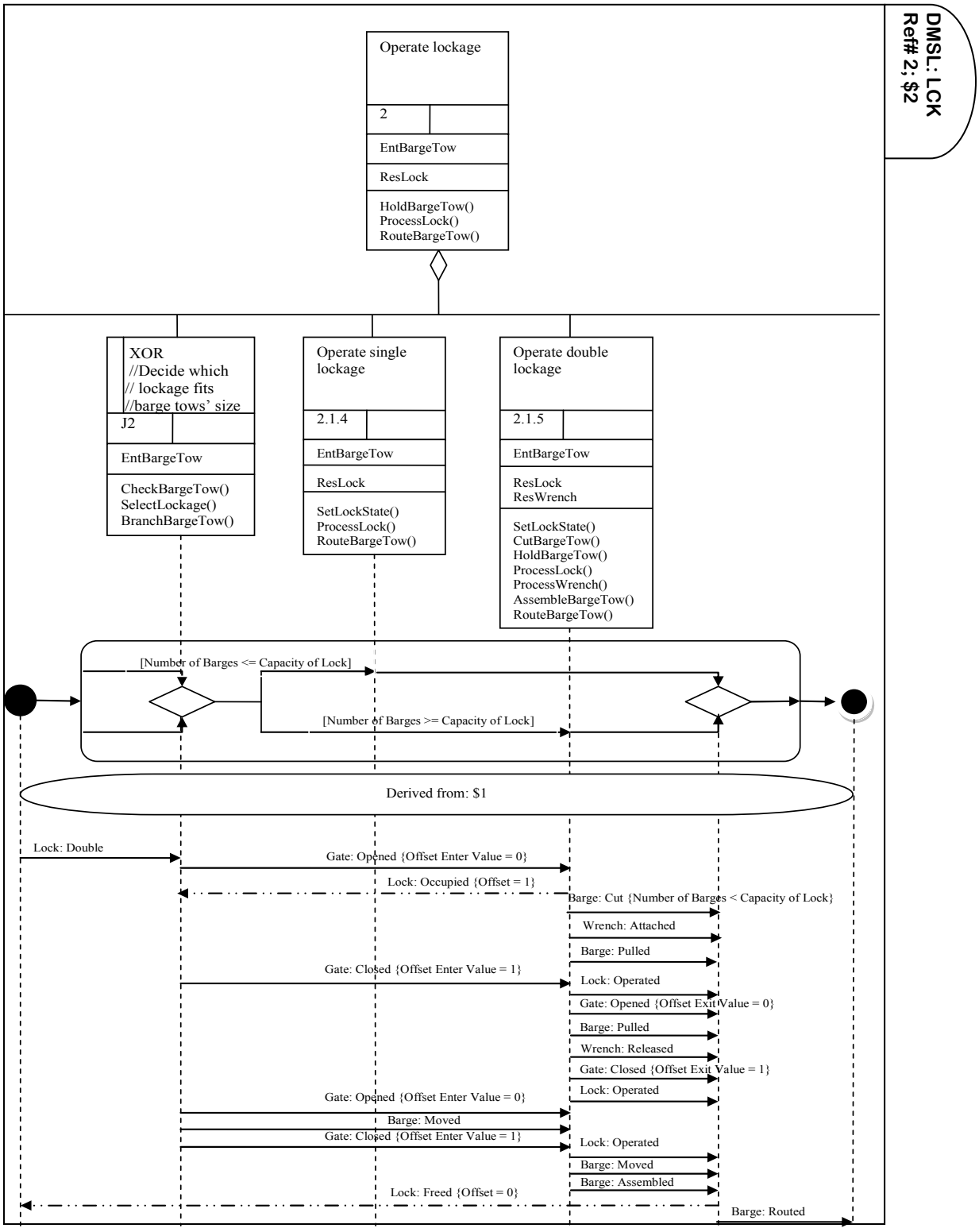

Figure 3 - 20: DMSL: LCK; sub-folder# 2; page# 2

Table 3 - 6: Description of Objects for DMSL: LCK

Object Name	Type	Description	Parameters
EntBargeTow	Entity	A barge-tow is represented as a target entity to be observed in the inland waterway system. A barge-tow entity consists of a set of barges and a tow boat.	: Identification# : Number of barges : Origin : Destination : Arrival time : Speed
ResLock	Resource	A lock is a resource that takes an action in raising or lowering barge-tow entities by filling or draining water. Also, the lock-enter allowance is controlled by its gates. The gates can be determined as internal or external resources.	: Name : File# : Resource# : Capacity : Activity time
ResWrench	Resource	An electric wrench is a resource used to pull a section of barges that are cut for the first lockage.	: File# : Resource# : Capacity : Activity time

Table 3 - 7: Description of Operations for DMSL: LCK

Operation Name	Type	Actor	Description	Attributes	Global Variables
AssembleBargeTow()	General/ Extended	EntBargeTow	An action is to accumulate one or more set of barges that are cut with a tow boat into a single entity	: Identical values : Number of barges	
AssignBargeTow() {…}	General	EntBargeTow	A simulation action is to assign identical attributes to define the characteristics of each barge-tow entity that represent a set of barges and a tow boat.	: Identification# : Number of barges : Origin : Destination :Arrival time : Speed	
BranchBargeTow()	General/ Extended	EntBargeTow	A number of branches are provided at a location for an entity to take upon conditions or probabilities		: Condition expression
CreateBargeTow()	General/ Extended	EntBargeTow	A barge-tow entity is created by a mean of containing a set of barges and a tow boat.	: First arrival : Arrival rate :Current time : Max# entities	
CollectTime()	General	EntBargeTow	Statistical data of time spent in the system are collected	: Travel time	: ID : Label

Table 3 - 8: Description of Operations for DMSL: LCK (Cont.)

Operation Name	Type	Actor	Description	Attributes	Global Variables
CheckBargeTow()	Extended	EntBargeTow	An action is to check how many barges the entity is containing to make a decision for selecting a lockage type.	: Number of barges	
CutBargeTow()	Extended/ Specific	EntBargeTow	An action is to split a specific number of barges that are allowed to enter a lock. There are many ways to cut, upon policies and sizes of each lock	: Identical batch size : Number of barges	
HoldBargeTow()	General/ Extended	ResLock	"Hold" can be determined as an action to control the flow of entities.		: Delay time
ProcessLock()	General	ResLock	An action is taken at a lock by a mean of delay-activity time.	: Resource# : Capacity of lock	: Activity time
ProcessWrench()	General	ResWrench	Electric wrench is used when a double lockage is required.	: Resource#	: Activity time
SelectLockage()	Extended	ResLock	A decision-making action is to select either single or double lockage configuration upon the sizes of the barge-tow entities	: Resource# : Capacity of lock	
SetLockState()	Extended	ResLock	An action (of sending a signal) verifies a status of the lock (e.g., busy or idle)	: Resource#	: Offset value
RouteBargeTow()	General	EntBargeTow	Each barge-tow entity is routed or moved through the system on designated routes. Delay time might be specified on each route.		: Distance
TerminateBargeTow()	General	EntBargeTow	Each barge-tow entity is terminated when it leaves the system		

Moreover, a full network statement of DMSL: LCK can be created by arranging operations in those SMUs into sequential activities. To make a network statement, follow these steps:

(1) Begin with Ref# 0 (the topmost level of decomposition) to check how many SMUs (modules) are in this folder (level of decomposition).

(2) Start with the left-most SMU.

(3) Check if the SMU leads to a new Ref# by using its abstraction-level number to search for a folder having the Ref# as same as the number.

- If no, go to Step (4).
- If yes, start with the new Ref# and go to Step (6).

(4) Transform the SMU's operations into a statement format and put them in a sequential order (stated in the sequence section).

(5) Check if there is another SMU next to it.

- If no, go to Step (8).
- If yes, go to Step (7).

(6) Repeat Step (2) until there is no further decomposition.

(7) Start with the next SMU and go to Step (3).

(8) End the network statement.

According to the instructions above, the simulation developers are able to obtain such the network statement as shown in DMSL: LCK.

DMSL: LCK; Ref# 0 – 2:

```
1  CreateBargeTow, First arrival, Arrival rate, Current time, Max# of entities;
2  AssignBargeTow, Identification#, Number of barges, Origin, Destination, Arrival time, Speed;
3  RouteBargeTow, Distance;
4  CheckBargeTow, Number of barges;
5  SelectLockage, Resource#, Capacity of lock, Number of barges;
6  SetLockState, Rerouce#, Offset value;
7  BranchBargeTow, Condition expression;
8  Condition, Number of barges <= Capacity of lock;
9  SetLockState, Resource#, Offset value;
10 RouteBargeTow, Distance;
11 ProcessLock, Resource#, Capacity of lock, Activity time;
12 RouteBargeTow, Distance;
13 SetLockState, Resource#, Offset value;
14 Condition, Number of barges >= Capacity of lock;
15 SetLockState, Resource#, Offset value;
16 CutBargeTow, Identical batch size, Number of barges;
17 HoldBargeTow, Delay time;
18 ProcessWrench, Resource#, Activity time;
19 ProcessLock, Resource#, Capacity of lock, Activity time;
20 RouteBargeTow, Distance;
21 SetLockState, Resource#, Offset value;
22 RouteBargeTow, Distance;
23 SetLockState, Resource#, Offset value;
24 ProcessLock, Resource#, Capacity of lock, Activity time;
25 RouteBargeTow, Distance;
26 AssembleBargeTow, Identical batchsize, Number of barges;
27 SetLockState, Resource#, Offset value;
28 RouteBargeTow, Distance;
29 CollectTime, Travel time, ID, Label;
30 RouteBargeTow, Distance;
31 TerminateBargeTow;
```

The design of network statement is intended to increase the readability of the process layer's representations without encumbering the simulation developers with extraneous information requirements (based on the idea of Pritsker and O'Reilly 1999).

3-9. Additional References

1. Bandy, D.B. (1987). Simulation of Fox River locks boat traffic. Presented at St. Louis ORSA/TIMS Joint National Meeting.
2. Bandy, D.B. (1988). Fox River locks SLAM simulation model. *Proceedings of the 1988 Winter Simulation Conference*, eds., M. Abrams, P. Haigh, and J. Comfort, 815-820. San Diego, California.
3. Bandy, D.B. (1991). Simulation of Fox River locks boat lift. *Proceedings of the 1991 Winter Simulation Conference*, eds., B.L. Nelson, W.D. Kelton, and G.M. Clark.
4. Bandy, D.B. (1996). Simulation of the Illinois waterway locks system. *Proceedings of the 1996 Winter Simulation Conference*, eds., J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain.
5. Pritsker, A. A. B. and J. J. O'Reilly. 1999. *Simulation with Visual SLAM and AweSim*. 2nd ed. New York: John Wiley & Sons.
6. U.S. Army Corps of Engineers. (n.d.). *Navigation*. Retrieved October 15, 2006, from http://www.swl.usace.army.mil/navigation/lock.html.
7. U.S. Army Corps of Engineers. (n.d.). *Navigation*. Retrieved November 27, 2007, from http://www.navlocks-hpo.usace.army.mil.

CHAPTER 4

Transformation of Conceptual Simulation Modeling

Abstract

Conceptual simulation modeling (CSM) is a critical approach that breaks through the barriers of cross-domain communication in Modeling and Simulation (M&S). However, standard symbols, notation, and diagrams created in CSM need to be transformed into contextualized documentation, so that their semantics of structural and behavioral contents within a simulation context can be represented in a more executable and readable form. The search for a formal transformation approach is crucial to establish a bridge between human concepts and simulation content. This chapter includes a pilot study on the transformation of a conceptual simulation model developed by Integrated Simulation Acknowledge Procedure (ISAP), based on model composability and simulation interoperability.

4-1. Introduction

In Modeling and Simulation (M&S), conceptual simulation modeling (CSM) plays a significant role in breaking through communication barriers between domain experts and simulation developers. The objective of CSM aims to deliver a robust and reusable simulation modeling framework for a target domain, describing its structural and

behavioral characteristics that represent both reality and simulation contents. Often this framework is used as a blueprint for domain specific simulation development. However, CSM is still based upon a human-to-human (readable/understandable) platform, which creates challenges in transferring concepts and requirements to computer simulation-based implementation.

Even though a CSM framework provides a set of processes, objects, functions, and relations needed to compose a simulation model, these can only be used at the generic or basic levels by a means of logical flows and component/pattern designs – to structure overall architecture but an independent-implementation environment for domain specific simulation. CSM tools are still incapable of transforming their conceptual models into executable simulation models automatically. This causes the information exchange to be inaccurate and incomplete. However, the automatic transformation does not guarantee that the information will not be lost when the concepts are transformed into executable codes. This is because at transformation stage domain experts are no longer involved in the process. In fact, it is important for the domain experts to check the entire development process of both conceptual models and simulation models to confirm if they still include all the aspects of their requirements.

The problem to be addressed is that most domain experts are unable to read/understand not only the computer/simulation programming languages but also the formal modeling languages available these days (Valenti, Panti, and Cucchiarelli 1998). This requires that an individual take special training before being able to read/understand such a modeling language. The main reason is that there are only a few modeling languages that incorporate an explanation generation approach such as PPP (explanation

component), as described in a study by Nijhuis (2005), to enrich conceptual models with user interaction like, i.e., UML. User interaction consists of an explanation component with natural language and a knowledgebase component, allowing the user (e.g., domain expert and simulation developer) to ask a question and to answer follow-up questions; and to store and get information from the knowledgebase. Another reason is that conceptual models are directly translated into complex programming languages such as C++ and JAVA when encoding a host simulation language (e.g., SIMAN/Arena and Visual SLAM/AweSim). The difficulty exists not only for the domain experts but also for the inexperienced simulation developers.

Based on these concerns, Setavoraphan and Grant (2008) proposed an Integrated Simulation Acknowledge Procedure (ISAP) to enhance recent modeling methods and to response the user's requirements in domain specific simulation development. In the paper, fundamental symbols, notation, and diagrams are represented in natural language, which can be transformed into simulation-language-like statements. Unfortunately, the paper is limited to cover all the aspects necessary to support the alignment of conceptualization and implementation. Therefore, a simulation-contextual document has been developed in this study to represent the transformation of ISAP using an intermediate simulation language that supports both model composability and simulation interoperability. This type of documentation aims at creating open space for the development of domain specific simulation on any simulation environments.

4-2. Key Concepts

An initiative for facilitating better understanding of the transformation approach is taken by distinguishing models and simulations. The Modeling and Simulation (M&S) community defines this distinction by a means of metadata structures in support of the development processes (Tolk and Turnitsa 2007). Models are the results of conceptualization of a problem to be solved, while simulations are implementations of models executable over time. Information generated and executed by these domains are different, which initially generates a gap between them by a means of conceptualization and implementation. Therefore, a need for information exchange between them becomes crucial for reducing or eliminating this gap.

In fact, conceptual models in most cases are not technically captured in a computable way. Hence, difficulties exist in evaluating if the information exchange can be aligned conceptually or not (Davis and Anderson 2003). Only the alignment of concepts, nevertheless, is not sufficient to fulfill the gap. The content of the information exchange reference must also be specified as contextualized information, which is built from a set of user declarations under a set of validity constraints (Analyti et al. 2007). Given such a contextualized framework, specifications of conceptualizations can be formalized into a context that simulation systems can really exchange and understand. The context can be viewed as a knowledge representation providing a common language for this cross-domain communication. Structuring a meaningful context, thus, leads to a key approach in the transformation process.

According to Hemel, Kats, and Visser (2008), "*the essence of the (transformation) approach is to shift the knowledge about these implementation details*

from the minds of programmers to the templates of code generators that automatically translate models into implementations." This statement points out that the context for transformation should initially be pursued at the conceptual level. The purpose is to acknowledge and synergize the requirements and specifications of two domains so that components that are created in the contextualized framework contain information easily transformed across the domains. To achieve this goal, the implementation of two key concepts: model composability and simulation interoperability, must be taken into account.

4-2-1. Model Composability

An interesting definition describing model composability can be found in the literature by Morse et al. (2003):

"*Given a set of components, structured descriptions or specifications of the components, sometimes called meta-data or meta-models, can be used to guide the process of selecting components for a specific purpose and determining if a set of components can be effectively composed.*"

In brief, composability is the process of combining and recombining components in different compositions (Petty 2004). A summary of composability can be found in the previous study (Setavoraphan and Grant 2008). However, details related to transformation are not included.

In general, syntactic and semantic composability are considered core concepts for model composability. Syntactic composability is concerned with the compatibility of implementation details (Petty and Weisel 2003); semantic composability, on the other

hand, is concerned with validity of the composition (Weisel, Petty, and Mielke 2003). However, as mentioned earlier, transformation is a process of structuring a context understandable by simulation systems. Hence, another type of composability needs to be addressed. Pragmatic composability is concerned with the context of the simulation and with the determination of whether the composed simulation meets the intended purpose of the modeler (Hofmann 2002).

In practice, semantic and pragmatic composability is a much harder problem (Fishwick and Miller 2004; Moradi 2007) in conducting conceptual models, compared to syntactic composability. One of the reasons is that the formats of conceptual models, in general, are not designed to contain semantic and pragmatic information about what they intend to simulate. This results in having a framework that provides interface specifications and rules which only facilitate technical aspects of compositions (e.g., syntactic composability), for example, the High Level Architecture (HLA) (Moradi 2008). Within a poorly structured semantic and pragmatic format, there appear the difficulties not only in reusing components but also in transforming these components from conceptualization to implementation.

To handle this issue, the first step is to have a clear understanding of what is being composed (e.g., components) and what is the result of composition (e.g., product from the components). Based upon a number of answers found in the literature, they can be referred to as levels of composability (Petty and Weisel 2003) as follows:

- *Application:* Applications such as simulations are composed together to build simulation events, exercises or experiments.

- *Federate:* Refer to specific HLA meanings, this level allows combining, recombining, and editing simulations to build a set of distributed simulations that communicate in run-time.
- *Package:* Simulations are composed by using pre-assembled packages of models to form a subset of the specific domain.
- *Parameter:* This level is focused on configuration of pre-existing simulation by using parameters.
- *Module:* Modules are composed into software executables.
- *Model:* Models of smaller scales are composed into composite models of larger scales.
- *Data:* Databases are developed through composition of sets of different data (e.g., entities, sources, and aspects).
- *Entity:* Entities are composed into groupings, whereas groupings are composed into higher level groupings.
- *Behavior:* Behaviors at lower level are composed into higher level behaviors, which are to be executed by computer generated forces or in constructive simulations.

The levels of composability provide information that help specify meanings, characteristics, and requirements of sources (e.g., conceptual components) and targets (e.g., executable components). Thus, the mappings and relations between component types can be taken into management and control, which generates rules and constraints for the specializations of transformations (Koch 2006).

The second step is to develop proper documentation used as guideline and control for (re)configuration and adaptation in composability. Davis and Anderson (2003) point out that in practice there is always a need for some degrees of component adjustment and adaptation before being able to compose a set of components. It can be done through reconfiguration of simulations by, for example, adjusting interfaces, making changes in the existing simulation codes, etc. In either case, proper documentation is needed for the adaptations (Fishwick and Miller 2004). "*It is much easier for a human to read and understand a textual description of a component than a program code, and use it as a basis for selecting and adapting the component*" (Moradi 2008). Hence, proper documentation aims to provide a good support to pursue successful composability at the conceptual and description level – in a formal way. Furthermore, an additional format for architectures and environments that facilitate transformation of the components can be created and implemented within documentation. This type of documentation brings potential path to progress in the entire development process for domain specific simulation. Documentation in this research study is developed upon technology-based concepts, which are further explained in Section 4-3.

4-2-2. Simulation Interoperability

Within the modeling and simulation (M&S) community, interoperability has been of major concern to bridge the gap between implementation focused methods and conceptual models (Tolk and Muguira 2003). Interoperability in M&S is closely related to composability; thus, it is of interest to clarify its definitions. The IEEE defines interoperability as "*the ability of two or more systems or components to exchange*

information and to use the information that has been exchanged" (IEEE 1990). Meanwhile, its definition in the M&S context has been described as the ability of different simulations connected in a distributed system to collaboratively simulate a common scenario (Petty 2002). Furthermore, the distinction between interoperability and composability is defined in the literature by Page et al. (2004) as the following aspects:

- Interoperability deals with exchange of data elements based on a common data interpretation related to implementation details, which can be mapped to the levels of syntactic and semantic interoperability.
- Composability addresses the alignment of issues on the conceptual modeling level to provide meaningful conceptualization used for being implemented by the simulation systems.

The underlying idea of distinguishing between interoperability and composability is to recognize what type components are (e.g., interoperable or composable) and what the result of configuration of the components is (e.g., levels of interoperability and composability). According to Petty (2002), components that are interoperable in one configuration, but cannot be composed together in other ways are not composable. Furthermore, having components with unbalance levels between interoperability and composability makes the meanings and validations of transformation become infeasible.

Originally, transformation is acknowledged as an approach dealing with information exchange between two different systems, which requires an achievement of meaningful interoperability as well. Hence, the focus is put on the specifications of what type of interoperability being structured and what level of information being

interchanged. First of all, similar to composability, there are two types of interoperability: technical and substantive (Petty 2002). In technical interoperability, components require to be compatible with an interoperability protocol which is responsible of exchange of information between components. Technical interoperability is also called syntactic interoperability. On the other hand, substantive or semantic interoperability is satisfied if the exchanged information is semantically meaningful, which is based on the definition settings. Furthermore, substantive interoperability is often collaborative along with pragmatic interoperability when determine whether the exchanged information differs from the intention (Pokraev et al. 2005).

Second, in order to reach meaningful interoperability, the levels of information being exchanged between two systems need to be managed at the conceptual level to avoid ambiguous interpretation (Tolk and Muguira 2003). Five levels of interoperability are defined in the literature as follows:

- *Level 0 – System Specific Data:* No interoperability exists between two systems because data is specifically used within each system.
- *Level 1 – Documented Data:* Data is documented using a common protocol and interface such as HLA.
- *Level 2 – Aligned Static Data:* Data is documented using a common reference model such as ontology.
- *Level 3 – Aligned Dynamic Data:* The use of the data within the component is well defined in standard documentation and is visible to the integrator such as Unified Modeling Language (UML), Extensible Markup Language (XML) in Simulation Reference Markup Language (SRML).

- *Level 4 – Harmonized Data:* Conceptual model is documented to control the consistency of semantic connections between unrelated data concerning the executable code.

As obviously seen, to reach the higher level of interoperability that is related to the availability and management of information exchange is relied on how well documentation is made to capture and describe data.

The idea of documenting interoperability is similar to documentation for model composability in which technology-based concepts are involved. A question here is that how documentation based technological concepts can be created to support transformation. A thought of what is the most critical requirement of the M&S transformation process becomes a key to answering it. Intensively, transformation requires the high-level collaboration between the "semantic" composition of components and the "meaningful" interoperation of information. One of the technology-based concepts found in a set of literature that show the success of transformation is Semantic Web. Within the terminology of Semantic Web, new possibilities for documenting a contextualized framework using its related concepts such as ontology, XML, and SRML, to facilitate transforming conceptualization into implementation are conducted. These concepts have been discussed in the section 4-3.

4-3. Related Technology-based Concepts

According to the W3C director Berners-Lee (2001), the Semantic Web is an extension of the current World Wide Web technology in which the semantics of

information on the web is well defined. Through Semantic Web technologies, it is possible to structure everything in languages to make formal description of contents, understandable by people and computers. Hence, the Semantic Web is about a verbal communication tool for collecting data relating to real world objects and exchanging the data in a formal way.

The reason behind the use of the Semantic Web in this study is to develop a methodology that supports transforming conceptual models into simulations. Its potential has inspired this work to document a contextualized framework using available technologies and languages within the Semantic Web context for the transformation process. The following sub-sections present an overview of the related technologies.

4-3-1. Ontology

To achieve proper contextualized documentation for transformation, semantics of composability and interoperability need to be well captured at the conceptual level. Ontology is defined as a formalization of a specification of a conceptualization (Tolk and Turnitsa 2007) in a sense to overcome the challenges of M&S composablity and interoperability. Its formulation for practical applications is described: "*if a formal specification concisely and unambiguously defines concepts such that anyone interested in the specified domain can consistently understand the concept's meaning and its suitable use, then that specification is an ontology*" (Tolk and Blais 2005). Accordingly, ontology aims at providing common and unambiguous meaning of information to establish a joint terminology/frame of reference of conceptual meanings between

components (e.g., entities) within a domain as well as their interactions between the components (e.g., events).

The following example helps to visualize how ontology works. Ontology defines that there is a concept of an object called Vehicle, and that a Vehicle requires fuel. This object can then be declared with terms and properties in some formal way to convey that; for instance, a Car is a Vehicle. Furthermore, using logical inference that a Car is a Vehicle and Vehicles require fuel, it can be concluded that a Car requires fuel. According to Moradi (2008):

"*The point of declaring this type of ontology is so that if two entities (for example two computers that try to mimic intelligence) have agreed upon using this ontology, and one of the entities can mention the word Car, then the other entity knows that this Car they are talking about is a Vehicle and nothing else.*"

Given its definition and example, it can be seen that ontology shares many structural similarities with object orientation by a means of fundamental aspects such as encapsulation, polymorphism, and inheritance.

Table 4 - 1: Ontology to Object-oriented mapping

Ontology Domain (retrieved from Wikipedia 2008)	**Object-oriented Domain (Rumbaugh et al. 1991)**
Individual concept (instance)	Object
Collections/types of concepts	Classes
Properties	Attributes
Function terms	Operations and methods
Relations	Links and associations

Ontology to Object-oriented mapping retrieved from the table above shows that a concept can potentially be structured into a form of an object that has its own class and relations between other objects/classes. Furthermore, its information (e.g., attributes, characteristics, properties, etc.) can be captured, exchanged, specialized, inherited, and reused within a specific domain (e.g., a frame of reference). Put together a set of objects, classes, information, and relations, the concept becomes less abstract but more semantic to be used in modeling.

Although the object orientation is (often) used as the fundamental methodology to develop conceptual models, it still lacks mechanisms to control and formalize the levels of semantics of conceptualizations. Vasilecas and Bugaite (2007) state that the development of conceptual model using ontology based approach is needed because the semantic content can be transformed into information system (e.g., simulation) artifacts. Hence, the costs and time can be reduced not only in conceptual modeling but also in simulation modeling.

Base Object Model (BOM), for example, has recently been developed using ontology-based approach to provide a component framework for facilitating reusability, composability, and interoperability (SISO 2006). A BOM is developed based on the assumption that piece-parts of models and simulations can be extracted and reused as modeling building-blocks and components. Specifically, BOMs are meant to provide an end-state of a conceptual simulation model and to be used as a foundation for the design of executable software code and integration of interoperable simulations. Based on the aspects found in a BOM, concepts can be captured and described in terms of both static (e.g., reality) and dynamic (e.g., interaction) view to support considering what the model

or simulation will intend to do. Furthermore, a BOM is a document that defines not only the template components for capturing the information needed to describe a simulation component but also the XML schema for interchanging the information between conceptual models and simulations. The concept of BOM brings an insight of applying the ontology-based approach and XML to develop a conceptual simulation model including simulation transformability. The implementation of this idea has been taken in the entire research study so that ISAP is capable of constructing a process-oriented and component-based framework to contain and represent sufficient semantics of simulation components with respect to the ontology context. Therefore, there is the possibility for ISAP to be documented by using XML to facilitate its transformation process. Detailed discussion will be given in the section of model transformation.

Another interesting issue about ontology is the levels of abstraction. There are three main levels: upper (foundation) ontology, core ontology, and domain (domain-specific) ontology (Fishwick and Miller 2004). Upper ontology is a model that captures basic concepts of real world. It provides a framework in which the building-blocks of reality can be described, independent of any specific domain. Thus, concepts defined in the upper ontology are generally applicable across a wide range of domain ontologies. The upper ontology then can be used as a knowledge base for building more specialized ontologies, which provides reusable knowledge and semantics to support interoperability between different specialized ontologies. The core ontology captures concepts and their relations in a field of practice. The domain ontology models a specific domain and specializes the core ontologies.

Moreover, the ontological spectrum introduced by Daconta et al. (2003) is used to describe a range of semantic models of increasing expressiveness and complexity to capture the information required, as shown in the following categories:

- Dictionaries and glossaries are lists of controlled vocabularies not underlying concepts and are the weakest semantics in this spectrum.
- Thesauri are well known-order and structured controlled vocabularies that facilitate retrieval of documents and achieve consistency in recorded documents.
- Taxanomies are tree structures of classifications for a given set of objects, which are the first form reflecting the idea of concepts.
- Ontologies represent the formalization of an exhaustive and rigorous conceptual schema within a given domain, which are helpful for capturing the meaning of the underlying concepts.
- Logical models are representing semantically the strongest methods of the ontological spectrum, giving knowledge representations in particular.

