

A FRAMEWORK FOR DEVELOPING CONCEPTUAL MODELS OF THE MISSION
SPACE FOR SIMULATION SYSTEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

N. ALPAY KARAGÖZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

JUNE 2008

Approval of the Graduate School of Informatics

Prof.Dr. Nazife BAYKAL
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy.

Prof.Dr. Yasemin YARDIMCI
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Doctor of Philosophy.

Assoc. Prof. Dr. Onur DEMİRÖRS
Supervisor

Examining Committee Members

Prof. Dr. Semih BİLGEN (METU, EEE) _____

Assoc. Prof. Dr. Onur DEMİRÖRS (METU, II) _____

Assist. Prof. Dr. Kayhan İMRE (HU, CS) _____

Assoc. Prof. Dr. Veysi İŞLER (METU, CENG) _____

Dr. Altan KOÇYİĞİT (METU, II) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: N. ALPAY, KARAGÖZ

Signature :

ABSTRACT

A FRAMEWORK FOR DEVELOPING CONCEPTUAL MODELS OF THE MISSION SPACE FOR SIMULATION SYSTEMS

Karagöz, N. Alpay

Ph.D., Department of Information Systems

Supervisor: Assoc. Prof. Dr. Onur Demirörs

June 2008, 160 pages

The simulation world defines conceptual modeling as a tool that provides a clear understanding of the target domain or problem. Although there are some approaches offering useful insights on conceptual modeling in the simulation development lifecycle, they do not provide adequate guidance on how to develop a conceptual model. This thesis study presents a framework for developing conceptual models for simulation systems that is based on the idea that the modelers will develop conceptual models more effectively by following a defined conceptual modeling method, using a domain specific notation and a tool. The conceptual model development method is defined in a step-by-step manner and explanations about the notation and tool are provided when required. A multiple-case study involving two cases is conducted in order to evaluate the applicability of the method for conceptual modeling and validate the expected benefits.

Keywords: Conceptual Modeling, Metamodel, Domain Specific Modeling, Conceptual Models of the Mission Space

ÖZ

SİMÜLASYON SİSTEMLERİ GÖREV UZAYI KAVRAMSAL MODELİ GELİŞTİRMEK İÇİN BİR ÇERÇEVE

Karagöz, N. Alpay

Doktora, Bilişim Sistemleri Bölümü

Tez Yöneticisi: Doç. Dr. Onur Demirörs

Haziran 2008, 160 sayfa

Simülasyon dünyası kavramsal modellemeyi hedef alanı veya problemi net bir şekilde anlamaya yarayan bir araç olarak tanımlanmaktadır. Kavramsal modelin simülasyon geliştirme yaşam döngüsündeki konumu ile ilgili yararlı bakış açıları sunan yaklaşımlar olmasına rağmen, bunlar bir kavramsal modelin nasıl geliştirileceği ile ilgili ayrıntılı rehberlik bilgisi içermezler. Bu tez çalışması, modelleyicilerin tanımlı bir kavramsal modelleme sürecini izleyerek, alana özgü bir modelleme dilini ve bunu destekleyen bir aracı kullanarak simülasyon sistemleri için daha etkin kavramsal modeller geliştirmelerini sağlayacak bir çerçeve tanımını sunmaktadır. Kavramsal model geliştirme yöntemi adım adım tanımlanmış ve bu adımlarda kullanılan notasyon elemanları açıklanmıştır. Tanımlanan çerçevenin kavramsal modellemeye uygulanabilirliğini değerlendirmek ve beklenen kazanımları geçerli kılmak amacıyla iki durumdan oluşan çoklu-durum çalışması gerçekleştirilmiştir.

Anahtar Kelimeler: Kavramsal Modelleme, Metamodel, Alan Özgü Modelleme, Görev Uzayı Kavramsal Modeli

To my family,

İsmail & Glsm & Kubilay KARAGZ

ACKNOWLEDGMENTS

This thesis is a result of a long journey whereby I have been accompanied and supported by many people. I believe that this is not only an academic requirement but rather a session of a joyful learning process. It is my pleasure now to express my gratefulness to all of the people who were with me during this journey of learning.

The first person I would like to thank is my supervisor Onur Demirörs for his continuous guidance, inspiration and enthusiasm. I have met him in 1999 when I started my MSc. and since then he has made a deep impression on me with his ability in building a perfect balance between providing direction and encouraging self-governance.

I wish to express my warm and sincere thanks to my committee members Semih Bilgen and Altan Koçyiğit for their insightful comments and suggestions. I would also like to thank the committee members Veysi İşler and Kayhan İmre for their valuable comments.

Many thanks go to members of the Gen.Kur.BİLKARDEM especially Ziya İpekkan, Orhun Molyer and Aşkın Erçetin. Without their participation our research project would not have been a success story. I would also like to thank to the members of the KAMA project who developed the KAMA tool which enabled me try many of the ideas easily. Thanks also go to Banu Aysolmaz, H. Halil Kızılca for applying the approach on their own and providing valuable feedbacks and all other people for participating in the case studies and providing valuable comments.

My most sincere appreciation goes to my friends, Ali Yıldız, Utkan Eryılmaz, Oktay Türetken, Özgür Tanrıöver and Ayça Tarhan. They never hesitated to provide support whenever I needed it. Our extensive discussions and their comments were invaluable. My heartfelt thanks go to Ceren; she is the very special person who supported me in the latest tours of this marathon and made me believe that I could accomplish this challenge.

Finally, I am deeply grateful to my parents and my brother for their endless trust, support and encouragement during these years*. My thanks and apologies to others whom I may have inadvertently forgotten to mention.

** Benden her türlü desteği hiçbir zaman esirgemeyen ve bana her zaman inanan Anneme, Babama ve abim Kubilay'a en içten teşekkürlerimi sunuyorum.*

TABLE OF CONTENTS

ABSTRACT	iv
ÖZ	v
DEDICATION	vi
ACKNOWLEDGMENTS	vii
TABLE OF CONTENTS	viii
LIST OF TABLES	xi
LIST OF FIGURES	xii
LIST OF ABBREVIATIONS	xiv
CHAPTER	
1. INTRODUCTION	1
1.1. The Context	1
1.2. The Problem	7
1.3. The Solution Approach	8
1.4. Research Strategy	11
1.5. Organization of the Thesis	11
2. RELATED RESEARCH	13
2.1. The Simulation Development Process and Conceptual Modeling	13
2.2. Conceptual Modeling Approaches, Frameworks and Methods	16
2.2.1. Federation Development and Execution Process (FEDEP)	16
2.2.2. Synthetic Environment Development and Exploitation Process (SEDEP)	21
2.2.3. Conceptual Models of the Mission Space (CMMS)	24
2.2.4. Defense Conceptual Modeling Framework (DCMF)	25
2.3. Base Object Model (BOM)	28
2.3.1. Model Identification	29
2.3.2. Conceptual Model	30
2.3.3. Model Mapping	31
2.3.4. Object Model Definition	31
2.3.5. BOM Integration	32

2.4.	Summary	32
3.	KAMA APPROACH	34
3.1.	Introduction	34
3.2.	The Notation	35
3.2.1.	Design Overview	35
3.2.1.	KAMA and UML	37
3.3.	The Conceptual Modeling Process Definition	38
3.3.1.	Acquire knowledge about the mission space	39
3.3.2.	Analyze the information sources and search for similar conceptual models.....	41
3.3.3.	Define conceptual model elements and develop conceptual model diagrams	41
3.3.4.	Review the conceptual model	42
3.3.5.	Refine the conceptual model.....	43
3.3.6.	Verify and validate the conceptual model.....	43
3.3.7.	Update the conceptual model with respect to the V&V findings	43
3.4.	The KAMA Tool.....	43
3.4.1.	High Level Functional Requirements	43
3.4.2.	Architecture of the Tool.....	47
3.4.3.	Usage of the Tool.....	48
4.	THE KAMA MODELING NOTATION.....	49
4.1.	Overview.....	49
4.1.1.	KAMA Architecture	49
4.1.2.	KAMA Metamodel	50
4.1.3.	How the KAMA Notation is Described	51
4.2.	Foundation Package	52
4.2.1.	Overview.....	52
4.2.2.	Abstract Syntax.....	52
4.2.3.	Class Descriptions.....	55
4.3.	Mission Space Package.....	58
4.3.1.	Overview.....	58
4.3.2.	Abstract Syntax.....	58
4.3.3.	Abstract Syntax.....	59
4.3.4.	Class Descriptions.....	59
4.3.5.	Diagrams.....	75
4.4.	Structure Package.....	77
4.4.1.	Overview.....	77
4.4.2.	Abstract Syntax.....	78
4.4.3.	Class Descriptions.....	78
4.4.4.	Diagrams.....	85

4.5.	Dynamic Behavior Package.....	89
4.5.1.	Overview.....	89
4.5.2.	Abstract Syntax.....	89
4.5.3.	Class Descriptions.....	90
4.5.4.	Diagrams.....	94
5.	APPLICATION OF THE NOTATION.....	96
5.1.	Research Strategy.....	96
5.2.	Multiple-case study design.....	97
5.2.1.	Research Questions.....	97
5.3.	Case Study 1.....	98
5.3.1.	Background.....	98
5.3.2.	Case Study Plan.....	101
5.3.3.	Conduct of the Case Study 1.....	104
5.3.4.	Discussion and Findings for the Case Study 1.....	113
5.4.	Case Study 2.....	116
5.4.1.	Background.....	116
5.4.2.	Case Study Plan.....	117
5.4.3.	Conduct of the Case Study 2.....	120
5.4.1.	Discussion and Findings for the Case Study 2.....	126
5.5.	Lessons Learned and Discussions.....	128
6.	CONCLUSIONS.....	133
6.1.	Contributions.....	133
6.2.	Limitations and Future Work.....	136
	REFERENCES.....	140
	VITA.....	145

LIST OF TABLES

Table 1: Conceptual Modeling Approaches.....	32
Table 2: OMG Metadata Architecture	37
Table 3: Conceptual Model Element Counts	100
Table 4. Schedule Summary (Case Study 1).....	101
Table 5: Mapping.....	113
Table 6. Schedule Summary (Case Study 2).....	117

LIST OF FIGURES

Figure 1: Zeigler's Basic Elements of the Modeling and Simulation.....	2
Figure 2: Relationship between Modeling and Simulation.....	2
Figure 3: KAMA Approach.....	9
Figure 4: KAMA Package Relationships.....	10
Figure 5: FEDEP High Level Process Flow.....	17
Figure 6: Perform Conceptual Analysis.....	20
Figure 7: DCMF Process - Main Parts.....	26
Figure 8: BOM Composition.....	30
Figure 9: Conceptual Modeling Process.....	40
Figure 10: High Level Use Case Diagram.....	46
Figure 11: Architecture of the Tool.....	47
Figure 12: KAMA Architecture.....	50
Figure 13: KAMA Metamodel.....	51
Figure 14: Foundation Package.....	53
Figure 15: Core Metamodel.....	53
Figure 16: Relationships Metamodel.....	54
Figure 17: Behavior Metamodel.....	54
Figure 18: Mission Space Metamodel.....	58
Figure 19: Mission Space Diagram.....	75
Figure 20: Task Flow Diagram.....	77
Figure 21: Structure Metamodel.....	78
Figure 22: Entity Ontology Diagram.....	86
Figure 23: Entity Relationship Diagram.....	87
Figure 24: Command Hierarchy.....	88
Figure 25: Organization Structure Diagram.....	89
Figure 26: Dynamic Behavior Metamodel.....	90
Figure 27: Entity State Diagram.....	95
Figure 28: An example task definition diagram.....	100
Figure 29: Mission Space Diagram for Case Study 1.....	106

Figure 30: Sample Task Flow Diagram for Case Study 1.....	107
Figure 31: Sample Entity Relationship Diagram for Case Study 1.....	108
Figure 32: Sample Entity Ontology Diagram for Case Study 1.....	109
Figure 33: Sample Command Hierarchy Diagram for Case Study 1.....	110
Figure 34: Sample Organization Structure Diagram for Case Study 1.....	111
Figure 35: Sample Entity State Diagram for Case Study 1.....	112
Figure 36: Sample Mission Space Diagram for Case 2.....	122
Figure 37: Sample Task Flow Diagram for Case 2.....	123
Figure 38: Sample Entity Ontology Diagram for Case 2.....	124
Figure 39: Sample Entity State Diagram for Case 2.....	124

LIST OF ABBREVIATIONS

5Ws	Who-What-Where-When-Why
BILKARDEM	Bilimsel Karar Destek Merkezi
BOM	Base Object Model
BPMN	Business Process Modeling Notation
CMMS	Conceptual Models of the Mission Space
CORBA	Common Object Request Broker Architecture
CWM	Common Warehouse Metamodel
C4ISR	Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance
CM	Conceptual Model
CMMI	Capability Maturity Model Integrated
C3I	Command, Control, Communications and Intelligence
DCMF	Defense Conceptual Modeling Framework
DMSO	Defense Modeling and Simulation Office
DBMS	Data Base Management Systems
DIF	Data Interchange Format
FEDEP	Federation Development and Execution Process
FDMS	Functional Descriptions of the Mission Space
FOM	Federation Object Model
GUI	Graphical User Interface
HLA	High Level Architecture
IDEF1X	ICAM Definition Languages

IEEE	Institute of Electrical and Electronics Engineers
KAMA	Kavramsal Modelleme Aracı
KM3	KnowledgeMetaMetaModel
METU	Middle East Technical University
MSM	Mission Space Model
MOF	Meta Object Facility
MVC	Model, View, Controller
MS	Modeling and Simulation
NATO	North Atlantic Treaty Organization
NC3TA	NATO Consultation, Command and Control Agency
OMG	Object Management Group
OMT	Object Model Template
OCL	Object Constraint Language
SEDEP	Synthetic Environment Development and Exploitation Process
SISO	Simulation Interoperability and Standardization Organization
SEDE	Synthetic Environment Development Environment
SE	Synthetic Environment
SPO	Subject-Predicate-Object
SOM	Simulation Object Model
SEI	Software Engineering Institute
SIW	Simulation Interoperability Workshop
SGKS	Sınır Gözetleme ve Kontrol Sistemi
SysML	System Modeling Language
UML	Unified Modeling Language
XML	Extensible Markup Language

CHAPTER 1

INTRODUCTION

Simulation systems have a widespread usage not only in the defense industry but also in other industries such as automotive, production, planning, training etc. Simulations play an important role in understanding the problem domain, doing early analysis and decision making. Simulations are not replications of real world systems but rather limited representations of selected aspects of the real world. It is important to understand and analyze the simulation systems' requirements in an effective way in order to accomplish a successful simulation system development project.

Simulation developers tend to model every issue without worrying too much about their necessity. This tendency increases the complexity of the simulation development projects and at the same time decreases the success rate. Therefore, it is important to define the essential requirements of a simulation system so that the model includes the minimum amount of detail required to achieve the project's objectives. The activity of abstracting the real system's parameters and interactions into a limited and approximated representation that contains the features and behaviors of interest is generally called conceptual modeling.

1.1. The Context

The terms model, modeling and simulation are at the heart of conceptual modeling. According to the Merriam Webster's Online Dictionary, model as a noun has several possible definitions [36]. The most appropriate definitions in our context are "4: a usually miniature representation of something; *also*: a pattern of something to be made" and "5: an example for imitation or emulation". Model or modeling as a verb is defined as "3b: to produce a representation or simulation of" and "4: to construct or fashion in imitation of a particular model".

Simulation is defined as "3a: the imitative representation of the functioning of one system or process by means of the functioning of another". All these terms share the same basis that they are representations of something. While models are generally static, simulations are dynamic and they show the change of a model over time.

Zeigler describes the basic elements of modeling and simulation as the real system, the model and the computer in [75]. He defines modeling as a relationship between the real world and models, whereas simulation is a relationship between the model and the computer.

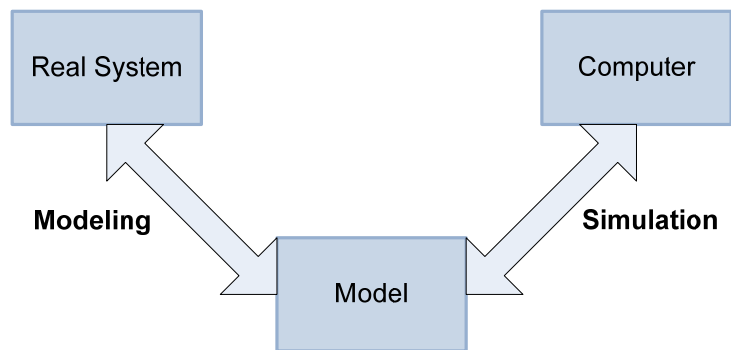


Figure 1: Zeigler's Basic Elements of the Modeling and Simulation

Rothenberg [57] stated that “modeling in its broadest sense is the cost effective use of something in place of something else for some purpose”. Every model is used to represent something in real life and it must be reasonable in terms of cost to use this model instead of using the real life entity for this purpose. Therefore, modeling is suitable for real life tasks which are dangerous, risky or expensive to perform or for situations that are impossible to observe directly. Banks and Carson [6] describe a simulation as a dynamic, digital implementation of a model over time that generates an “artificial history” of the modeled systems. This artificial history is then observed to draw inferences concerning the operation of the real system. They also mention one aspect of simulation development that separates it from systems or software engineering that is the purpose of simulation modeling is to further study the system.

Chapman in [11] defined the relation between modeling and simulation from a different perspective with special emphasis on the conceptual modeling phase. The diagram in Figure 2 is derived from the conceptual modeling framework defined by Chapman and depicts the relationship between modeling and simulation.