The main purpose of using the ontological spectrum is to fill several gaps identified by Robinson (2006) in modeling conceptualization and implementing models into simulations. Hence, the use of a common language has been introduced for information exchange to control and represent the levels of semantics between abstract and executable thinking (Tolk and Turnitsa 2007). It exploits common artifacts in capturing and providing information that not only human but also computer can read and understand. Obviously, these common artifacts become particular definitions and supportive elements

for transformations between domains. Among available languages, XML has been selectively used as a common language in this study.

4-3-2. XML

Extensible Markup Language (XML) is a markup language derived from Standard Generalized Markup Language (SGML), originally designed for the exchange of data on the Web (Quin 2003). XML is a simple and flexible text format for describing a class of data objects called XML documents. XML documents consist of elements which contain either parsed or unparsed data, specified using tags (words bracketed by '<' and '>'). Parsed data are made of characters that form character data (e.g., attributes) for describing the elements and form markup for encoding the contents of document, described by a-name-value pairs. XML provides this markup mechanism to impose constraints on the document's storage layout and logical structure. The example below represents how the author can be described in an XML document.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Author>
        <Name>Kitti Setavoraphan</Name>
        <Institution>
                <Name>University of Oklahoma</Name>
                <Location>
                        <City>Norman</City>
                        <State>Oklahoma</State>
                        <Post>
                                <Code>OK</Code>
                                <Zip>73019</Zip>
                        </Post>
                        </State>
                        <Country>USA</Country>
                </Location>
        </Institution>
</Author>
```

Using this tree-structured text style, related data and networked data can naturally be described at various levels of complexity. The hierarchical structure in an XML document is formed in a fashion similar to the object-oriented structures laying out the foundation for the development of most simulation software packages. The most precise semantic mapping of conceptual structures onto both the XML structures and object-oriented structures is the Document Object Model (DOM). The DOM provides access to the various nodes of a document such as document, element, and attribute, and to node lists. Moreover, XML marks a shift toward data structures that can be defined by grammars (Daum 2003), so that it has enough expressiveness to satisfy the requirements of conceptualization-to-implementation communication in modeling and simulation. Thus, XML becomes a preferred protocol for accessing and establishing simulation modeling specifications (Lu, Qiao, and McLean 2003) regardless of simulation platform.

As a grammar-driven language, XML allows a modeler to define his own tags and structure of documents using an XML schema (Lim 2004). The schema can also be viewed as a common vocabulary for a particular application that involves exchanging documents (W3C 2002). The vocabulary aims at defining the structures of the XML document in terms of constraints in a particular schema. There are two types of constraints used in a schema: content definition and data type constraints (Walsh 1999). Content definition constraints describe the order and the sequence of elements, whereas data type constraints describe valid units of the data. An XML schema, thus, permits the modeler to specify rules to structure the content of an XML document (e.g., elements, attributes, relationships, and data types) that can be validated with simulation software (Reichenthal 2002).

Since its syntax is simple, an XML document can be read by both human and computers (Fishwick 2002). Not only does transformability of the XML document potentially emerge but also collaboration between the domain experts and the simulation developers does improve. As a result, XML is considered as a core language suited for documenting a contextualized framework to represent the contents of both domains at an intermediate information-exchange level – that is a more computer-readable form.

4-3-3. Simulation Reference Markup Language (SRML)

According to Reichenthal (2002), "*simulation is a process that attempts to predict aspects of the behavior of some system by creating an approximate model of it.*" The underlying concept here is that a model must contain the ability to describe behavior of all items comprising a simulation. As mentioned above, an XML document is constructed upon object-based descriptions to represent (physical) data structure of a particular system. However, use of plain XML does not imply semantic behavior of the data (W3C 2002).

The Simulation Reference Markup Language (SRML) is a formal language for describing simulations using similar constructs of XML, developed by the Boeing Company. Gustavson and Chase (2004) state that "*SRML is like HTML in that it provides for executable content using the same kinds of mechanisms such as object models, scripting, plugs-in, and the ability to dynamically download and assemble content.*" SRML is designed to combine XML and those features, especially scripting, to encode both structure and behavior of entire simulations using classes and scripts (Reichenthal

2008). Moreover, simulations described in SRML can be executed in its runtime environment, Simulation Reference Simulator (SR Simulator).

In an SRML document, the structure of XML constructs is organized in a way that makes it practical to represent (Reichenthal and Johanson 2008):

- A set of interconnected items, both hierarchically and networked;
- Individual item behavior via scripts;
- Item classes, polymorphism, and multiple-inheritance;
- A means for synchronous, asynchronous, and scheduled communication; and
- Random events.

The example below illustrates how a simple SRML document is constructed.

```
<Simulation>
        <Script Type='text/javascript'>
        ...
        </Script>
        <ItemClass Name='Vehicle'>
                <Vehicle type="Passenger">
                        <Script Type='text/javascript>
                        ...
                        </Script>
                </Vehicle>
        </ItemClass>
        <Vehicle Quantity='4'/>
</Simulation>
```

In general, SRML uses a set of tags as any other XML-based language to create structured representations, including pre-designated attributes to describe abstract items. Meanwhile, internal and external structures are described and validated by XML and simulation-specific schemas (W3C 2002). For a better understanding, an explanation of the basic tags used in the example above is given. The <Simulation></Simulation> tags

encapsulate elements and scripts corresponding to an entire simulation. A simulation element contains zero or more Script elements as children to specify the main simulation behavior, using a <Script></Script> tag-set. This script can be written in JavaScript, VBScript, or any other script-language; however, JavaScript is set by default for an SRML script. The next part of the document is followed by a number of <ItemClass></ItemClass> tags, which define the different types of items required for the simulation. In the above example, a Vehicle is defined as an ItemClass element within a <ItemClass> tag-set. Each ItemClass element can typically be created by defining properties, structure, and behavior of the item. It can also contain zero or more Script elements describing the functionality of the item defined. After the <ItemClass>-tags, a set of instances of items that is being used in the simulation may or may not be specified. The example above shows that four vehicles are created. In addition to these simple constructs, the SRML document can contain other elements, for example, Links, Events, and so forth to support specific domains under simulation. For more information on SRML, see the W3C SRML website (http://www.w3.org/TR/SRML/).

To be concluded at this point, "*SRML should not be considered a programming language, but rather a composition language for integrating XML data models with behavior*" (Reinchenthal 2004). With these characteristics, SRML can provide enough expressive power to model most anything for purposes of simulation by a means of structural and behavioral documentation, which can be executed on computers. Furthermore, simulation models constructed in this form of SRML document supports both composability and interoperability as well. Hence, SRML has successfully been used in transforming conceptual models such as BOMs into simulations (Gustavson and

Chase 2004; Reichenthal, Gustavson, and Cruz 2003; Moradi, Nordvaller, and Ayani 2006; Moradi 2008).

Nevertheless, SRML includes a large number of features, some of which are not applicable in the scope of this research study – especially with ISAP. At the beginning, this research study has been set to avoid any burdens with designing complex high-level definitions of any host simulation or programming languages. It is focused on providing a friendly-user methodology that facilitates the development of domain specific simulation environments. The transformation of conceptualization has become a part of the methodology, creating a simulation document independently applied for implementation on any simulation environments. Hence, only has the architecture of the SRML language been utilized in documenting ISAP conceptual simulation models.

4-4. Model Transformation

This section presents a methodology based on contextualized-framework documentation to support the transformation of conceptual simulation models developed under ISAP. The document follows the SRML-based architecture using XML and JavaScript to capture and describe structural and behavioral components in a domain specific simulation. Even though ISAP uses three layers (e.g., Initialization, Process, and Termination) to represent a generic structure of the domain specific simulation, it is possible to develop all-in-one-type documentation to satisfy the individual layers' requirements. This type of documentation must allow selecting and representing the information from ISAP that can potentially be exchanged and mapped into a simulation environment on a conceptual level. The main purpose is to reduce some concerns related

to constraints and knowledge in implementation, so that the document can encapsulate the contents within the simulation context into an organized and easily-readable module.

An advantage of documenting ISAP information into a form of modularization is that each SMU can be transformed, integrated, distributed, and reused independently in separate documents, by separate model developers.

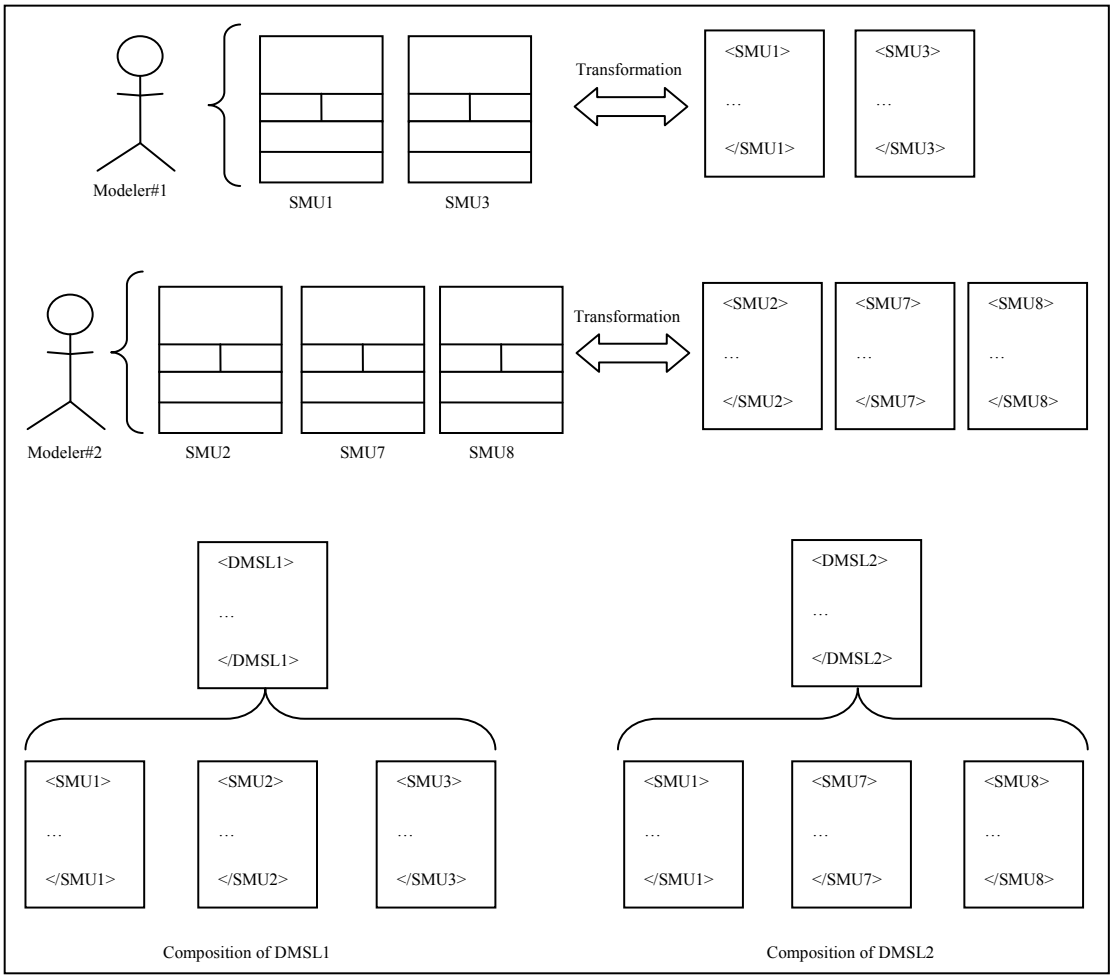

Figure 4 - 1: General approach of documentation for ISAP

A transformational module contains information not only from the targeted SMU but also from surrounding information such as inputs, relations, and sequences, described in the ISAP layers. This is to ensure that the transformational module provides sufficient

semantics of its construction as well as a generic simulation module/building-block in which the information are formalized into specific data components for future applications. Figure 4-2 shows the initial data components and their relations of a simulation module.

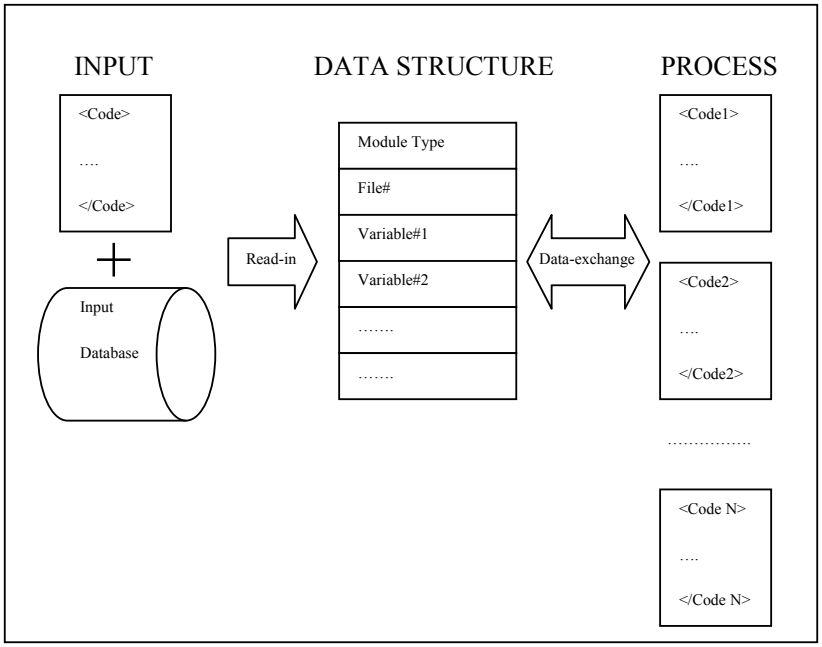

Figure 4 - 2: Initial structure of a simulation module

It is seen from the figure above that a simulation module is enabled by a collaboration of three data components: inputs, data structure (e.g., arrays and parameters), and process (e.g., functions). Its mechanisms require a set of instructions (codes) that are iteratively used to:

- Receive input data;
- Create parameters and a one-dimensional array to store input data;
- Generate one or more processes to execute parameters; and
- Return new data set to the storage.

Look at the mechanisms; it shows that all these instructions are written to describe how the data exchange and the logical execution of the simulation module implicitly take place. From this point of view, there are two major needs for an SMU to perform its transformation process: data descriptions and behavioral functions. Refer to the SRML-based architecture, data such as entities, properties, and logic can be described by XML, whereas behavioral functions such as create, assign, and queue can be represented by scripts (e.g., JavaScript). The illustration is given in Figure 4-3.

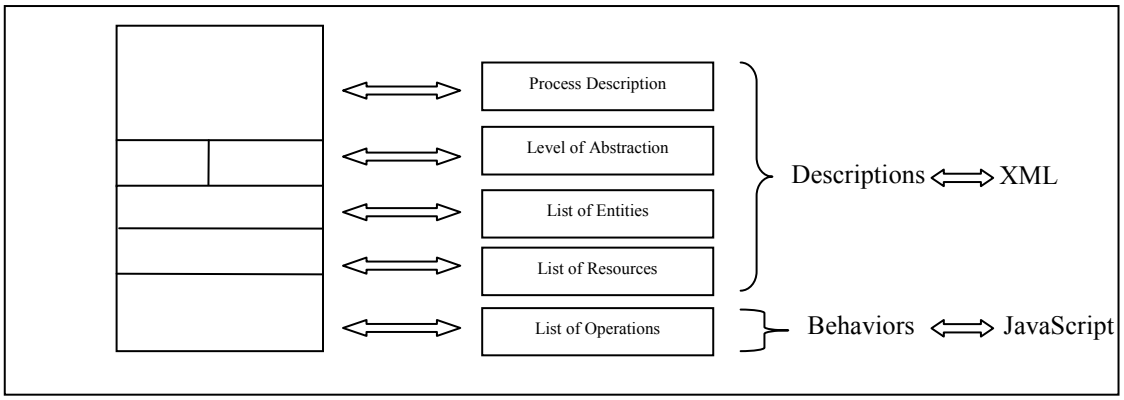

Figure 4 - 3: A scenario of transformation process

Later, each individual transformational module needs to be coupled into a network representing an entire simulation, which requires more additional descriptions and scripts to make the connection between them. The following subsections demonstrate how to apply this approach for the transformation of ISAP.

4-4-1. Selection

The transformation process starts with the selection of SMUs and their relevant information at the lowest abstraction level. The idea is to provide true semantics of the composition of transformational modules and the interoperation among them when put together in documentation. In ISAP, an entire system is logically viewed as a document folder so that the levels of abstraction can be decomposed into one or more sub-folders in a tree-hierarchical order. However, due to the fact that decomposition in ISAP exploits aggregation, the transformational module at the lowest level of abstraction may or may not contain all the inherit properties from its root. A loss of information does affect the completeness of implementation. Hence, it is necessary to derive and maintain descriptions and behaviors from the origins within documentation.

The lockage-operation example demonstrated in Chapter 3 is applied in this study to represent a continuous development process – from CSM to contextualized documentation. Basically, the selection begins at the network statement created from ISAP. The network statement contains the information of operations that are arranged based on logical sequences and decomposition levels. Then, it is to check back with its DMSL to find which SMUs those operations belong to. Finally, a list of SMUs is taken into consideration for the selection, which , in this example, consists of SMU *Inform Arrival of Barge-tows*, XOR *Decide Which Lockage Fits Barge tows' size*, SMU *Operate Single Lockage*, and SMU *Operate Double Lockage*.

4-4-2. Modularization

The next step is to transform each SMU into a transformational module using XML and JavaScript to specify its descriptive and behavioral characteristics.

XML allows the simulation developers to define their own tags to describe elements, properties, relations, and so forth. However, to make consistency through this research study, the following tags are set by defaults:

Table 4 - 2: Default-setting tags

Tag Name	Description
<DMSL></DMSL>	Encapsulate the entire dynamic modeling subsystem that represents a domain specific simulation.
<SMU></SMU>	Define different types of modules that need to be transformed and are present in the simulation.
<Entity></Entity>	Define different types of entities that contain their own properties (e.g. attributes) and flow through the simulation
<Resource></Resource>	Define different types of resources that contain their own properties (e.g. attributes) and process the entities.
<Script></Script>	Define different behaviors for the item corresponding to the enclosing element (e.g. DMSL or SMU). Each script may contain functions, procedures, or variables that override the previous script.
<Attribute/>	Define and assign attribute name and attribute type to the element.
<Variable/>	Define and assign global variable name and global variable type to the element.
<Link></Link>	Define relation types and destinations for the current SMU.

Assignment of properties to each element depends on what requirements are needed and how to translate those requirements by using descriptions. Properties are necessary for representing and, in some cases, adding more descriptions that cannot be covered by the ISAP conceptual models. It needs to be realized that levels of complexity in describing

properties are based on the quality and quantity of information that are derived from conceptualization and required for simulation. In Listing 4-1, for example, properties assigned to EntBargeTow and ResLock describe characteristics that specifically distinguish between an entity-type element and a resource-type element, providing acknowledgement of their contexts logically used for either within or separate modules (SMUs) to be referenced by functions.

Listing 4 - 1: XML-based descriptions of entity and resource

```
<Entity Name ="EntBargeTow">
  <Attribute atribname="Identification#" atribtype="interger"/>
  <Attribute atribname="NumberBarges" atribtype="integer"/>
  <Attribute atribname="Origin" atribtype="integer"/>
  <Attribute atribname="Destination" atribtype="integer"/>
  <Attribute atribname="ArrivalTime" atribtype="real"/>
  <Attribute atribname="Speed" atribtype="real">
</Entity>

<Resource Name ="ResLock">
  <Attribute atribname="Name" atribtype="string"/>
  <Attribute atribname="File#" atribtype="integer"/>
  <Attribute atribname="Resource#" atribtype="integer"/>
  <Attribute atribname="CapacityLock" atribtype="integer"/>
  <Attribute atribname="ActivityTime" atribtype="real"/>
</Resource>
```

Properties described in XML terms must also be relevant and accessible for functions specified in scripting – which is a critical concern for modularization by a means of interoperable-connection mechanisms – that controls interoperation of modules in separate development.

The previous study (Setavoraphan and Grant 2008) states that functions (or operations) within an SMU can be used to create one or more simulation modules/building-blocks as found in SIMAN and Visual SLAM. Furthermore, a study by Reinchenthal and Gustavson (2003) shows the use of behavioral markup (JavaScript) in SRML to describe the process blocks in SIMAN. As a result, each function attached to an SMU can be viewed as a block that contains encapsulation of elements, parameters, and sub-functions/methods as a self-describing process module. This block then can be transformed into a simulation module/building-block that can be reused in other compositions.

However, there exists a difficulty in applying this transformation method. It is still unable to avoid dealing with JavaScript and simulation programming which require knowledge, skills, and experiences in creating a module to function as expected. To encounter this difficulty, an approach to establish an intermediate simulation language based on JavaScript and host simulation programming has been enforced to the study of transformation. This language is not focused on implementation but rather on description to specify transformations and mediations between domains. It offers more logical and more flexible to create documentation that can be further developed in a host simulation language or mapped into simulation building-blocks available in generic simulation software. There is a question of whether documentation of transformation itself can be implemented into a simulation. The answer is "yes" if it is created under the environment runtime of a simulator such as Simulation Reference Simulator developed by Boeing (see http://www.w3.org/TR/SRML/). However, the scope of this research study leads to a finding of methodology that facilitates the development of a domain specific simulation

environment rather than a finding of validation that approves the correctness of documentation.

To reduce the trade-off caused by this approach, simulation developers are allowed to construct their own user-callable functions that can be used/reused and expected to be available – in host simulaiton languages (e.g., SIMAN/Arena and Visual SLAM/AweSim). User-callable functions are meant to work as a set of fundamental support functions for performing all commonly encountered functions such as event scheduling, statistics collection, and random sample generation. With these fundamental user-callable functions, the simulation developers are able to reduce difficulties in describing and specifying functions defined in the ISAP conceptual models by having none or a minimum of coding. Moreover, mapping is easier to be made because there are some common characteristics (similarities) between those in the intermediate simulation language and the target-host simulation languages with respect to meanings and specifications of functionalities that can be paired (more discussion in the next chapter).

Although levels of appropriateness of specifying user-callable functions are relied on the simulation developers' expertise in host simulation languages, the key of creation is to delivering a concrete perception of what each user-callable function is and how it works. Thus, it is essential for the simulation developers to provide references for user-callable functions in terms of function structures and descriptions, including properties related. Also, object classes that are used to reference functions and properties need to be defined. Visual SLAM, for example, includes VSLAM (the general simulator object), VSENTITY (an object for referencing an entity), VSENTRY (an object which maintains an entity's position within a file), and VSNODE (an object used to reference the

functions and properties of a network node) to support arguments to some of the functions and properties (Pritsker and O' Reilly 1999). Moreover, Visual SLAM allows users to define their own object classes on purposes. Objects, in general, are referenced to both functions and properties in the following manner: object.function and object.property.

In this study, the construction of fundamental user-callable functions have been developed based on user-callable and user-written visual basic functions available in Visual SLAM (Pritsker and O' Reilly 1999). The purpose is to make contextualized documentation for transformation more consistent and more effective. However, there is only one general object class named IP and user-defined object class being used to reference functions and properties – to decrease any complexity in exploitation. The following table lists some properties and user-callable functions that have been used in the documentation of DMSL: LCK and been generalized for other host simulation languages.

Table 4 - 3: Referenced properties for DMSL: LCK

References	Description
NewEntity()	Create a new entity.
CurrentEntity()	Return the current entity.
CloneEntity()	Clone the entity.
TerminateEntity()	Terminate the entity.
Release(Resource#, Units)	Release number of units of the resource#.
Seize(Resource#, Units)	Allocate number of units of the resource#.
NARES(Resource#)	Return the number of available units of the resource#.
NIUSE(Resource#)	Return the number of busy units of the resource#.
Resource()	Allocate a resource and assign its calling number
Schedule(Event, Entity, Time)	Schedule an event of type Event to occur at time TNOW + Time for the current entity.
Assign(Attribute 1, Attribute 2, …)	Assign one or more attributes to the entity.
LocateEntity(Event, Resource, Entity)	Locate the entity in the target resource
Intlc(run)	Check the initial run
TNOW	Current simulated time

Listing 4-2 shows the transformation of SMU Inform Arrival of Barge-tows into a module in documentation using properties and user-callable functions in the able tables.

Listing 4 - 2: A transformational module of SMU *Inform Arrival of Barge-tows*

```
<SMU Name = "Inform arrival of barge-tows">
    <Entity Name = "EntBargeTow">
      <Attribute atribname="Identification#" atribtype="interger"/>
      <Attribute atribname="NumberBarges" atribtype="integer"/>
      <Attribute atribname="Origin" atribtype="integer"/>
      <Attribute atribname="Destination" atribtype="integer"/>
      <Attribute atribname="ArrivalTime" atribtype="real"/>
      <Attribute atribname="Speed" atribtype="real">
     </Entity>

    <Script Type="text/javascript">
      <![CDATA[
```

```
    function CreateBargeTow()//Create and schedule entities
    {
      //Define variables used in this function
      var FirstArrival = 0;
      var ArrivalRate;
      var CurrentTime = TNOW;
      var MaxEntities;

      //Create a new entity
      set NewEntBargeTow = IP.NewEntity();
      set NewEntBargeTow.ArrivalTime = IP.TNOW;
      IP.Schedule("FirstArrival", NewEntBargeTow,
(NewEntBargeTow.ArrivalTime+ArrivalRate);

      //Schedule the next entities
      for (i=1; i<=Max# entities; i++)
        {
          set NextEntBargeTow = IP.CloneEntity();
          set NextEntBargeTow = IP.TNOW;
          IP.Schedule("NextArrival", NextEntBargeTow,
(NextEntBargeTow.ArrivalTime+ArrivalRate));
        }
    }

    function AssignBargeTow()//Assign attributes to the BargeTow
entities
    {
      var Identification#;
      var NumberBarges;
      var Origin;
      var Destination;
      var ArrivalTime;
      var Speed;

      //Define the current EntBargeTow entity and assign attributes to
it
      set CurrentEntBargeTow = IP.CurrentEntity();
      CurrentEntBargeTow.Assign(Identification#, NumberBarges, Origin,
Destination, ArrivalTime, Speed);
```

```
    }

    function RouteBargeTow()//Schedule the current EntBargeTow entity
for travelling
    {
      var Distance;
      var Speed;
      var DelayTime = Distance/Speed;
      IP.Schedule("Decision", CurrentEntBargeTow,
CurrentEntBargeTow.DelayTime);
    }
      ]]>
   </Script>
```

4-4-3. Integration

The final step is to combine a set of transformational modules and the root of a dynamic modeling subsystem. Like a general simulation environment, it contains zero or more global variables that can be accessed and used by its children, and specifies the main simulation behavior or main function (script) that controls the overall operations of the simulation. The integration of these elements leads to a complete simulation documentation being used as the future reference for implementation. Listing 4-3 illustrates a sample of documentation for the lockage operation system, LCK.

Listing 4 - 3: Partial documentation for DMSL: LCK

```
<DMSL Name ="LCK">
  <Variable varname="Offset" vartype="boolean"/>
  <Variable varname="Offset enter value" vartype="boolean"/>
  <Variable varname="Offset exit value" vartype="boolean"/>
  <Script Type="text/javascript">
    <![CDATA[
```

```
  //Initialize variables for the first run
  function Initial()
  {
    Intlc(run);//Check the initial run
    if (run = 1)
      {
        var Offset = 0;
        var Offset enter value = 0;
        var Offset exit value = 0;
      }
   }
    ]]>
</Script>

<SMU Name = "Inform arrival of barge-tows">
  <Entity Name = "EntBargeTow">
    <Attribute atribname="Identification#" atribtype="interger"/>
    <Attribute atribname="NumberBarges" atribtype="integer"/>
    <Attribute atribname="Origin" atribtype="integer"/>
    <Attribute atribname="Destination" atribtype="integer"/>
    <Attribute atribname="ArrivalTime" atribtype="real"/>
    <Attribute atribname="Speed" atribtype="real">
    </Entity>

  <Script Type="text/javascript">
    <![CDATA[

  function CreateBargeTow()//Create and schedule entities
  {
    //Define variables used in this function
    var FirstArrival = 0;
    var ArrivalRate;
    var CurrentTime = TNOW;
    var MaxEntities;

    //Create a new entity
    set NewEntBargeTow = IP.NewEntity();
```

```
      set NewEntBargeTow.ArrivalTime = IP.TNOW;
      IP.Schedule("FirstArrival", NewEntBargeTow,
(NewEntBargeTow.ArrivalTime+ArrivalRate);