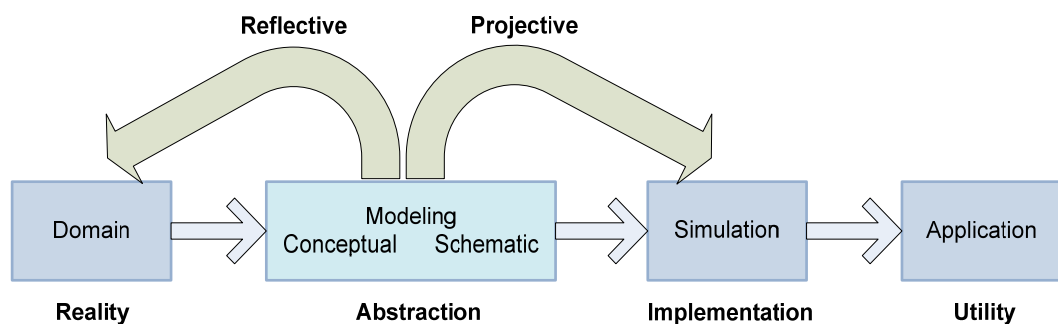


Figure 2: Relationship between Modeling and Simulation

Chapman identifies an important feature of modeling phase as being both reflective and projective. Modeling is reflective because it links the model back to the real world domain that is being represented. And modeling is projective because it links the model forward to its intended application. Pace mentions these features by defining the conceptual model as a bridge between the simulation requirements and the simulation design [45].

Chapman, as well as many other authors uses the term abstraction while defining the modeling phase. Abstraction is an intuitive technique which requires the modeler define the essential features of a real world domain and represent these features in a different form. Simplification, together with abstraction is an important technique used in the modeling phase. These are both analytical techniques that have been used by the scientists in problem solving. Simplification is the removal of unimportant details in order to define simpler relationships. Law [31] and Widman et.al [72] claim that a model should not, actually cannot, represent the real world completely, but rather provide an idealized approximation that will be adequate for the requirements of the simulation

According to Chapman's definition in [11], modeling is an abstraction of reality, which involves conceptual and schematic models. Conceptual models are defined as "the first abstractions of real world domain and descriptions of system objects and their relationships". Conceptual models are implementation independent models that can be used to develop schematic models, which are implementation dependent models that establish a basis for the simulation code. These models are also called as the conceptual models of the mission space and conceptual models of the simulation space respectively. Haddix prefers to use the terms conceptual model and conceptual design for representing this distinction in [21].

The simulation world defines conceptual modeling as a tool that provides a clear understanding of the target domain or problem. Therefore, conceptual models should be abstract and simple representations of the real world, providing effective means for communication. This abstraction involves some level of simplification of reality [75]. In the simulation life cycle, conceptual models should be captured early based on sponsor objectives defining what is intended and then should serve as a "frame-of-reference" for the subsequent development phases. Despite this general goal, there are many perspectives on conceptual models with varying interpretations. Most of the definitions of conceptual model agree that it is related with the early stages of simulation development life cycle. Aside from this general understanding, "conceptual model" is an overloaded term [30].

Merriam Webster's Online Dictionary defines terms concept as "2: an abstract or generic idea generalized from particular instances" and conceptual as "of, relating to, or consisting of concepts". When these definitions are considered together we can come up with the meaning of conceptual model as "a miniature representation of something generalized from particular instances".

Balci [4] states that the early phases of a simulation study are not just visited once, but that they are continually returned to through a series of iterations in the life-cycle of a project. As such, conceptual modeling is not a one-shot process, but one that is repeated and refined a number of times during a simulation study.

Nance [39] separates the ideas of conceptual model and communicative model. The conceptual model exists in the mind of a modeler; the communicative model is an explicit representation of the conceptual model. He also specifies that the conceptual model is separate from model execution. In other words, the conceptual model is not concerned with how the computer-based model is coded. Fishwick [16] takes a similar view, stating that a conceptual model is vague and ambiguous. It is then refined into a more concrete executable model. The process of model design is about developing and refining this vague and ambiguous model and creating the model code. In these terms, conceptual modeling is a sub-set of model design, which also includes the design of the model code.

Robinson [52] offers the following definition for a conceptual model: “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.” This definition highlights the non-software specificity of the conceptual model and the components of such a model.

These definitions are basically from researchers working in non-military simulation world and they are provided to give a complete picture of the simulation domain. Robinson [51] classifies the simulation domain into two groups as military and business-oriented and describes the similarities, and differences between them. He states that the military simulations often entail large scale models developed by teams of developers. There is much interest in model reuse and distributed simulation whereas business oriented simulations tend to be smaller in scale, involve lone modelers normally using a visual interactive modeling tool [49], and the models are often thrown-away on completion of a project. Interest in distributed simulation is moderate, mostly because the scale and life-time of the models does not warrant it [53]. Robinson has concentrated on discrete-event based simulations and pointed out useful issues; however, we think that the labels used for the classification may be misleading. Business-oriented simulation systems such as used in the automotive industry, training simulators such as driver training, simulators that support kids learning science may carry the exact properties stated by Robinson as belonging to military simulations. In this context, the prime interest of this thesis study is in simulation systems such as command and control, training, strategic and tactic level military simulations, and similar systems that include user interaction, larger in scale and possibly distributed. Although we think that our framework can be applied to the discrete-event based simulation systems with slight modifications, we have not yet performed a case study to confirm this thought. The following paragraphs briefly summarize the views of the researchers who are focused on the military simulation domain.

Pace is one of the most influential authors in this debate and defines conceptual modeling as “translating modeling requirements into a detailed design framework from which the software that will make up the simulation can be built” [45]. Pace states that the conceptual model is largely independent of software design and implementation decisions. We can deduce from his view that conceptual models serve as a bridge between requirements analysis and design phases. This bridge should also include definitions for translations which will facilitate transformations between phases. He also defines the ingredients of a conceptual model as assumptions, algorithms, characteristics, relationships and data [47].

Lacy et al. [30] identified many different perspectives on conceptual modeling and tried to reach a consensus on the definition of a conceptual model. The paper describes a plethora of views, but concludes by identifying two types of conceptual model. A *domain-oriented* model that provides a detailed representation of the problem domain and a *design-oriented* model that describes in detail the requirements of the model and is used to design the model code.

Haddix [21] points out that there is some confusion over whether the conceptual model is an artifact of the user or the designer. To clarify this confusion, Haddix defines a conceptual model as “the ultimate definition of the requirements” and adds another definition, conceptual design, which represents the “initial descriptions of the system’s implementation”. He believes that a fully expressed conceptual model should serve as the basis for the verification, validation and accreditation activities. That is, conceptual models should be transformed into more detailed models which will then be used for verification and validation activities.

FEDEP (Federation Development and Execution Process) describes the conceptual model as “an implementation-independent representation that serves as a vehicle for transforming objectives into functional and behavioral capabilities” [24]. This statement includes many essential aspects of conceptual models that can be listed as: independency from specific platform requirements, being well-structured in order to provide effective communication, including transformation definitions between objectives and capabilities and providing traceability

Sargent views conceptual model as “a mathematical, logical, or verbal representation of a problem entity (domain) for a particular study” [60]. He focuses on validation of computerized models and simulations using conceptual models. Accordingly, he claims that conceptual models should be structured enough to provide means for validation.

Davis describes conceptual modeling as “laying out the theories and algorithms formally to create implementation independent specifications” [14]. He emphasizes the importance of separating model design from detailed design and states that conceptual models should be formal representations and should not include implementation details. Davis also suggests that conceptual models should be developed as staged models, such that model design is used to capture “what” will be developed and detailed design includes “how” the model will be developed.

DMSO (Defense Modeling and Simulation Office) one of the pioneering organizations in modeling and simulation; defines a conceptual model as a representation of the combined view of user's and developer's that is used to describe the statement of the content. Any assumptions, limitations, and algorithms should also be included in this representation [15]. DMSO extends this definition and introduces the term CMMS (Conceptual Models of the Mission Space) which can be defined as "simulation-implementation-independent functional descriptions of the real world processes, entities, and environment associated with a particular set of missions" [63]. In other words, conceptual models should propose a common language for both users and developers to effectively communicate and should also be structured so that it includes logic and algorithm rules.

Similarly, Johnson states that conceptual models should provide the necessary information and methods for constructing modeling and simulation representations and should capture the features of the problem space and the target domain [26]. He also suggests that the conceptual models should represent a simulation-neutral view of the real-world, acting as a bridging function between the domain expert (war fighter in the CMMS context) and the simulation developer. A conceptual model captures "what" a model should represent based on the features of the problem space and target domain and should be independent from specific simulation attributes.

Sargent and Haddix state that conceptual models will be mainly used for verification and validation purposes, therefore their definitions require utilization of formal methods. DMSO and Johnson states that conceptual model should be used as a means for understanding the problem domain. Since it is used as a part of problem analysis, it should provide methods that can effectively be used both by users and developers. FEDEP, Davis, and Pace focus on the simulation-implementation independency of the conceptual models and the necessity of transformation definitions. Main idea in their definitions is dividing the development of the model into stages where the level of abstraction is decreased and level of detail is increased at subsequent stages, and providing definitions to accomplish the transformations.

In this thesis we interpret the conceptual model as;

- a simplified representation of the real system,
- a specification of the problem including limitations and assumptions,
- a common language for both users and developers,
- an implementation independent representation,
- an artifact that can be used for validation and verification activities,
- an output of an iterative and repetitive conceptual modeling process, with the model being continually revised throughout the modeling study.

1.2. The Problem

Conceptual models are usually underused in simulation development. Borah states in [7] that “the community of engineers engaged in simulation development need to step back and develop a clear understanding of the role of conceptual modeling in the simulation development process. Once the role of conceptual modeling is established then the requisite elements of a sound conceptual model can be defined as a part of ongoing discussions of sound simulation development practices”. To understand the situation, we need to ask the following question; what is the role of a conceptual model in simulation development?

Zeigler states that one of the important aspects in modeling is communication [75]. He states that “the long-term contribution of any modeling effort lies in the benefits it affords, either by direct use or by guidance for further development, to science and industry”. Accordingly the model should help modelers, domain experts and the users of the model a clear understanding of the context and to visualize it within the framework of their prior conceptions about how things work. This is what we should expect from a conceptual model in the simulation development life cycle.

Most of the ambiguity concerning a conceptual model is caused by the employment of many different disciplines in a simulation development project such as modeling and simulation, systems engineering and software engineering. These disciplines interpret a conceptual model from different points of views, which result in different expectations. What we need is a high-level approach to conceptual modeling which covers the requirements of all these related disciplines and defines the interactions among them. An informal conceptual model would be sufficient in the early simulation development phases, which should then be elaborated and more formally represented to be useful for systems and software developers. The initial conceptual model could be effectively used for validation purposes so that it can form a sound basis for a detailed conceptual model.

Although many of the researchers acknowledge the importance of conceptual modeling in the simulation development lifecycle, there are very few studies and publications on the conceptual model development methods and notations. Most of the time the information required for developing a mission space conceptual model is written in free text by the subject matter experts. Pace [48] describes four approaches for documenting the conceptual models; which are ad hoc method, design accommodation, CMMS paradigm, and scientific paper approach. All of these approaches utilize free text notation for representing the conceptual models.

The free text notation causes ambiguous and recurrent definitions, is difficult to interpret by machines, and does not provide adequate guidance to the modeler [66]. There are also more formal notations used for conceptual modeling, such as UML (Unified Modeling Language), BPMN (Business Process Modeling Notation) and IDEF1X [58], [10] and [23]. Each of these methods provides different approaches; UML follows a more object-oriented approach, BPMN is more

process-oriented, IDEF1X is a more data-oriented approach. However, these are general modeling notations which do not necessarily address the specific requirements of the mission space conceptual modeling.

It should be noted that although some tools and techniques can be provided and successfully employed for effective conceptual modeling, the usefulness of a conceptual model is related with the skills and experience of the modeler in charge. As Shannon has written in [62], “The art of modeling can be mastered by those who possess the necessary skills of ingenuity, insight, and resourcefulness, as well as an extensive exposure to the systems and physical phenomena they are trying to model. There is no hard and fast rule about how the problem is originally formulated, i.e., how one looks at it in the first place. There are no magic formulas for deciding what should be included in the model in the form of variables and parameters, descriptive relationships and constraints, and criterion for judgment of effectiveness. Remember that nobody solves the problem; rather, everybody solves the model that he has constructed of the problem. This concept helps to keep the model and the art of modeling in the proper perspective.”

As a result of the essence of the “art of modeling”, there would not be a single conceptual model developed in a one shot process for any simulation system. Conceptual models are “living” documents that are iteratively developed, and evolve as required by the simulation development process from an informal description to a more formal description to communicate between the diverse groups (sponsors/users, modelers, systems engineers, and software designers) participating in a simulation development effort.

Since conceptual modeling is mostly related with the problem definition phase, late removal of any defect injected at this phase will cost too much effort and time sometimes even leading to unrecoverable situations. Conceptual models can form a basis for verification and early validation activities that may dramatically decrease the number of defects injected at the early phases of the simulation development life cycle.

The problem statement inspiring this thesis can be summarized as follows:

There is not a widely accepted conceptual modeling approach that includes a notation capable of representing a wide range of simulation systems, providing an effective bridge between the simulation requirements and simulation design and forming a basis for early validation of the simulation systems.

1.3. The Solution Approach

In order to resolve the defined problems in the previous section, we propose an approach for developing conceptual models of the mission space. Our approach includes 3 major components, which are, the conceptual modeling notation, process definition and a software tool that supports the notation and the process as shown in Figure 3.

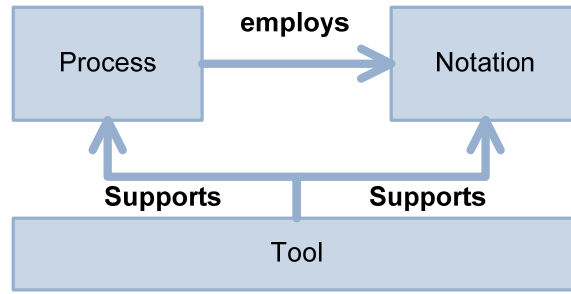


Figure 3: KAMA Approach

Although the focus of the thesis is on the notation component, we knew that notation solely would not be adequate for effective conceptual modeling. Therefore, we defined a process for the purpose of properly performing the case studies in Section 3.3. Additionally, a tool was developed within the scope of a research and development project. Our major support for the development efforts was during the requirements elicitation phase. The high level requirements of the tool are presented in Section 3.4.

As the first step of our solution approach, we performed a survey on the related literature and investigated existing conceptual models. We identified the essential features of a conceptual model. Then we examined possible notations that could be used for representing conceptual models, which are explained in Section 2. Based on our examinations, we proposed the KAMA notation, which is a domain specific modeling language designed for conceptual modeling. The modeling language consisted of domain specific graphical notation so that it could easily be understood and used by the modelers and the domain experts. It is also based on UML so that it would be easier to transfer the knowledge in the conceptual model to the simulation design activities. The use of UML also allowed us define a domain specific notation by reusing the existing infrastructure.

KAMA notation is composed of metamodel elements that are represented as graphical diagram elements. The 7 types of diagrams are used to represent both the structural and behavioral aspects of a conceptual model. The available metamodel elements within each diagram are explained and any semantic rules associated with these metamodel elements and their relationships are provided. The relationship of KAMA with UML results in a semi-formal notation.

KAMA notation is composed of 4 major packages, which are named as Foundation, Mission Space, Structure and Dynamic Behavior packages. This package structure is inherited from the UML Infrastructure document and used as an effective representation to show the logical groupings within the metamodel. Another benefit of this structure is exposed by the use of the Foundation package. This package abstracts all of the metamodel elements that are directly inherited from UML, therefore it can be thought of as an interface package. The other packages include metamodel elements that inherit from the metamodel elements in the Foundation package

as shown in Figure 4. Mission Space and Dynamic Behavior packages depend on the metamodel elements in the Structure package.

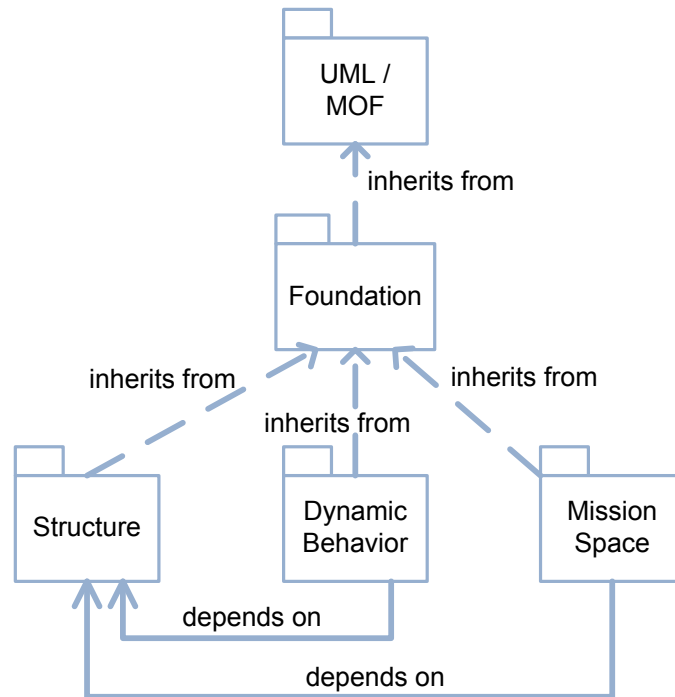


Figure 4: KAMA Package Relationships

All of the diagram elements in the KAMA notation are titled in order to map the specific terminology of the domain experts and modelers, thus resulting in a domain specific modeling notation. The modelers are not required to have technical expertise in other domains such as object modeling, systems modeling, data modeling etc. Nevertheless, knowledge in these domains would be an advantage. By providing an abstraction layer that is close to the modelers and domain experts, we aimed to decrease the number of unrelated concepts and let the modeler focus on the essential aspects of the mission space.

The modeler is guided by the process definition provided in Section 3.3; however it is not mandatory to follow the exact order of activities. The modeler has the freedom to follow any process keeping in mind the pre-requisite relationships among the diagrams such as an entity state diagram requiring an entity to be defined or a task flow diagram requiring the existence of a mission or a task.

Although not mandatory for utilization of the notation, usage of the KAMA tool enables additional functionality such as effectively manage, report, verify and navigate among the model elements.

1.4. Research Strategy

The primary research objective of this study is to develop a notation for the use of modelers and domain experts in conceptual modeling. We performed a survey on the related literature and investigated existing approaches. Based on our findings, we developed a domain specific modeling language. In order to identify improvement opportunities and observe the applicability of the notation, a multiple-case study involving two case studies was conducted.

The first case study was conducted as part of a research project that was realized by a consortium including two companies and the METU (Middle East Technical University). The objective of the study was to examine the drawbacks of the notation, look for improvement suggestions and identify the high level requirements for the tool that is going to be developed for supporting the conceptual modeling activities. The study has been performed by three researchers, two of which developed the conceptual model and the other one verified.

The second case aimed to evaluate the applicability of the notation and observe its limitations in an industrial setting. The study has been conducted in a systems/software development company. The modelers in charge of developing the conceptual model and the domain experts who will verify the conceptual model took a training course on conceptual modeling. The modelers were also trained on the simulation domain by the domain experts. The modelers developed the conceptual model iteratively and we reviewed the conceptual models before being submitted to the customer. The domain experts who were also representatives of the customer were involved in the verification process through the joint reviews. We observed the execution of the conceptual modeling process, usage of the notation and interviewed with selected modelers and domain experts in order to evaluate the notation.

1.5. Organization of the Thesis

The thesis is organized as six chapters.

Chapter 1 briefly summarizes the context, the problem definition, the solution approach and the research strategy.

Chapter 2 provides related research on the simulation development lifecycle, conceptual modeling notations and modeling and simulation.

Chapter 3 provides a high level view of our approach, which is composed of a notation, process definition and a tool. Since our focus is on the conceptual modeling notation, we provided an overview of the design of the notation and its relationship with UML. The process definition is briefly summarized and the high level requirements of the tool are depicted as a use case diagram and major features are explained.

Chapter 4 describes the modeling notation in detail. The architecture of the notation is provided as package diagrams and a metamodel diagram is given, which shows all of the metamodel elements and their relationships. The sections of this chapter are organized according to the packages. Each section includes a metamodel diagram and detailed definitions of the metamodel elements in that package. The diagrams related to each package are explained by examples at the end of each section.

Chapter 5 presents the implementation of the notation in two case studies. The chapter presents the research strategy, case study design, case study plans and findings of the case studies. The details of the implementations, their results as well as the lessons learned and discussions are provided for each case study.

Chapter 6 presents the conclusions obtained and summarizes the contribution of this research. Possibilities for further investigation are also provided in this chapter¹.

¹*The terms conceptual model and conceptual models of the mission space are used interchangeably throughout the thesis. Unless otherwise specified, both of these terms represent the conceptual models of the mission space.*

CHAPTER 2

RELATED RESEARCH

Conceptual model can be used as a tool in the problem analysis and requirements analysis phases of simulation development. Since it is related with the problem domain, it should be independent of the design and implementation issues. It can also be used as a basis for the verification and validation activities. In order to establish this basis, effective frameworks and representation techniques should be used. The following section summarizes the related literature about these topics.

2.1. The Simulation Development Process and Conceptual Modeling

Most of the researchers agree that it is essential to develop conceptual models at the early stages of simulation development life cycle [4], [15], [46], [24], [37]. Aside from this general understanding, Lacy states that “conceptual model” is an overloaded term and describes “well-documented and widely presented perspectives” on conceptual modeling [27]. All of the perspectives compared by Lacy are based on the military simulation domain. Robinson attempts to clarify the meaning and usage of the conceptual models both in the military and business-oriented simulation domains with a focus on the operational research studies [55].

Conceptual modeling is related with the problem domain and can be used as a tool to identify the components of the problem in terms comprehensible to both modelers and subject matter experts [60], [26], [56], [37]. Robinson highlights the importance of following a structured problem definition process in order to define a combined view effectively. He describes possible scenarios in understanding the problem situation and offers simple methods. In cases where the “problem situation is neither well understood nor [clearly] expressed”, he suggests that formal problem structuring methods should be utilized [56]. Balci and Nance [5] defines a methodology for problem formulation in simulation. Mojtahed et.al [37] takes this approach one step further and treats the conceptual analysis phase as a knowledge engineering activity.

Problem definition in simulation development is a part of the requirements analysis phase; therefore conceptual models are the products of this phase. Specifications, assumptions and

constraints related with the domain of interest should be included in the conceptual model. These specifications may include the entities, tasks, actions and interactions among the entities, which will form a basis for the design phase [15]. Pace [45] describes the conceptual model as a bridge between requirements analysis and design phases. Since the boundaries of these phases cannot be sharply defined, there is confusion over whether the conceptual model is a product of the user or the designer [21]. In order to reduce this confusion, Haddix defines a conceptual model as “the ultimate definition of the requirements” and makes another definition, conceptual design, which represents the “initial descriptions of the system’s implementation”. However, SISO [64] conflicts with these definitions in its BOM (Base Object Model) standard by stating that the BOMs are defined to “provide an end-state of a simulation conceptual model and can be used as a foundation for the design of executable software code and integration of interoperable simulations”.

Being a product of the requirements analysis phase, conceptual models should be independent of the software design and implementation decisions [63], [46], [24]. This aspect of the conceptual model is based on a software development viewpoint. Johnson [26] introduces a slightly different aspect of the conceptual model as providing a “simulation-neutral view of the real world”. He suggests that the simulation specific attributes, even if they are not related with the design phase, should be kept out of a conceptual model. Thus, the conceptual model should include the specifications of a simulation system and it can be realized by different simulation implementations.

It is an established practice in the software engineering field to initiate verification and validation activities as early as possible in the software development life cycle [12]. Software requirements specification is used for ensuring that the developers are producing what the customer really wants. Similarly, early validation of a simulation system is essential for the success of a simulation development project. Conceptual models can be used as a basis for the verification, validation and the accreditation activities [60], [21]. Sargent underlines that the conceptual model should be structured enough to provide means for validation. However, a complete validation will be possible after the simulation system has been completed. Any defects found during verification and validation activities should be corrected by revisiting the prior phases including the conceptual modeling phase. Hence, conceptual modeling is not a one shot process but rather an iterative one that should be performed in many cycles throughout a simulation study [4], [73].

The simulation community produced various framework definitions covering the abovementioned perspectives on conceptual modeling. The CMMS (Conceptual Models of the Mission Space) project originated by the U.S. Department of Defense is one of the first initiatives providing detailed guidance on conceptual model development activities. Prior to this work, there were detailed framework definitions [75], [62], [5] regarding the whole simulation development lifecycle, but with less guidance on the conceptual modeling phase. The latest framework definition, which belongs to Robinson [56] includes detailed definitions for developing conceptual

models. Robinson's work is distinct from others that it is based on business-oriented simulation domain rather than the military domain.

The conceptual modeling phase begins with understanding the problem situation and defining the context [15], [45], [24], [37], [56]. The context definition phase includes determining the simulation objectives, identifying the authoritative information sources and defining the assumptions and constraints about the simulation system. This context definition is called as mission space [15], [45], [24], [37]. Pace [48] underlines the importance of recording a history of changes made on the assumptions and constraints. The next phase is developing the content, which includes defining the entities, tasks, interactions, inputs, outputs and relationships among all these elements. The output of this phase may be structured information represented in plain text, tables or diagrams that will provide a basis for understanding, sharing and reviewing.

Although a well-defined representation technique is necessary, some frameworks prefer not to impose a specific notation for representing the conceptual models [24], [48], [56]. Pace [48] describes four approaches for documenting the conceptual models, which are ad hoc method, design accommodation, CMMS paradigm, and scientific paper approach. These approaches utilize free text notation for representing the conceptual models. Robinson [56] uses tabular structure for representing the conceptual models, while also mentioning the usefulness of the diagrammatic representation. However, the free text notation causes ambiguous and recurrent definitions, is difficult to interpret by machines, and does not provide adequate guidance to the modeler [66]. Mojtahed et.al [37] introduced the KnowledgeMetaMetaModel (KM3) as both a language and a tool to construct conceptual models. They state that their intent in developing KM3 was not to construct a unified model description language, but rather provide a way to "capture system structures and behaviour in an object-oriented and rule-based way". The abstract syntax of KM3 is defined as a class-diagram and the concrete syntax is textually represented. Mojtahed et.al [37] state that various graphical representations can be used; however they do not define a method explaining how to associate these graphical representations to the concrete syntax.

Shannon [62] claims that modeling is more of an 'art' than 'science', therefore it is generally assumed difficult to define methodical ways to develop conceptual models. However, artistic skills improve better by the reduction of routine work as a result of following disciplined and systematic methods. The evolution of other engineering fields, such as systems and software shows that defining and using well-defined modeling notations, following defined processes and utilizing software tools help build an effective discipline. Conceptual modeling frameworks should include these three elements in order to provide a complete solution.

Robinson [56] provides a range of other methods and notations that are used to represent conceptual models. Process flow diagrams, activity cycle diagrams, petri-nets, event graphs, digraphs, The UML (Unified Modeling Language), object models and simulation activity diagrams

are examples to these notations. UML is one of the most popular modeling languages used for analysis and design of software. As a result of its success in the software modeling field, UML has been utilized in domains other than software such as systems modeling, business process modeling, data modeling and software process modeling. UML and SysML (System Modeling Language) which is an extension to UML have also been used for developing conceptual models [50], [19]. However, UML has not been extensively used in the conceptual modeling field.

2.2. Conceptual Modeling Approaches, Frameworks and Methods

This section defines some of the existing approaches, frameworks and methods related with conceptual modeling. Although these approaches, frameworks and methods may include extensive information, this section focuses on the conceptual modeling related parts of them. The following sub-sections explain FEDEP, SEDEP, CMMS (FDMS), DCMF and other related approaches.

2.2.1. Federation Development and Execution Process (FEDEP)

FEDEP (Federation Development and Execution Process) [24] defines the processes and procedures that are intended for the HLA (High Level Architecture) [25] users. It does not aim to define low level management and systems engineering practices native to HLA² user organizations but rather aims to define a higher-level framework into which such practices can be integrated and tailored for specific use. Single simulation in a federation is called a federate and federation can be defined as a set of simulations. Therefore, the federation conceptual model includes the simulation conceptual model. The purpose of FEDEP architecture is to facilitate interoperability among simulations and promote reuse of simulations and their components.

FEDEP provides process definitions that encourage the use of conceptual models in the simulation development life cycle. These definitions are originated by the Department of Defense and then standardized by IEEE as a recommended practice. The designers of HLA identified the need for a flexible process according to which HLA applications will be developed. The main idea was avoiding unnecessary constraints on the construction and execution processes because these processes could vary significantly within or across user applications. However, it is then realized that it is possible to define a process at a more abstract level which should be followed by all HLA

² *HLA is general purpose architecture for distributed computer simulation systems. HLA includes Interface Specification, Object Model Template, and HLA Rules for defining an architecture that will allow computer simulations communicate to other computer simulations regardless of the computing platforms.*

developer organizations. Figure 5 shows the top level view of this FEDEP process flow which includes seven steps. Among these, the second step entitled “Perform Conceptual Analysis” is within the scope of this thesis and therefore it will be explained in detail. The other steps are briefly explained and provided only to depict a complete view.

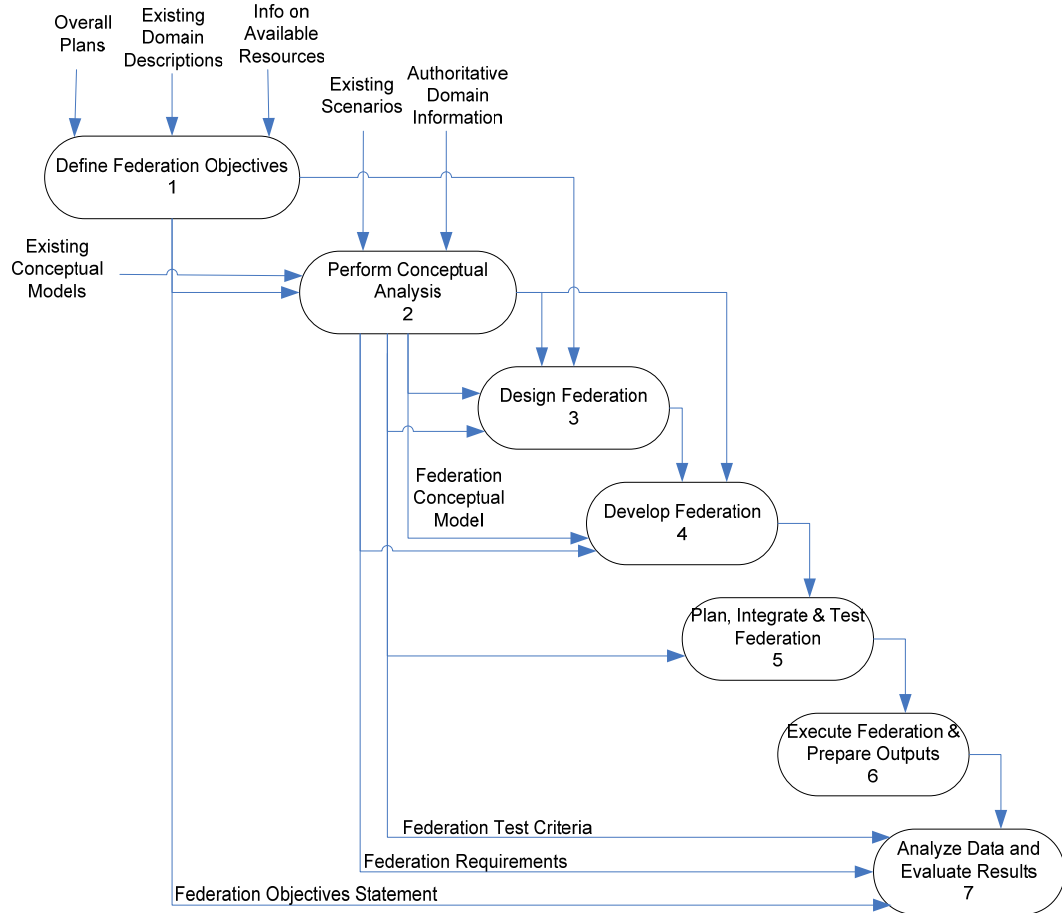


Figure 5: FEDEP High Level Process Flow

Step 1 – Define federation objectives: The federation user, the sponsor, and the federation development team define and agree on a set of objectives and document what must be accomplished to achieve those objectives. This step includes two key activities; a) identify user/sponsor needs and b) develop objectives. The aim of the first activity is to develop a clear understanding of the problem to be addressed by the federation; therefore it is very essential for developing a valid conceptual model. This understanding is generally recorded as a “statement of needs” document which may vary widely in terms of scope and degree of formalization. Despite this variance, it is generally accepted that this document should at a minimum include; high-level descriptions of critical systems of interest, expectations about required fidelity and required behaviors for simulated entities, key events that must be represented in the federation scenario and output data requirements. In addition, the needs statement should indicate the constraints related

with the development of the federation such as funding, personnel, tools, facilities, due dates, non-functional requirements etc.

The needs statement is then analyzed and refined into a more detailed set of specific objectives for the federation. The aim of this activity is developing more concrete, complete and measurable goals and also performing an early assessment of feasibility of the federation and risks related to development. This activity requires a close collaboration between the federation sponsor and the federation developer.

Step 2 – Perform conceptual analysis: The purpose of this step is to develop an appropriate representation of the real world domain based on the characteristics of the problem space defined in the previous step. The federation objectives are transformed into more concrete federation requirements. This step includes three key activities; a) develop scenario, b) develop federation conceptual model, c) develop federation requirements. These key activities are explained in detail in section 2.2.1.a.

Step 3 – Design federation: The purpose of this step is to produce the design of the federation that will be implemented in Step 4. This step includes three key activities; a) identify existing federation participants (federates) that are suitable for reuse, b) create new federates and federate components if required and allocate the required functionality to federates and c) develop a detailed plan for federation development and implementation. The conceptual model developed in the previous step may be utilized to identify reusable federates and/or federations. In addition to the technical reusability parameters, managerial parameters such as availability, security, maintainability should be considered. Although HLA provides a common architecture for developing reusable federates at the technical level, interoperability at the conceptual model level is an important issue that should also be considered [69].

Step 4 – Develop federation: The purpose of this step is to develop the FOM (Federation Object Model), modify federates if necessary, and prepare the federation for integration and test. This step includes four key activities; a) develop the FOM, b) establish federation agreements, c) implement federate designs, and c) implement federation infrastructure. FOM is a specification defining the information exchanged at runtime to achieve a given set of federation objectives. This includes object classes, object class attributes, interaction classes, interaction parameters, and other relevant information. After the federation requirements are defined and allocated to federates, FOM is developed to support the data exchanges required among the federates to meet the federation objectives.