      //Schedule the next entities
      for (i=1; i<=Max# entities; i++)
        {
          set NextEntBargeTow = IP.CloneEntity();
          set NextEntBargeTow = IP.TNOW;
          IP.Schedule("NextArrival", NextEntBargeTow,
(NextEntBargeTow.ArrivalTime+ArrivalRate));
        }
    }

    function AssignBargeTow()//Assign attributes to the BargeTow
entities
    {
      var Identification#;
      var NumberBarges;
      var Origin;
      var Destination;
      var ArrivalTime;
      var Speed;

      //Define the current EntBargeTow entity and assign attributes to
it
      set CurrentEntBargeTow = IP.CurrentEntity();
      CurrentEntBargeTow.Assign(Identification#, NumberBarges, Origin,
Destination, ArrivalTime, Speed);
    }

    function RouteBargeTow()//Schedule the current EntBargeTow entity
for travelling
    {
      var Distance;
      var Speed;
      var DelayTime = Distance/Speed;
      IP.Schedule("Decision", CurrentEntBargeTow,
CurrentEntBargeTow.DelayTime);
    }
```

```
      ]]>
   </Script>

  <Link Name="J2" Type="Precedence">
  <Link Target="XOR: Decide which lockage fits barge tows' size">
  </Link>
  </SMU>

  <SMU Name ="XOR: Decide which lockage fits barge tows' size">
    <Entity Name ="EntBargeTow">
     <Attribute atribname="Identification#" atribtype="interger"/>
     <Attribute atribname="NumberBarges" atribtype="integer"/>
     <Attribute atribname="Origin" atribtype="integer"/>
     <Attribute atribname="Destination" atribtype="integer"/>
     <Attribute atribname="ArrivalTime" atribtype="real"/>
     <Attribute atribname="Speed" atribtype="real">
    </Entity>

    <Resource Name ="ResLock">
     <Attribute atribname="Name" atribtype="string"/>
     <Attribute atribname="File#" atribtype="integer"/>
     <Attribute atribname="Resource#" atribtype="integer"/>
     <Attribute atribname="CapacityLock" atribtype="integer"/>
     <Attribute atribname="ActivityTime" atribtype="real"/>
    </Resource>
    <Script Type="text/javascript">
    <![CDATA[

    function CheckBargeTow()//Retrieve the value of number of barges
from the current EntBargeTow entity
    {
      var NumberBarges;
      set CheckNumberBarges = CurrentEntBargeTow.NumberBarges;
    }

    function SelectLockage()//Retrieve the capacity value from the lock
Resource#
    {
```

```
      var LockCapacity;
      set ResLock = IP.Resource();
      set LockCapacity = ResLock.CapacityLock;
     }

    function BranchBargeTow()
    {
      var LockCapacity;
      if (CheckNumberBarges <= LockCapacity)
      IP.LocateEntity("Operate single lockage", ResLock,
CurrentEntBargeTow);
      else if (CheckNumberBarges > LockCapacity)
      IP.LocateEntity("Operate double lockage", ResLock,
CurrentEntBargeTow);
    }

    ]]>
    </Script>

   <Link Name="LockType" Type="Precedence with condition(s)">
   <Link Target="Operate single lockage"/>
   <Link Target="Operate double lockage"/>
   </Link>
   </SMU>

   <SMU Name ="Operate single lockage">
    <Entity Name ="EntBargeTow">
      <Attribute atribname="Identification#" atribtype="interger"/>
      <Attribute atribname="NumberBarges" atribtype="integer"/>
      <Attribute atribname="Origin" atribtype="integer"/>
      <Attribute atribname="Destination" atribtype="integer"/>
      <Attribute atribname="ArrivalTime" atribtype="real"/>
      <Attribute atribname="Speed" atribtype="real">
      </Entity>

    <Resource Name ="ResLock">
      <Attribute atribname="Name" atribtype="string"/>
      <Attribute atribname="File#" atribtype="integer"/>
```

```
      <Attribute atribname="Resource#" atribtype="integer"/>
      <Attribute atribname="CapacityLock" atribtype="integer"/>
      <Attribute atribname="ActivityTime" atribtype="real"/>
    </Resource>

    <Script Type="text/javascript">
     <![CDATA[
    function SetLockState()
    {
      if (NIUSE(ResLock) >=1)//Lock is occopied
        {
          var Offset = 1;//State is busy
          var Offset enter value = 1;//Enter gate is closed
          var offset exit value = 1;//Exit gate is closed
        }
       else (NIUSE(ResLock) <=0)//Lock is available
        {
          var Offset = 0;//State is idle
          var Offset enter value = 0;//Enter gate is opened
          var Offset exit value = 0;//Exit gate is opened
        }
    }

   function ProcessLock()
   {
      var ActivityTime;
      if (NARES(ResLock) >0)
       {
         IP.Seize(ResLock, 1);
         IP.SetLockState();
         IP.Schedule("Lockage", CurrentEntBargeTow,
(CurrentEntBargeTow.TNOW+ActivityTime));
         IP.Release(Reslock, 1);
       }
   }

   function RouteBargeTow()//Schedule the current EntBargeTow entity
for exiting lockage
```

```
   {
      var Distance;
      var Speed;
      var DelayTime = Distance/Speed;
      IP.Schedule("Exit", CurrentEntBargeTow,
CurrentEntBargeTow.DelayTime);
   }

    ]]>
  </Script>
  <Link Name="Exit" Type="Precedence">
  <Link Target="Set departure of barge-tows"/>
  </SMU>
```

//The rest of documentation includes the descriptions and specifications of SMU *Operate Double Lockage* and SMU *Set Departure of Barge-tows*. The construction follows the methodology described in these sub sections. This listing is just aimed to show how documentation of transformation is developed.

```
</DMSL>
```

4-4-4. Revision

Like other documentation, a numerous iterative revisions and editions are very vital in clarifying the semantics of a transformational document. These actions must be taken within not only apiece of modules nor the entire document but also conceptual models and simulations. The patterns of the actions are: a) Vertical search for appropriateness of decomposition (top-down) and for semantics of composition (bottom-up); and b) Horizontal search for degrees of implementation (left-to-right) and for levels of communication (right-to-left). The purpose of following these patterns is to generate

and keep up the semantics of model composability and simulation interoperability throughout the development process for domain specific simulation. The final documentation, hence, becomes a handbook to facilitate any construction of simulation models under a specific domain.

4-5. Conclusions

Many research studies have been found with their efforts to encourage the M&S community to recognize the importance of semantics of model composability and simulation interoperability when building conceptual models. They expect not only to improve this cross-domain communicational tool but also to promote its potential utilization in other applications. To satisfy these expectations, the use of contextualized-framework documentation has been introduced to facilitate model transformation, which leads conceptual models to have more expressive and meaningful representations in the levels of implementation.

An Ontology-based approach has added the capability to documentation to describe both structural and behavioral simulation characteristics in a more executable and readable way, using the Semantic Web technologies like XML and SRML. The derived concepts provide this study a thoughtful approach to develop an intermediate simulation language to compose a transformational document. Furthermore, this type of documentation contains a set of modules that can possibly be translated into composable and reusable simulation modules/building-blocks or any host simulation languages, as will be seen in this dissertation. However, it is not possible to translate all of the modules in the document directly into those targets. Therefore, future research should be focused

on finding a methodology to support the translation of documentation, based on ontology mapping and knowledge-base selecting algorithms.

The next chapter is focused on mapping the descriptions of systems developed using Transformation into implemented simulation tools.

4-6. References

1. Analyti, A., M. Theodorakis, N. Spyrotos, and P. Constantopoulos. 2007. Contextualization as an independent abstraction mechanism for conceptual modeling. Accessed http://www.sciencedirect.com on November 17, 2008.
2. Berners-Lee, T. 2001. The Semantic Web. *Scientific American*, 284, issue 5:28.
3. Daconta, M. C., L. J. Obrst, K. T. Smith. 2003. *The semantic web: A guide to the future of XML, web services and knowledge management*. John Wiley & Sons.
4. Daum, B. 2003. Modeling business objects with XML schema. *Morgan Kaufmann*, 2003.
5. Davis, P. K. and R. H. Anderson. 2003. Improving the composability of Department of Defense Models and Simulations. Prepared for the Defense Modeling and Simulation Office.
6. IEEE Standard Computer Dictionary: *A Compilation of IEEE Computer Glossaries*, New York: IEEE, 1990.
7. Fishwick, P. A. 2002. Using XML for simulation modeling. In *Proceedings of the 2002 Winter Simulation Conference*, 616-622.
8. Fishwick, P. A. and J. A. Miller. 2004. Ontologies for modeling and simulation: issues and approaches. In *Proceedings of the 2004 Winter Simulation Conference*.

9. Gustavson, P. and T. Chase. 2004. Using XML and BOMs to rapidly compose simulations and simulation environments. In *Proceedings of the 2004 Winter Simulation Conference*.

10. Hemel, Z., L. C. L. Kats, and E. Visser. 2008. Code generation by model transformation. A Case Study in Transformation Modularity. In *International Conference on Model Transformation*.

11. Hofmann. M. 2002. Introducing pragmatics into VV&A. In *Proceedings of the European Simulation Interoperability Workshop*, London, UK.

12. Koch, N. 2006. Transformation techniques in the model-driven development process of UWE. In *ICWE 2006 Workshops*.

13. Lim, N. 2004. RUBE_QM: A 3D simulation and modeling approach for Queuing Systems. *Master Thesis*, University of Florida.

14. Lu, R. F., G. Qiao, and C. McLean. 2003. Nist XML simulation interface specification at Boeing: A case study. In *Proceedings of the 2003 Winter Simulation Conference*, ed. S. Chick, P. J. Sanchez, D. Ferrin, and D. J. Morrice. 1230-1237.

15. Moradi, F. 2007. Component-based simulation model development using BOMs and web services. In *Proceedings of the first Asia Modeling Symposium*.

16. Moradi, F. 2008. A Framework for component based modeling and simulation using BOMs and Semantic Web Technology. *Doctoral Thesis*, KTH Computer Science and Communication.

17. Moradi, F., P. Nordvaller, and R. Ayani. 2006. Simulation model composition using BOMs. In *Proceedings of the Tenth IEEE International Symposium on Distributed Simulation and Real-Time Applications*.
18. Morse. K., M. Petty, P. Reynolds, W.. Waite, P. Zimmerman. 2003. Findings and recommendations from *the 2003 Composable Mission Space Environments Workshop*.
19. Nijhuis. M. 2005. Generating natural language explanations for conceptual models. In *the 3rd Twente Student Conference on IT*.
20. Page, E. H., R. Briggs, and J. A. Tufarolo. 2004. Toward a family of maturity models for the simulation interconnection problem. In *Proceedings of IEEE Spring Simulation Interoperability Workshop*.
21. Petty, M. D. 2002. Semantic Composability and XMSF. In *XMSF Technical Challenges Workshop 2002*, Monterey, CA.
22. Petty, M. D. 2004. Simple composition suffices to assemble any composite model. In *Proceedings of the Spring 2004 Simulation Interoperability Workshop*, Orlando, Florida.
23. Petty, M. D. and E. W. Weisel. 2003. A composability lexicon. In *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, Orlando, Florida.
24. Pokraev, S., M. Reichert, M. W. A. Steen, and R. Wieringa. 2005. Semantic and pragmatic interoperability: A model for understanding. *EMOI-INTEROP*, 2005.
25. Pritsker, A. A. B. and J. J. O'Reilly. 1999. *Simulation with Visual SLAM and AweSim*. 2nd ed. New York: John Wiley & Sons.

26. Quin, L. 2003. Extensible Markup Language (XML). Accessed http://www.w3c.org/XML/ on October 28, 2008.
27. Reichenthal, S. W. 2002. Re-introducing web-based simulation. In *Proceedings of the 2002 Winter Simulation Conference*.
28. Reichenthal, S. W. 2004. SRML Case study: Simple self-describing process modeling and simulation. In *Proceeding of the 2004 Winter Simulation Conference*.
29. Reichenthal, S. W. 2008. On the integration and distribution of XML-Based simulations. From steven.w.reichenthal@boeing.com, on October 20, 2008.
30. Reichenthal, S. W. and P. L. Gustavson. 2003. Manufacturing BOMs with SRML for process-oriented federations. Simulation Interoperability Standards Organization. In *Spring Simulation Interoperability Workshop*, Orlando, Florida.
31. Reichenthal, S. W., P. L. Gustavson, and J. de la Cruz. 2003. Case study: Prototyping a mega-BOM with SRML for next-generation combat support. Simulation Interoperability Standards Organization. In *Spring Simulation Interoperability Workshop*, Orlando, Florida.
32. Reichenthal, S. W. and B. E. Johanson. 2008. Case study: Lessons learned in availability modeling. Retrieved from steven.w.reichenthal@boeing.com on October 20, 2008.
33. Robinson. S. 2006. Issues in Conceptual Modeling for Simulation: Setting a research agenda. In *2006 OR Society 3rd Simulation Workshop*, Ashorne, UK.
34. Rumbaugh, J., M. Blaha, W. Premerani, F. Eddy, and W. Lorensen. 1991. *Object-oriented modeling and design*, New York: Prentice Hall.

35. Setavoraphan, K. and F.H. Grant. 2008. Conceptual Simulation Modeling: The Structure of Domain Specific Simulation Environment. In *Proceedings of the 2008 Winter Simulation Conference*.
36. Simulation Interoperability Standards Organization. 2006. Base Object Model (BOM) Template Specification. *SISO-STD-003-2006*.
37. Tolk, A. and C. L. Blais. 2005. Taxonomies, ontologies, and battle management languages – recommendations for the Coalition BML Study Group. In *Proceedings of IEEE Spring Simulation Interoperability Workshop*.
38. Tolk, A. and J. A. Muguira. 2003. The Levels of conceptual interoperability model. In *the 2003 Fall Simulation Interoperability Workshop*, Orlando, Florida.
39. Tolk, A. and C. D. Turnitsa. 2007. Conceptual modeling of information exchange requirements based on ontological means. In *Proceedings of the 2007 Winter Simulation Conference*.
40. Valenti, S., M. Panti, and A. Cucchiarelli. 1998. Overcoming communication obstacles in user-analyst Interaction for functional requirements elicitation. *ACM SIGSOFT Software Engineering Notes*, 23, issue 1 (January): 50-55.
41. Vasilecas, O. and D. Bugaite. 2007. An algorithm for the automatic transformation of ontology axioms into a rule model. *CompSysTech*, 2007.
42. W3C. 2002. Simulation Reference Markup Language element specification 3.0. Accessed http://www.w3.org on November 17, 2008.
43. Walsh, N. 1999. Understanding XML Schemas. Accessed http://www.xml.com/pub/a/1999/07/schemas/index.html on October 28, 2008.

44. Weisel, E. W., M. D. Petty, and R. R. Mielke. 2003. Validity of models and classes of models in semantic composability. In *Proceedings of the Fall 2003 SIW*, Orlando, Florida.

45. Wikipedia. 2008. Ontology (information science). Accessed http://en.wikipedia.org/wiki/Ontology_(computer_science) on November 1, 2008.

CHAPTER 5

Domain Specific Ontological Mapping: An Integrated Approach

"Reproduced with automatic permission from [Setavoraphan, K. and Grant, F. H. (2009) Domain specific ontological mapping: An integrated approach, being in progress to submit to Journal of Computers & Industrial Engineering]. It has been modified somewhat to reflect current advances in this research."

Abstract

To establish a domain specific simulation environment (DSSE) from conceptual simulation models (CSMs), one of the most efficient and easiest solutions is to map CSMs onto an existing simulation environment or a host simulation language. Based on this idea, a simulation developer can exploit available resources (e.g., building blocks or callable functions) similar to the CSMs to develop his/her own domain specific simulation environments that provide, e.g., reusable model constructs/functions and their callable libraries. However, this mapping is not as easily done as it may seem. This is because mapping requires not only a common layer for information/knowledge exchange but also a framework and pattern for mapping. Methodologies such as ontology mapping, simulation block building, and visual subnetwork modeling have been applied to develop an integrated approach that facilitates mapping in this research study.

5-1. Introduction

The development of a domain specific simulation environment (DSSE) consists of three major pieces: structure, content, and simulation environment application. First, the structure of the DSSE can be laid out by using the conceptual simulation modeling

(CSM) approach to generate a blueprint, describing physical and behavioral characteristics of both reality and simulation domain. CSM also delivers a communication tool for domain experts and simulation developers to support their collaborations. Among CSM methods and tools available in Modeling and Simulation (M&S) communities, Integrated Simulation Acknowledge Procedure (ISAP) has been selected to manage and access critical aspects (e.g., static/dynamic components, functional layers, and representations) required for structuring and composing DSSE (see Setavoraphan and Grant (2008)). However, CSM itself is not complete enough to develop DSSE because all the details of needed simulation components cannot be included within this kind of knowledge-based representation – using symbols, notations, and diagrams.

To avoid inconsistency and invalidity in using conceptual simulation models, those symbols, notations, and diagrams need to be transformed into a contextualized document. This process aims to retrieve the appropriate simulation contents from CSMs. Making contextualized documentation helps not only eliminate irrelevance but also improve the semantics of the contents. To conduct such a transformational document, the Semantic Web terminology has been used with the exploitation of an ontological analysis approach, including a common language (e.g., XML) and an integrated descriptive and behavioral language (e.g., SRML). This contextualized documentation later plays a critical role together with CSMs in developing DSSE.

DSSE is determined as a simulation environment application that provides reusable simulation model constructs to represent domain specific system elements. Basically, a DSSE application can be developed under: a) an original simulation environment where everything is uniquely created; or b) an existing simulation

environment where the availability of resources exists for accessing and utilizing by the simulation developers. Both conditions have advantages and disadvantages. However, the focus of this study is to develop a methodology that facilitates the development of a DSSE application – which not only minimizes cost and time but also maximizes productivity and efficiency. Based on this methodology, it is assumed that the simulation developers already have a simulation environment application for generic uses. Theoretically, it is possible to transform the simulation environment application to be a DSSE application by mapping the structure and simulation contents into those available resources, so that the resources are caused to function as defined in specifications. Mapping, thus, is seen as the key solution in developing a DSSE application under the existing simulation environment.

Mapping is taken into consideration to enable individuals to keep their own world views and at the same time to share knowledge across domains (Ehrig and Sure 2004). By a means of sharing knowledge, mapping is required to deal with semantic interoperability, the issue of allowing the exchange meaningful information/knowledge between applications/domains (Bouquet et al. 2003). However, the problem here is that the representations of information/knowledge in each domain are depicted in different ways. To solve this problem, one needs to view the information/knowledge representations in the framework of an ontology which provides a joint terminology and frame of reference of specifications and semantics of conceptualization. The use of ontologies helps create a common layer where the conceptualization of information/knowledge can be transformed and categorized into a set of standard representation elements such as entities, properties, and relations. When the

information/knowledge from alternative domains is agreed in such a sense of semantic similarities, it allows the individuals to recognize what to be mapped. However, to have the efficient and correct exchange of information/knowledge between domains through mapping, the framework and pattern of mapping must be clearly specified.

As described above, mapping by a means of ontology mapping is seen as a solution to facilitate the exchange information/knowledge between domains. There are a number of articles in the literature focusing on research in ontology mapping in different areas to provide definitions, techniques, algorithms, and representations of mapping (see Ehrig and Sure 2004; Marques 2005; Kalfoglou and Schorlemmer 2003). The concepts retrieved from the literature are used to support creating a specific framework and pattern of mapping to match the characteristics and requirements of the materials at hand (e.g., CSMs, contextualized documentation, and simulation environment application). This means that mapping is not just about sharing information/knowledge between different ontologies, but it also includes the levels of similarities in many aspects (e.g., structure, contents, and representations). Therefore, the methodologies of simulation block building and visual subnetwork modeling have been integrated into ontology mapping to construct a solid and robust framework and pattern of mapping between conceptualization and a simulation environment application – to generate a DSSE application.

This research study is organized as follows: Section 5-2 describes the key concepts that include ontology mapping, simulation block building, and visual subnetwork modeling. The implementation of the concepts is demonstrated in Section 5-3. Finally, the conclusions and recommendations are given in Section 5-4.

5-2. Key Concepts

This section is aimed to provide a survey of the key concepts found in a number of articles in the literature related to ontology mapping, simulation block building, and visual subnetwork modeling. There is a particular purpose lying within each of these concepts. Ontology mapping is important for transferring simulation contents between domains. Simulation block building is applied to design the structure of the mapping destination that will encapsulate the simulation contents being used in a simulation environment application. Finally, visual subnetwork modeling provided in Visual SLAM and AweSim is used to configure and enforce the simulation building blocks to function.

Moreover, the selection of definitions, methods, techniques, and tools provided for the methodologies has been made based upon the overall framework designed for the development of a DSSE application. This framework is expected to arrange and control the similarities of structure, simulation contents, and simulation environment applications for both conceptualization and application. The idea behind the framework of similarity is to eliminate complexity, irrelevance, and inconsistency in transferring semantics of information/knowledge during the transitions of representation formats from one to another.

5-2-1. Ontology Mapping

"*An ontology is an explicit, formal specification of a shared conceptualization of a domain of interest*" (Gruber 1993). The purpose of using a terminology of ontology is to "*reduce or eliminate conceptual and terminological confusion among the members of a user community who need to share various kinds of (electronic) documents and*

information" (Navigli, Velardi, and Gangemi 2003). To accomplish this purpose, a set of relevant concepts (e.g., entities, instances, relations, and properties) that characterize a given domain needs to be identified and defined properly, which later becomes a mutual point of interest for mapping of two ontologies.

According to Rahm and Bernstein (2001), an ontology mapping process is defined as a set of activities required to transform instances of a source ontology into instances of a target ontology. For a clearer picture, Ehrig and Staab (2004) define the term of ontology mapping: "*Given two ontologies O_1 and O_2, mapping one ontology onto another means that for each entity (concept C, relation R, or instance I) in ontology O_1, we try to find a correspond entity, which has the same intended meaning, in ontology O_2.*" Also, ontology mapping can be defined in different terms such as alignment, merging, articulation, fusion, integration, morphism, and so on, depending on the application and intended outcome (see the details in Kalfoglou and Schorlemmer 2008).

Recently, there are numerous frameworks that provide a methodological approach to ontology mapping. For example, a cooperative framework for integrating ontologies described by Breis and Bejar (2002) is a system having the algorithm that supports the integration by using taxonomic features and synonymous concepts in the two ontologies. Madhavan et al. (2002) develop a framework that enables mapping between ontologies in different representation languages without first translating the ontologies into a common language. A framework for ontological structures to support ontology sharing, namely IFF, is proposed by Kent (2000), which represents ontologies as logics and ontology sharing as a specifiable ontology extension hierarchy. Among the frameworks available up to date, there exists a set of commonalities in their approaches and processes to

ontology mapping. These commonalities are assembled and identified in the MAFRA conceptual framework (Maedche et al. 2002), providing the critical clues that lead to the possibilities for mapping between conceptualization and a simulation environment application.

Maedche and Staab (2000) develop MAFRA as a mapping framework for distributed ontologies in the Semantic Web. MAFRA is built on the idea that mapping existing ontology will be easier than creating a common ontology. This is because only a smaller community is involved in the mapping process. Also, this framework aims to detect similarities of entities contained in two different ontologies – being the critical mechanisms of this mapping framework. Thus, the framework of MAFRA discovery reveals the essential modules that support the exploitation of semantic similarities in ontology mapping, as described in follows:

- *Lift and Normalization*: The main purpose of this module is to raise all data to be mapped onto the same representation level, which copes with syntactical and structural language heterogeneity (Visser et al. 1997). Maedche and Staab (2000) states that "*both ontologies must be normalized to a uniform representation, …, thus eliminating syntax differences and making semantics differences between the source and the target ontology more apparent*."
- *Similarity*: This module aims to support mapping discovery by establishing similarities between entities from the source and target ontology. The mapping approach is based on different similarity measures, which have been proposed in the literature by Rahm and Bernstein (2001), Doan et al. (2002), and Maedche and Staab (2000).

- *Semantic Bridging*: This module is responsible for establishing correspondence between entities from the source and target ontology based on the similarities found between them. The goal in specifying the semantic bridge ontology is to maintain and exploit the existent constructs and minimize extra constructs (Maedche and Staab 2000).

Referring to the concepts of these modules, it can be concluded that the key role in conducting ontology mapping is the ability to establish similarities between the source and target ontology and to specify a common representation framework (or space) for mapping these similarities. In Section 5-3, a work of ontology mapping based on the similarity-centric approach is demonstrated.

5-2-2. Simulation Block Building

The BETADE program at Delft University of Technology, Netherland, has been applied to support distributed working, designing, and modeling in order to construct distributed applications or models entirely out of reusable building blocks. The working definition used in the BETADE research program is (Verbraeck et al. 2002):

"*A building block is a self-contained, interoperable, reusable, and replaceable unit, encapsulating its internal structure and providing useful services or functionality to its environment through precisely defined interfaces. A building block may be customized in order to match the specific requirements of the environment in which it is 'plugged' or used*."

To apply a building block in such an environment, it needs to clarify the relationship between a building block and components or other related terms. Verbraeck et al. (2002) states: "*A component is the implementation of a building block in a software*

environment. The interface (functionality) of the building block and the component are therefore different presentations of the same thing." The authors also argue that a building block on its own – without domain specific context, communicating, co-operating or even competing with other building block – cannot provide the functionality a user requires.

In order to provide functionality, an application is constructed by an aggregate of more than one building block, where lower level building blocks are combined into new higher level building blocks and interact each other to function as the user specifies. The aggregation of building blocks is based on the object-oriented paradigm in order to support reusability of building blocks in applications or models, which can be illustrated as in Figure 5-1. It is seen that a set of building blocks consists of model building blocks (1^{st} level) that are constructed of building block elements (2^{nd} level) (Verbraeck and Valentin 2002). Each building block element communicates using a standard interface used for formal entries for messages and entity passing and represents a specific functionality. Therefore, different kinds of model building blocks can be designed and constructed by using different building block elements.

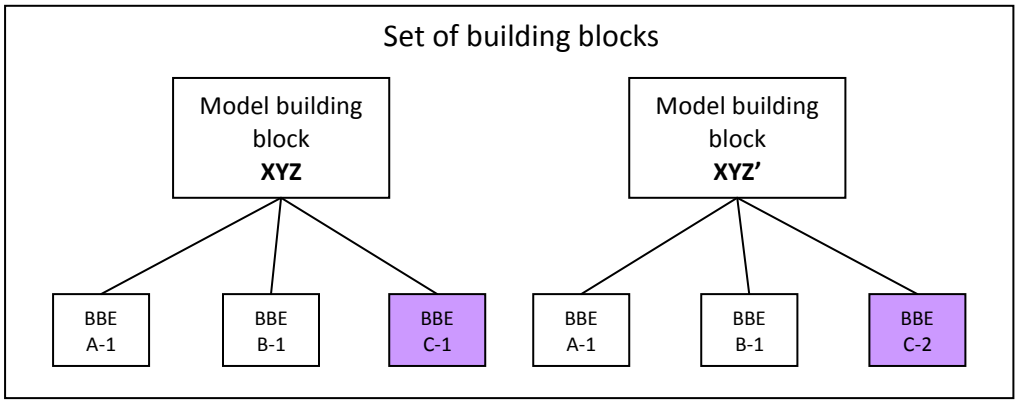

Figure 5 - 1: Example of a set of model building blocks using building block elements (Valentin and Verbraeck 2002, p567)

Recently, building blocks have been applied in simulation studies to support the development of discrete event simulation models (see examples in Verbraeck and Versteegt 2000; Valentin 2002; and Saanen 2002). Similar model constructs are used over and over again, especially when developing different simulation models within the same domain. In this case, it seems very logical to structure and package the repetitive model constructs into building blocks, and make them available for the modeler for repeated use (Valentin 2002). Moreover, these building blocks can be collected together in categorized libraries that emerge a set of specific vocabularies for future callable references in simulation modeling – which later possibly turns into a domain specific simulation language (DSSL).

The main purpose in establishing a DSSL is to define and specify semantics, relations, and constraints associated with those domain concepts in a representation format of domain specific simulation elements – being used as interfaces of communication. Like a natural language, when the popularity of using and creating domain specific simulation elements within DSSL increases, it is able to create its own

simulation environment (community) to support the development of simulation models (communication) using those elements (vocabularies). This kind of simulation environment can be determined as a domain specific simulation environment (DSSE). Following this logical reasoning, it can be concluded that building blocks play a critical role in the development of DSSE by a means of structuring, defining semantics, and configuring functionalities.

According to Snowdon et al. (1998), building blocks must support easy model development, which means a set of building blocks can be viewed as a conceptual model that can be reused and directly/easily transferred to the simulation model. The statement leads this research study to find out a connection between conceptualization and simulation, in which a building block becomes a data bridge of two different domains. In addition to the characteristics as of DSSL, the building block is also capable of communicating and translating concepts either implicitly or explicitly of its environment. This allows the building block to merge with other existing DSSLs by customization/configuration to match the specific requirements of the new environments. As the result, it is not necessary to think of the development of building blocks in a traditional way – which starts from a scratch.

The literature by Verbraeck and Valentin (2001) shows that building blocks can be developed in existing simulation environments such as Arena, Automod, eM-plant, and Taylor ED, whose system architecture is based on object orientation. The authors also define the characteristics of simulation building blocks used in the existing simulation environments: "*Building blocks can range from just one very basic functionality (like executing a simple function) till very complex building blocks with*

hundreds of functionalities, no matter what kind of problem, and no matter how large the set of available building blocks is." Their statement substantiates that such an object-oriented simulation environment is fit best with the characteristics of simulation building blocks since it provides the most important mechanisms – composition. A simulation building block can be composed of either basic simulation building block elements (e.g., create, queue, and resource) or customized/specialized simulation building block elements (e.g., conveyor, crane, and AGV) in a specific way. However, configuration of the building block is still needed to have well defined interface for connecting it to other building blocks and to fit in its environments. Therefore, experiences and knowledge in employing specific simulation environments are critical to succeed in composing simulation building block elements and connecting them together to function.

Based on the concepts described through this subsection, the author is able to make a hypothesis that we have explored the successful development of simulation building blocks using an existing object-oriented simulation environment. Visual SLAM and AweSim, thus, have been chosen as a simulation modeling language/environment. A brief discussion of Visual SLAM and AweSim, including its feature that supports building simulation blocks, is given in the next subsection.

5-2-3. Visual Subnetwork Modeling

Prior to have a better understanding why Visual SLAM and AweSim have been selected for this research study, it is necessary to recognize the idea behind the development of this simulation modeling language/environment. Pritsker and O' Reilly

(1999) provides an explanation related to their perspectives and essential concepts for building blocks:

"*For many years, it has been desired to develop a modeling language that is modular and hierarchical. Modularity would allow submodels to be developed and used as building blocks for a total systems model. Hierarchical models would display the aggregate features of a model and allow the details to be viewed by driving the view to less aggregate displays of the model. These properties would allow for the building of submodels by team members which then could be integrated into a system model. To achieve these capabilities, modeling languages were designed using object-oriented concepts*."

This idea, thus, becomes a fact that the object orientation is taken in Visual SLAM. It aims to employ object-oriented concepts and coding within the network worldview (or objects) of the Visual SLAM simulation language. In the meantime, AweSim is a simulation problem-solving environment for Visual SLAM, providing extensive input, output, and integration capabilities to facilitate the use of Visual SLAM by users. For the development of simulation building blocks, both Visual SLAM (modeling language) and AweSim (mechanisms/environment) are needed.

Since Visual SLAM employs object-oriented concepts, it allows for defining a subnetwork as an object class. An entity is routed to the subnetwork for a particular instance of that object class. For example, different machines that perform similarly to process parts can be modeled as a subnetwork. The subnetwork for the processing to be done by the specific machine is modeled by Visual SLAM network elements and by passing parameters to define the node and activity characteristics for the subnetwork instance. Because of the object nature of a subnetwork, it can be referred to as a visual subnetwork or VSN.

Subnetworks always contain two important modeling aspects: modularity and hierarchy. Each subnetwork encapsulates the data (e.g., subnetwork variables, entity attributes, and parameters) in such a way that a self-contained block/module is created and is able to connect with other subnetworks. Modularity allows for the subnetwork to be reused in different locations within a large network model and to be built for use by other modelers. Moreover, for the hierarchical modeling aspect, entities are transferred from a calling network to a subnetwork. The calling network can be the main network or a subnetwork that is one level higher than the subnetwork that it calls. The hierarchy is also related to the different levels of detail specified in each subnetwork.

When looking at these capabilities, there appear similarities between VSNs and simulation building blocks. The similarities by a means of the object-oriented world view facilitate not only information mapping but also physical and behavioral modeling to develop and use simulation building blocks through VSNs. Therefore, in practice, a VSN can be generated as a model building block, whereas a network node/branch in Visual SLAM network model can be used as a building block element.

Moreover, the construction of VSNs is supported by mechanisms and tools available in the simulation environment, AweSim. As a result, the simulation developers do not need to worry whether the VSNs match with the requirements of their environment. In another case, the simulation developers are also able to customize a VSN in order to function as they require by creating user-written functions (user-codes) and use them via the interface points called EVENT and ENTER nodes. This available feature provides complete modeling flexibility for the configurations of VSNs. The simulation building blocks created in Visual SLAM will be maintained in AweSim's

libraries, which allow the simulation developers to reuse, modify, add, and delete those for the future simulation projects under the same domain. Later, the network libraries become vocabularies that are used only in a domain specific simulation, which automatically creates a simulation environment that supports modeling specific problems.

For the details of the syntax and semantics associated with VSNs, network nodes, and user-written functions, see Simulation with Visual SLAM and AweSim, The User Manual Guide, and Visual SLAM Quick Reference Manual by Pritsker and O' Reilly (1999).

5-3. Concept Implementation

The concepts of ontology mapping, simulation block building, and visual subnetwork modeling are integrated as a paramount approach for the development of a domain specific simulation environment (DSSE) using Visual SLAM simulation modeling language with AweSim mechanisms. This section is focused on the demonstration of implementation of this approach, associating with the previous studies of conceptual simulation modeling (CSM) and transformation of CSM. Moreover, this demonstration still continues using the example of lockage operations found in Chapter 3 and 4, respectively, to close the series of development processes.

Prior to start the demonstration, it must be clear that the key issue of this study is to map the conceptual simulation models collaborated with their contextualized documentation into components available in an existing simulation environment application with respect to simulation requirements and constraints. Therefore, the demonstration includes only the processes of mapping two ontologies and building

simulation blocks by using VSNs. The expectation of this study is to illustrate the process and to obtain a DSSE for lockage (inland waterway) operations.

5-3-1. Mapping CSM with Visual SLAM

As discussed in Subsection 5-2-1, this study employs a similarity-centric approach to perform ontology mapping between conceptualization and simulation. Using this approach, the simulation developers must be able to establish similarities between the source ontology (e.g., CSMs) and the target ontology (e.g., Visual SLAM) and specify a common representation framework/space for mapping these similarities. However, it must be understood by simulation developers that similarity mapping concerns not only the concepts to be mapped but also the structure to be generated (for encapsulating the concepts). The mapping between CSM and Visual SLAM, thus, means to the transformation of both structure and semantics of SMUs into VSNs (as simulation building blocks).

Both SMUs and VSNs are considered as objects having the aspects of modularity and hierarchy, which can be constructed as building blocks at different levels of detail. Therefore, it is critical to limit the detailed levels of their structures before mapping their concepts. Refer to the structure of building blocks, there are only two levels: model building blocks (1^{st} level) and building block elements (2^{nd} level). Technically, model building blocks should be a direct translation of the defined instances from CSM (e.g., SMUs) because the model building blocks represent the world-view of the domain expert in terms of standard functionalities used/reused in reality. Meanwhile, building block elements are deterministically used as either internal or external functionalities of the

model building blocks, providing the semantics of their construction and interconnection to match the user requirements.

When considering and comparing the structures of SMUs and VSNs, it seems possible to perform one-to-one mapping between them to generate model building blocks and building block elements with respect to the levels of detail. In general, an SMU can be viewed as a VSN as well as a model building block, whereas the operations within the SMU can be transformed into a set of Visual SLAM network nodes that perform as building block elements. For the structure mapping, however, it is natural to refine the structure of CSMs by adding details (e.g., tools or structures) for implementation in order to link them together for testing in a host simulation language. In this case, it is not always necessary that the results of mapping between CSMs and Visual SLAM will follow the same pattern previously described. This is because the status of being either a model building block or a building block element of SMUs/VSNs is depended on the correctness of implementation. A set of SMUs are only used as a core design for the development of simulation building blocks, while adding the details for implementation to create VSNs or network nodes is relied on the determination of the simulation developers. The structure mapping, therefore, becomes a more or less abstraction issue for future discussion.

To handle the problem caused by the structure mapping, a critical support role in this similarity-centric approach is taken by ontology mapping. This study is set to apply the framework of MAFRA (Maedche and Staab 2000) to deal with semantic similarities of the concepts/ontologies between two domains. Focusing on similarities of semantics rather than those of structures is helpful for making decision not only in proposing

candidates for mapping but also in adding details for implementation (if necessary). As a result, the simulation developers are able to determine which candidates can be used, what levels of detail (e.g., model building block or building block element) they can be constructed, and how they can be linked together for testing. This provides flexibility in building simulation blocks on the host simulation language like Visual SLAM. However, the MAFRA mapping framework contains some requirements prior to map semantic similarities between ontologies.

The requirements have been described in terms of modules which include lift and normalization, similarity, and semantic bridge. The main objective of these modules is to facilitate defining and establishing a specific framework that fits the surroundings of both source ontology and target ontology. There are three critical conditions to keep in mind. First, both ontologies must be normalized to a uniform representation (or the same representation level). Second, similarities between entities from the source and target ontology must apparently be established. Third, there must be a space for the similarities of two ontologies to be mapped. Following these conditions helps the author to develop a tool, called Similarity Mapping Plane (SMP), to support ontology mapping between conceptualization and simulation.

Prior to exploit SMP, it is important for the simulation developers to be able to specify what source (ontology) to be mapped. On the other hand, it means how to exchange information/data between ontologies. To deal with the information/data exchange, the first step is to categorize the information/data into generic formats – which are object and content formats – since the target of mapping, obviously, is simulation

instances and simulation contents. The object format includes SMUs, VSNs, and network nodes, whereas the content format contains descriptions, properties, and so on.

The next step is to select mapping objects. For example, SMU *Inform Arrival of Barge-tows* is selected as an object to be constructed as a VSN. Meanwhile, its operations such as CreateBargeTow(), AssignBargeTow(), and RouteBargeTow() are also set as objects to be transformed into network nodes (as shown in Figure 5-2). In practice, building a VSN begins with defining and specifying a set of network nodes and all the required parsing parameters. This allows the simulation developers to perform mapping at the second level (operations ↔ network nodes) prior to reach the first level (SMUs ↔ VSNs).

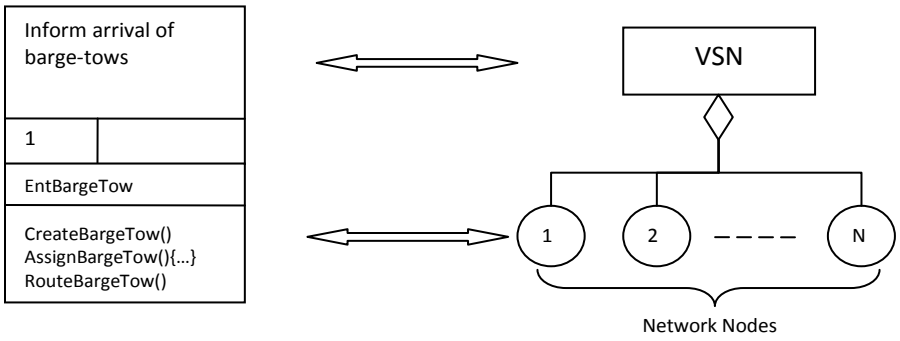

Figure 5 - 2: An example of mapping between an SMU and a VSN

After the mapping objects have been selected, one or more candidates must be nominated from the target of mapping per a selected mapping object by briefly scanning if there is any semantic similarity by a means of functionalities. This would be the easiest way to obtain a number of candidates. It seems to be a time-consuming activity and to require experiencing in the host simulation language – to do manually searching. At this rate of searching, each candidate is placed onto SMP as well as the selected mapping object.

However, both of them cannot be completely mapped until their contents are provided in the same representation level of similarities to be supportive decisions.