Although FOM defines the set of data that is exchanged among the federates, there may be other operational agreements that must be established. Such agreements are necessary to establish a fully consistent, interoperable, distributed simulation environment. For example, an agreement should be established on utilizing the conceptual models for gaining an understanding and agreement on

the behavior of all federation objects. It is essential for the success of federation development that all parties talk in the same language and understand each other.

Step 5 – Plan, integrate, and test federation: The purpose of this step is to perform all necessary federation integration and testing activities, and test the federation prior to execution. This step includes three key activities; a) plan execution, b) integrate federation and c) test federation. The federation execution environment is fully defined and an execution plan is developed in this step. All of the federation participants are brought into a unifying operating environment and all of the federation participants are tested to show that they can interoperate to the degree required to achieve federation objectives.

Step 6 – Execute federation and prepare outputs: The purpose of this step is to execute the federation and to pre-process the output data from the federation execution.

Step 7 – Analyze data and evaluate results: The purpose of this step is to analyze and evaluate the data acquired during the federation execution and to report the results back to the user/sponsor.

a. Conceptual Modeling in FEDEP

The second step of FEDEP includes development of the conceptual model. FEDEP defines the conceptual model as “An abstraction of the real world that serves as a frame of reference for federation development by documenting simulation-neutral views of important entities and their key actions and interactions.” And the federation conceptual model is defined as “the document that describes what the federation will represent, the assumptions limiting those representations, and other capabilities needed to satisfy the user’s requirements. Federation conceptual models are bridges between the real world, requirements, and design.”

In order to comply with these definitions, this step of the FEDEP process begins with developing federation scenarios that are based on the federation requirements. The relationship between the activities of this step, the consumed inputs and produced outputs are depicted in Figure 6. Federation scenarios define the boundaries of conceptual modeling activities. Authoritative information sources should be identified prior to scenario construction. A federation scenario includes “the types and numbers of major entities that must be represented by the federation, a functional description of the capabilities, behavior, and relationships between these major entities over time, and a specification of relevant environmental conditions that impact or are impacted by entities” in the federation. Initial conditions (e.g., geographical positions for physical objects), termination conditions, and specific geographic regions should also be provided.

FEDEP does not impose use of any tool for scenario generation but lists the possible styles as textual scenario descriptions, event-trace diagrams and graphical illustrations of geographical positions of the physical objects. This scenario or scenarios can be interpreted as a high level view of the federation and includes the entities, behaviors, and events that need to be represented in the

federation scenario, the geographic areas of interest, the environmental conditions of interest and the initial and termination conditions for the scenarios.

Following the scenario definition, a conceptual representation of the problem space is developed based on the interpretation of the user needs and federation objectives. The federation conceptual model provides an implementation-independent representation that serves as a vehicle for transforming federation objectives into functional and behavioral descriptions for system and software designers. The model also provides a crucial traceability link between the stated federation objectives and the eventual design implementation. This model can be used as a basis for the latter federation development steps and can highlight problems early through a validation process that involves the user/sponsor.

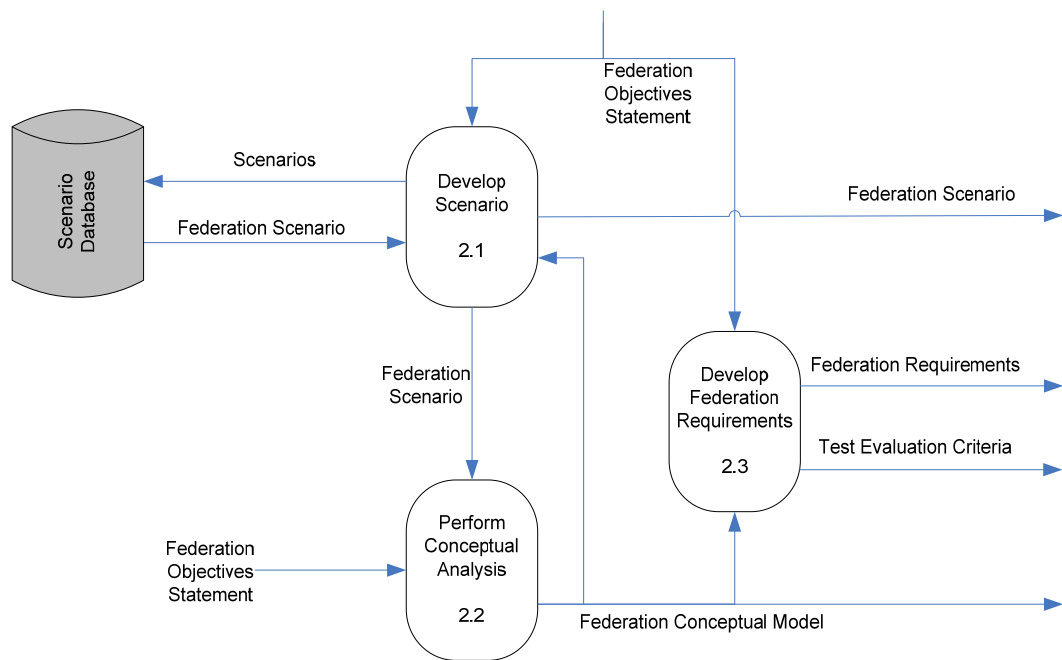


Figure 6: Perform Conceptual Analysis

The federation conceptual model development activities start with defining entities, actions and the assumptions and limitations regarding these. Defining entities include identifying static and dynamic relationships between entities, and identifying the behavioral and transformational (algorithmic) aspects of each entity. Static relationships may include association that shows a simple relationship among any entity, generalization that shows is-a relationship, and aggregation that shows part-whole relationship. Dynamic relationships may include temporally ordered sequences of entity interactions with associated trigger conditions. The characteristics of entities and interaction parameters used among entities may also be defined during this activity. FEDEP does not impose any specific notation for representing conceptual models, but states that it is

important that this notation should be appropriate to develop conceptual models that provide insight into the real world domain.

Since the objective of conceptual model is representing the real world domain, it should be reviewed by the user/sponsor to ensure the adequacy of domain representation. As most of the researchers have pointed out, conceptual modeling is not a one-shot activity but an iterative one, therefore any changes caused by these reviews should be performed under control. As the conceptual model evolves, it is transformed from a general representation of the real world domain to a more specific expression of the capabilities of the federation as constrained by the federates and available resources.

Although FEDEP definitions do not explicitly merge simulation and software development processes, the link is built through federation conceptual model and federation requirements. Detailed federation requirements are developed as the federation conceptual model is evolved. These requirements, should be traceable back to the federation objectives, should be testable and should provide enough guidance needed to design and develop the federation. The federation requirements should consider the specific execution management needs of all federation users, and explicitly address the issue of fidelity, so that fidelity requirements can be considered during selection of federation participants. In addition, any design or development constraints on the federation should be refined and described to the degree of detail necessary to guide federation implementation.

FEDEP is one of the first initiatives that shows the position of conceptual modeling in the simulation development lifecycle, together with its boundaries, inputs and outputs. However, it does not provide enough detail on conceptual modeling to enable modelers develop conceptual models. Not imposing a modeling notation makes the FEDEP flexible, but on the other hand decreases the chances of use. In this form, it will be very useful at the strategic level but not adequate at the tactical level. The modelers need more detailed guidance, sample conceptual models, need to see the concrete relationship between conceptual model and simulation requirements and design and data to show the return on investment resulting from the utilization of a conceptual model.

2.2.2. Synthetic Environment Development and Exploitation Process (SEDEP)

SEDEP (Synthetic Environment Development and Exploitation Process) is developed based on FEDEP with the purpose of providing additional information to the synthetic environment community. The synthetic environment term is used as a more generic definition for federation. SEDEP aims to cover both military and civil applications, and tries to extend FEDEP by losing its association with the HLA [17]. SEDEP is a product of the EUCLID RTP 11.13 initiative [22].

SEDEP adds a step at the start of the FEDEP process for analyzing the user needs in order to identify existing synthetic environment assets that could be re-used and to develop the project plan. Another step is added at the end of the process for analyzing the execution outputs and evaluating the results. One of the major differences from the FEDEP is that it generalizes the terminology so as not specific to the HLA. SEDEP also introduces a repository to provide a means for storing information and exchanging data among different process activities, probably supported by different tools.

One of the main deliverables of SEDEP is the development of a prototype SEDE (Synthetic Environment Development Environment) that is comprised of five main components as listed below [17];

- SE Process: Definition of activities and tasks for creating and utilizing a synthetic environment
- SE Management Tool: Management and configuration control of synthetic environment data
- SE Tools: Support for various activities defined by synthetic environment process
- Repository: Facility for exchanging data between synthetic environment tools and for storing data
- SE Knowledge Base: Information for users about available synthetic environment assets and best practices for creating and utilizing synthetic environments

SEDEP is intended to support all of the stakeholders who are involved in the execution of the process steps. The main roles are; problem setter, problem solver, synthetic environment developer, synthetic environment operator, exercise controller and infrastructure controller. SEDEP is defined by following a similar structure to the FEDEP definition. There are 8 steps, each of which may be detailed by using activities. [22].

- Step 0 - Analyze user needs
- Step 1 - Define federation user requirements
- Step 2 - Define federation system requirements
- Step 3 - Design Federation
- Step 4 - Implement Federation
- Step 5 - Integrate & Test Federation
- Step 6 - Operate Federation
- Step 7 - Perform Evaluation

The first 3 steps are of interest to our study; therefore we will focus on these steps.

The purpose of Step 0 is to “define and analyze user needs in order to understand what results the synthetic environment should provide” [17]. It includes planning of the particular SEDEP iteration and also tailoring of the process, if required. Step 0 includes analyzing the user input step where the problem setter and the problem solver ensure that all the relevant inputs to the process have been identified, obtained and understood. Problem solver tries to identify whether there are any data similar to the current SEDEP iteration as the next step. The third step aims to develop a project plan. This step is an addition to the FEDEP definition.

The purpose of Step 1 is to define federation user requirements, scenario and evaluation objectives. The evaluation objectives are major sources for determining the capability the federation and needs to be considered early in the process. This step maps to the Define Federation Objectives step of the FEDEP. The federation user requirements are analyzed, detailed and quantified using measurable criteria. Then the problem setter defines the functional specification of the federation scenario, which may reuse existing scenarios. Then the Problem Setter and Problem Solver define evaluation objectives, which include the behaviour, skills, characteristics, tactics, procedures, functionality, etc. that should be analysed and evaluated based on the user requirements.

The purpose of Step 2 is “to define specification for a federation that will satisfy the user requirements” [22]. This step corresponds to the Define Federation Conceptual Model step of the FEDEP. As the first activity, a conceptual model of the federation describing all the static and dynamic relationships between the entities that are required to implement the scenario is developed. The conceptual model development activity is generally performed iteratively. Federation system requirements and the evaluation criteria are developed in the subsequent phases.

Lemmers defines the conceptual model as a conceptual representation of the intended problem space produced by the problem solver based on her interpretation of the constraints, user requirements and the scenario definition [32]. Conceptual model includes the elements and the relationships among these elements that are required to represent the mission space. This model must be independent of any simulation specifications or tools. Lemmers also suggests that the conceptual model should be reviewed before proceeding to the next step.

Lemmers proposes a UML based conceptual model format that includes two kinds of customizations; specific stereotypes and well-formedness rules. The stereotypes are SEelement, SEdata and SRelation. SEelement inherits from the UML class and is used to represent an element that participates in the specified synthetic environment. SEelement classes have the capability to communicate with other SEelement classes. SEelement classes may be composed of other SEelement classes. SEdata is a class that represents a data type that can be exchanged by SEelements.

For each type of relation between SElements a SRelation Model is defined. Most common SRelation types are; generalization association that is used to facilitate modeling of commonalities by means of an abstraction hierarchy and aggregation association that is used to facilitate composition of SElements into more complex SElements. Well-formedness rules are used to properly define the relationships among SElements.

Following the conclusion of RTP 11.13, further development of SEDEP has stopped. However, the SEDEP v2.0 is publicly available and synthetic environment developers are strongly encouraged to use the SEDEP.

2.2.3. Conceptual Models of the Mission Space (CMMS)

The United States' Defense Modeling and Simulation Office has initiated a project to promote the use of conceptual models and has led an effort to provide an integrated framework and toolset for developing CMMS (Conceptual Models of the Mission Space) [15]. The main cause for this effort was the interoperability problems among many simulation projects developed in many different platforms. DMSO defined the CMMS as “the first abstractions of the real world that serve as a frame of reference for simulation development by capturing the basic information about important entities involved in any mission and their key actions and interactions”. CMMS provides;

- a disciplined procedure by which the simulation developer is systematically informed about the real world problem to be synthesized,
- an information standard the simulation subject matter expert employs to communicate with and obtains feedback from the military operations subject matter expert,
- a real world, military operations basis for subsequent, simulation-specific analysis, design, and implementation, and eventually verification, validation, and accreditation/certification, and
- a singular means for identifying re-use opportunities in the eventual simulation implementation by establishing commonality in the real world activities.

The objectives of the CMMS are to “enhance interoperability and reuse of models and simulations by accessing descriptions of real-world operational entities, their actions, and interactions through the identification of authoritative sources of information for models and simulations and the integration of information from independent knowledge acquisition sources” as specified in [15].

CMMS defines mission spaces for each mission area and aims to develop conceptual models for these mission spaces. The MSM (Mission Space Model) is a by-product of a simulation's front-end analysis and they are simulation and implementation independent functional descriptions. These functional descriptions represent a view of real world operations, entities, actions, tasks, interactions, environmental factors and relationships among all of them. Since CMMS is defined

for all stakeholders, it serves as a bridge between the subject matter experts and the developers. Subject matter experts act as authoritative knowledge sources when validating the mission space models.

CMMS is a framework that includes tools for gathering and storing knowledge, reusing this knowledge and providing a common repository for information storage, tools for conversions among various model representation notations. CMMS is composed of four main components;

- mission space models: consistent representations of real world military operations,
- technical framework: standards for knowledge creation and integration, includes;
 - a common syntax and semantics for describing the mission space
 - a process definition for creating and maintaining conceptual models
 - data interchange standards for integration and interoperability of mission space models
- common repository: a DBMS for registration, storage, management, and release of conceptual models
- supporting tools, utilities and guidelines

According to CMMS process definition, the four basic steps in conceptual model development are as follows;

- Collect authoritative information about the simulation context,
- Identify entities and processes to be represented in the simulation,
- Develop simulation elements (representational abstraction) and
- Define interactions and relations among simulation elements.

After some time CMMS has been renamed as FDMS (Functional Descriptions of the Mission Space) but the project faded away through the end of 1999 and the FDMS also faded away around 2002.

2.2.4. Defense Conceptual Modeling Framework (DCMF)

The Swedish Defense Research Agency found the idea of CMMS concept very promising and initiated a project to further study the conceptual modeling concepts and improve the CMMS. As they made progress in their research, they realized that they are moving further from the original CMMS concepts and renamed the project as DCMF (Defense Conceptual Modeling Framework) [37]. The major tasks were analyzing the CMMS in greater depth, studying the knowledge

acquisition and knowledge elicitation phases, analyzing the language issues such as ontology, terminology, common syntax and semantics and developing new methods when required.

Lundgren et.al. [33] identified the problems and limitations related with CMMS and proposed solutions to these problems as listed below:

- Unsupported knowledge acquisition: A complete methodology for knowledge acquisition in this domain should be developed.
- Lack of clarity of modeling elements: Modeling elements should be logically grouped by using meta-levels or abstraction levels.
- Need for alternative knowledge representations: A new method should be developed for knowledge representation.
- Limitations of processes: An action-centric approach should be used to add dynamic knowledge without limiting the process

Based on these findings, the objectives of DCMF were defined as “to capture authorized knowledge of military operations, to manage, model and structure the obtained knowledge in an unambiguous way; and to preserve and maintain the structured knowledge for future use and reuse.” DCMF claims that the conceptual model should be “(a) well documented, (b) readable and usable for a person as well as a machine, (c) composable, (d) traceable the whole way back to the original sources, and finally (e) usable as a basis for simulation models.”

DCMF project includes a process definition, a language definition, a list of available tools and analysis methods related with conceptual modeling. One of the major improvements to FEDEP is DCMF’s knowledge engineering focus on the conceptual analysis phase. The DCMF process consists of four main parts as shown in Figure 7.

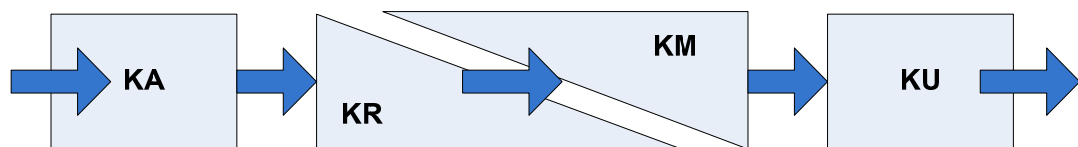


Figure 7: DCMF Process - Main Parts

Knowledge Acquisition is the learning phase which focuses on acquiring information and knowledge. DCMF defines this phase in 3 steps; the first step includes the determination of the focused context, the second step is the identification of the authorized knowledge sources, and the third step includes the actual acquisition of knowledge which is sometimes called knowledge elicitation. DCMF suggests the use of structured and well-documented techniques for knowledge elicitation such as interviews, prototyping, questionnaires etc. It is also noted that these kinds of analyses may include linguistic processes. A typical linguistic process includes phonetic, lexical,

morphological, syntactic and semantic analyses of existing documents or voice records that are used as information sources.

Knowledge Representation phase aims to analyze the structure of and formalize the acquired information. The human readable and probably ambiguous information is transformed into a machine readable and unambiguous format. This can be done by using methods like SPO (Subject-Predicate-Object), 5Ws (Who-What-Where-When-Why) and KM3 which are explained in [37]. These analyses will probably result in an ontology, which gives the context of the domain, the definitions of terms and their relationships and interactions.

Knowledge Modeling phase focuses on the semantic analysis and modeling of the information. Although previous phase may produce usable artifacts, building a common general model at the right level of abstraction requires further study. Different models can be generated based on the same set of data. These models should be suitable for future use and reuse. In order to provide this facility, the DCMF proposes using knowledge components that represent smaller knowledge parts. This approach provides flexibility, increases the rate of reuse and composability of conceptual models. Knowledge modeling also involves the merging of these knowledge components or conceptual models; therefore it will be a good idea to store these artifacts in a knowledge repository.

Knowledge Use deals with the actual use of the artifacts produced as a result of the previous phases. DCMF suggests using effective mechanisms that provide different visualizations of the knowledge for various users. These users may include the sponsor, consumer, producer and controller. The original intent of a knowledge component and any changes made to it should be recorded for an effective usage mechanism.

KM3 represents a specification, a tool and a language. KM3 is a specification for the creation of generic and reusable conceptual models. It is a tool for structuring knowledge in the form of generic templates. It is a common language that enables different stakeholders in developing conceptual models. KM3 follows an activity-centric approach and represents activities as KM3 actions. KM3 specification includes both static and dynamic descriptions. The static descriptions are specified by the attributes of an object whereas the dynamic descriptions are specified by the inclusion of rules into the object descriptions.

The KM3 metamodel consists of four main components; model element, attribute, state and rules. Model element is similar to the class concept in the object oriented world and it is the fundamental construct of the KM3. The objects that are parts of the activities and the activity objects are represented as model elements. A model element may be associated with attributes and states. There are four different types of model elements; EntityType, RoleInAction, RoleInOrganisationType and ActionType.

Attribute is similar to the attribute concept in the object oriented world and it is used to define measurable characteristics of a model element. An attribute is an optional part of a model element, but when defined, it belongs to at least one model element. DCMF defines various attribute types depending on the type of information represented by the attribute.

A state is a set of attributes with values associated with a model element. Any change in the value of an attribute results in a state change. These changes are restricted by the rules defined as StateDefinition. A valid state means that values of all of the attributes comply with the StateDefinition rules.

The dynamic behavior is represented by rules in the KM3. All changes of model elements are described by rule definitions, which specify the conditions under which an action starts and ends. A rule is composed of an activity role and an atomic formula. Atomic formulas can be combined conjunctively (OR-Connections) or disjunctively (AND-Connections) to create complex formulas.

The main purpose of the DCMF can be described as “to facilitate and support *development, reuse and interoperability* between simulation models.” DCMF accomplishes this purpose by a) providing a common language for all stakeholders and serving as a bridge between the military experts and the developers, b) creating libraries of validated conceptual models with certified quality levels and c) using the KM3 language as an enabler for transforming the conceptual model into other formats.

2.3. Base Object Model (BOM)

BOM (Base Object Model) is a SISO (Simulation Interoperability and Standardization Organization) standard that intends to “provide a component framework for facilitating interoperability, reuse, and composability.” The SISO BOM Product Development Group produced the “BOM Template Specification” [64] and the “The Guide for BOM Use and Implementation” [65] for describing the BOM related concepts. The objective of BOM is to encourage reuse, support composability, and help enable rapid development of simulations and simulation spaces.

The BOM Template Specification defines the format and syntax for describing the elements of a template for representing BOMs. It specifies the syntax and the semantics of the elements of BOM. This specification also provides a DIF (Data Interchange Format) for the representation of BOMs using the XML (Extensible Markup Language). The “Guide for BOM Use and Implementation” introduces methodologies for creating and implementing BOMs in the context of a larger simulation environment. The document provides guidance for BOM development, integration and use in supporting simulation development. The guide also includes examples of UML diagrams that may be used to represent BOM tables.

BOM standard sticks to the FEDEP definition and defines the conceptual model as “A description of what the [simulation or federation] will represent, the assumptions limiting those representations, and other capabilities needed to satisfy the user’s requirements.” BOMs are defined to “provide an end-state of a simulation conceptual model and can be used as a foundation for the design of executable software code and integration of interoperable simulations.” in [64]. This definition states that the BOM is closer to the solution domain and the developer rather than the problem domain and the domain expert.

A BOM is composed of a group of interrelated elements, which are the model identification, conceptual model information, model mapping information and object model definition as shown in Figure 8. The components of this template can be represented using tabular format or UML diagrams. In addition, the BOM DIF enables the transfer of the BOM information between tools. It should be noted that, although the BOM specification uses the HLA (High Level Architecture) OMT (Object Model Template) constructs, it does not restrict the use of a BOM to HLA specific implementations.

The model identification component is used to associate important identifying information with the BOM. The conceptual model component includes different views to represent the conceptual model information. This information should be transformed into an object model, which is preferably a composition of HLA Object Classes, Interaction Classes and Data Types. In order to enable this transformation, the required mapping information is provided as Model Mapping component. The Lexicon component is used to document the terms and ensure that they are consistently used in the correct form.

The following sections explain the components of a BOM, with an emphasis on the conceptual model component.

2.3.1. Model Identification

One of the goals for using BOMs is to facilitate reuse; therefore each BOM has to contain a minimum but sufficient degree of descriptive information such as name, type, version, security classification, point of contact, etc. This information is mostly based on the IEEE Std. 1516.2-2000 [25] and represented as a table. Every BOM is required to have a Model Identification table that includes the name, type, version, point of contact and other required information as specified in the IEEE Standard 1516.2-2000 [25].

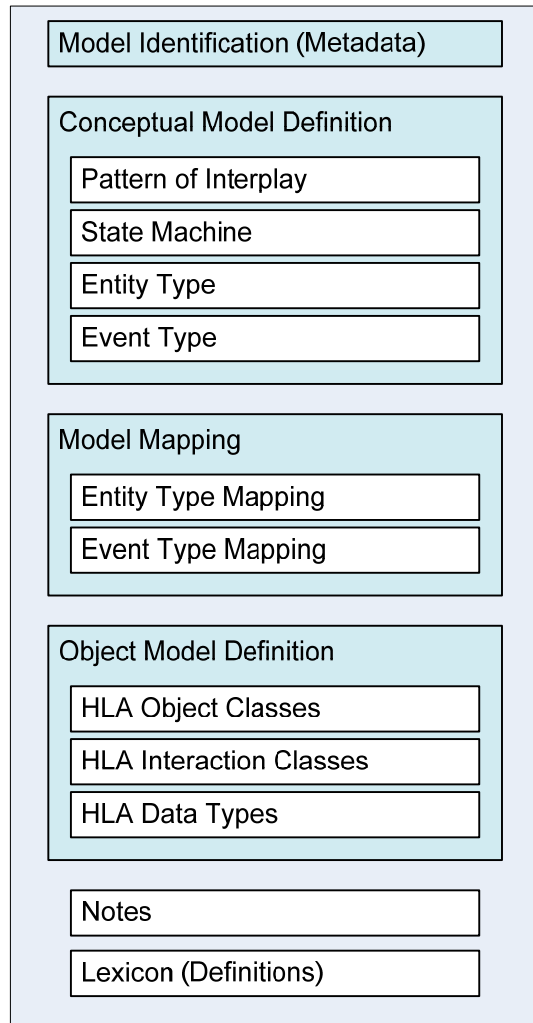


Figure 8: BOM Composition

2.3.2. Conceptual Model

The conceptual model of a BOM describes how the pattern of interplay within the conceptual model takes place, the various state machines that may be represented, the entity type and event types defined in the conceptual model. This definition is closely matched with the FEDEP steps related with conceptual modeling and provides a description of what the simulation component, simulation or federation “will represent, the assumptions limiting those representations and other capabilities needed to satisfy the user’s requirements.”

The pattern description is used to identify the sequences of actions necessary for fulfilling the pattern of interplay that may be represented by a BOM. In addition to the main course of events, the variations and exceptions are also represented as pattern descriptions. A pattern may be composed of many actions each of which includes the sender entity, the receiver entity and optionally the variations and exceptions.

The state machine description is used to identify the behavior states of a conceptual entity that are required to support one or more patterns of interplay. A state machine table includes the name of the state machine, the conceptual model entities that support the states defined and the behavior states that are supported by a conceptual entity. A name and exit condition is defined for each state. Each exit condition identifies an exit action and the next state upon satisfying the exit action.

The entity type is used to identify the conceptual entity types required to support the patterns of interplay and executing the various state machines. An entity type may play the role of a sender or receiver in a pattern of interplay or may be associated with a state machine. The entity type is identified by a name and associated characteristics. An example entity type may be a “waiter” having the “name” and “assigned tables” as characteristics.

The event type is used to “identify the type of conceptual events used to represent and carry out the actions variations and exceptions within a pattern of interplay”. The two types of BOM events are BOM Triggers and BOM Messages. A BOM trigger is an undirected event that may occur as a result of a change in the state of an entity and affects other entities that have interest in such observable changes. The source entity and the trigger condition are known, but the target entities cannot be identified. A BOM Message is a directed event which identifies both of the source and target entities. A Message is an event type with a target entity and a trigger is an event type with a trigger condition.

2.3.3. Model Mapping

The model mapping provides a mechanism for mapping between the elements of the conceptual model and the class structure elements of the Object Model Definition that are described using HLA OMT specification constructs. The two types of mapping supported are Entity Type Mapping and Even Type Mapping.

The entity type mapping provides a mechanism for mapping between the entity types of a conceptual model definition and the class structures of an Object Model Definition. An entity type is mapped into an HLA object or interaction class, which is composed of characteristics and attributes.

An event type is mapped into an HLA object or interaction class, which is also mapped with a source characteristics, target characteristics, content characteristics and trigger condition.

These mappings are means for transforming conceptual model elements into object model elements.

2.3.4. Object Model Definition

The object model definition defines the structure of an object and interaction class, and their associated attributes and parameters. HLA Object classes include HLA attributes and HLA

interaction classes include HLA parameters. This BOM component also includes the inheritance relationships between classes.

2.3.5. BOM Integration

As BOM Specification explains, BOMs are used to represent the conceptual models in accordance with the FEDEP steps. These steps include; selecting BOMS that support an aspect of a conceptual model among the existing ones, developing new BOMs if required, integrating these BOMs to create BOM assemblies and generating FOM / SOM from these assemblies. BOM specification also admits that although use of HLA is not a mandatory subsequent step, it is likely that BOM assemblies are intended to support an HLA based federation. This feature of BOM makes it dependent on the simulation specifications, which contradicts with the simulation-independency of a conceptual model. However, BOM can be used as an effective tool to transform conceptual models of the mission space into simulation space, which can be called as simulation conceptual model.

2.4. Summary

Abovementioned frameworks and methods either do not include detailed guidance on how to develop conceptual models or describe notations for representing conceptual models. They point out the requirements of a conceptual model, the inputs needed and the likely outputs that are produced.

Table 1: Conceptual Modeling Approaches

Approach	Method / Process	Notation	Tool Support
FEDEP	Includes process definition intended for HLA.	No specific notation is imposed.	No tool support.
SEDEP	Includes process definition for synthetic environments.	A UML based conceptual model format is proposed.	No tool support.
CMMS (FDMS)	Process definition does not include detailed guidance.	Project started with the aim of defining a common syntax and semantics.	No tool support.
DCMF	Includes process definition.	Includes KM3 notation for representing conceptual models.	Tool support exists.
BOM	Process definition does not include detailed guidance.	Includes a text based syntax and semantics definition. UML may also be used.	Tool support exists.
Robinson	Includes a process definition.	No specific notation is imposed.	No tool support.

A comparison of existing approaches is provided in Table 1 with respect to being a complete conceptual modeling framework. A complete framework solution for developing conceptual

models of the mission space should include a process definition (or a method) for guiding the conceptual modelers and the domain experts, a notation for representing the conceptual models and a tool for supporting the process and the notation. BOM does not include a detailed process definition for developing conceptual models and DCMF focuses on the knowledge acquisition activities of the conceptual modeling phase. Although Robinson's paper [56] is entitled as a framework, it does not include a notation specification and tool support for developing conceptual models. The CMMS approach was promising when the objectives are considered, however the project faded away without being able to provide a common syntax and semantics for conceptual modeling. FEDEP includes a high-level overview of the conceptual analysis phase by providing the boundaries of the phase, required inputs and the produced outputs; however detailed guidance for developing conceptual models is lacking.

There are a number of notations, which are not specifically designed but that can be used for conceptual modeling. Pace suggests the use of scientific paper method [45], van der Zee proposes petri nets, event graphs and UML object models [71], Robinson uses process flow diagrams [52], Ryan and Heavey [59] use simulation activity diagrams. Recent studies promote the utilization of UML [8] and SysML [19] for conceptual modeling. BPMN and IDEF1X [10], [23] are two other alternative notations. These notations provide different approaches; UML follows a more object-oriented approach, BPMN is more process-oriented and IDEF1X is a more data-oriented approach. However, these are general modeling notations designed and used for modeling systems, software or processes which do not necessarily address the specific requirements of the mission space conceptual modeling.

Our main motivation in developing a domain specific language for conceptual modeling is based on the ineffective use of modeling notations which are not specifically designed for conceptual modeling and that cannot be easily utilized by the SMEs. Most of the time the information required for developing a mission space conceptual model is written in free text by the subject matter experts. The free text notation causes ambiguous and recurrent definitions, is difficult to interpret by machines, and does not provide adequate guidance to the modeler [66].

CHAPTER 3

KAMA APPROACH

This chapter presents the KAMA approach that includes the conceptual modeling method, notation and a supporting tool. Section 3.1 introduces the basic components of our approach. Section 3.2 provides an overview of the notation and the relationship between KAMA and UML and Section 3.3 describes the conceptual modeling process definition. Section 3.4 summarizes the high-level functional requirements, architecture and usage of the KAMA tool.

3.1. Introduction

Conceptual models are not frequently developed using formal or semi formal notations. These models were not exhaustively utilized throughout the simulation development lifecycle and they were not shared among the simulation development projects. Also, the modelers do not emphasize the conceptual modeling phase in the simulation development life cycle. Therefore, we need a mechanism to increase the visibility of the conceptual modeling phase in the simulation development lifecycle and provide the necessary infrastructure to support effective development of conceptual models.

We proposed a solution with three major components. In order to provide an effective development instrument and increase the utilization of the conceptual models we designed the KAMA notation. Section 3.2 describes the major design goals for the notation and Chapter 4 explains the notation in detail.

To increase the visibility of the conceptual modeling phase, we revisited the simulation development lifecycle and proposed a refined conceptual modeling process. However, our aim was not to make a complete process definition but rather provide a concrete set of activities with well-defined inputs and outputs. This process definition, which is described in Section 3.3, is also used as a structured method for the execution of the case studies for this study.

The last component of our approach was a tool that supports both the notation and the process definition. We aimed to provide an effective infrastructure for developing, managing, sharing and

verifying conceptual models by the assistance of the KAMA tool. The features and the architecture of the tool are summarized in Section 3.4.

It should be noted that the focus of this study is on designing and explaining a notation for developing conceptual models of the mission space. The other components of our approach can be thought as auxiliary work products that support our thesis.

3.2. The Notation

The conceptual models should serve the needs of the analysis phase and should be used as an input to the design phase of the simulation development life cycle. In order to comply with the first need, the modeling notation should be usable and understandable by the domain experts and for the second need the developed models should be transferrable to the design phase by some means.

We analyzed the literature for available notations and provided the results under the summary heading in Section 2. The disadvantages and deficiencies of existing notations directed us to define a new notation. The deficiencies included mainly the lack of domain-specific modeling approach, the inadequacy of the notations for representing the conceptual models, difficulties the domain experts face in understanding the notation. The dominance of UML in systems and software modeling and our experience on object modeling and process modeling resulted in a decision to base our notation on UML. As a result we defined a domain specific conceptual modeling notation based on a metamodel architecture to represent the conceptual models of the mission space.

3.2.1. Design Overview

This section explains the major goals of designing the notation and their rationale. The first goal is identified for providing a common language for the conceptual modelers and the domain experts. The domain experts need not be knowledgeable in modeling languages; therefore we decided to define a domain specific notation. UML is a software modeling notation, which is later on positioned as a family of languages that provides an infrastructure for defining new languages. The dominance of UML in the software and systems modeling lead us define the second goal. The third goal is directly related with the definition of the conceptual models of the mission space in the literature.

a. Domain specific notation

Design Goal: The KAMA should be a domain specific modeling notation designed for the use of conceptual modelers and the domain experts

The KAMA notation uses a terminology specific to the domain of interest in defining the conceptual models. We named the metamodel elements after an analysis of existing conceptual models and conceptual modeling approaches. All of the metamodel elements map to a concept in

the mission space. The diagrams as well as the model elements serve to the specific needs of the conceptual modeling domain. However, since there are various simulation domains, these metamodel elements can be renamed and semantically redefined to comply with the requirements of different domains. Keeping in mind that the conceptual modeling steps reside on the analysis phase of the simulation development lifecycle, we proposed diagrams that would prevent the tendency to proceed into the design phase during developing the conceptual model.

b. Based on UML

Design Goal: The KAMA metamodel should reuse the UML metamodel where applicable

Our first idea was using UML Profile as the extension mechanism; however, we were not able to represent all of the required concepts using UML profiles. The two most important diagrams in KAMA, mission space and task flow diagrams included relationships (such as achieves, quantifiedBy, produces, inputTo) and model elements (workproduct, role, measure, objective) that could not take place together in a valid UML model.