In order to normalize the contents of two ontologies into a uniform representation, a framework of similarities must be drawn from the contents available in the target of mapping. The contents of the Visual SLAM network nodes, including Visual SLAM (e.g., support, user-written, and user-callable) functions, are represented in terms of descriptions, statement forms, input listings, and specifications within documentation (as seen in the documents for Visual SLAM and AweSim by Pritsker and O' Reilly). All the contents described in the documentation are determined as entities of the target ontology (as stated by Maedche and Staab 2002). With these entities, the simulation developers are able to establish a scope of the similarity framework to seek for similar entities in the source ontology. However, it needs to be noted here that the representation of those entities in the source ontology must be on the same level as well as documentation.

When considering the source ontology to be mapped, it is found out that the entities retrieved from CSMs are also represented in a context of documentation as well. For example, there appear the entities like description, property, input statement, and function for CreateBargeTow() provided in descriptive tables, network statements, and, especially, contextualized documentation for DMSL: LCK (see Chapter 3 and 4). The entities from the source and target ontology, therefore, become normalized on the documentation basis and ready for selection. The selection of entities is made on the following criteria:

- The entities must clearly represent the main contents of both ontologies.
- The entities must be included in both ontologies and can be matched up directly.

- The entities must provide details that facilitate the simulation developers to measure or weigh the degrees of similarity between entities.

For the above example, there is only one candidate to be mapped with CreateBargeTow(), which is the CREATE node. Table 5-1 shows how to construct SMP and to weigh the degrees of similarity between entities from the source and target ontology.

Apparently, the degrees of similarity can be weighed in a sense of scoring numeric evaluation. The range of evaluation might be varied, depending on individuals' judging criteria and experiences. However, to make this mapping example easy to understand, the evaluation is set at three different degree levels of similarity with numeric scores in (): none (0), likely similar (1), and similar (2). The scores given for each entity are then summed up at the bottom of SMP. The more total score is; the higher possibility of mapping is.

Moreover, the total score can be used as an indicator to consider if the target ontology needs to be added by any other details for implementation. If so, there are three methods for the simulation developers to add those details. The first method is to edit the structure of the target ontology by adding partial specific functionalities to the origin (e.g., modification of the network nodes). The second method is to recreate the target ontology by making a new complete set of specific functionalities (e.g., simulation programming for the functionalities). Finally, the third method is to add one or more extra extensions to the target ontology for being used as supporting roles (or

surroundings). This happens when the target ontology cannot be self-contained enough to responding to the contents of the source ontology to be mapped or to developing itself

Table 5 - 1: Similarity Mapping Plane for CreateBargeTow()

Source	Weight	Target
Instance Name: CreateBargeTow()	1	*Node* Name: CREATE
Description A barge-tow entity is created by a mean of containing a set of barges and a tow boat.	2	*Description* Entities are generated within the network.
Properties :First arrival :Arrival rate :Current time :Max# entities	1	*Inputs* :Time between creations (TBC) :Time of first creation (TF) :Maximum creations (MC) :Mark variable which will store the time of creation (MV) :Number of branches (M)
Input statement CreateBargeTow, First arrival, Arrival rate, Current time, Max# of signal entities;	1	*Input format* CREATE, TBC, TF, MV, MC, M;

Explanation
function CreateBargeTow()
{
var FirstArrival = 0;
var ArrivalRate;
var CurrentTime = TNOW;
var MaxEntities;
 	
//Create a new entity
 	
Set NewEntBargeTow = IP.NewEntity();
Set NewEntBargeTow.ArrivalTime = IP.TNOW;
IP.Schedule("FirstArrival", NewEntBargeTow, (NewEntBargeTow.ArrivalTime+ArrivalRate);
 	
//Schedule the next entities
 	
for (i=1; i<=Max# entities; i++)
Set NextEntBargeTow = IP.CloneEntity();
Set NextEntBargeTow = IP.TNOW;
IP.Schedule("NextArrival", NextEntBargeTow, (NextEntBargeTow.ArrivalTime+ArrivalRate));
 	
}
	1	*Explanation*
		CREATE NODE
		:The first entity is created at a time specified by the value of TF;
		:The time between creations of entities after the first is specified by the variable TBC;
		:The time at which the entity is created can be assigned to a variable MV;
		:Entities will continue to be created until a limit is reached, specified by MC

Total scores	6/10	Likely similar

Weight by degrees of similarity (score): None (0); Likely similar (1); and Similar (2).

into a stand-alone instance (e.g., network node or functional module). It can be seen in many cases that to represent a true semantic of a specific function requires a composition of a set of detailed functions; for instance, a PROCESS function is composed of SEIZE, DELAY, and RELEASE function. Adding the details for implementation, the simulation developers need to closely collaborate with the simulation experts for advising and revising in their works.

In addition to facilitate the similarity mapping, SMP also provides an extra joint entity, called Explanation, to support revealing the true semantics of simulation functionalities for both ontologies. Explanation entity in the source ontology is derived from its contextualized documentation, whereas the one in the target ontology may be retrieved either from the documents for the host simulation language or from the simulation developers' understanding. Since most of the target ontologies are the Visual SLAM network nodes which lack detailed explanation how they function, the simulation developers are required to test each of them to have recognition of their functionalities and usage. As a result, the simulation developers can describe these Visual SLAM network nodes in terms of basic function procedures (or processing steps). For other cases such as having simulation functionalities already described by the documents or specifically created by the simulation developers, they can be directly put onto SMP for comparing similarities with those descriptive functions from the contextualized documentation. Providing the Explanation entity is very helpful not only for completing the ontology mapping between conceptualization and simulation but also for making a final decision whether to utilize the target ontology in developing a simulation building block.

5-3-2. VSNs and DSSE

The ontology mapping between conceptualization and simulation results in providing the definition of simulation building blocks for implementation. These definitions help the simulation developers to specify an implementation framework for considering which simulation functionality (e.g., network node) needs to be modeled and how it can be tested in a demo (simulation) model. Having the implementation framework is to ensure that every time new simulation building blocks being created match the requirements of the simulation environment they are plugged into. It is recommended to apply a black-box approach (Valentin and Verbraeck 2002) to select functionalities and set up a starting environment for testing. This approach is to start implementing the functionalities needed to get a working simulation building block in details with respect to develop a test model. This helps the simulation developers to get insight in the benefits or weaknesses of the simulation building blocks regarding visualization, representation, ease-of-use, output, and use in model development process (Valentin and Verbraeck 2002).

In this research study, the implementation of simulation building blocks is performed in the simulation environment of Visual SLAM and AweSim by using the feature, called visual subnetwork modeling. It allows the simulation developers to implement the definitions of simulation building blocks into the Visual SLAM network elements and visual subnetworks (VSNs). The details of using the network modeling language of Visual SLAM, however, will not be discussed here, so it is important to study Simulation with Visual SLAM and AweSim (Pritsker and O' Reilly 1999) prior to have a better understanding of this demonstration.

Intuitively, the correctness of syntax of the simulation language seems to be the most important issue in developing a VSN. However, this statement is not completely correct. It is critical for the simulation developers to realize that the main purpose of building a VSN as a simulation building block is to deliver the reusability and flexibility in modeling. Moreover, the VSN must be understandable and accessible for both simulation developers and domain experts. It will be useless if the VSN cannot represent the true semantics of functionality corresponding to the real-world process. In reality, there are a set of processes that keep specifically being reused in a domain – which are recognized as standard routines. These standard routines later become callable references used for communication not only in the problem domain but also in the simulation domain. Therefore, each VSN must contain enough information/data to provide a semantic functionality that matches the user requirements.

Figure 5-3 depicts the VSN named SLCK that represents as a model building block to function for the single lockage operation. Operating a single lockage occurs when the number of barges is less than the capacity of the lock, which requires only one lock resource. Within the VSN, as represented as building block elements, a set of the Visual SLAM network nodes are created to convey the semantic of functionality of the single lockage operation, as shown in Figure 5-4.

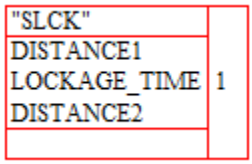

Figure 5 - 3: The VSN named SLCK

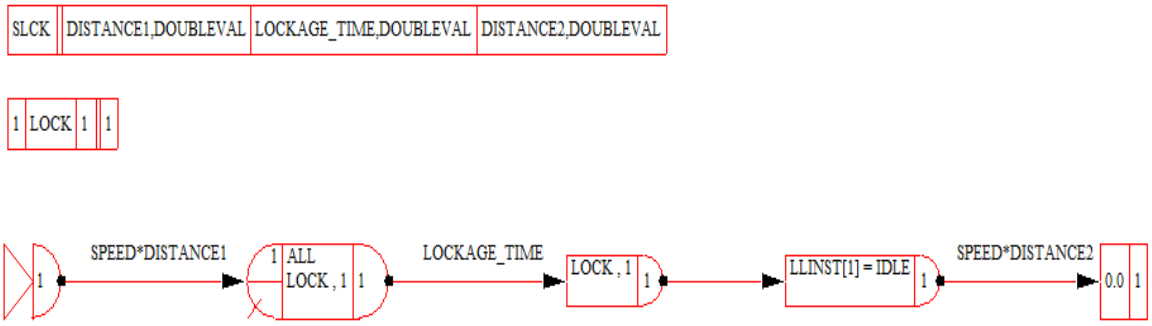

Figure 5 - 4: The Visual SLAM network nodes within the VSN named SLCK

Nevertheless, some restrictions regarding nodes and statements for use within VSNs are imposed in building the VSN (see Pritsker and O' Reilly 1999). This creates the difficulties in encapsulating all the information/data within the VSN as a complete module to function as required. When encountering this kind of situation, the simulation developers are allowed to separate some information/data from the original module and place them aside (or surround) the modified module as the supportive options/extensions. As the result, a combination of the supportive options/extensions and the modified module can be viewed as a big module that still provides the same semantic as well as the original one. Another case is that the original module cannot longer be a module after the information/data have been separated apart. The separated information/data, thus, can be arranged or grouped together as a set of information/data elements to represent the original module instead. This idea not only offers flexibility but also reduces intensity in building VSNs. However, to strengthen the idea, the decomposition and composition approaches need to be strictly performed – to obtain the best combination or the most appropriate set of information/data elements.

An example of the combination of the supportive options/extensions and the modified module is given in Figure 5-5. The original module is expected to represent

SMU *XOR Decide Which Lockage Fits Barge-tows' Size*, which can be replaced by the combination of the Visual SLAM network nodes and a VSN.

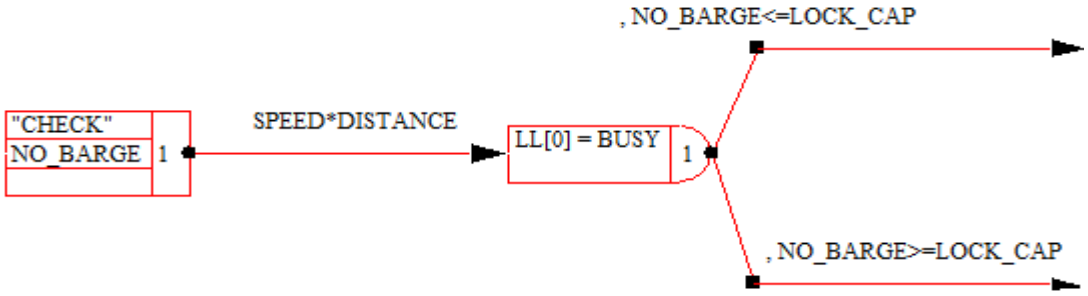

Figure 5 - 5: A combination that represents SMU *XOR Decide Which Lockage Fits Barge-tows' Size*

Another example is given in Figure 5-6 to show a set of the Visual SLAM network nodes that covey the meaning of SMU *Inform Arrival of Barge-tows*.

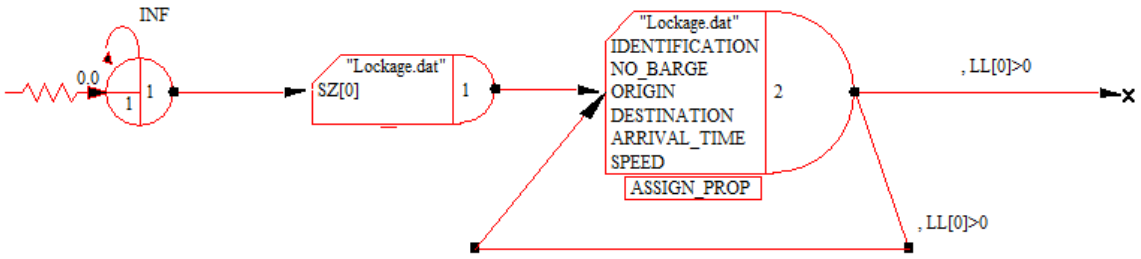

Figure 5 - 6: A set of the Visual SLAM network nodes that represent SMU *Inform Arrival of Barge-tows*

Basically, the VSNs are collected and stored in the AweSim library of subnetworks. The library provides the ability to reuse the VSNs in order to build simulation models to solve other problems within the domain of lockage operation. Moreover, it allows for the simulation developers to add, edit, or delete the VSNs in the library with respect to the requirements of simulation modeling. For a period of time, the collection of VSNs in the library will become vocabularies specifically used to describe

this problem domain in many scenarios related. Figure 5-7 depicts the AweSim library of subneetworks that stores the VSNs created for the lockage operation.

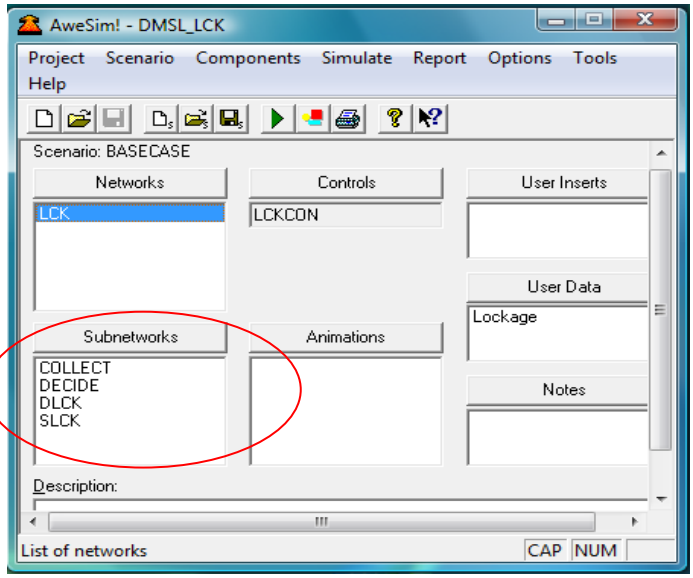

Figure 5 - 7: The AweSim library of subnetworks for DMSL_LCK

Here is a question: How to reuse those combinations and information/data elements in other simulation studies of this domain? It can be seen that the AweSim library of subnetworks is available only for storing the VSNs. To resolve this problem, it is critical for the simulation developers to employ the pattern-based approach. This approach aims to define a framework of reusable solution to a commonly occurring problem in modeling as a pattern. As a result, the simulation developers are able to transform and maintain the combinations and information/data elements as patterns to be reused in many situations. The patterns can be stored in the AweSim library of networks as available networks (not considered as the main networks) for future references, shown in Figure 5-8.

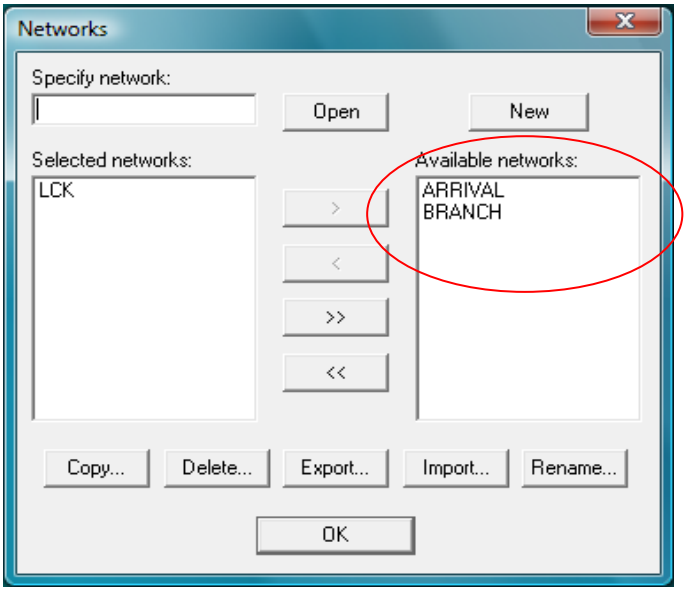

Figure 5 - 8: The AweSim library of networks storing patterns

Having the libraries for reusable VSNs and patterns available for modeling the lockage problem, it leads to a point that the simulation developers are able to establish the AweSim project as a domain specific simulation environment (DSSE). As obviously seen, the DSSE named DMSL_LCK provides a variety of tools for the development of simulation studies related to the lockage operation, facilitated by the Visual SLAM simulation language and AweSim mechanisms. Based on this methodology, each AweSim project can be developed at the level of DSSE for a particular problem domain – as long as it contains enough tools for communication and reuse in modeling simulation for the domain.

5-4. Conclusions

The heart of this research study is to propose the integrated approach that facilitates the development of a domain specific simulation environment (DSSE). Use of the DSSE may be for one time use (application) or multiple uses (language like). The

integrated approach consists of the methodologies of ontology mapping, simulation block building, and visual subnetwork modeling, which also collaborates with the conceptual simulation modeling (CSM) and transformation approach. The idea behind the integrated approach is to design a framework and pattern of mapping between the structures and contents derived from conceptualization and an existing simulation environment application/host simulation language. This brings the simulation developers the abilities to create their own simulation environments to support simulation modeling for a specific problem domain. Not only flexibility in modeling will their DSSEs provide but also reusability in generating simulation models related to the domain.

Nevertheless, the idea of mapping and simulation block building has not received extensive attentions from most simulation language developers. It seems difficult and complicated for them to begin with laying out the structures by using conceptual simulation models, defining and specifying the simulation contents by making contextualized documentation, and encapsulating and mapping those information/data into a module by constructing simulation building blocks. Often, they prefer to use logical models and jump right away to develop simulation models. Programming is also another choice of their preferences for modeling simulation for their particular problems. It might be a comfort zone for them to deal with simulation modeling.

Another reason is that to obtain good production from using the integrated approach is depended on how detailed the source ontology (e.g., CSMs and contextualized documentation) can be and how much expertise in simulation modeling (including simulation environment applications and languages) the simulation developers have. This is a big barrier that not only blocks them from using the approach effectively

and efficiently but also enforces them to deny involving with it at the end. Moreover, there is still room for improvement of the representations and applications to gain more insights and effectiveness to the approach in terms of details for implementation. It is found out that the simulation developers are lost in translation and unable to link the elements for testing. To reach the optimum goal for this research study; therefore, it is critical to have collaborations from individuals in different major areas such as domain experts, simulation experts, software engineers, and computer programmers. The author personally believes that this approach can be developed as a fundamental applied for the Modeling and Simulation (M&S) communities.

5-5. References

1. AweSim: Total Simulation Project Support. 1999. *User's guide*. Version 3.0. Symix Systems.
2. Bouquet, P., B. Magnini, L. Serafini, and S. Zanobini. 2003. A SAT-based algorithm for context matching. In *IV International and Interdisciplinary Conference on Modeling and Using Context*, Standford University, California, USA.
3. Breis, J. F. and R. M. Bejar. 2002. A cooperative framework for integrating ontologies. *International Journal of Human-Computer Studies*, 46(6), 707-728.
4. Doan, A., J. Madhavan, P. Domingos, and A. Halevy. 2002. Learning to map between ontologies on the semantic web. In *Proceedings of WWW-2002, 11th International WWW Conference*, Hawaii.

5. Ehrig, M. and S. Staab. 2004. QOM – Quick Ontology Mapping. In *International Semantic Web Conference*, 683-697.
6. Ehrig, M. and Y. Sure. 2004. Ontology mapping – an integrated approach. In *ESWS 2004*, 76-91.
7. Gruber, T. R. 1993. Towards principles for the design of ontologies used for knowledge sharing. In *Formal Ontology n Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, Kluwer Acadmic Publishers.
8. Kalfoglou, Y. and M. Schorlemmer. 2003. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18(1), 1-31.
9. Kent, R. 2000. The information flow foundation for conceptual knowledge organization. In *Proceedings of the 6th International Conference of the International Society for Knowledge Organization (ISKO)*, Toronto, Canada.
10. Madhavan, J., P. A. Bernstein, P. Domingos, and A. Halevy. 2002. Representing and reasoning about mappings between domain models. In *Proceedings of the 18th National Conference on Artificial Intelligence*, Edmonton, Alberta, Canada.
11. Maedche,, A. and S. Staab. 2000. Semi-automatic engineering of ontologies from texts. In *Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering*, Chicago, IL, USA.
12. Maedche, A., B. Motik, N. Silva, and R. Volz. 2002. Mafra: A mapping framework for distributed ontologies. In *Proceedings of the 13th European Conference and Knowledge Engineering and Knowledge Management*, Madrid, Spain.

13. Marques, D. 2005. A survey of recent research in ontology mapping. Available via <http://sfu.ca/~mhatala/iat881/2005/DM-OntologyMapping.pdf> [accessed October 5, 2008].

14. Navigli, R., P. Velardi, and A. Gangemi. 2003. Ontology learning and its application to automated terminology translation. *IEEE Intelligent Systems*, 18(1), 22-31.

15. Pritsker, A. A. B. and J. J. O'Reilly. 1999. *Simulation with Visual SLAM and AweSim*. 2nd ed. New York: John Wiley & Sons.

16. Rahm, A. and A. Bernstein. 2001. A survey of approaches to automatic schema matching. *The Very Large Databases Journal*, 10(4), 334-350.

17. Saanen, Y. A. 2002. Chapter16: The application of advanced simulations for the engineering of logistic control systems. In A. Verbraeck, A. Dahanayake (Eds.). *Building blocks for effective telematics application development and evaluation*. Delft University of Technology, Natherlands.

18. Setavoraphan, K. and F. H. Grant. 2008. Conceptual simulation modeling: The structure of domain specific simulation environment. In *Proceedings of the 2008 Winter Simulation Conference.* pp. 975-986.

19. Snowdon, J. L., S. El-Taji, M. Montevecchi, E. MacNair, C. A. Callery, and S. Miller. 1998. In *Proceedings of the 1998 Winter Simulation Conference*.

20. Valentin, E. C. 2002. Chapter15: Building blocks for modeling of passengers at airports. *Building blocks for effective telematics application development and evaluation*. Delft University of Technology, Netherlands.

21. Valentin, E. C. and A. Verbraeck. 2002. Guidelines for designing simulation building blocks. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yucesan, C. –H. Chen, J. L. Snowdon, and J. M. Charnes. pp. 563-571. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
22. Verbraeck, A. and E. Valentin. 2001. The use of building blocks to enhance flexibility and maintainability of simulation models and simulation libraries. In: N. Giambiasi, C. Frydman (Eds.), *Proceedings ESS'2001 – 13th European Simulation Symposium 2001*, pp. 973-979.
23. Verbraeck, A. and E. Valentin. 2002. Simulation building blocks for airport terminal modeling. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yucesan, C. –H. Chen, J. L. Snowdon, and J. M. Charnes. pp. 563-571. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
24. Verbraeck, A. , Y. Saanen, Z. Stojanovic, E. Valentin, K. van der Meer, A. Meijer, and B. Shishkov. 2002. Chapter 2: What are building blocks?. In A. Verbraeck, A. Dahanayake (Eds.). *Building blocks for effective telematics application development and evaluation*. Delft University of Technology, ISBN 90-5638-092-3. pp. 8-21.
25. Verbraeck, A. and C. Versteegt. 2000. A bridge between the design and implementation of complex transportation systems – Linking simulation models and physical systems. In: Dietmar P. F. Moller (Ed.), *ESS 2000 – Simulation in Industry*, pp. 238-243 Ghent: SCS Publications.
26. Visual SLAM. 1997. *Quick reference manual.* Version 2.0. Pritsker Corporation.

27. Visser, P. R. S., D. M. Jones, T. J. M. Bench-Capon, and M. J. R. Shave. 1997. An analysis of ontology mismatches: Heterogeneity versus interoperability. In *AAAI 1997 Spring Symposium on Ontological Engineering*, Stanford, CA, USA.

CHAPTER 6

Case Study: DSSE Development Illustration

Abstract

This chapter is aimed at illustrating the methodology developed for developing domain specific simulation environments (DSSEs). The methodology is an integration of conceptual simulation modeling (CSM), contextualized documenting, ontology mapping, simulation block building, and visual subnetwork modeling. To illustrate the use of the methodology, the application of lockage operations on an inland waterway network on the McClellan-Kerr Arkansas River is taken as a case study.

6-1. Introduction

There have been a number of simulation models developed to provide a variety of scenarios and results for the analysis of lockage operations on inland waterway navigation systems. However, none of them can be reused in other simulation studies, though, they are in the same domain of interest. A lack of reusability in simulation modeling leads to higher cost and more time when conducting a simulation study. To solve this problem, we critically consider the approach of domain specific simulation environments (DSSEs). The main purpose of the approach is to create a simulation environment that is able to provide reusable and accessible tools (or structures) to facilitate the development of simulation studies for a specific problem domain. To develop such a DSSE, it requires using concepts, approaches, and applications related to simulation and modeling. Here have been numerous attempts by researchers and

simulation developers/modelers to develop methodologies and tools to support the construction of DSSEs, giving alternative outcomes – depended on purposes, requirements, and perspectives of the users.

In this study, the implementation of the methodology developed by the author is applied in an application of lockage operations on an inland waterway system on the McClellan-Kerr Arkansas River to illustrate how effective it will be in a simulation practice. This methodology encourages a simulation developer to conduct the development of a DSSE with recognition of structures, contents, and simulation environment application. Therefore, to apply this methodology, the simulation developer needs to complete three phases which are conceptualization, transformation, and mapping, to accomplish the development.

This case study is focused only on a partial segment of the inland waterway network on the McClellan-Kerr Arkansas River to minimize the size of model and explanation. Also, we have attempted to make the demonstration as direct as possible to illustrate the procedures. We also assume some knowledge of Visual SLAM and Awesim.

6-2. Problem Description

This problem statement is taken from the US Army Corps of Engineers (2008). The McClellan-Kerr Arkansas River Navigation System (MKARNS) is reliable, year-round waterway into the Southwest. On this 445-mile long waterway, there is a series of navigation pools connected by 18 locks and dams to enable vessels to overcome a 420-foot difference in elevation from the Mississippi River to the head of navigation at

Catoosa, Oklahoma. The MKARNS was designed for ease of navigation by multi-barge tows, with ample channel and lock dimensions and bridge clearances, where the locks and dams are operated 24 hours a day by the Corps of Engineers. Figure 6-1 shows the locations of the locks and dams on the MKARNS.

The average size of locks is 110 ft. x 600 ft., which can accommodate eight jumbo barges without double lockage. If there are more than eight barges in the group, double lockage with tow haulage is needed. Tow haulage is a procedure for drawing barges through a lock by using equipment (e.g., wrench) on the lock itself to minimize the maneuvering of a towboat when a tow exceeds the length of the lock. (Note: see the detailed descriptions of single and double lockage in Chapter 3).

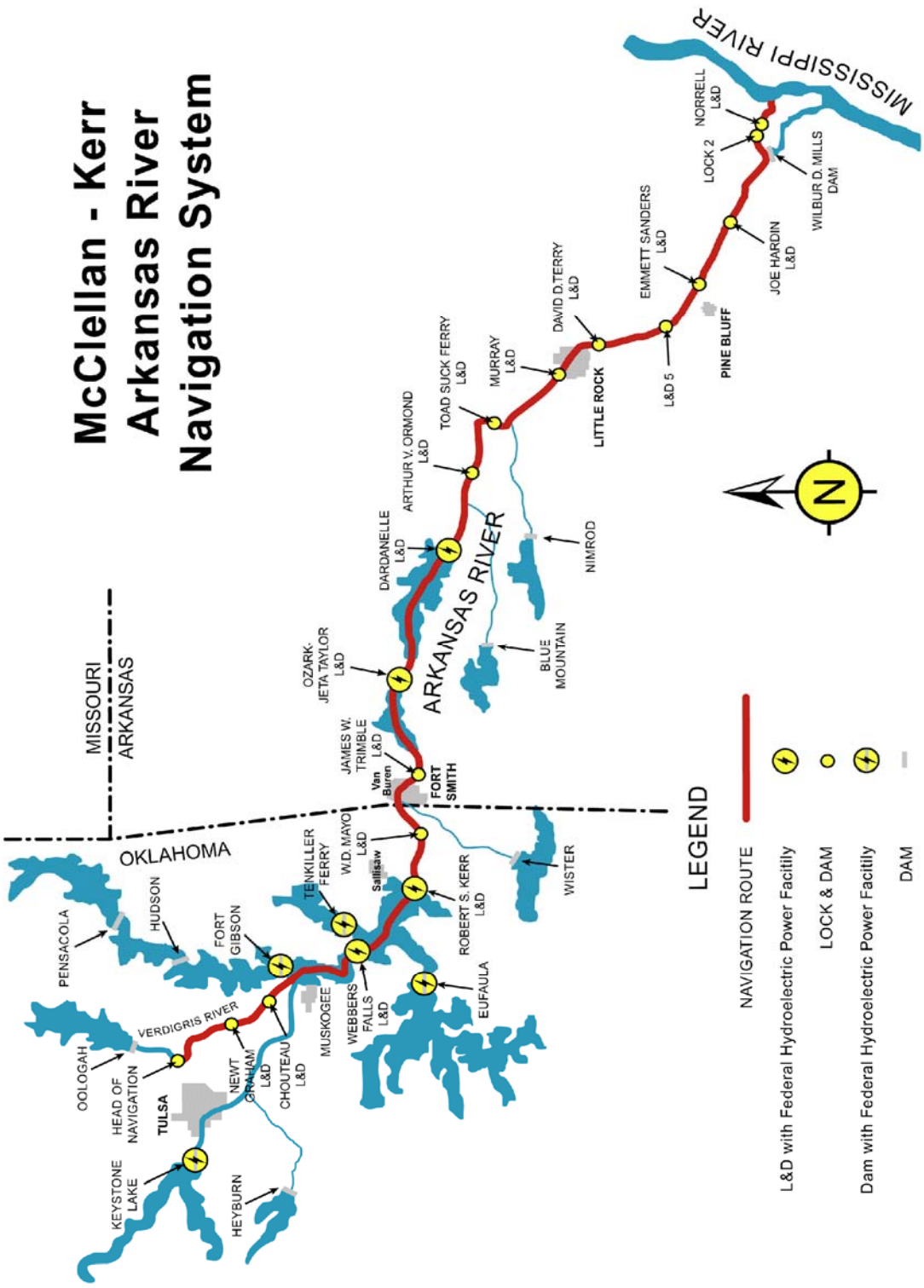

Figure 6 - 1: Locations of the locks on the MKARNS (U.S. Army Corps of Engineers 2008)

In this case study, the focus is on the general lockage operations that service barge-tows travelling either upstream or downstream through the locks between Port of Muskogee (downstream) and Tulsa Port of Catoosa (upstream) located in Oklahoma. Generally, several barge-tows are found to have traveled at a maximum speed of 15 mph and a top actual speed of around 7 mph along the river, whereas the average entrance speed is around of 4 mph. There are two locks: Chouteau (#17) and Newt Graham (#18) between these ports, with the in-between distance of around 7.6, 20.2, and 23.4 miles, respectively (the US Army Corps of Engineers 2008). The average time of single lockage operation is 10-15 minutes, while using tow haulage approximately takes 15-25 minutes to complete. It is desired to simulate the operations of the locks for one-day period (24 hours) to obtain the average time in system and the average waiting time of each barge-tow, including the average utilization of the locks.

6-3. Methodology

This section represents how to develop a DSSE for the lockage operations on the MKARNS by following the three-phase-design approach. Phase 1 is to develop conceptual simulation models (CSMs) using ISAP to generate a blueprint of the overall structure of the DSSE. The process of transformation of the CSMs will be illustrated in Phase 2. Finally, in Phase 3, the mapping between conceptualization and simulation will be taken to develop simulation building blocks using Visual SLAM and AweSim. These three phases are not necessarily executed independently. In practice, there will be significant overlaps and iterative feedback loops in the modeling and simulation process,

which requires the simulation developers to solve different types of problem between these phases.

6-3-1. Phase 1: Conceptualization of Problem Domain

Prior to develop a DSSE for the lockage operations on the MKARNS, the structure describing the physical and behavioral characteristics of the target domain is needed. To obtain such an accurate structure of the domain, it is critical to limit and describe how things work and what is to be solved within the domain by starting with a problem domain (Valentin and Verbraeck 2002). To formulate the problem domain, the simulation developers must be able to understand the problem context, set specific modeling objectives, and define the system to be modeled. Problem descriptions, system boundaries and components, and desired results are the outcomes of the formulation of the problem domain, which is considered as conceptualization. In this case study, Integrated Simulation Acknowledge Procedure (ISAP) is taken to support the conceptualization of the problem domain to generate conceptual simulation models (CSMs).

According to Setavoraphan and Grant (2008), ISAP consists of three layers: Initialization Layer (IL), Process Layer (PL), and Termination Layer (TL), which is developed through three phases based on the simulation and modeling design approach. The results of using ISAP are represented in the following CSMs according to each layer.

6-3-1-1. Initialization Layer (IL)

In IL, initial information about the simulation experiment to be performed such as number of simulation runs, number of attributes/variables, and time to begin/end simulation are specified. This is a process to design and define a set of parameterized references whose settings can be modified per experimentation. Figure 6-2 illustrates the design of IL: Lock.

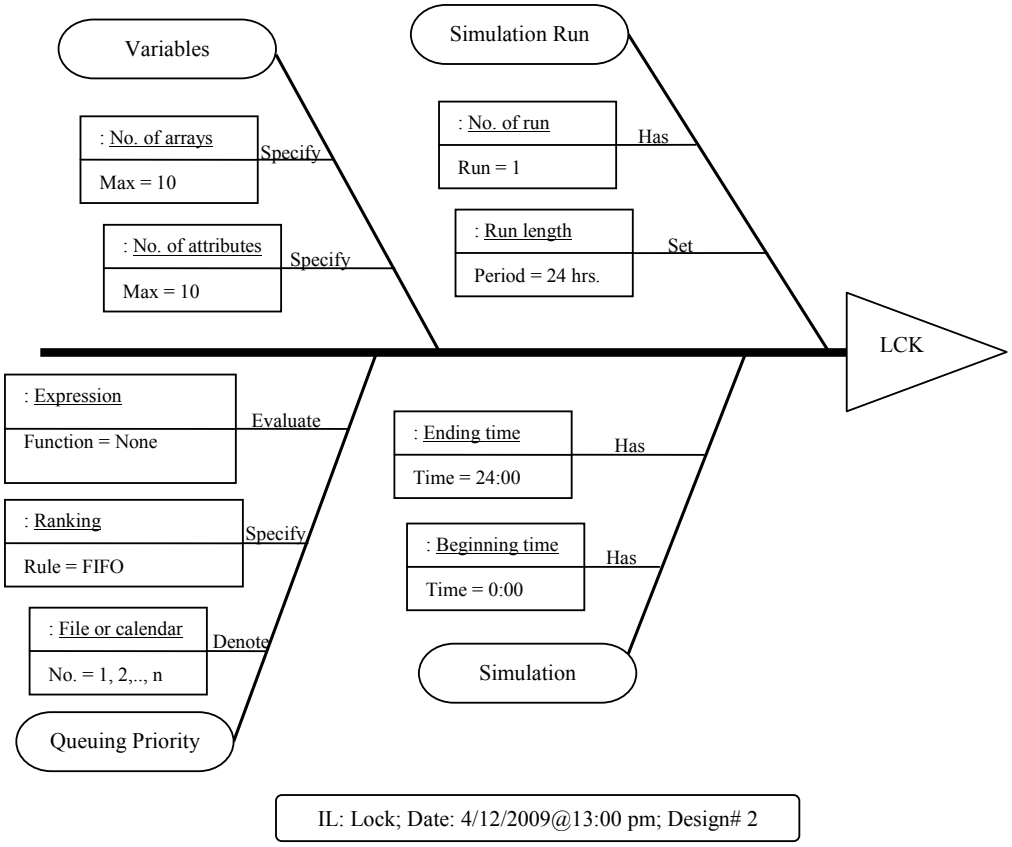

Figure 6 - 2: A diagram representing initialization of Lock

Creating an IL seems to be difficult for a beginner to realize what to do and what to input. The best suggestion is that he/she should find out which is the destination for his/her DSSE to be built – a host simulation language or a simulation environment application. In this case study, AweSim is our target, so we can deploy its structure and input parameters as references for designing a parameterized framework. To avoid using irrelevant parameters, filter and arrange them in categories that positively impact overall experimentation and construction of his/her DSSE. Organize them within a diagram that well conveys information to an individual's perception in a way of a mind-map. Finally, revise and update the diagram to response to correctness and modification of parameters. After a few trials, IL can be designed on his/her own purpose, giving better initial parameters that help specify and sharpen the scope of modeling.

6-3-1-2. Termination Layer (TL)

TL is aimed to provide the setting procedures of terminating simulation and printing out a simulation output report, which specifies a frame of reference for parameterization and termination of simulation. The frame of reference can be portrayed in a tabular-cell pattern that contains a set of data fields and information. Table 6-1 shows the data fields for parameterization with assignments required for this case study.

Table 6 - 1: Description of data fields for parameterization with assignments

	Fields*					
1	2	3	4	5	6	7
TIS	Arrival time of entity ATRIB[1]	Departure time of entity TNOW				
LUT	Lock number	Lock busy time {0}	Lock idle time {0}	Total simulation run {∞}		
WAT	Queue number {0}	Arrival time of entity at queue	Departure time of entity from queue TNOW			
TER	Maximum entities {∞}	Time limit {∞}				

Where,

TIS = Time in system of each barge-tow;

LUT = Utilization of each lock;

WAT = Waiting time of each barge-tow;

TER = Termination of simulation;

The beginner can follow the guidelines previously mentioned in IL. Obtaining a good list of output parameters and terminating criteria is depended upon how well his/her modeling objectives are set. This requires not only the details of information from the domain problems but also the individual's experience in modeling. This means that good communication between domain experts and simulation developers are critical for exchange of information; however, the responsibility in translating the information into simulation requirements belongs to the simulation developers. Therefore, the

relationships among inputs, a simulation system, and outputs must be drawn as a big picture to support the individual's understanding in their effects and decision making for the selection of appropriate parameters.

6-3-1-3. Process Layer (PL)

PL becomes the most critical part of ISAP because the physical and structural characteristics of the problem domain and simulation domain are formulated here to generate the core structure of the DSSE, constructed under two different modeling subsystems. The physical characteristics are described in the static modeling subsystem, whereas the behavioral characteristics are represented in the dynamic modeling subsystem. According to Figure 6-1, the static modeling subsystem representing the physical layout of the locks and ports on the MKARNS can be created as shown in Figure 6-3.

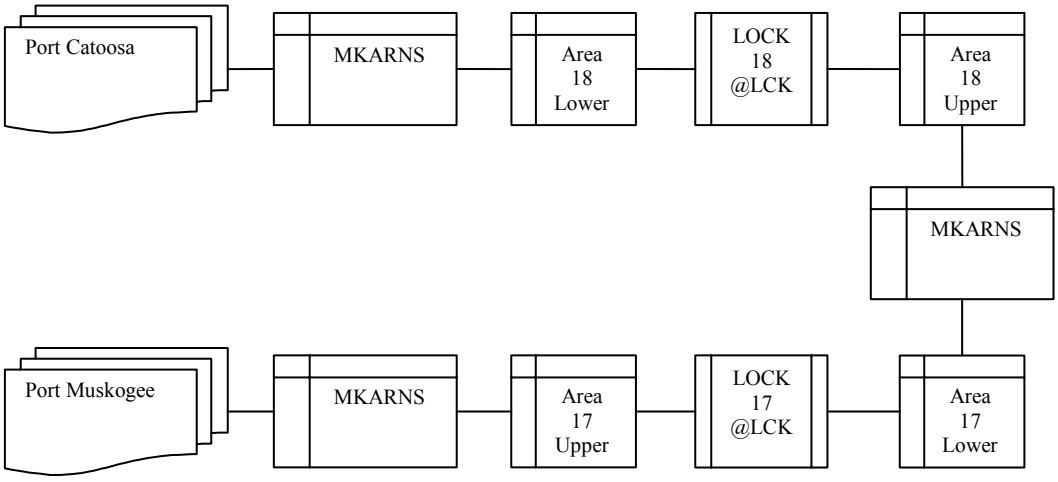

Figure 6 - 3: A static modeling subsystem of the problem domain

The static modeling subsystem provides an insight to determine which segment of the physical layout needs to be processed. As seen in Figure 6-3, only at LOCK 17 and LOCK 18 are the processes taken place for representing the lockage operations. It is assumed that the same processes are operated at these two locks. The next step is to specify and describe the lockage operations in a dynamic modeling subsystem to represent, for example, relationships, attributes, and dynamic flows of those processes.

The beginner can start with drawing a logical flow diagram to help organize ideas, concepts, and information into a pattern of descriptive processes. Later, add details, e.g., entities, resources, sub-processes, and attributes, as needed to the logical flow diagram to provide a better understanding of the processes. Revise it a few times before transforming it into a dynamic modeling subsystem diagram. A good diagram should help the individual visualize what happens in those dynamic flows of the processes. Losing a focus in the details causes difficulties in translating and mapping conceptual simulation models in later states.

There is no restriction in using tools or software applications for developing dynamic modeling subsystem diagrams, including other diagrams shown in this research study. Microsoft Words, for example, might be convenient for many individuals but not for everyone. Therefore, it needs to ensure that creating a diagram is not an obstacle in using the integrated methodology. The following figures illustrate the dynamic modeling subsystem, DMSL: LCK, built for this case study.

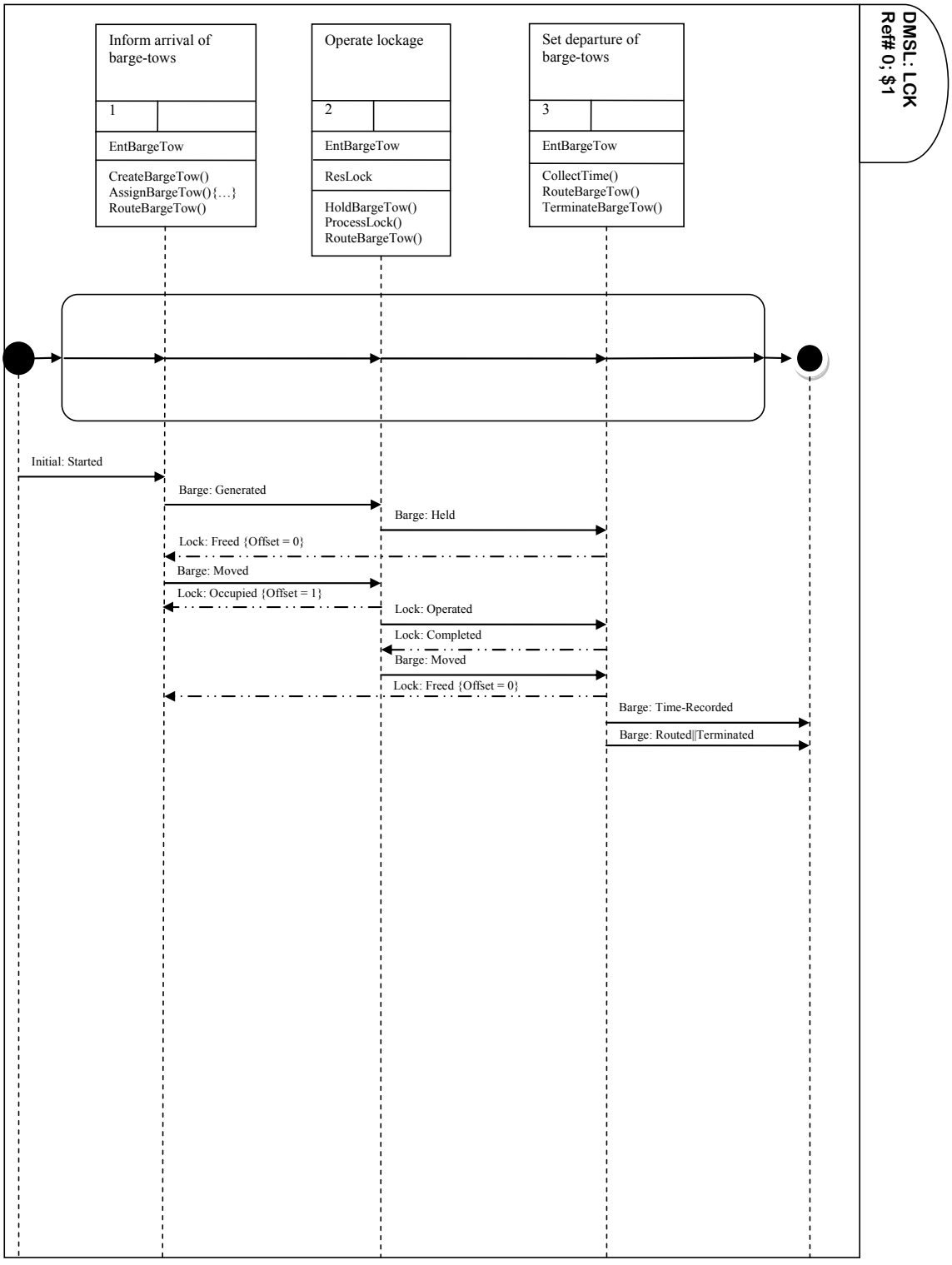

Figure 6 - 4: DMSL: LCK; sub-folder# 0; page#1

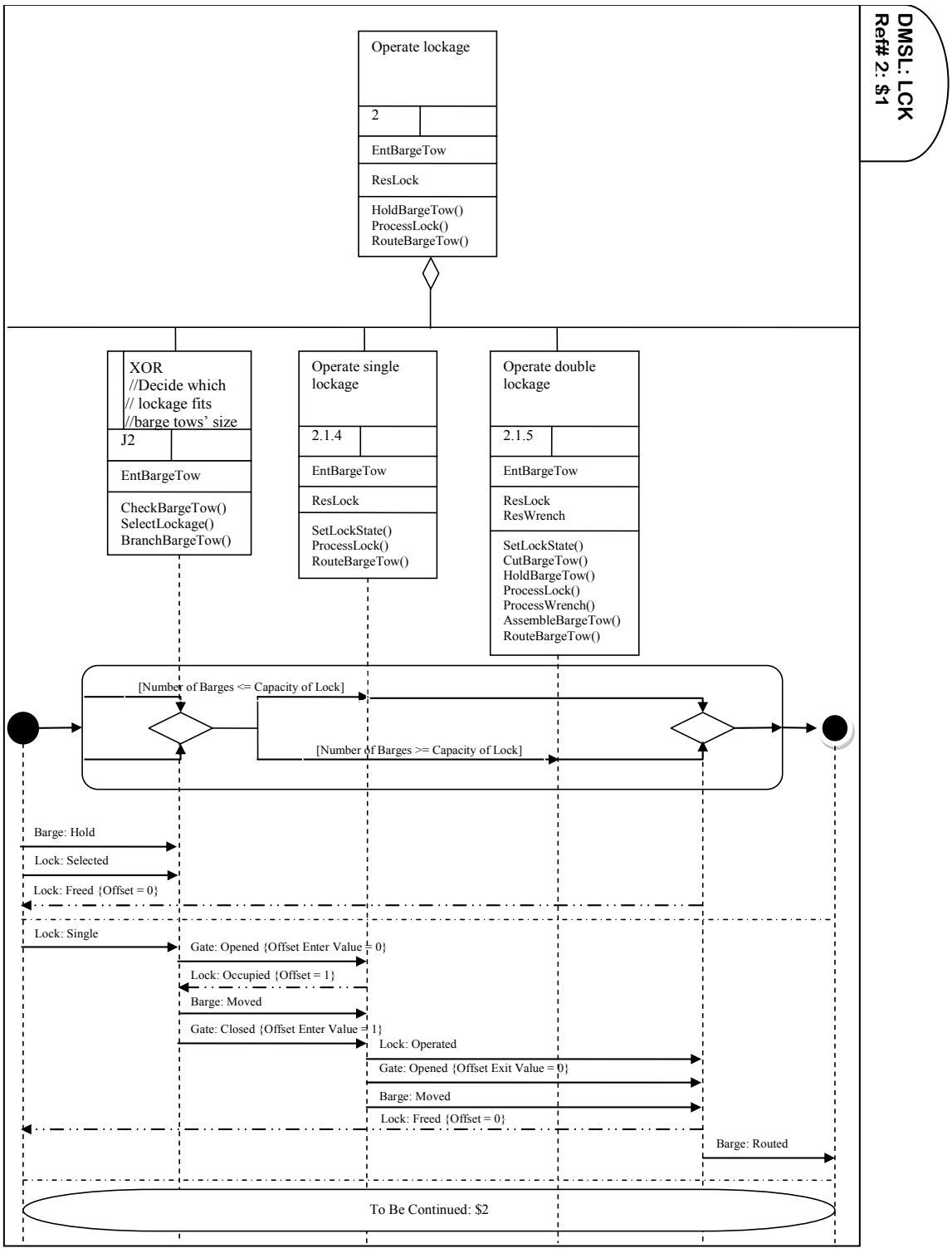

Figure 6 - 5: DMSL: LCK; sub-folder# 2; page# 1

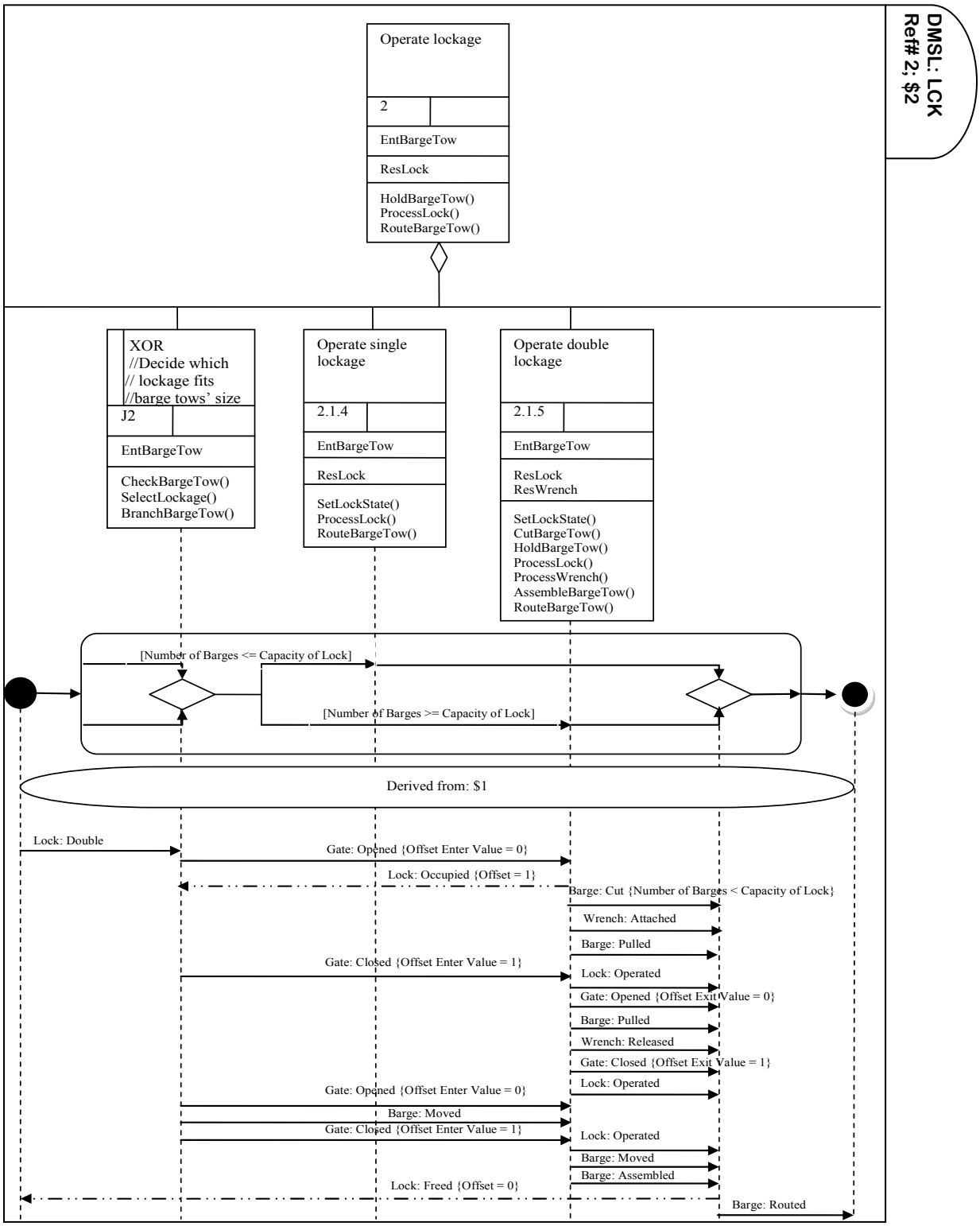

Figure 6 - 6: DMSL: LCK; sub-folder# 2; page# 2

Following the figures, tables of descriptions of the objects and operations are provided to support understanding of DMSL: LCK, as shown in Table 6-2, 6-3, and 6-4, respectively.

Table 6 - 2: Description of Objects for DMSL: LCK

Object Name	Type	Description	Parameters
EntBargeTow	Entity	A barge-tow is represented as a target entity to be observed in the inland waterway system. A barge-tow entity consists of a set of barges and a tow boat.	: Identification# : Number of barges : Origin : Destination : Arrival time : Speed
ResLock	Resource	A lock is a resource that takes an action in raising or lowering barge-tow entities by filling or draining water. Also, the lock-enter allowance is controlled by its gates. The gates can be determined as internal or external resources.	: Name : File# : Resource# : Capacity : Activity time
ResWrench	Resource	An electric wrench is a resource used to pull a section of barges that are cut for the first lockage.	: File# : Resource# : Capacity : Activity time

Table 6 - 3: Description of Operations for DMSL: LCK

Operation Name	Type	Actor	Description	Attributes	Global Variables
AssembleBargeTow()	General/ Extended	EntBargeTow	An action is to accumulate one or more set of barges that are cut with a tow boat into a single entity	: Identical values : Number of barges	
AssignBargeTow() {…}	General	EntBargeTow	A simulation action is to assign identical attributes to define the characteristics of each barge-tow entity that represent a set of barges and a tow boat.	: Identification# : Number of barges : Origin : Destination :Arrival time : Speed	
BranchBargeTow()	General/ Extended	EntBargeTow	A number of branches are provided at a location for an entity to take upon conditions or probabilities		: Condition expression
CreateBargeTow()	General/ Extended	EntBargeTow	A barge-tow entity is created by a mean of containing a set of barges and a tow boat.	: First arrival : Arrival rate :Current time : Max# entities	
CollectTime()	General	EntBargeTow	Statistical data of time spent in the system are collected	: Travel time	: ID : Label

Table 6 - 4: Description of Operations for DMSL: LCK (Cont.)

Operation Name	Type	Actor	Description	Attributes	Global Variables
CheckBargeTow()	Extended	EntBargeTow	An action is to check how many barges the entity is containing to make a decision for selecting a lockage type.	: Number of barges	
CutBargeTow()	Extended/ Specific	EntBargeTow	An action is to split a specific number of barges that are allowed to enter a lock. There are many ways to cut, upon policies and sizes of each lock	: Identical batch size : Number of barges	
HoldBargeTow()	General/ Extended	ResLock	"Hold" can be determined as an action to control the flow of entities.		: Delay time
ProcessLock()	General	ResLock	An action is taken at a lock by a mean of delay-activity time.	: Resource# : Capacity of lock	: Activity time
ProcessWrench()	General	ResWrench	Electric wrench is used when a double lockage is required.	: Resource#	: Activity time
SelectLockage()	Extended	ResLock	A decision-making action is to select either single or double lockage configuration upon the sizes of the barge-tow entities	: Resource# : Capacity of lock	
SetLockState()	Extended	ResLock	An action (of sending a signal) verifies a status of the lock (e.g., busy or idle)	: Resource#	: Offset value
RouteBargeTow()	General	EntBargeTow	Each barge-tow entity is routed or moved through the system on designated routes. Delay time might be specified on each route.		: Distance
TerminateBargeTow()	General	EntBargeTow	Each barge-tow entity is terminated when it leaves the system		

Descriptions should be concise enough to specify meanings and purposes of use of the objects and operations identified in the dynamic modeling subsystem diagrams. Also, labeling an object/operation should be meaningful and consistent so that it will not create any conflicts when used in transformational documentation.

The next step is to translate DMSL: LCK into a network statement to increase the readability of the process layer's representations, as shown below:

DMSL: LCK; Ref# 0 – 2:

1 CreateBargeTow, First arrival, Arrival rate, Current time, Max# of entities;
2 AssignBargeTow, Identification#, Number of barges, Origin, Destination, Arrival time, Speed;
3 RouteBargeTow, Distance;
4 CheckBargeTow, Number of barges;
5 SelectLockage, Resource#, Capacity of lock;
6 SetLockState, Rerouce#, Offset value;
7 BranchBargeTow, Condition expression;
8 Condition, Number of barges <= Capacity of lock;
9 SetLockState, Resource#, Offset value;
10 RouteBargeTow, Distance;
11 ProcessLock, Resource#, Capacity of lock, Activity time;
12 RouteBargeTow, Distance;
13 SetLockState, Resource#, Offset value;
14 Condition, Number of barges >= Capacity of lock;
15 SetLockState, Resource#, Offset value;
16 CutBargeTow, Identical batch size, Number of barges;
17 HoldBargeTow, Delay time;
18 ProcessWrench, Resource#, Activity time;
19 ProcessLock, Resource#, Capacity of lock, Activity time;
20 RouteBargeTow, Distance;
21 SetLockState, Resource#, Offset value;
22 RouteBargeTow, Distance;
23 SetLockState, Resource#, Offset value;
24 ProcessLock, Resource#, Capacity of lock, Activity time;
25 RouteBargeTow, Distance;
26 AssembleBargeTow, Identical batchsize, Number of barges;
27 SetLockState, Resource#, Offset value;
28 RouteBargeTow, Distance;
29 CollectTime, Travel time, ID, Label;
30 RouteBargeTow, Distance;
31 TerminateBargeTow;

6-3-2. Phase 2: Transformation of Conceptual Simulation Models

This phase is to transform the CSMs into contextualized documentation, so that their semantics of structural and behavioral contents within a simulation context can be represented in a more executable-readable form. The contextualized documentation is developed by using the Semantic Web technologies such as XML and SRML. Moreover, user-callable functions are created to support the explanation of the documentation, as shown in Table 6-5.

Table 6 - 5: Referenced properties and callable functions for DMSL: LCK

References	Description
NewEntity()	Create a new entity.
CurrentEntity()	Return the current entity.
CloneEntity()	Clone the entity.
TerminateEntity()	Terminate the entity.
Release(Resource#, Units)	Release number of units of the resource#.
Seize(Resource#, Units)	Allocate number of units of the resource#.
NARES(Resource#)	Return the number of available units of the resource#.
NIUSE(Resource#)	Return the number of busy units of the resource#.
Resource()	Allocate a resource and assign its calling number
Schedule(Event, Entity, Time)	Schedule an event of type Event to occur at time TNOW + Time for the current entity.
Assign(Attribute 1, Attribute 2, …)	Assign one or more attributes to the entity.
LocateEntity(Event, Resource, Entity)	Locate the entity in the target resource
Intlc(run)	Check the initial run
TNOW	Current simulated time

From DMSL: LCK, the contextualized documentation can be generated as shown below:

```
<DMSL Name ="LCK">
  <Variable varname="Offset" vartype="boolean"/>
  <Variable varname="Offset enter value" vartype="boolean"/>
  <Variable varname="Offset exit value" vartype="boolean"/>
  <Script Type="text/javascript">
    <![CDATA[

    //Initialize variables for the first run

    function Initial()
    {
      Intlc(run);//Check the initial run
      if (run = 1)
        {
          var Offset = 0;
```

```
          var Offset enter value = 0;
          var Offset exit value = 0;
        }
     }
      ]]>
  </Script>

  <SMU Name = "Inform arrival of barge-tows">
    <Entity Name = "EntBargeTow">
      <Attribute atribname="Identification#" atribtype="interger"/>
      <Attribute atribname="NumberBarges" atribtype="integer"/>
      <Attribute atribname="Origin" atribtype="integer"/>
      <Attribute atribname="Destination" atribtype="integer"/>
      <Attribute atribname="ArrivalTime" atribtype="real"/>
      <Attribute atribname="Speed" atribtype="real">
      </Entity>

    <Script Type="text/javascript">
      <![CDATA[

    function CreateBargeTow()//Create and schedule entities
    {
      //Define variables used in this function
      var FirstArrival = 0;
      var ArrivalRate;
      var CurrentTime = TNOW;
      var MaxEntities;

      //Create a new entity
      set NewEntBargeTow = IP.NewEntity();
      set NewEntBargeTow.ArrivalTime = IP.TNOW;
      IP.Schedule("FirstArrival", NewEntBargeTow,
(NewEntBargeTow.ArrivalTime+ArrivalRate);

      //Schedule the next entities
      for (i=1; i<=Max# entities; i++)
        {
          set NextEntBargeTow = IP.CloneEntity();
          set NextEntBargeTow = IP.TNOW;
          IP.Schedule("NextArrival", NextEntBargeTow,
(NextEntBargeTow.ArrivalTime+ArrivalRate));
        }
    }

    function AssignBargeTow()//Assign attributes to the BargeTow
entities
    {
      var Identification#;
      var NumberBarges;
      var Origin;
      var Destination;
      var ArrivalTime;
      var Speed;

      //Define the current EntBargeTow entity and assign attributes to
it
      set CurrentEntBargeTow = IP.CurrentEntity();
```

```
      CurrentEntBargeTow.Assign(Identification#, NumberBarges, Origin,
Destination, ArrivalTime, Speed);
    }

    function RouteBargeTow()//Schedule the current EntBargeTow entity
for travelling
    {
      var Distance;
      var Speed;
      var DelayTime = Distance/Speed;
      IP.Schedule("Decision", CurrentEntBargeTow,
CurrentEntBargeTow.DelayTime);
    }
      ]]>
   </Script>

  <Link Name="J2" Type="Precedence">
  <Link Target="XOR: Decide which lockage fits barge tows' size">
  </Link>
  </SMU>

  <SMU Name ="XOR: Decide which lockage fits barge tows' size">
    <Entity Name ="EntBargeTow">
     <Attribute atribname="Identification#" atribtype="interger"/>
     <Attribute atribname="NumberBarges" atribtype="integer"/>
     <Attribute atribname="Origin" atribtype="integer"/>
     <Attribute atribname="Destination" atribtype="integer"/>
     <Attribute atribname="ArrivalTime" atribtype="real"/>
     <Attribute atribname="Speed" atribtype="real">
    </Entity>

    <Resource Name ="ResLock">
     <Attribute atribname="Name" atribtype="string"/>
     <Attribute atribname="File#" atribtype="integer"/>
     <Attribute atribname="Resource#" atribtype="integer"/>
     <Attribute atribname="CapacityLock" atribtype="integer"/>
     <Attribute atribname="ActivityTime" atribtype="real"/>
    </Resource>
    <Script Type="text/javascript">
    <![CDATA[

    function CheckBargeTow()//Retrieve the value of number of barges
from the current EntBargeTow entity
    {
      var NumberBarges;
      set CheckNumberBarges = CurrentEntBargeTow.NumberBarges;
    }

    function SelectLockage()//Retrieve the capacity value from the lock
Resource#
    {
      var LockCapacity;
      set ResLock = IP.Resource();
      set LockCapacity = ResLock.CapacityLock;
     }

    function BranchBargeTow()
```

```
    {
      var LockCapacity;
      if (CheckNumberBarges <= LockCapacity)
      IP.LocateEntity("Operate single lockage", ResLock,
CurrentEntBargeTow);
      else if (CheckNumberBarges > LockCapacity)
      IP.LocateEntity("Operate double lockage", ResLock,
CurrentEntBargeTow);
    }

    ]]>
    </Script>

   <Link Name="LockType" Type="Precedence with condition(s)">
   <Link Target="Operate single lockage"/>
   <Link Target="Operate double lockage"/>
   </Link>
   </SMU>

   <SMU Name ="Operate single lockage">
    <Entity Name ="EntBargeTow">
      <Attribute atribname="Identification#" atribtype="interger"/>
      <Attribute atribname="NumberBarges" atribtype="integer"/>
      <Attribute atribname="Origin" atribtype="integer"/>
      <Attribute atribname="Destination" atribtype="integer"/>
      <Attribute atribname="ArrivalTime" atribtype="real"/>
      <Attribute atribname="Speed" atribtype="real">
      </Entity>

    <Resource Name ="ResLock">
      <Attribute atribname="Name" atribtype="string"/>
      <Attribute atribname="File#" atribtype="integer"/>
      <Attribute atribname="Resource#" atribtype="integer"/>
      <Attribute atribname="CapacityLock" atribtype="integer"/>
      <Attribute atribname="ActivityTime" atribtype="real"/>
    </Resource>

    <Script Type="text/javascript">
     <![CDATA[
    function SetLockState()
    {
      if (NIUSE(ResLock) >=1)//Lock is occopied
        {
          var Offset = 1;//State is busy
          var Offset enter value = 1;//Enter gate is closed
          var offset exit value = 1;//Exit gate is closed
        }
       else (NIUSE(ResLock) <=0)//Lock is available
        {
          var Offset = 0;//State is idle
          var Offset enter value = 0;//Enter gate is opened
          var Offset exit value = 0;//Exit gate is opened
        }
    }

   function ProcessLock()
   {
```

```
      var ActivityTime;
      if (NARES(ResLock) >0)
       {
         IP.Seize(ResLock, 1);
         IP.SetLockState();
         IP.Schedule("Lockage", CurrentEntBargeTow,
(CurrentEntBargeTow.TNOW+ActivityTime));
         IP.Release(Reslock, 1);
       }
   }

   function RouteBargeTow()//Schedule the current EntBargeTow entity
for exiting lockage
   {
      var Distance;
      var Speed;
      var DelayTime = Distance/Speed;
      IP.Schedule("Exit", CurrentEntBargeTow,
CurrentEntBargeTow.DelayTime);
   }

    ]]>
  </Script>
  <Link Name="Exit" Type="Precedence">
  <Link Target="Set departure of barge-tows"/>
  </SMU>

  <SMU Name ="Operate double lockage">
    <Entity Name ="EntBargeTow">
      <Attribute atribname="Identification#" atribtype="interger"/>
      <Attribute atribname="NumberBarges" atribtype="integer"/>
      <Attribute atribname="Origin" atribtype="integer"/>
      <Attribute atribname="Destination" atribtype="integer"/>
      <Attribute atribname="ArrivalTime" atribtype="real"/>
      <Attribute atribname="Speed" atribtype="real">
      </Entity>

    <Resource Name ="ResLock">
      <Attribute atribname="Name" atribtype="string"/>
      <Attribute atribname="File#" atribtype="integer"/>
      <Attribute atribname="Resource#" atribtype="integer"/>
      <Attribute atribname="CapacityLock" atribtype="integer"/>
      <Attribute atribname="ActivityTime" atribtype="real"/>
    </Resource>

    <Resource Name ="ResWrench">
      <Attibute atribname="File#" atribtype="integer"/>
      <Attribute atribname="Resource#" atribtype="integer"/>
      <Attribute atribname="CapacityWrench" atribtype="integer"/>
      <Attribute atribname="ActivityTime" atribtype="real"/>
      <Resource>

     <Script Type="text/javascript">
     <![CDATA[

    function SetLockState()
    {
```

```
      if (NIUSE(ResLock) >=1)//Lock is occopied
        {
          var Offset = 1;//State is busy
          var Offset enter value = 1;//Enter gate is closed
          var offset exit value = 1;//Exit gate is closed
        }
       else (NIUSE(ResLock) <=0)//Lock is available
        {
          var Offset = 0;//State is idle
          var Offset enter value = 0;//Enter gate is opened
          var Offset exit value = 0;//Exit gate is opened
        }
    }

    function CutBargeTow()
    {
      var FirstBatchSize;
      var SecondBatchSize;
      var NumberBarges;
      if (NumberBarges > CapacityLock)
        {
           set FirstBatchSize = CapacityLock;
           set SecondBatchSize = NumberBarges - CapacityLock;
        }
    }

    function HoldBargeTow()
    {
      var DelayTime;
      IP.Schedule("ProcessLock", CurrentBargeTow.FirstBatchSize,
CurrentEntBargeTow.DelayTime);
      IP.Schedule("ProcessLock", CurrentBargeTow.SecondBatchSize,
CurrentEntBargeTow.DelayTime);
    }


    function ProcessLock()
    {
      var ActivityTime;
      if (NARES(ResLock) >0)
        {
          IP.Seize(ResLock, 1);
          IP.SetLockState();
          IP.Schedule("Lockage", CurrentEntBargeTow.FirstBatchSize,
(CurrentEntBargeTow.FirstBatchSize.TNOW+ActivityTime));
          IP.Release(ResLock, 1);
        }
     }

    function ProcessWrench()
    {
      var WrenchCapacity
      var ActivityTime;
      set ResWrench = Resource();
      set WrenchCapacity = ResWrench.CapacityWrench;
      if (NARES(ResWrench) > 0)
       {
```

```
          IP.Seize(ResWrench, 1);
          IP.Schedule("Wrench", CurrentEntBargeTow.FirstBatchSize,
(CurrenEntBargeTow.FirstBatchSize.TNOW+ActivityTime));
          IP.Release(ResWrench, 1);
       }
    }

    function ProcessLock()
    {
      var ActivityTime;
      if (NARES(ResLock) >0)
       {
          IP.Seize(ResLock, 1);
          IP.SetLockState();
          IP.Schedule("Lockage", CurrentEntBargeTow.SecondBatchSize,
(CurrentEntBargeTow.SecondBatchSize.TNOW+ActivityTime));
          IP.Release(ResLock, 1);
        }
     }

    function AssembleBargeTow()
    {
      set CurrentEntBargeTow.NumberBarges =
CurrentEntBargeTow.FirstBatchSize + CurrentEntBargeTow.SecondBatchSize;
    }

   function RouteBargeTow()//Schedule the current EntBargeTow entity
for exiting lockage
   {
      var Distance;
      var Speed;
      var DelayTime = Distance/Speed;
      IP.Schedule("Exit", CurrentEntBargeTow,
CurrentEntBargeTow.DelayTime);
   }
     ]]>
    </Script>
    <Link Name="Exit" Type="Precedence">
    <Link Target="Set departure of barge-tows"/>

  </SMU>

  <SMU Name ="Set departure of barge-tows">
    <Entity Name ="EntBargeTow">
      <Attribute atribname="Identification#" atribtype="interger"/>
      <Attribute atribname="NumberBarges" atribtype="integer"/>
      <Attribute atribname="Origin" atribtype="integer"/>
      <Attribute atribname="Destination" atribtype="integer"/>
      <Attribute atribname="ArrivalTime" atribtype="real"/>
      <Attribute atribname="Speed" atribtype="real">
      </Entity>
    <Script Type="text/javascript">
      <![CDATA[

    function CollectTime()
    {
      var TravelTime;
```

```
      var ID;
      var Label;
      set TravelTime = CurrentEntBargeTow.ArrivalTime - TNOW;
      IP.Collect(TravelTime, ID, Label);
    }

    function RouteBargeTow()
    {
      var Distance;
      var Speed;
      var DelayTime = Distance/Speed;
      IP.Schedule("Terminate", CurrentEntBargeTow,
CurrentEntBargeTow.DelayTime);
    }

    function TerminateBargeTow()
    {
      IP.TerminateEntity(CurrentEntBargeTow);
    }

    ]]>
  </Script>

  </SMU>
```

6-3-3. Phase 3: Mapping and Building

In the final phase, the simulation developer is able to make the transition from conceptualization to simulation. Using a tool, Similar Mapping Plane (SMP), is very crucial for mapping between the source ontology (e.g., CSMs and contextualized documentation) and the target ontology (e.g., Visual SLAM). SMP allows the simulation developer to nominate candidates for mapping and determine which one is the most appropriate selection for implementing in simulation. The following tables show how to map two ontologies by using SMP.

Table 6 - 6: Similarity Mapping Plane for CreateBargeTow()

Source	Weight	Target
Instance Name: CreateBargeTow()	1	*Node* Name: CREATE
Description A barge-tow entity is created by a mean of containing a set of barges and a tow boat	2	*Description* Entities are generated within the network.
Properties :First arrival :Arrival rate :Current time :Max# entities	1	*Inputs* :Time between creations (TBC) :Time of first creation (TF) :Maximum creations (MC) :Mark variable which will store the time of creation (MV) :Number of branches (M)
Input statement CreateBargeTow, First arrival, Arrival rate, Current time, Max# of signal entities;	1	*Input format* CREATE, TBC, TF, MV, MC, M;

Explanation
`function CreateBargeTow()`
`{`
`var FirstArrival = 0;`
`var ArrivalRate;`
`var CurrentTime = TNOW;`
`var MaxEntities;`
 	
`//Create a new entity`
 	
`Set NewEntBargeTow = IP.NewEntity();`
`Set NewEntBargeTow.ArrivalTime = IP.TNOW;`
`IP.Schedule("FirstArrival",`
`NewEntBargeTow,`
`(NewEntBargeTow.ArrivalTime+ArrivalRate);`
 	
`//Schedule the next entities`
 	
`for (i=1; i<=Max# entities; i++)`
`{`
`Set NextEntBargeTow = IP.CloneEntity();`
`Set NextEntBargeTow = IP.TNOW;`
`IP.Schedule("NextArrival",`
`NextEntBargeTow,`
`(NextEntBargeTow.ArrivalTime+ArrivalRate)`
`);`
`}`
`}`
	1	*Explanation*
		CREATE NODE
		:The first entity is created at a time specified by the value of TF;
		:The time between creations of entities after the first is specified by the variable TBC;
		:The time at which the entity is created can be assigned to a variable MV;
		:Entities will continue to be created until a limit is reached, specified by MC

Total scores	6/10	Likely similar

Weight by degrees of similarity (score): None (0); Likely similar (1); and Similar (2).

Table 6 - 7: Similar Mapping Plane for AssignBargeTow()

Source	Weight	Target
Instance Name: AssignBargeTow()	1	*Node* Name: ASSIGN
Description A simulation action is to assign identical attributes to define the characteristics of each barge-tow entity.	2	*Description* Values are assigned to Visual SLAM variables at each arrival of an entity to the node.
Properties :Identification# :Number of barges :Origin :Destination :Arrival time :Speed	1	*Inputs* :Visual SLAM global or entity variable (VAR) :Expression (VALUE) :Number of branches (M)
Input statement AssignBargeTow, Identification#, Number of barges, Origin, Destination, Arrival time;	1	*Input format* ASSIGN, {{VAR, VALUE}, repeats}, M;

Explanation
`function AssignBargeTow()`
 	
`//Assign attributes to the BargeTow entities`
`{`
`var Identification#;`
`var NumberBarges;`
`var Origin;`
`var Destination;`
`var ArrivalTime;`
`var Speed;`
 	
`//Define the current EntBargeTow entity and assign attributes to it`
 	
`Set CurrentEntBargeTow = This.EntBargeTow.CloneEntity();`
 	
`CurrentEntBargeTow.Assign(Identification#, NumberBarges, Origin, Destination, ArrivalTime, TravelTime, Speed);`
`}`
	1	*Explanation*
		ASSIGN NODE
		:Values are prescribed to the attributes of an entity passing through the ASSIGN node; or
		:Values are prescribed to the system variables that pertain to the network in general

Total scores	6/10	Likely similar

Weight by degrees of similarity (score): None (0); Likely similar (1); and Similar (2).

(Note: In advanced modeling, READ node can be used to assign values from an external file to attributes instead of ASSIGN node. However, to assign values to global variables still needs using ASSIGN node. SetLockState() is also be mapped to this node.)

Table 6 - 8: Similar Mapping Plane for RouteBargeTow()

Source	Weight	Target
Instance Name: RouteBargeTow()	1	*Node* Name: ACTIVITY
Description Each barge-tow entity is routed or moved through the system on designated routes.	2	*Description* Branches are used to model activities. Only at branches are explicit time delays prescribed for entities flowing through the network.
Properties :Distance	0	*Inputs* :Activity number (A) :Duration specified for the activity (DUR) :Condition for selecting the activity and can be a probability specification (COND) :End node label (NLBL) :Number of parallel identical servers (N) :Activity identification (ID)
Input statement RouteBargeTow, Distance;	0	*Input format* ACTIVITY, A, DUR, CONDITION, NLBL, N, ID;

Explanation
`function RouteBargeTow()`
 	
`//Schedule the current EntBargeTow entity for travelling`
`{`
`var Distance;`
`var Speed;`
`var DelayTime = Distance/SPEED;`
`IP.Schedule("Decision",`
`CurrentEntBargeTow,`
`CurrentEntBargeTow.DelayTime);`
`}`
	2	*Explanation*
		ACTIVITY
		:Branches emanate entities to simultaneously flow through them;
		:The duration of an activity is the time delay that an entity encounters as it flows through the branch representing the activity

Total scores	5/10	Likely similar

Weight by degrees of similarity (score): None (0); Likely similar (1); and Similar (2).

(Note: By the characteristics of ACTIVITY, it can also be used to function for BranchBargeTow(), CheckBargeTow(), and SelectLockage() by defining expressions for ACTIVITIES.)

Table 6 - 9: Similar Mapping Plane for ProcessLock()

Source	Weight	Target
Instance Name: ProcessLock()	1	*Node* Name: AWAIT
Description An action is taken at a lock by a mean of delay activity time.	2	*Description* The AWAIT node is used to store entities waiting for UR units of resource RES or waiting for gate GATE to open.
Properties :Resource# :Capacity of lock :Activity time	1	*Inputs* :File number (IFL) :Label of a component previously defined with a RESOURCE (RESORGATE) :Units required (UR) :Resource allocation rule (RULE) :Queue capacity (QC) :Condition of queue (FULLCOND) :Number of branches (M)
Input statement ProcessLock, Resource#, Capacity of lock, Activity time;	1	*Input format* AWAIT, IFL, {{RESORGATE, UR}, repeats}, RULE, QC, FULLCOND, M;

Explanation
`function ProcessLock()`
`{`
`var ActivityTime;`
`if (NARES(ResLock) >0)`
`{`
`IP.Seize(ResLock,1);`
`IP.SetLockState();`
`IP.Schedule("Lockage", CurrentBargeTow,`
`(CurrentBargeTow.TNOW+ActivityTime));`
`IP.Release(ResLock, 1);`
`}`
`}`
	2	*Explanation*
		AWAIT node
		: The AWAIT node delays an entity in file IFL until UR units of resource or group RES are available.
		:When required resources are available, the entity seizes the UR units of RES.

Total scores	7/10	Likely similar

Weight by degrees of similarity (score): None (0); Likely similar (1); and Similar (2).

(Note: To use AWAIT node, it also requires RESOURCE block and FREE node to complete the process. ProcessWrench() can be mapped to this Visual SLAM nodes and block as well.)

Table 6 - 10: Similar Mapping Plane for CollectTime()

Source	Weight	Target
Instance Name: CollectTime()	1	*Node* Name: COLCT
Description Statistical data for time spent in the system are collected.	2	*Description* Statistics can be collected on any expression at a COLCT.
Properties :Travel time :ID :Label	1	*Inputs* : Statistics index (N) :Expression whose value is to be observed (VARIABLE) :Identifying label (ID) :Number of histogram cells (NCEL) :Lower limit of first cell (HLOW) :Cell width (HWID) :Number of branches (M)
Input statement CollectTime, Travel time, ID, Label;	1	*Input format* COLCT, N, VAR, "ID", NCEL, HLOW, HWID, M;

Explanation
`function CollectTime()`
`{`
`var TravelTime;`
`var ID;`
`var Label;`
`set TravelTime =`
`CurrentEntBargeTow.ArrivalTime - TNOW;`
`IP.Collect(TravelTime, ID, Label);`
`}`
	1	*Explanation*
		COLCT node
		:The value of a Visual SLAM expression is recorded as an observation every time an entity arrives to the node.

Total scores	6/10	Likely similar

Weight by degrees of similarity (score): None (0); Likely similar (1); and Similar (2).

Table 6 - 11: Similar Mapping Plane for CutBargeTow()

Source	Weight	Target
Instance Name: CutBargeTow()	1	*Node* Name: UNBATCH
Description An action is to split a specific number of barges that are allowed to enter a lock.	2	*Description* An entity is split into multiple entities.
Properties :Identical batch size :Number of barges	1	*Inputs* :Number of copies to make of the entity (NCLONE) :Number of branches (M)
Input statement CutBargeTow, Identical batch size, Number of barges;	1	*Input format* UNBATCH, NCLONE, M;

Explanation
`function CutBargeTow()`
`{`
`var FirstBatchSize;`
`var SecondBatchSize;`
`var NumberBarges;`
 	
`if (NumberBarges > CapacityLock)`
`{`
`set FirstBatchSize = CapacityLock;`
`set SecondBatchSize = NumberBarges - CapacityLock;`
`}`
`}`
	1	*Explanation*
		UNBATCH node
		:The arriving entity is duplicated and NCLONE identical entities are released from the UNBATCH node.

Total scores	6/10	Likely similar

Weight by degrees of similarity (score): None (0); Likely similar (1); and Similar (2).

(Note: In practice, this function may or may not be required since it is possible to employ the logic that one unit can be put in multi-processing instead of splitting it into two parts for shortening the processes. Thus, AssembleBargeTow() that can possibly mapped into BATCH node would be ignored. This helps make modeling less complicated.)

The results of mapping ontologies on SMP help the DSSE developer to determine which Visual SLAM network nodes or functions are best fit to the construction of simulation building blocks by using the feature of visual subnetworks (VSNs). A collection of the VSNs is created as a library for reusing and accessing for other simulation studies under the same problem domain. However, as mentioned above, not every mapping result can be linked and implemented for testing. It is important for the simulation developer to add the details for implementation to those given products of mapping to satisfy the requirements of the simulation. Moreover, configurations and modifications in the Visual SLAM network nodes or functions are necessary for the accomplishment of simulation modeling. Finally, the quality of the simulation model is depended on the simulation developer's experience and expertise in Visual SLAM and AweSim, including logic and skills in simulation and modeling.

Figure 6-7 represents a simulation model for the travels of barge-tows from Port Muskogee to Tulsa Port of Catoosa through Lock#17. This simulation model has been developed from the results of mapping between conceptualization and simulation.

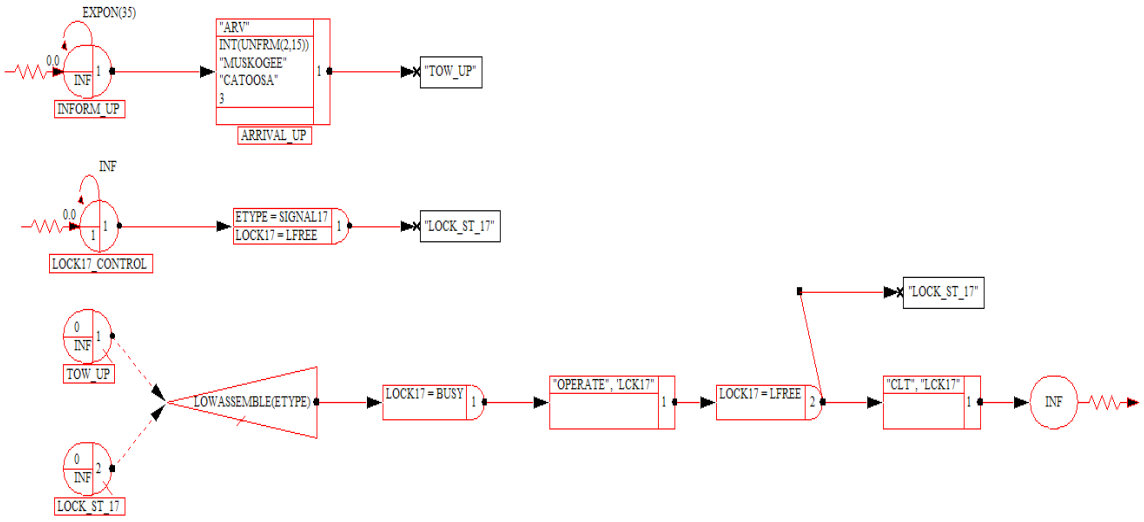

Figure 6 - 7: A simulation model for the lockage operations at Lock#17

As mentioned in Chapter 5, due to some restrictions of using VSNs, there appear combinations of the Visual SLAM network nodes and VSNs, patterns of the Visual SLAM network nodes, and stand-alone VSNs in the model. Obviously, it is unable to create simulation building blocks for representing every SMU retrieved from CSMs. The following figures provide the structures (building block elements) of the VSNs: "ARV", "OPERATE", "SINGLE", "DOUBLE", and "CLT", respectively.

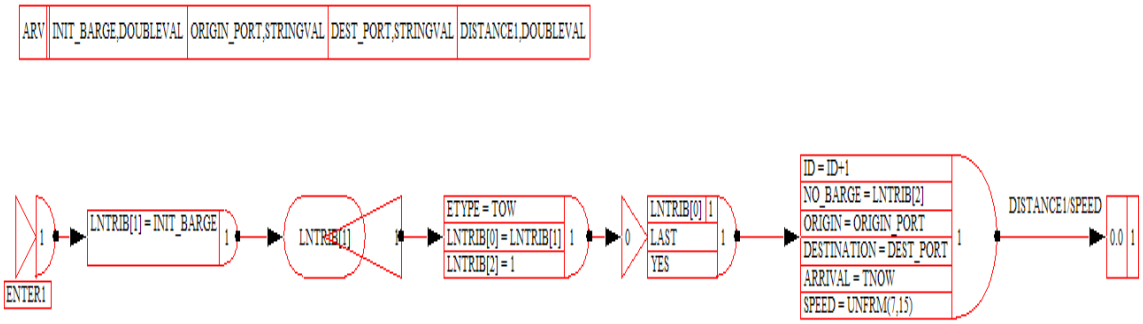

Figure 6 - 8: Building block elements for the VSN: "ARV"

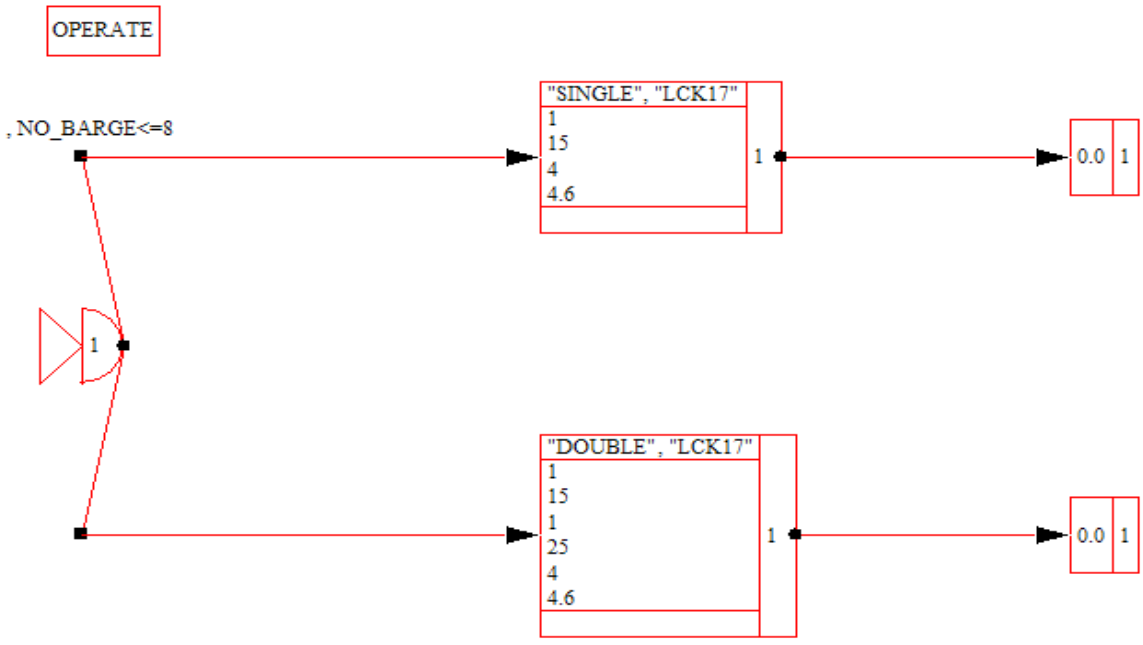

Figure 6 - 9: Building block elements for the VSN: "OPERATE"

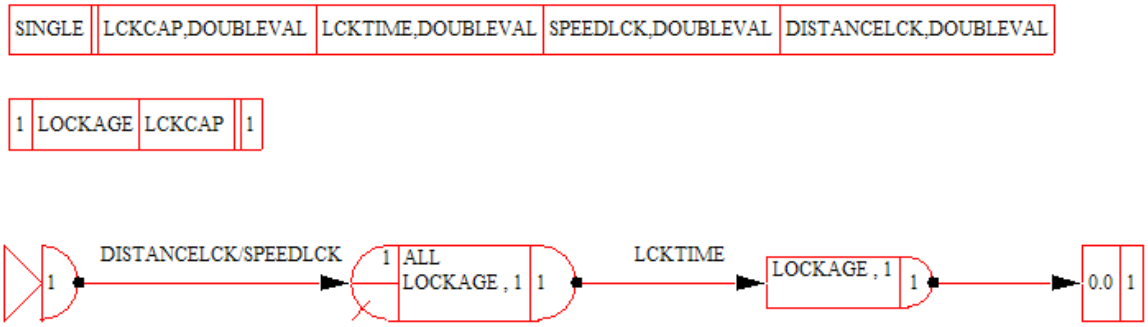

Figure 6 - 10: Building block elements for the VSN: "SINGLE"

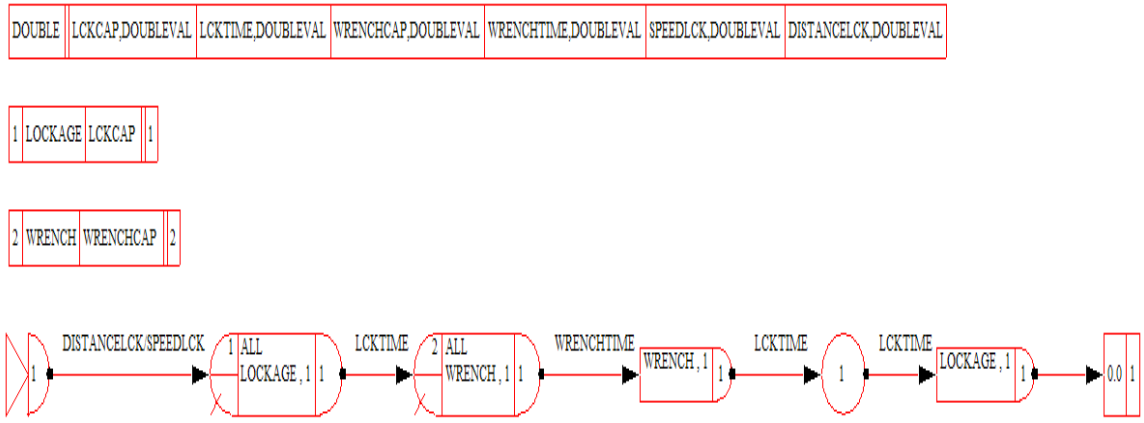

Figure 6 - 11: Building block elements for the VSN: "DOUBLE"

Figure 6 - 12: Building block elements for the VSN: "CLT"

For this simulation model, the network statements and control statements are given in Figure 6-13, whereas the subnetwork statements are provided in Figure 6-14.

```
Network statements:

INFORM_UP: CREATE,EXPON(35),0.0,,INF,1,,,,,{50,110};
        ACTIVITY,,,,,,,,,{1,3,,,};
ARRIVAL_UP: CALLVSN,"ARV",,{INT(UNFRM(2,15)),"MUSKOGEE","CATOOSA",3},1,,,,,,{130,110};
        ACTIVITY,,,,"TOW_UP",,,,,,{3,-1,,,250,110};
LOCK17_CONTROL: CREATE,INF,0.0,,1,1,,,,,{60,180};
        ACTIVITY,,,,,,,,,{5,7,,,};
        ASSIGN,{{ETYPE,SIGNAL17},{LOCK17,LFREE}},1,,,,,,,,{140,180};
        ACTIVITY,,,,"LOCK_ST_17",,,,,,{7,-1,,,250,180};
TOW_UP: QUEUE,1,0,INF,NONE,{ONE_SELECT_1},,,,,,{50,230};
        ;CONNECTOR{9,11}
ONE_SELECT_1: SELECT,LOWASSEMBLE(ETYPE),NONE,NONE,{TOW_UP,LOCK_ST_17},,,,,,,{100,260};
        ACTIVITY,,,,,,,,,{11,13,,,};
        ASSIGN,{{LOCK17,BUSY}},1,,,,,,,,{230,260};
        ACTIVITY,,,,,,,,,{13,15,,,};
        CALLVSN,"OPERATE","LCK17",,1,,,,,,{330,260};
        ACTIVITY,,,,,,,,,{15,17,,,};
        ASSIGN,{{LOCK17,LFREE}},2,,,,,,,,{430,260};
        ACTIVITY,,,,"LOCK_ST_17",,,,,,{17,-1,,,480,210,540,210};
        ACTIVITY,,,,,,,,,{17,20,,,};
        CALLVSN,"CLT","LCK17",,1,,,,,,{530,260};
        ACTIVITY,,,,,,,,,{20,22,,,};
        TERMINATE,INF,,,,,,,,,{630,260};
LOCK_ST_17: QUEUE,2,0,INF,NONE,{ONE_SELECT_1},,,,,,{50,290};
        ;CONNECTOR{23,11}


Control statements:

GEN,"Kitti Setavoraphan","MKARNS",4/9/09,1,YES,YES;
LIMITS,,10,,10,10,10;
INITIALIZE,0.0,1440,YES,,NO;
EQUIVALENCE,{{TOW,1},{SIGNAL17,17},{LFREE,0},{BUSY,1},{ARRIVAL,ATRIB[1]},{SPEED,ATRIB[2]}
,{ID,LTRIB[1]},{NO_BARGE,LTRIB[2]},{ORIGIN,STRIB[1]},{DESTINATION,STRIB[2]},{LOCK17,LL[0]
}};
INTLC,{{LL[0],0}};
NET;
FIN;
```

Figure 6 - 13: Visual SLAM network and control statements for Lock#17 simulation model

Subnetwork statements:

```
VSN,ARV,{{INIT_BARGE,DOUBLEVAL,
},{ORIGIN_PORT,STRINGVAL,},{DEST_PORT,STRINGVAL,},{DISTANCE1,DOUBLEVAL,
}},,,,,,,,{40,30};
        LIMITSVSN,-1,10,-1,-1,10,-1,,,,,{0,0};
ENTER1: ENTERVSN,,1,,,,,,,,{50,90};
        ACTIVITY,,,,,,,,,{3,5,,,};
        ASSIGN,{{LNTRIB[1],INIT_BARGE}},1,,,,,,,,{80,90};
        ACTIVITY,,,,,,,,,{5,7,,,};
        UNBATCH,LNTRIB[1],1,,,,,,,,{220,90};
        ACTIVITY,,,,,,,,,{7,9,,,};
        ASSIGN,{{ETYPE,TOW},{LNTRIB[0],LNTRIB[1]},{LNTRIB[2],1}},1,,,,,,,,{290,90};
        ACTIVITY,,,,,,,,,{9,11,,,};
        BATCH,0,LNTRIB[0],1,LAST,{LNTRIB[2]},YES,1,,,,{410,90};
        ACTIVITY,,,,,,,,,{11,13,,,};

ASSIGN,{{ID,ID+1},{NO_BARGE,LNTRIB[2]},{ORIGIN,ORIGIN_PORT},{DESTINATION,DEST_PORT},{ARRI
VAL,TNOW},{SPEED,UNFRM(7,15)}},1,,,,,,,,{500,90};
        ACTIVITY,,DISTANCE1/SPEED,,,,,,,,{13,15,,,};
        RETURNVSN,0.0,1,,,,,,,,{680,90};
VSN,OPERATE,,,,,,,,,{40,20};
        ENTERVSN,,1,,,,,,,,{50,120};
        ACTIVITY,,,NO_BARGE<=8,,,,,,{2,5,,,50,60};
        ACTIVITY,,,,"CASE_CALLVSN_1",,,,,,{2,8,,,50,180};
        CALLVSN,"SINGLE","LCK17",{1,15,4,4.6},1,,,,,,{180,60};
        ACTIVITY,,,,,,,,,{5,7,,,};
        RETURNVSN,0.0,1,,,,,,,,{330,60};
CASE_CALLVSN_1: CALLVSN,"DOUBLE","LCK17",{1,15,1,25,4,4.6},1,,,,,,{180,180};
        ACTIVITY,,,,,,,,,{8,10,,,};
        RETURNVSN,0.0,1,,,,,,,,{330,180};
VSN,SINGLE,{{LCKCAP,DOUBLEVAL,},{LCKTIME,DOUBLEVAL,},{SPEEDLCK,DOUBLEVAL,},{DISTANCELCK,D
OUBLEVAL,}},,,,,,,,{40,30};
        RESOURCE,1,LOCKAGE,LCKCAP,{1},,,,,,{40,60};
        ENTERVSN,,1,,,,,,,,{50,110};
        ACTIVITY,,DISTANCELCK/SPEEDLCK,,,,,,,{3,5,,,};
        AWAIT,1,{{LOCKAGE,1}},ALL,,NONE,1,,,,,{160,110};
        ACTIVITY,,LCKTIME,,,,,,,{5,7,,,};
        FREE,{{LOCKAGE,1}},1,,,,,,,,{280,110};
        ACTIVITY,,,,,,,,,{7,9,,,};
        RETURNVSN,0.0,1,,,,,,,,{380,110};
VSN,DOUBLE,{{LCKCAP,DOUBLEVAL,},{LCKTIME,DOUBLEVAL, },{WRENCHCAP,DOUBLEVAL,
},{WRENCHTIME,DOUBLEVAL, },{SPEEDLCK,DOUBLEVAL, },{DISTANCELCK,DOUBLEVAL,
}},,,,,,,,{30,20};
        RESOURCE,1,LOCKAGE,LCKCAP,{1},,,,,,{30,50};
        RESOURCE,2,WRENCH,WRENCHCAP,{2},,,,,,{30,80};
        ENTERVSN,,1,,,,,,,,{40,120};
        ACTIVITY,,DISTANCELCK/SPEEDLCK,,,,,,,{4,6,,,};
        AWAIT,1,{{LOCKAGE,1}},ALL,,NONE,1,,,,,{150,120};
        ACTIVITY,,LCKTIME,,,,,,,{6,8,,,};
        AWAIT,2,{{WRENCH,1}},ALL,,NONE,1,,,,,{260,120};
        ACTIVITY,,WRENCHTIME,,,,,,,{8,10,,,};
        FREE,{{WRENCH,1}},1,,,,,,,,{370,120};
        ACTIVITY,,LCKTIME,,,,,,,{10,12,,,};
        GOON,1,,,,,,,,,{480,120};
        ACTIVITY,,LCKTIME,,,,,,,{12,14,,,};
        FREE,{{LOCKAGE,1}},1,,,,,,,,{530,120};
        ACTIVITY,,,,,,,,,{14,16,,,};
        RETURNVSN,0.0,1,,,,,,,,{620,120};
VSN,CLT,,,,,,,,,{41,33};
        ENTERVSN,,1,,,,,,,,{50,80};
        ACTIVITY,,,,,,,,,{2,4,,,};
        COLCT,,TNOW-ARRIVAL,"TIS",,,1,,,,{120,80};
        ACTIVITY,,,,,,,,,{4,6,,,};
        RETURNVSN,0.0,1,,,,,,,,{260,80};
```

Figure 6 - 14: Visual SLAM subnetwork statements for Lock#17 simulation model

Based on this methodology, the simulation developer can further develop a simulation model for the lockage operations at both Lock# 17 and Lock# 18 in which the barge-tows from the downstream of MKARNS travel, as shown in Figure 6-15.

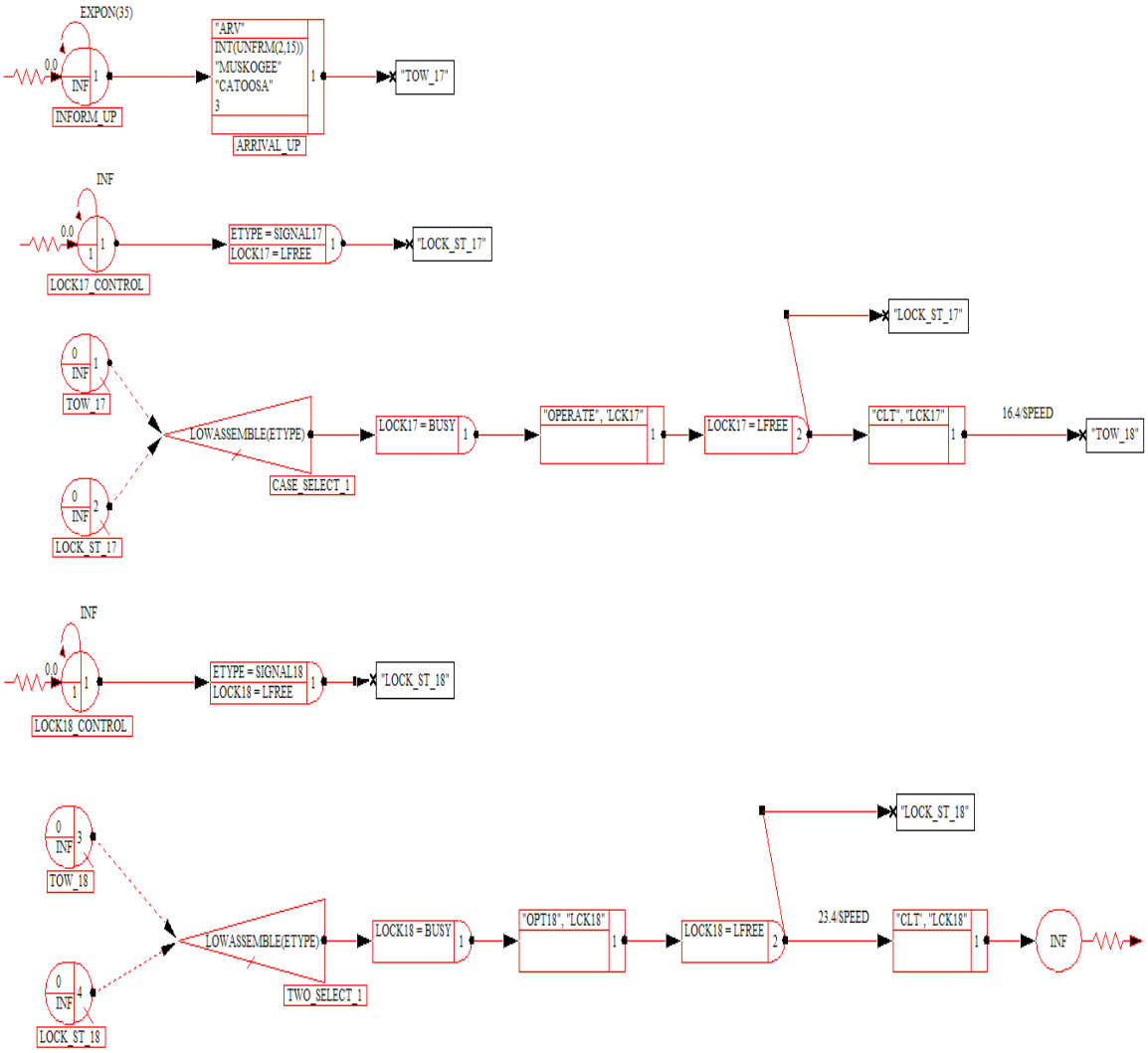

Figure 6 - 15: A simulation model for the lockage operations at Lock# 17 and Lock# 18

As well, its network statements, control statements, and extended subnetwork statements are given in the following figures:

```
Network statements:

INFORM_UP: CREATE,EXPON(35),0.0,,INF,1,,,,,{50,110};
        ACTIVITY,,,,,,,,,,{1,3,,,};
ARRIVAL_UP: CALLVSN,"ARV",,{INT(UNFRM(2,15)),"MUSKOGEE","CATOOSA",3},1,,,,,,{130,110};
        ACTIVITY,,,,"TOW_17",,,,,,{3,-1,,,240,110};
LOCK17_CONTROL: CREATE,INF,0.0,,1,1,,,,,{60,180};
        ACTIVITY,,,,,,,,,,{5,7,,,};
        ASSIGN,{{ETYPE,SIGNAL17},{LOCK17,LFREE}},1,,,,,,,,{140,180};
        ACTIVITY,,,,"LOCK_ST_17",,,,,,{7,-1,,,250,180};
TOW_17: QUEUE,1,0,INF,NONE,{CASE_SELECT_1},,,,,,{50,230};
        ;CONNECTOR{9,11}
CASE_SELECT_1: SELECT,LOWASSEMBLE(ETYPE),NONE,NONE,{TOW_17,LOCK_ST_17},,,,,,,{100,260};
        ACTIVITY,,,,,,,,,,{11,13,,,};
        ASSIGN,{{LOCK17,BUSY}},1,,,,,,,,{230,260};
        ACTIVITY,,,,,,,,,,{13,15,,,};
        CALLVSN,"OPERATE","LCK17",,1,,,,,,{330,260};
        ACTIVITY,,,,,,,,,,{15,17,,,};
        ASSIGN,{{LOCK17,LFREE}},2,,,,,,,,{430,260};
        ACTIVITY,,,,"LOCK_ST_17",,,,,,{17,-1,,,480,210,540,210};
        ACTIVITY,,,,,,,,,,{17,20,,,};
        CALLVSN,"CLT","LCK17",,1,,,,,,{530,260};
        ACTIVITY,,16.4/SPEED,,"TOW_18",,,,,,{20,-1,,,660,260};
LOCK_ST_17: QUEUE,2,0,INF,NONE,{CASE_SELECT_1},,,,,,{50,290};
        ;CONNECTOR{22,11}
LOCK18_CONTROL: CREATE,INF,0.0,,1,1,,,,,{60,370};
        ACTIVITY,,,,,,,,,,{24,26,,,};
        ASSIGN,{{ETYPE,SIGNAL18},{LOCK18,LFREE}},1,,,,,,,,{140,370};
        ACTIVITY,,,,"LOCK_ST_18",,,,,,{26,-1,,,230,370,240,369};
TOW_18: QUEUE,3,0,INF,NONE,{TWO_SELECT_1},,,,,,{50,430};
        ;CONNECTOR{28,30}
TWO_SELECT_1: SELECT,LOWASSEMBLE(ETYPE),NONE,NONE,{TOW_18,LOCK_ST_18},,,,,,,{120,470};
        ACTIVITY,,,,,,,,,,{30,32,,,};
        ASSIGN,{{LOCK18,BUSY}},1,,,,,,,,{240,470};
        ACTIVITY,,,,,,,,,,{32,34,,,};
        CALLVSN,"OPT18","LCK18",,1,,,,,,{330,470};
        ACTIVITY,,,,,,,,,,{34,36,,,};
        ASSIGN,{{LOCK18,LFREE}},2,,,,,,,,{430,470};
        ACTIVITY,,,,"LOCK_ST_18",,,,,,{36,-1,,,480,420,560,420};
        ACTIVITY,,23.4/SPEED,,,,,,,{36,39,,,};
        CALLVSN,"CLT","LCK18",,1,,,,,,{560,470};
        ACTIVITY,,,,,,,,,,{39,41,,,};
        TERMINATE,INF,,,,,,,,,{660,470};
LOCK_ST_18: QUEUE,4,0,INF,NONE,{TWO_SELECT_1},,,,,,{50,490};
        ;CONNECTOR{42,30}

Control statements:

GEN,,,,1,YES,YES;
LIMITS,,10,,10,10,10;
INITIALIZE,0.0,1440,YES,,NO;
EQUIVALENCE,{{TOW,1},{SIGNAL17,17},{SIGNAL18,18},{LFREE,0},{BUSY,1},{ARRIVAL,ATRIB[1]},{S
PEED,ATRIB[2]},{ID,LTRIB[1]},{NO_BARGE,LTRIB[2]},{ORIGIN,STRIB[1]},{DESTINATION,STRIB[2]}
,{LOCK17,LL[0]},{LOCK18,LL[1]}};
INTLC,{{LL[0],0},{LL[1],0}};
NET;
FIN;
```

Figure 6 - 16: Visual SLAM network and control statements for Lock# 17 and Lock# 18 simulation model

```
VSN,OPT18,,,,,,,,,{60,20};
        ENTERVSN,,1,,,,,,,,{60,120};
        ACTIVITY,,,NO_BARGE<=8,,,,,,,{2,5,,,60,70};
        ACTIVITY,,,,"OPT18_CALLVSN_1",,,,,,{2,8,,,60,170};
        CALLVSN,"SGL18","LCK18",{1,10,4,4.2},1,,,,,,,{150,70};
        ACTIVITY,,,,,,,,,,{5,7,,,};
        RETURNVSN,0.0,1,,,,,,,,{280,70};
        OPT18_CALLVSN_1: CALLVSN,"DBL18","LCK18",{1,10,1,20,4,4.2},1,,,,,,,{150,170};
        ACTIVITY,,,,,,,,,,{8,10,,,};
        RETURNVSN,0.0,1,,,,,,,,{280,170};
VSN,SGL18,{{LCKCAP,DOUBLEVAL, },{LCKTIME,DOUBLEVAL, },{SPEEDLCK,DOUBLEVAL,
},{DISTANCELCK,DOUBLEVAL, }},,,,,,,,{70,30};
        RESOURCE,3,LK18,LCKCAP,{3},,,,,,,{70,70};
        ENTERVSN,,1,,,,,,,,{80,120};
        ACTIVITY,,DISTANCELCK/SPEEDLCK,,,,,,,,{3,5,,,};
        AWAIT,3,{{LK18,1}},ALL,,NONE,1,,,,,{200,120};
        ACTIVITY,,LCKTIME,,,,,,,,{5,7,,,};
        FREE,{{LK18,1}},1,,,,,,,,{310,120};
        ACTIVITY,,,,,,,,,,{7,9,,,};
        RETURNVSN,0.0,1,,,,,,,,{390,120};
VSN,DBL18,{{LCKCAP,DOUBLEVAL,},{LCKTIME,DOUBLEVAL,},{WRENCHCAP,DOUBLEVAL,},{WRENCHTIME,DO
UBLEVAL,},{SPEEDLCK,DOUBLEVAL,},{DISTANCELCK,DOUBLEVAL,}},,,,,,,,{60,30};
        RESOURCE,3,LK18,LCKCAP,{3},,,,,,,{60,70};
        RESOURCE,4,WR18,WRENCHCAP,{4},,,,,,,{60,100};
        ENTERVSN,,1,,,,,,,,{70,160};
        ACTIVITY,,DISTANCELCK/SPEEDLCK,,,,,,,,{4,6,,,};
        AWAIT,3,{{LK18,1}},ALL,,NONE,1,,,,,{190,160};
        ACTIVITY,,LCKTIME,,,,,,,,{6,8,,,};
        AWAIT,4,{{WR18,1}},ALL,,NONE,1,,,,,{290,160};
        ACTIVITY,,WRENCHTIME,,,,,,,,{8,10,,,};
        FREE,{{WR18,1}},1,,,,,,,,{390,160};
        ACTIVITY,,LCKTIME,,,,,,,,{10,12,,,};
        GOON,1,,,,,,,,,{490,160};
        ACTIVITY,,LCKTIME,,,,,,,,{12,14,,,};
        FREE,{{LK18,1}},1,,,,,,,,{550,160};
        ACTIVITY,,,,,,,,,,{14,16,,,};
        RETURNVSN,0.0,1,,,,,,,,{630,160};
```

Figure 6 - 17: Extended Visual SLAM subnetwork statements for Lock# 17 and Lock# 18 simulation model

Finally, Table 6-12 represent a sample of results for the expected outputs (as designed in Termination Layer) from running the simulation model for 24 hours.

Table 6 - 12: The results from running the simulation model of Lock# 17 and Lock# 18

Lock#	Average Time in System	Average Lock Utilization	Average Waiting Time
17	255.493 minutes	82%	222.211 minutes
18	306.833 minutes	55%	14.489 minutes

6-4. Conclusions

The case study of the lockage operations on MKARNS illustrates that the methodology developed is effective in supporting the development of simulation models. It leads the processes of simulation and modeling to the effectiveness and efficiency in not only generating simulation models but also improving an individual's thinking and decision making. This means that the methodology provides such standard tools and procedures that facilitate communication and collaboration among team members (or anyone involving) and specify frameworks for the development. With these tools, frameworks, and procedures, the individual is able to develop his/her own simulation environment to support simulation studies related to a specific domain such as the lockage operations. While effective, considering the restrictions of Visual SLAM and AweSim and the limitations of the individual (e.g., skills and knowledge), this methodology could be made more applicable through the development of user-friendly interface tools that are beyond the scope of this research.

6-5. References

1. AweSim: Total Simulation Project Support. 1999. *User's guide*. Version 3.0. Symix Systems.
2. Pritsker, A. A. B. and J. J. O'Reilly. 1999. *Simulation with Visual SLAM and AweSim*. 2nd ed. New York: John Wiley & Sons.
3. Setavoraphan, K. and F. H. Grant. 2008. Conceptual simulation modeling: The structure of domain specific simulation environment. In *Proceedings of the 2008 Winter Simulation Conference.* pp. 975-986.

4. U.S. Army Corps of Engineers. (2008). *Navigation*. Retrieved May 15, 2008, from http://www.swl.usace.army.mil/navigation/lock.html.

5. U.S. Army Corps of Engineers. (2008). *Navigation*. Retrieved May 15, 2008, from http://www.swl.usace.army.mil/PROJMGT/ArkNav.Final_Report/Chapter_1_.doc.

6. Valentin, E. C. and A. Verbraeck. 2002. Guidelines for designing simulation building blocks. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yucesan, C. –H. Chen, J. L. Snowdon, and J. M. Charnes. pp. 563-571. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

7. Visual SLAM. 1997. *Quick reference manual.* Version 2.0. Pritsker Corporation.

CHAPTER 7

Conclusions and Future Research

This chapter is divided into general conclusions, comparisons, and recommendations for future research.

7-1. General Conclusions

The purpose of this research study was to provide a robust and rigid methodology that enhanced the current modeling and simulation (M&S) knowledge for building domain specific simulation environments (DSSEs). Three critical issues were addressed to outline a direction and framework for this dissertation in order to develop the methodology, which were: the appropriateness of conceptual simulation models; the semantics of transformational conceptual simulation models; and the expressiveness of use of simulation models. To address these issues, it was necessary not only to handle each of them individually but also to resolve all of them together, including their interactions. The methodology is decomposed into a trilogy of methodologies corresponding to each issue as well as integrating them into a powerful M&S tool corresponding to the entire development of DSSEs.

At the beginning of study, the center focus was on a DSSE approach that allowed simulation developers to obtain both maintainability/sustainability in modeling and controllability in simulation. Under the DSSE approach, simulation developers were able to include the entire development processes for a DSSE – from designing its overall structure through controlling the semantics and contents of its model constructs until

implementing the model constructs for a specific use. Moreover, the approach was not restrictively applied to only create a new simulation language environment or application. On the other hand, it can potentially be embedded into an existing simulation language environment or application – to build a reusable modeling environment that can solve future design problems as they arise. With this significant potential, the DSSE approach is a key solution reducing not only the gap between reality and simulation but also the barriers created by existing simulation technologies.

As the research progressed, the DSSE approach involved the development of particular methodologies that addressed the key components of Conceptualization, Documentation, and Translation. Each methodology provided relevant concepts and techniques to resolve a specific issue. In fact, each individual methodology can either be directly applied or be initially disregarded in the development of a DSSE. However, to represent the relationship, completeness, and correctness of structure, content, and context of simulation in the DSSE, an integration of these methodologies was intensively needed. This aimed to increase the levels of syntactic and semantic interoperability/composability of data and components from conceptualization through implementation.

The first step of developing a DSSE using the integrated methodology initiated with conceptualization of a domain problem. In the chapter 3, Integrated Simulation Acknowledge Procedure (ISAP) was introduced to be as a conceptual simulation modeling (CSM) tool for capturing and transforming the concepts in a specific problem domain into a set of descriptive processes, static and dynamic modeling components, interactions, and rules/algorithms defined within a simulation modeling framework. The

center idea of ISAP was to provide not only an appropriate framework that specified both structural and behavioral characteristics of a DSSE but also a blueprint that gave designs and instructions for the development process.

Under the ISAP framework, a DSSE can be divided into three layers: Initialization Layer (IL), Process Layer (PL), and Termination Layer (TL) to match characteristics and architecture of general host simulation languages/environments. Frames of references defining experimental conditions, input and output parameters, and process descriptions and interactions were portrayed by a means of knowledge representations using symbols, notation, and diagrams for these layers. Using knowledge representations was a key to success in reducing or eliminating complexities in capturing real world concepts, communicating between simulation developers and domain experts, and mapping the concepts into simulation concepts/requirements. As a result, errors from the conceptualization process can be minimized, which helped save time and cost for simulation projects.

Even though, in many cases, conceptual simulation models can be directly used as construction guidelines for building simulation models, there always appeared a trouble called “lost in translation” of, e.g., semantics and contents of simulation during transformation. This became another focus of this dissertation on bridging a gap between conceptualization and implementation. Chapter 4 was dedicated to study key factors and concepts in transforming CSMs into executable simulation models. A finding of the study showed that in order to have more expressive and meaningful representations for transformation of CSMs at the level of implementation, semantics of model composability and simulation interoperation must be clearly specified at the conceptual

level. Accordingly, the integrated methodology was initially designed to support this continuous development process by representing CSMs in a pattern of descriptive model components and using natural language for descriptions – which can be transformed into simulation-like-language statements. The statements, therefore, became a data bridge that facilitated controlling and transferring semantics of model composability and simulation interoperability between conceptualization and implementation.

However, to represent both structural and behavioral characteristics of simulation in an executable way, the simulation-like-language statements needed to be translated into programmable-and-simulation documentation that computer/simulation systems can really exchange and understand. Also, to avoid the difficulty in reading/understanding the documentation, an intermediate simulation language was developed for contextualizing those statements in a more readable form – by using the Semantic Web languages such as XML and SRML. In a consequence, it was able to create contextualized documentation that included details of modeling transformation and supported semantics/data exchange between conceptualization and implementation. The documentation then can be either instantly implemented or ontologically mapped onto a host simulation language or a simulation runtime environment.

Instant implementation of contextualized documentation seemed to be a critical resolution for the "lost in translation" problem. This was because all descriptions and functionalities specified in simulation contexts can initially be executed by a simulation system without any modifications. Nevertheless, there was a need for building a specific simulator that contained, e.g., runtime environment, database, event calendar, and libraries of callable functions, to support the implementation process, which was an

unavoidable time-and-cost consuming activity. As a result, the resolution was switched to focus on ontology mapping, described in Chapter 5.

Having been developed by using the Semantic Web languages, this type of contextualized documentation provided essential aspects of both object-orientation and ontology. This allowed not only delivering unambiguous meaning of data of model transformation but also representing semantic behavior and structure of data in a set of component modules – that matched the common architecture of any host simulation languages/environments. As a result, there was a possibility to translate these modules into a set of simulation building blocks, creating a pattern and framework for mapping in a sense of semantic similarities. With assistance of ontology mapping, most of the conceptual modules specified in the contextualized documentation can be mapped onto basic simulation components and be composed into simulation building blocks on a host simulation language/environment.

The integrated methodology was applied to build a DSSE for an inland waterway (lockage) operation problem on Visual SLAM and AweSim, demonstrated in Chapter 6 The main reason for choosing this host simulation language/environment was that its model components can be translated into network and control statements when implementation. Also, its open-ended architecture allowed us to add specific components to the simulation environment by using Visual Basic and C/C++ programming language and to develop simulation building blocks by using the features of visual subnetworks (VSNs). All these characteristics, including other integrating capabilities of AweSim, e.g., to store, retrieve, browse, and communicate with externally written software

applications, helped facilitate us in ontology mapping as well as building a set of libraries of VSNs to be reused in the DSSE.

The outcomes from exploiting the integrated methodology for the development of DSSEs returned to us in many beneficial aspects. First, it helped improve both individual and team's thinking and decision making process in solving a variety of real world problems, which CSM was the critical supporting mechanism in managing complexities and communicating concepts. Second, it generated standard tools, frameworks, and procedures that an individual can apply to develop his/her own DSSE corresponding to resolve specific problems. This also helped enhance the capability of the existing host simulation language/environment such as Visual SLAM and AweSim to be more supportive for alternative simulation modeling. Finally, it led simulation developers to focus more on optimizing performance of what simulation applications available in hands rather than looking for new replacements. This study showed us that methodology was more important than technology, and it just needed to be appropriately embedded into one of the right technologies. If it can be done so, not only individuals but also organizations can increase cost savings and confidence in applying simulation for problem solving.

The advantages mentioned above seemed to be the extraordinary results that might create impacts to the M&S society in terms of either individuals or organizations. However, the true contribution of this research study was a state-of-the-art methodology used to build simulation applications that are self documenting, easy to expand, and easy to use by users that go beyond the model builder, as simulation application is today. The product can be either a one-time-use simulation application (model) or a reusable

simulation modeling environment. In addition, this dissertation aimed to help improve the quality of simulation training and education by illustrating the essential architecture and behavior of simulation at conceptual levels which can reduce/eliminate difficulties in learning and complexities in building simulation. Prospect or current simulation learners, therefore, were less reluctant to learn/use simulation to solve their problems. Nevertheless, the methodology seemed not to be the ultimate weapon breaking the barriers between reality and simulation world because of some restrictions created by the methodology itself, simulation technologies, and users' skills and knowledge. Room for improvement in many aspects, thus, was still opened for the future work.

7-2. Comparison with Current Methods

It would be useful to consider how the proposed methodology works when compared to standard approaches. This is, however, difficult, since the proposed methodology does not have the advanced user interface tools which are typically available and influence model development productivity. We can, however, consider some general comparison characteristics to get an idea of how productivity can be enhanced when applying the new technology.

To clarify the characteristics the proposed methodology possesses in breaking through the limitations of modeling and simulation created by the current approaches, the comparison can be divided into two categories: objective and subjective comparison. Objective comparison requires numeric measurements, whereas subjective comparison needs reasonable descriptions. Since we do not have details regarging model

development productivity to be measured, the following discussion, thus, is given in subjective terms for comparison.

For decades, domain specific simulation applications/environments have been strictly developed based on standard approaches similar to the conventional software engineering processes, which can be categorized into two main streams: pure programming and user-interface (e.g., nodes/blocks, tools) application. In either way of creation, they both share one common – "sketch-to-ash" – framework. This means each project originates from raw ideas which later are transformed into thousands of programming-code patterns to create a complete domain specific simulation application/environment for solving only one particular problem. Sooner or later the product/software becomes obsolete because editing/configuring those programming codes to match new requirements is a too complicated task and too risky investment for time and budget.

However, the difficulty in programming is not a big concern for this comparison. We rather focus on the differences in terms of the key modeling and simulation issues that make impacts to both development and application of domain specific simulation.

Table 7- 1: Subjective comparison between integrated and standard methodology

Issues	Integrated Methodology	Standard Methodology
Controllability	: Moderate control if built on existing simulation host language/environment : 100% control plus appropriate design if newly built	: 100% control, but probably non-applicable
Reusability	: Organized processes for modeling : Retrievable components for remodeling	: Ad hoc : Unorganized processes for modeling : Required expert knowledge for remodeling
Maintainability	: Well-documented for future reference : Public accessed for maintenance	: Non-reference : Private accessed for maintenance
Composability	: Semantic composition : Easy configuration	: Often syntactic composition : Complicated configuration
Interoperability	: Open platform for implementation	: Limited platform for implementation

As shown in Table 7-1, five issues that have been seriously discussed through this dissertation are still determined to be critical characteristics for this subjective comparison. This aims to illustrate how the integrated methodology has enhanced the standard approaches.

Under the integrated methodology, simulation developers are encouraged to develop domain specific simulation on an existing simulation host language/environment, which could limit controllability in simulation. Unlike the standard approaches, every line of programming codes are written to fully support simulation controls. However, if apply the integrated methodology for developing a new domain specific simulation application/environment, what the simulation developers would obtain is not only 100% of simulation controllability but also the appropriateness in designing simulation controls. This is because domain experts are allowed to involve with the simulation developers through the whole development process, in which a mutual understanding is created – that keeps the simulation development being controllable and applicable. Without sharing perspectives from both sides, the power of simulation controllability could be seemingly useless.

Next, reusability has become one of the most beneficial characteristics provided by the integrated methodology. It offers well-organized processes for modeling simulation components. Also, the idea of simulation block building helps the simulation developers to be able to design architecture and patterns for configuring the simulation components to be reusable and retrievable for remodeling. Meantime, having unorganized modeling processes, the standard approaches seems to be ad hoc when

modeling reusable simulation components. This becomes a too specific job that requires expert knowledge for remodeling.

For the maintenance of domain specific simulation, the integrated methodology enforces the simulation developers to generate well-documented information that anyone can use as reference. This makes the maintenance job easier and more precise. Unlikely, the standard approaches are still relied on personal responsibility in maintaining individual simulation projects. As a result, maintenance becomes a high cost and time consuming activity.

In applicable terms, the integrated methodology restrictedly specifies the requirements for composing simulation components to create high productivity for simulation implementation. The result is that it can reduce or eliminate numerous compositions of the simulation components that do not give any semantics or solutions. With strict control and management of composition, configuration of composed simulation components can easily be performed. On the other hand, ignorance in semantic composition is often allowed to occur in the simulation products created by the standard approaches. Because of some limitations of programming languages or skills, it is difficult to avoid having syntactic composition to accomplish a task. This, later, can lead to troubles in configuration.

Finally, the most obvious characteristic that the integrated methodology has brought to the modeling and simulation community is the ability to build domain specific simulation on any platform of programming languages, host simulation languages/environments, or technologies. Even though the integrated methodology has been only applied with Visual SLAM and AweSim, its concepts and processes are not

limited to be applied for other implementation platforms. This is because ontological mapping plays a key role in simulation interoperation, which cannot be found in the standard approaches.

Though it seems to be too subjective for comparison, if carefully determine, it is found out that the integrated methodology shows numerous characteristics that benefit the ways of modeling and simulation – which lacks in today's approaches.

7-3. Future Work

Even though the framework of this research study was designed based on the underlying convergent concepts of Software Engineering (SE) and Knowledge Engineering (KE), the integrated methodology had only capacity for structuring and displaying information as formalisms of knowledge representations. When considering the development of a practical and user-friendly application, additional research is recommended to reach the level necessary to deliver a complete M&S tool corresponding to the requirements of non-expert simulation users. Additional research is suggested to address the following:

- The process could be more efficient by the development of graphical interface software to create graphical diagrams, notation, symbols, and tables, including XML and JavaScript documentation;
- An integrated software system would be useful which multi software applications such as Microsoft Office, Visual Studios, and AweSim, which creates and maintains work files and verifies the correctness of information; and

- Development of mechanisms facilitating connection, display, and storage for knowledge representations, documentation, model components, and references to be developed as knowledgebase for either current or prospect simulation projects.

These development topics will bring the integrated methodology to life and to make it more easily used by engineers, basically research in software engineering. SE could give us the power to enhance our current capability in representing the ideas and concepts to become more realistic and more practical terms of applications. In addition, SE could open space for others in different study areas to share their knowledge and involve with the software development processes to create better solutions. Under the SE umbrella, it allows us to come up with a three-phase development program that lays out perspectives and guidelines for the future works.

7-3-1. Phase I: User-friendly Software

The main task in this phase is to develop software that can assist simulation developers to shorten processing time in creating knowledge representations and documentation for a specific simulation project. Also, the software application could allow them to store their works into categories which are ready to be called and restored for modifications. To make a user-friendly software application, it is recommended to build it in a window-type format that can be run on Microsoft Windows or Mac OS. This application should provide a graphical interface that contains, e.g., function tools, icons, and display areas, for specific tasking modes such as building diagrams for Initial Layer, writing XML and JavaScript documentation, or updating callable user-function libraries.

Basically, this software application could be developed as an environment similar to, for example, Rhapsody C++® software by I-Logic Inc. which is used to support the software development using UML, C++, and Java. This means that the software environment could be designed to support not only basic functions but also programming compilers. Other features, moreover, could be added if needed, for example, database components and import/export links. The goal is to make software simple as possible for every level of users to develop knowledge representations and documentation effectively and efficiently.

7-3-2. Phase II: Embedded Simulation Software

The product from Phase I could resolve the difficulties in building knowledge representations and documentation. The next step is to embed the software application into a simulation application such as AweSim, which makes it to be one of the components (e.g., User Data, User Inserts, or Notes) displayed in the AweSim executive window. After embedded, it should be easier for simulation developers to call references from the (software) component for selecting and mapping network model components available in AweSim. Furthermore, it could help save time creating historical records of selection, mapping, and creation (in case of no match) for being used as future references. The goal of this embedded simulation software is to provide simulation developers convenience and accuracy in transferring conceptualization into simulation.

7-3-3. Phase III: Automation

The process of selecting and mapping in the embedded simulation software could still be considered a time-consuming activity. With the SE capabilities, it is possible to go beyond the current capacity of the embedded simulation software. Automation could be added as a mechanism that automatically transforms knowledge representations into XML and JavaScript documentation and translates documentation into model components. Also, it could provide recommendations for mapping. However, automation requires sophisticated algorithms to perform automatic transformation, translation, and mapping, which could create errors and delays in processing and analyzing results. The goal of automation is to shorten time for testing and verifying conceptualization in simulation.

This three-phase development program provides simulation developers a spectacular insight and inspiration to enhance the capabilities of simulation we have today and to go beyond the boundaries of any M&S methodologies. Finally, the integrated methodology developed in this research study is expected to be a strong stepping stone that leads both simulation application and education to be implemented in higher levels.