Then we decided to extend UML. KAMA metamodel has its base on Foundation package, which is based on a subset of the UML metamodel in which irrelevant features have been removed. The other packages in the KAMA metamodel are based on the Foundation package. UML metamodel elements are reused by adding constraints where possible. When constraints are inadequate, KAMA metamodel elements are derived from UML metamodel elements through multiple inheritance or subtyped from Foundation package elements with additional attributes and operations. All KAMA metamodel elements direct or indirect subtypes of the Foundation package elements.

By this approach, we utilized the substantial knowledge cumulated by the UML community. We provided an easy to use domain specific notation by redefining the syntax and semantics of UML and removing complex structural and behavioral features. Since we defined a MOF based metamodel, it is possible to transform KAMA models into other MOF-based modeling notations such as UML. The modelers and domain experts who are familiar with the UML will easily learn and use the KAMA notation.

c. Simulation and implementation independence

Design Goal: The KAMA model should be simulation and implementation independent, i.e. a conceptual model developed in KAMA may be realized by various simulations which may be developed in various development environments/languages.

KAMA metamodel consists of domain specific elements, which are not dependent on any simulation specific concept or environment. The specific needs of conceptual modeling domain have been taken into account in designing the notation and the metamodel elements defined in

KAMA notation allow defining concepts existing in the mission space. However, it should be noted that the notation does not have much control on the modeler, therefore the modeler may attempt to represent simulation and implementation dependent concepts using KAMA notation.

There exists no construct related with the implementation of a simulation system in KAMA.; The relation of KAMA and UML may presume that using object oriented analysis and design methods would be a more effective choice.

3.2.1.KAMA and UML

This section explains the role of UML in designing the KAMA notation. Following a short overview of MOF and UML, we depict the relationship between KAMA, UML and MOF.

a. Overview of MOF and UML

The MOF (Meta Object Facility) is an OMG (Object Management Group) technology for defining metadata and representing it as CORBA (Common Object Request Broker Architecture) objects. The MOF provides a set of CORBA interfaces that can be used to define and manipulate a set of interoperable metamodels. It also defines a simple meta-metamodel with sufficient semantics to describe metamodels in various domains. The MOF supports any kind of metadata that can be described using Object Modeling techniques. This metadata may describe any aspect of a system and the information it contains, and may describe it to any level of detail and flexible depending on the metadata requirements.

The MOF provides a 4-layer architecture for defining metamodels. These layers are depicted in Table 2. One can define a metamodel for any domain using basic constructs of the MOF metamodel. These are classes, associations, data types and packages.

Table 2: OMG Metadata Architecture

Meta-Level	MOF Terms	Examples
M3	meta-metamodel	The MOF Model
M2	metamodel, meta-metadata	UML metamodel, CWM metamodel
M1	model, metadata	UML model, CWM data
M0	object, data	modeled systems, warehouse data

The main difference between MOF and UML is that while the former is designed for modeling metadata, the latter is designed for object modeling. The UML is a graphical modeling language

for modeling software systems. Although the UML is not necessarily tied to any particular application area it has been effectively applied in object-oriented analysis and design of software systems. The modeling elements of UML support the specification of both structural and behavioral aspects of a software system. As shown in Table 2, UML is formally defined by a metamodel, which is actually MOF.

b. The relationship between KAMA and UML

One of the objectives of this thesis was to define a metamodel for representing conceptual models of the mission space. Thus, the KAMA metamodel defines elements and rules for developing conceptual models that represent the mission spaces.

The KAMA metamodel includes a Foundation package that is based on the UML metamodel. This Foundation package serves as the base package on which the KAMA metamodel is developed. It consists of a subset of the UML metamodel where the irrelevant aspects have been removed. Any metaclass within KAMA directly or indirectly inherits from some metaclass in the Foundation package. For example “Entity” and “Capability” metaclasses in the Structure package inherit from the “Class” and “Operation” metaclasses in the Foundation package respectively. This relation also shows the analogy between an entity in real life having capabilities and a class in a software system having operations, both for representing behavior.

UML is used in KAMA in 3 different roles:

- 1) The UML notation is used in the diagrammatic representations of the KAMA metamodel. KAMA metamodel is depicted using UML class diagrams.
- 2) Additional constraints on the KAMA metamodel are represented in OCL (Object Constraint Language), as defined in the UML specification.
- 3) UML or the part that corresponds to the MOF model is used as the foundation metamodel. UML, specifically a subset as represented by the Foundation package, is used as the core of KAMA. All other packages of the KAMA notation inherit classes and associations from the Foundation package.

3.3. The Conceptual Modeling Process Definition

The conceptual modeling process includes knowledge acquisition, representation of this knowledge in different forms, validating this knowledge and using this knowledge as a starting point for more detailed models.

This section briefly explains the conceptual modeling process defined for realizing the case studies. Our aim is not to introduce a comprehensive process definition, but to provide help for the modelers and domain experts who will develop the conceptual models. The process is depicted using a KAMA task flow diagram in Figure 9. Only the highest level tasks are shown in the

diagram and this is why the process is depicted as if it has a sequential flow. The iterative flow of the conceptual modeling process is explained in the following sub-sections. The detailed explanation of the process definition is given in [27].

The main tasks are derived from the process definitions of CMMS [15] and Pace [45] and then enriched with our experience on process and object oriented software modeling. Since the conceptual modeling process should be performed iteratively, the sequence of these tasks does not impose the order of execution unless otherwise specified. In some cases a task may need to use the outputs of another one and therefore cannot be performed prior to that task.

3.3.1. Acquire knowledge about the mission space

The first step in our process definition is knowledge acquisition about the mission space for which a conceptual model will be developed. As the modeler will reflect her knowledge and experience on the domain to the conceptual model, it is essential that she use the right and accredited knowledge resources. As Lundgren has pointed out in [33]; although the conceptual analysis requires proper usage of knowledge engineering activities, the community has not yet accumulated enough experience on this topic. This situation cannot be easily explained with the difficultness of the knowledge acquisition activities. The conceptual analysis and also the knowledge acquisition activities are not visible throughout the simulation development process and the simulation developers have a general tendency to move through the solution space.

This step should be executed first, but the modeler may need to revisit this phase and perform updates. Information gathered in this step should be recorded and properly stored for future reference. It would be of great benefit to apply a simple version management mechanism for keeping these information records under control and provide limited access if required by the project constraints. Mojtahed et.al. treat the whole conceptual modeling process as a knowledge engineering task and they propose a framework in [37]. We do not propose any method for knowledge acquisition activities but identify the necessary outputs that will be used in the subsequent phases of conceptual modeling.

- a. Identify simulation objectives: The first step is to identify the high level simulation objectives that should be defined in terms of units as measurable as possible. The initial objectives that are defined by the sponsor are enhanced to include the measures of performance parameters with the assistance of the domain experts and modelers. Defining the measures of performance parameters early in the simulation development lifecycle will play an important role in the acceptance of the simulation project. Therefore, it will also be beneficial to take the commitment of all parties involved in the knowledge acquisition phase after simulation objectives have been finalized.

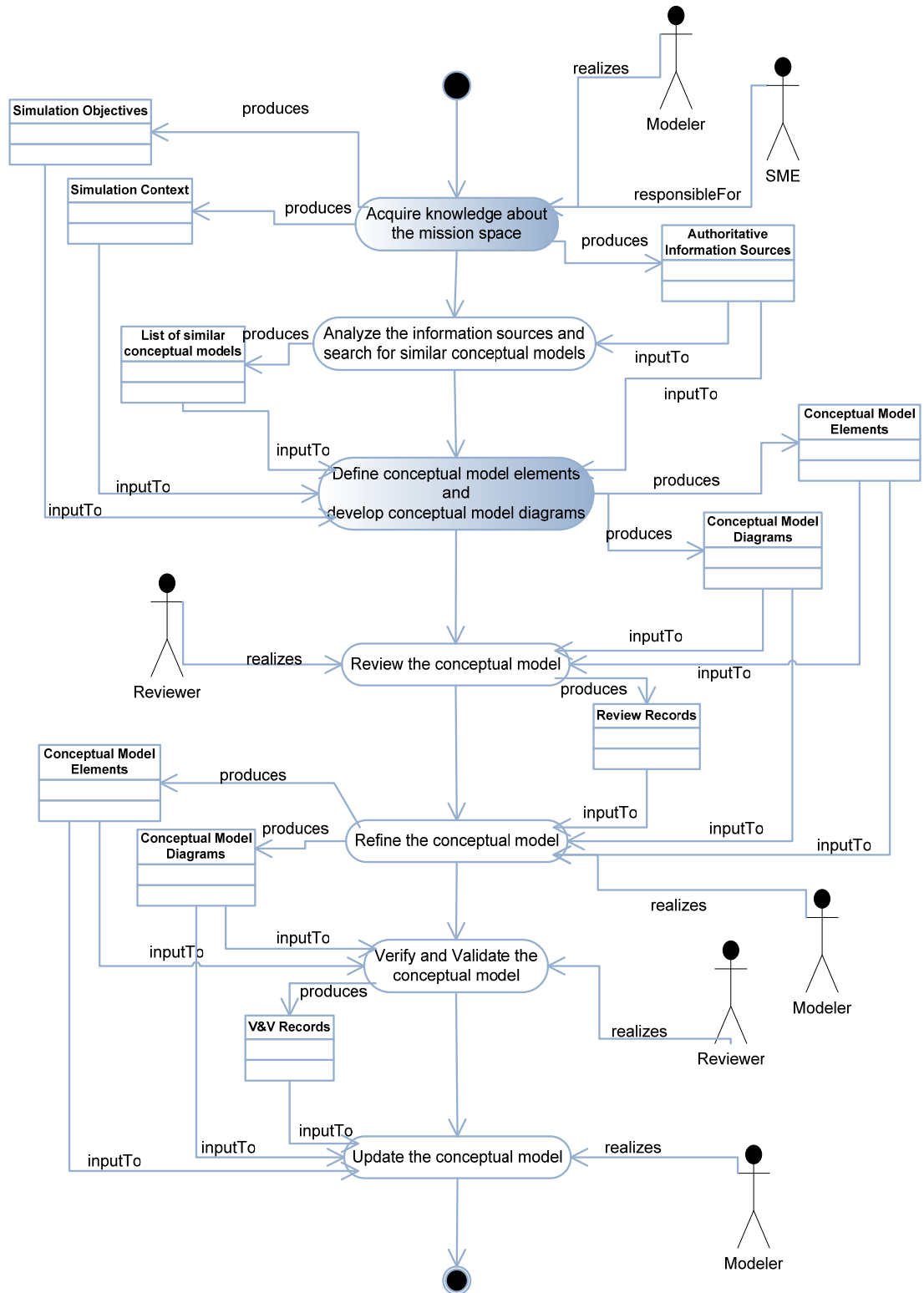


Figure 9: Conceptual Modeling Process

- b. Define the simulation context: The simulation context defines the boundaries of the mission space aligned with the simulation objectives. This may include the high level

needs, assumptions and constraints of the sponsor, the fidelity requirements of the simulation and the risks related with this information. The change history of this information should also be kept under control in order to track the changes to the source.

- c. Identify the sources of authoritative information: Following the identification of the objectives and definition of the simulation context the authoritative sources of information required to develop the conceptual model should be identified. These sources may include books, documents, files as well as human resources such as domain experts. The following information about these sources will be useful; where the source is located, how and when to access the source, the confidentiality of the information stored in the source, the date and version of the source and any other necessary properties. As mentioned before, the list of these sources may be frequently updated as required. Maintaining a traceability matrix between the conceptual model elements and the information sources will be helpful in the validation of the conceptual models. In addition, the methods that will be utilized for acquiring knowledge from the domain experts should be identified and planned appropriately.

3.3.2. Analyze the information sources and search for similar conceptual models

The information sources are analyzed to understand the domain better and draft conceptual model elements are identified. These are generally high level elements and do not contain too much detail. Another source of information may be the similar conceptual models that can be accessed. The modeler may search for conceptual models of the related projects to find and reuse common parts. Especially the static components of conceptual models (e.g. entities, work products) may be reused among different simulation projects. Storing the conceptual models in a common central repository that can be remotely accessed and searched will be beneficial for this step.

3.3.3. Define conceptual model elements and develop conceptual model diagrams

The modeler together with the domain expert begins identifying the conceptual model elements. As the conceptual model elements are being defined, the modeler notes the assumptions, constraints and the information resources related with the element. The modeler may choose to define a conceptual model element first and then develop the diagram or vice versa.

This is an iterative step by itself; in the first iteration the modeler may only entitle the conceptual model elements and briefly describe them, in the second iteration the modeler may provide the detailed attributes of the conceptual model elements and in the third iteration the modeler may define the relationships among the conceptual model elements. Although there is not a predefined sequence for defining the conceptual model elements, it will be better to begin with defining the high level missions that make up the mission space or defining the major entities that make up the

entity ontology. We do not intend to specify one of these options as superior to the other; however, we select one of them because we need to follow a sequence to write down these steps.

- a. Define missions: The modeler defines the highest level tasks in the mission space as missions. The modeler may also use missions for logically grouping the tasks, by keeping the most related tasks under the same mission. The modeler also defines the objectives and measures related with each mission. Identification of roles that are responsible for a mission may be deferred until the command hierarchy has been established. The modeler depicts the relationships among missions, roles, objectives and measures in the mission space diagram.
- b. Define tasks for each mission and sub-tasks for each task: After high level missions are defined, the modeler details these missions using task flow diagrams. Each task in a task flow diagram can also be detailed using task flow diagrams. The modeler defines the start and end points, the sequence of tasks, the decision points, the synchronization points, the work products, objectives, measures and roles associated with each task in this diagram. It is also possible that the modeler may reuse previously defined conceptual model elements such as tasks, work products, roles etc. The relationships among the work products are depicted using entity relationship diagrams.
- c. Define entities: As the missions and tasks are being defined, entities within the mission space may be concurrently identified. Sometimes, the entities may be well known even before the missions have been identified. Our experience shows that most of the entities can be directly derived from the statement of work documents. The entities are defined with the attributes and capabilities that make up the characteristics of an entity. The relationships among the entities are depicted using the entity ontology diagrams.
- d. Define actors and roles: Actors may be derived from the entities. Each role must be attached to an actor. Therefore, actors and roles are defined after the definition of missions and entities. The actors are defined in the command hierarchy diagrams and the roles are defined in the organizational structure diagram.
- e. Define the security and fidelity requirements: The access constraints on the conceptual model or parts of conceptual model are defined.

3.3.4. Review the conceptual model

After the conceptual model is developed, a modeler other than the developer of the conceptual model verifies the model with respect to defined checklists. In addition to this, modeler may request for a joint review with the domain expert and the sponsor to perform an early validation of the model. A walkthrough will be the most appropriate method for this review. The modeler

explains each diagram in the model during the walkthrough and tries to capture whether she has understood the mission space well and notes any inconsistencies or defects.

3.3.5. Refine the conceptual model

Based on the findings noted during the early review the modeler updates the model. Then the model is refined to reflect the following aspects.

- a. Define the behavior of entities: The modeler defines the state machines for the entities that have complex behavior using entity state diagrams. This is especially important for reactive entities which observe the behavior of other entities and react accordingly. The states that an entity can be in throughout its lifecycle and the transitions among these states are identified.
- b. Define high level algorithms: If a capability of an entity represents a complex behavior and requires further explanation, this explanation is provided at this step.

3.3.6. Verify and validate the conceptual model

After the conceptual model has been finalized the model is verified in accordance to the syntactic and semantics rules. It would be better to perform these checks automatically; however, should this option not available checklists can be utilized. Validation of the conceptual model may be accomplished through joint reviews with the domain experts and the sponsor. The verification and validation results should be reported.

3.3.7. Update the conceptual model with respect to the V&V findings

The modeler updates the conceptual model in order to resolve the V&V findings. Additional reviews may be performed if required.

3.4. The KAMA Tool

We needed to develop a tool to effectively support the usage of the notation following the defined process. This tool was developed as part of a research project by a group of researchers and software developers. This section presents the high level functional requirements of the tool, its architecture and its usage in the case studies.

3.4.1. High Level Functional Requirements

This section presents the high level functional requirements of the tool that supports the KAMA notation. The high level functional requirements are shown as a use case diagram in Figure 10.

- 1) Develop and manage conceptual models

The user can create, store, delete or update conceptual models. A conceptual model consists of conceptual model elements, diagrams and information related with these. User can operate on the conceptual model as if it is a composite entity, can save it as a new version. Security level can be assigned to a conceptual model.

2) Create and manage conceptual model elements

The user can create, store, delete or update conceptual model elements. All of the conceptual model elements are logically grouped and represented as an item in the model tree. Once defined, a conceptual model element can be used in many diagrams by dragging and dropping onto the diagram. The tool checks whether a conceptual model element can be used in a diagram by referring to the predefined rules in the metamodel. A conceptual model element includes detailed information in its properties tab. This information consists of general information, information resources, properties and relations sections. Properties section is customized for different types of conceptual model elements. Security level can be assigned to each conceptual model element.

3) Associate external files with conceptual model elements

The user can associate any file type with a conceptual model element. The tool keeps this association and stores the associated file in its local database. When the conceptual model is exported, these files are also exported together with conceptual model definitions. This feature is especially necessary for representing information which cannot be easily represented as diagrams such as geographic references. The tool will not be responsible for editing the associated files, but only store the link and the file itself.

4) Create and manage conceptual model diagrams

Diagrams are the means used to represent the model from different viewpoints. The user can create, store, delete and update diagrams. The tool presents a customized toolbar for each type of diagram. The tool keeps the rules related with each type of diagram. These rules are then verified by the tool to assist the user while creating diagrams. The user can drag and drop model elements from the model tree onto the diagram pane, or the user may also select to create empty diagram elements using the toolbar. The user can manage diagrams using drag and drop facility.

5) Provide a search facility

Reusing existing conceptual models and elements was one of the objectives of the research project; therefore the tool shall provide a powerful search mechanism. The user can perform search within and among conceptual models stored either locally or on the remote central repository. The user can preview the found conceptual models, conceptual model elements or diagrams and if the user decides to use any of them, s/he can download it and add it to the local database. If the element is located in the local database but in another conceptual model, the element is copied to the target conceptual model. The user can perform a complex search by filtering the result set.

- 6) Provide version management for conceptual models and conceptual model elements
The tool shall store the conceptual models and conceptual model elements with version information so that the user can access and use any version s/he needs. The user can restore a version, delete it or delete all versions of a conceptual model element. The versioning mechanism shall also handle the versions of the conceptual model elements residing on the central repository.
- 7) Provide controlled access to the conceptual models
The user can assign security levels to the conceptual models. The system administrator can assign access rights to the users. Any user with an access right lower than a conceptual model cannot read or update that conceptual model. These access rights and security levels can be updated by the system administrator.
- 8) User management (add/remove/change access rights)
The user can manage local users, and the system administrator can manage both the local and remote users. They can add or remove users and change access rights.
- 9) Provide a reporting mechanism
The developed conceptual models will probably include many conceptual model elements located in many diagrams. The tool shall provide a customizable reporting mechanism which will fit the user's documentation needs. Details of conceptual model elements, diagrams, verification results, rules used to verify the conceptual model shall be included in the reports.
- 10) Provide access to a remote central repository
The user can access the central repository for performing a search, downloading a conceptual model or a conceptual model element, and publishing a conceptual model for sharing. Conceptual model shall be internally verified prior to being published. Conceptual model versions shall be managed by the tool during these transfers. The system administrator shall be able to accept or reject the published conceptual model. When a conceptual model is published it shall be marked as accessible by other modelers and any changes made on it shall be controlled.
- 11) Verify the conceptual model
The tool shall be able to verify the conceptual model based on a predefined set of rules. These may be generic rules at the metamodel level or specific rules at the model level. The tool shall produce a verification results report, including both errors and warnings. An error indicates a conflict or a defect within the conceptual model that shall be corrected before the conceptual model is published to the central repository. Warnings indicate situations which may lead to an error but it is not compulsory to correct them as a prerequisite for publishing.

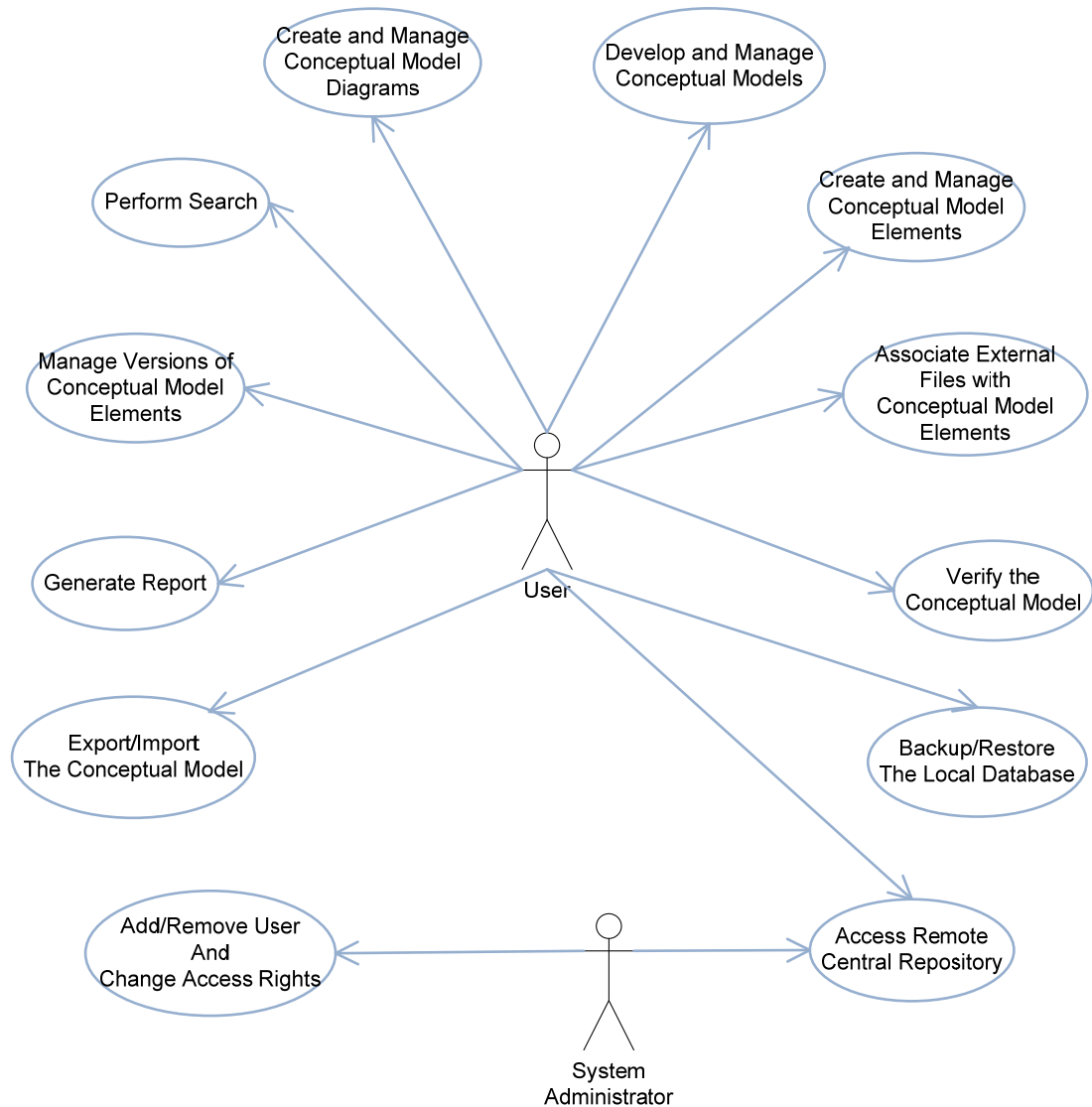


Figure 10: High Level Use Case Diagram

12) Export / import the conceptual model

The tool shall be able to export a conceptual model or a conceptual model element as a text file, preferably in XML format, and also import any valid text file into a conceptual model. All of the files associated with the conceptual model elements shall also be included as binary files. The tool shall perform an integrity check prior to importing a conceptual model from a text file.

13) Backup / restore the local database

The user shall be able to backup the conceptual models in the local database. The tool shall be able to restore from the backups. The user shall be able to schedule automatic backups.

14) Assist the modeler through an intuitive GUI

The tool shall include an intuitive graphical user interface, which will make the modeling process easier. A conceptual modeling wizard shall be used to guide the modeler, by reminding what to do as the next step.

15) n-dimensional viewing feature

The tool shall include a mechanism to elaborate on the model by providing different views. These views may be visually in three dimensions or logically in n-dimension. The logical dimensions may be interpreted as combined views including different types of diagrams or diagrams showing the relationships between diagrams.

3.4.2. Architecture of the Tool

The tool is designed to be a client application which uses a local database to store the conceptual models. A central repository is located on a server and clients access to this repository through web service interfaces. Client includes mainly the following product components; a graphical editor, model manager, version manager, report generator, model rule base manager, n-dimensional viewer, user manager, security manager, file access manager and data access manager. Figure 11 depicts the architecture of the tool. The architecture is composed of three logical layers, which represent view, control and model layers as defined in the MVC pattern [18].

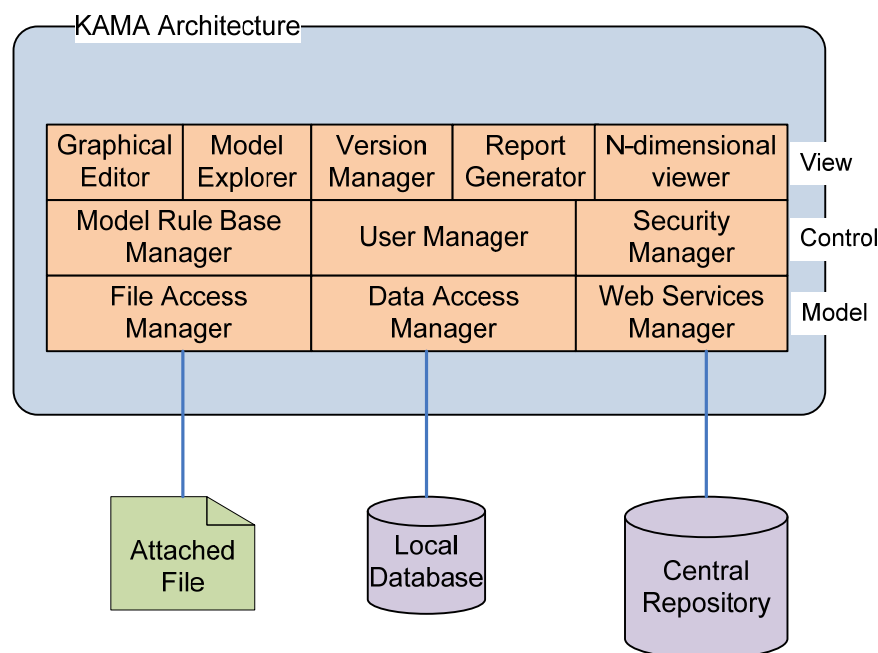


Figure 11: Architecture of the Tool

The view layer includes two-dimensional graphical editor to create diagrams, a model explorer to create and navigate among model elements, a version manager to create, store and access different versions of each model element, a report generator to see the model properties in detail and an n-dimensional viewer to produce views of the diagrams in three dimensions. The Control layer

includes manager packages which provide services for accessing the model elements, diagrams, users and security properties. The Model layer includes packages for providing data access services to the local database, central repository and files attached to the model elements. Clients store the data in the local database that has the same structure with the central repository. Data transfer among the clients is carried out via the XML files that are generated by the export / import utility of the tool.

3.4.3. Usage of the Tool

The tool is aimed to be used by the modelers and domain experts to produce conceptual models of the mission space. The elements that can be used in the models, the possible relationships among them, and the rules covering these relationships are defined in a metamodel. The tool provides an intuitive graphical user interface that supports the development of conceptual models by guiding the user to follow a defined process. The conceptual models are verified in accordance with a predefined set of rules and only then can be published into the central repository. Clients can perform search on these conceptual models and share the conceptual models they have developed with other clients.

We used the tool in the first case study to develop the conceptual model. This model was verified and then published into the central repository.

CHAPTER 4

THE KAMA MODELING NOTATION

This chapter presents the modeling notation we proposed for modelers to develop conceptual models of the mission space. The need for developing a conceptual modeling notation is discussed in Chapter 2. In this chapter we first explain the KAMA architecture and the structure used to describe the KAMA notation in Section 4.1. Section 4.2 explains the Foundation package which includes the model elements inherited from UML; these metamodel elements form a basis for the other packages. Section 4.3 describes the Mission Space package that includes the Mission Space and Task Flow diagrams. Section 4.4 explains the structural metamodel elements in the Structure package that includes Entity Ontology, Entity Relationship, Command Hierarchy and Organization Structure diagrams and Section 4.5 describes the Dynamic Behavior package, which includes the state machine representation and the Entity State diagrams.

4.1. Overview

The KAMA Metamodel uses packages and a hierarchical package structure to reduce complexity, promote understanding, and support reuse. The dependency of KAMA packages to the UML metamodel is encapsulated with the Foundation package. Foundation package includes metamodel elements that are directly inherited or derived from the UML metamodel. Rest of the KAMA metamodel elements in the other packages are derived from the elements in the Foundation package.

4.1.1. KAMA Architecture

The main packages that form the KAMA notation and the relationships among them are shown in Figure 12. This package structure provides only a logical grouping among the metamodel elements and does not imply any other meaning such as usage of any package as a separate metamodel. The Foundation Package is a subset of the UML and provides basic constructs for creating and describing metamodel classes for other KAMA packages. Mission Space package extends the Foundation package and includes metamodel elements that are used to represent the missions and tasks in a mission space. The reason for having a separate package for Mission Space is that this

package includes common metamodel elements that are used to represent both the structural and behavioral aspects of a conceptual model. Separating these common elements decreased complexity of the notation specification and provided a more understandable presentation of the notation. The Structure package is based on the Foundation package and includes the metamodel elements that are used to represent the static structure of a conceptual model.

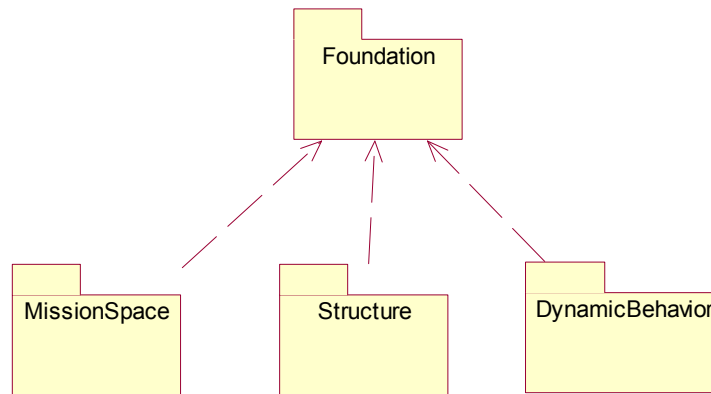


Figure 12: KAMA Architecture

The dynamic behavior package is based on the Foundation package and includes the metamodel elements that are used to represent the dynamic behavior of a conceptual model by focusing on the internal state changes of an entity.

4.1.2. KAMA Metamodel

The previous sections depicted the inheritance relations between the metamodel elements in the Foundation package and other packages. The relationships among the KAMA metamodel elements are shown in Figure 13. The metamodel diagram shows the multiplicity values, relations and role names.

There is a many to many relationship between a mission and a task. A task must belong to at least one mission and a mission must have at least one task. A task has an aggregation relation to a task, in which the one of the tasks plays the role of a taskNode. This structure is used to represent task flow diagrams using tasks. A task may be composed of many tasks and a task may be part of many tasks. A task may or may not produce a WorkProduct and zero or more WorkProducts may be input to a task.

Both a mission and a task may have one or more objectives associated with them via “achieves” relationship. There is a many to many relationship between an objective and a measure. Each objective may be quantified by one or more measures. A measure must be related with at least one objective. A mission may be associated with other missions via includes or extends relationships.

metamodel of that package. The Foundation package has a slightly different outline, because almost all of the metamodel elements in that package are directly used from the UML metamodel. Therefore, only brief definitions are given for these metamodel elements.

Under the heading for each package, class descriptions section provides detailed explanations for all of the model elements including the relations. Then the additional constraints related with these package elements are defined in OCL and the diagram(s) including the model elements in that package are introduced.

Class descriptions section includes the following information for each model element. Generalizations section identifies the super classes of that class, and any additional attributes are defined in the attributes section. Constraints related with the class are defined in the constraints section, semantics of the class is explained in the semantics section.

4.2. Foundation Package

4.2.1. Overview

The Foundation Package is a subset of the UML and provides basic constructs for creating and describing metamodel classes for other KAMA packages. Defining a subset of UML allows us to utilize the power of UML by focusing only on the related parts. The detailed definitions of these parts are defined in the UML infrastructure [43] and UML superstructure [44] documents; therefore this section contains high-level metamodel diagrams and brief explanations for the metamodel elements.

4.2.2. Abstract Syntax

The Foundation Package is divided into three logical sub-packages that are depicted in Figure 14.

Core metamodel depends on no other packages. Core metamodel contains basic model elements that are used by all other KAMA packages. The model elements in the Core metamodel and their relationships are depicted in Figure 15. All of the metamodel elements in the core package inherit from the Element. We introduced the ModelElement, which includes the common features for the KAMA metamodel elements that will eventually inherit from one of the metamodel elements in the Core package.

Relationships metamodel depends on the Core package and defines the classes that describe the basic relationships among KAMA model elements. The model elements in the Relationships metamodel and their relationships are depicted in Figure 16.

Behavior metamodel depends on the Core package and defines the classes that describe the behavior of KAMA model elements. The model elements in the Behavior metamodel and their relationships are depicted in Figure 17.

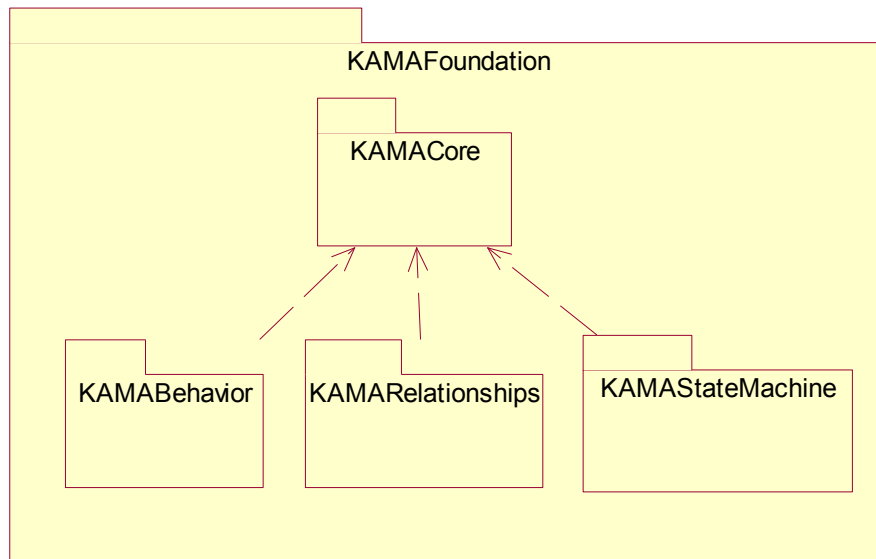


Figure 14: Foundation Package

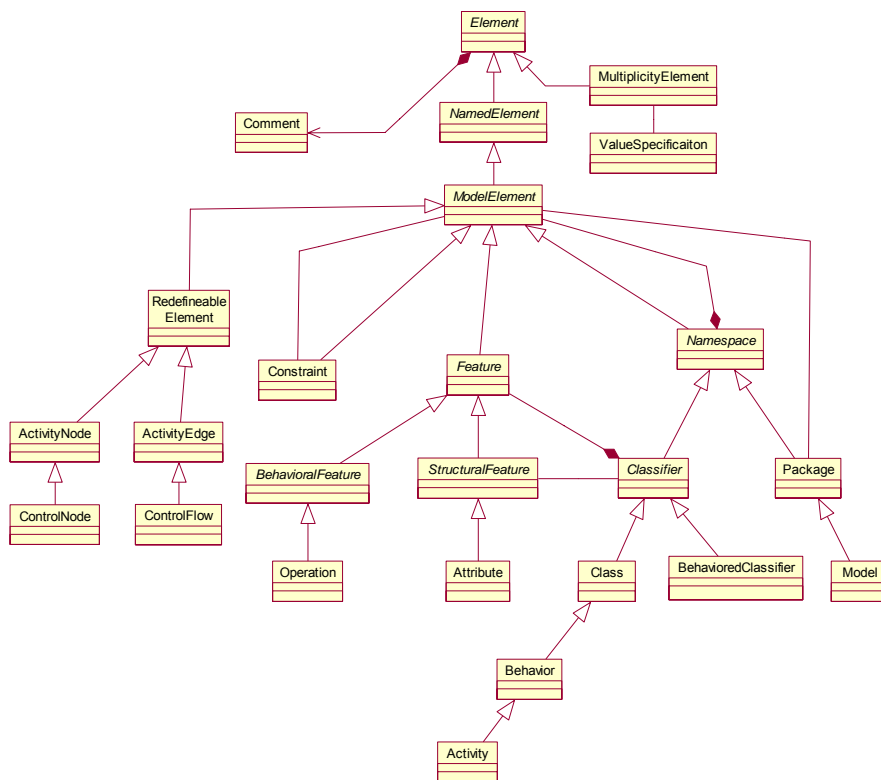


Figure 15: Core Metamodel

Although the elements in the Foundation Package are defined in detail in the UML specifications [43] and [44], the following section provides brief descriptions of the essential elements to preserve completeness.

4.2.3. Class Descriptions

a. Element

Element is an abstract metaclass, which is used as the common superclass for all of the metaclasses in the UML infrastructure library as well as the KAMA packages. It has the capability of owning other elements.

b. Named Element

A named element is an abstract metaclass that is used to represent elements in a model that may have a name. The name is used for identification of the named element within the namespace in which it is defined.

c. Model Element

A model element is an abstract metaclass that is used to represent common features in the KAMA metamodel elements. It is not part of the UML metamodel. A “Model Element” includes the following attributes;

- **id:** A unique id that is used as a distinguished identifier while referring to a KAMA model element.
- **name:** Explanatory name for the KAMA model element. This name can be shown on the diagram.
- **description:** A short description of the KAMA model element.
- **assumptions:** List of assumptions in free text form. The assumptions may also be stored including the date, information source regarding the assumption and change history.
- **constraints:** List of constraints in free text form. The constraints may also be stored including the date, information source regarding the constraint and change history.
- **geographicalInformation:** Any information related with the geography of a KAMA model element. This attribute may refer to external files such as maps, drawings etc.

d. Feature

A feature is an abstract metaclass that declares a behavioral or structural characteristic of instances of classifiers. There are two subclasses of Feature in the scope of KAMA notation, behavioral

feature and structural feature. A behavioral feature specifies that an instance of a classifier will respond to a designated request by invoking a behavior. A structural feature is a typed feature of a classifier that specifies the structure of instances of the classifier.

e. Namespace

A namespace provides a container for named elements, that is, it is a named element that can own other named elements. Each named element may be owned by at most one namespace and a namespace provides a means for identifying named elements by name.

f. Classifier

A classifier is a namespace whose members can include features. It is used to describe a set of instances that have features in common.

g. Class

Class is a kind of classifier whose features are attributes and operations. The purpose of a class is to specify a classification of objects and to specify the features that characterize the structure and behavior of those objects.

h. Operation

An operation is a behavioral feature of a classifier that specifies the name, type, parameters, and constraints for invoking an associated behavior. It is a specification of a transformation or query that an object may be called to execute.

i. Attribute

An attribute is structural feature represented as a named slot within a classifier that describes the values that instances of the classifier may hold.

j. Package

A package is used to group elements, and provides a namespace for the grouped elements.

k. Generalization

A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. The more specific element is fully consistent with the more general element and contains additional information. An instance of the more specific element may be used where the more general element is allowed.

l. Dependency

A relationship between two elements in which a change to one element (the supplier) may affect or supply information needed by the other element (the client).

m. Association

An association is a relationship among two or more specified classifiers that describes connections among their instances.

n. Association End

An association end is a structural part of an association that defines the participation of a class in the association. The properties that an association end may have include but not limited to; aggregation, multiplicity, navigability, ordering and visibility.

o. StateMachine

State machines can be used to express the behavior of part of a system. Behavior is modeled as a traversal of a graph of state nodes interconnected by one or more joined transition arcs that are triggered by the dispatching of series of (event) occurrences. During this traversal, the state machine executes a series of activities associated with various elements of the state machine.

p. State

A state is a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event. States are contained in a state machine that describes how the history of an object evolves over time in response to events.

q. PseudoState

A pseudostate is an abstraction that encompasses different types of transient vertices in the state machine graph. The only type of pseudostate used in KAMA is the initial state. A final state is not a pseudostate.

r. Vertex

A vertex is an abstraction of a node in a state machine graph. In general, it can be the source or destination of any number of transitions.

s. Transition

A transition is a relationship within a state machine between two states indicating that an object in the first state will perform specified actions and enter the second state when a specified trigger occurs.

4.3. Mission Space Package

4.3.1. Overview

Mission Space package extends the Foundation package and includes metamodel elements that are used to represent the missions and tasks in a mission space. Mission Space package elements are specifically defined for representing the conceptual models and in this sense differ from Foundation package elements. Main objective of the Mission Space package is to define the high-level missions, their inter-relationships, and objectives assigned to them so that the modelers will be able to represent the mission space. These missions will be detailed by task flow diagrams that include the tasks, the roles responsible for performing these tasks, related workproducts, objectives and measures.

4.3.2. Abstract Syntax

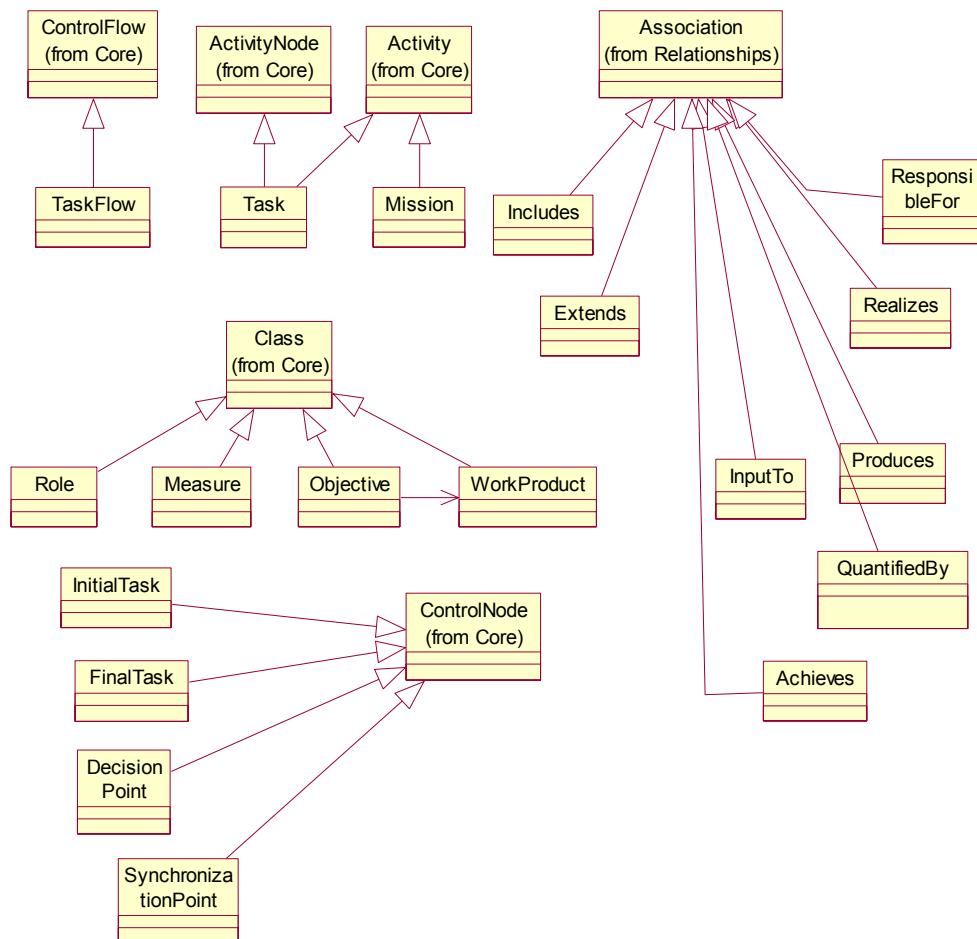


Figure 18: Mission Space Metamodel

4.3.3. Abstract Syntax

The inheritance hierarchy of the Mission Space package metamodel elements is depicted in Figure 18. The metamodel elements that inherit from the same Foundation package elements are distinguished by different constraints and semantics as described in Section 4.3.3.

4.3.4. Class Descriptions

a. Mission

Development of conceptual models of mission space begins with the definition of a mission space that is compatible with the simulation objectives. Mission can be defined as an ordered series of tasks with a well defined objective and is required to be implemented by the simulation system. The mission space of a simulation system can be represented by mission space diagrams at the highest level and thus the context of the conceptual model is clearly defined and the simulation developer is left with a well defined problem definition.

Missions represent the highest level tasks that a simulation system should realize. The detailed flow of a mission is shown by task flow diagrams as tasks connected by task flows. Mission can be connected to other missions by include and generalize relations. A mission should be related to simulation objectives or high level requirements specified in the statement of work document of the simulation system to be developed.

Generalizations

- Activity from Core

Attributes

- inputList: List of Workproducts that are connected to the Mission via inputTo relation.
- outputList: List of Workproducts that are connected to the Mission via produces relation.
- preConditions: The preconditions that are required for executing the Mission.
- postConditions: The postconditions that occur after the execution of the Mission.
- objectiveList: List of Objectives that are connected to the Mission.
- roleList: List of Roles that are connected to the Mission via responsibleFor or realizes relationships.
- measureList: List of measures that are connected to the objectives of a mission via quantifiedBy relationships.

Constraints

[C-MS-a-1] A mission should be related with at least a role

context Mission self.roleList.size() > 0

[C-MS-a-2] A mission should be related with at least an objective

context Mission self.objectiveList.size() > 0

[C-MS-a-3] A mission may not have a relation of type include or generalize to itself

context Includes S=self.source AND T=self.Target implies not (S=T)

context Generalizes S=self.source AND T=self.Target implies not (S=T)

Notation



A mission is represented by an ellipse, with the name of the mission placed at the center.

b. Task

A task is used to detail the mission. Tasks can be reused among different missions.

A task shares the same structure with a mission with some additional attributes and behavior. By inheriting both from Activity and ActivityNode, a task can also take place as a node in a task flow diagram. This property of task provides a recursive mechanism in defining the mission space. The modeler has the opportunity to develop the task tree until she thinks it is sufficient to stop. Tasks are connected to each other by task flows. Roles, objectives, and workproducts are connected to tasks in the task flow diagrams.

Generalizations

- Activity from Core
- ActivityNode from Core

Attributes

- inputList: List of Workproducts that are connected to the Task via inputTo relation.
- outputList: List of Workproducts that are connected to the Task via produces relation.
- preConditions: The preconditions that are required for executing the Task.
- postConditions: The postconditions that occur after the execution of the Task.

- objectiveList: List of Objectives that are connected to the Task.
- roleList: List of Roles that are connected to the Task via responsibleFor or realizes relationships.
- measureList: List of measures that are connected to the Objectives of a Task via quantifiedBy relationships.
- isExtensionPoint: Boolean value that is used to identify the Tasks that are extension points.
- extensionPointId: Integer value that is used to identify and refer to Tasks that are extension points.

Constraints

[C-MS-b-1] A task should be related with at least a role

context Task *self.roleList.size() > 0*

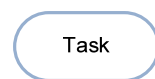
[C-MS-b-2] A task should be related with at least an objective

context Task *self.objectiveList.size() > 0*

[C-MS-b-3] A task has one incoming and one outgoing task flow

context Task *self.incomingFlows.size()=1 AND self.outgoingFlows.size()=1*

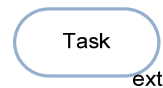
Notation



A task is represented by a rounded rectangle, with the name of the task placed at the center.



A task having a task flow diagram is shown with a filled rounded rectangle.



If a task is an extension point, it is identified with the “ext” label below the rounded rectangle.

c. Role

A role represents an active actor who is “responsible for” or “realizes” a task or a mission. The actors that are defined in the Command Hierarchy diagram should be instantiated as roles in the Organization Structure diagram in order to take part in the

Task Flow or Mission Space diagrams. Command Hierarchy diagrams are more stable diagrams that could be reused among simulation projects, whereas Organization Structure diagrams may include role definitions that are specific only to a particular mission.

A role may be used to represent a human being or another active entity which can play a role in performing a task or a mission.

Generalizations

- Class from Core

Attributes

- taskList: A list of missions and tasks the role is connected with “responsible for” or “realizes” relations.
- ownerList: A list of Actors that the role is connected with “owns” relation.

Constraints

[C-MS-c-1] A role should be owned by at least an actor

context Role *self.ownerList.size() > 0*

Notation



A role is represented by a stick figure, with the name of the role placed below. The head of the stick figure is filled to distinguish from the actor symbol.

d. Objective

An objective represents the goal of a mission or task. It should be related with the simulation objectives. Each mission should have at least an objective to be meaningful. The modeler may optionally choose to specify objectives for each task. The performance of an objective is determined by looking at the measures connected to the objective.

Generalizations

- Class from Core

Attributes

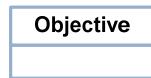
- `measureList`: List of measures connected to the objective.
- `performanceCriteria`: The criteria or formula that is based on `measureList` and calculated to determine the performance of the task or mission.

Constraints

[C-MS-d-1] An objective should be related with at least a measure

context Objective *self.measureList.size() > 0*

Notation



An objective is represented by a rectangle with two compartments, with the name of the objective placed at the top compartment.

e. Measure

A measure is a quantifiable element defined to determine the performance of an objective. An objective is supposed to be successful if it satisfies all of the measures connected to it.

Generalizations

- Class from Core

Attributes

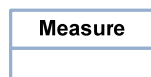
- `unit`: A quantifiable measurement unit such as meter, hour, count, etc.

Constraints

[C-MS-e-1] A measure should be related with at least an objective

context Measure *exists o:Objective in objectives where o.measureList.includes(self)*

Notation



A measure is represented by a rectangle with two compartments, with the name of the measure placed at the top compartment.

f. WorkProduct

A work product represents any input consumed or any output produced by a task. A work product may be produced by a task and consumed within the context of the same task as well as it can be consumed by another task. Production of a work product does not necessarily mean that the work product is created by the task; it may also imply a modification.

A work product may be composed of smaller components or may take part in an inheritance hierarchy of work products. The relationships among the work products can be shown in the Entity Relationships diagram.

A work product carries only data by means of its attributes, and therefore does not have any capability associated with it.

Generalizations

- Class from Core

Attributes

- No additional attributes

Constraints

[C-MS-e-1] A work product does not have any capabilities

context WorkProduct *self.capabilityList.size() = 0*

Notation



A workproduct is represented by a rectangle with two compartments, with the <<Workproduct>> label and name of the workproduct placed at the top compartment. The bottom compartment is used to show the attributes of the workproduct.

g. DecisionPoint

A decision point is a control node where the modeler represents a decision situation. A decision point has one incoming connection and may have multiple outgoing connections. A decision node may also represent multiple incoming connections merged into a single outgoing connection. All of the outgoing connections from a decision node should have guard conditions as labels. These guard conditions must not conflict with each other.

Generalizations

- ControlNode from Core

Attributes

- No additional attributes.

Constraints

[C-MS-g-1] The connections coming into and going out of a decision point must be task flows

context DecisionPoint *self.incoming.oclIsTypeOf(TaskFlow) AND self.outgoing.oclIsTypeOf(TaskFlow)*

[C-MS-g-2] The guard conditions on multiple outgoing connections from a decision point must not be the same

context DecisionPoint *self.outgoingConnections->forAll(c1, c2 | c1 <> c2 implies c1.guardCondition <> c2.guardCondition)*

[C-MS-g-3] Every outgoing connection from a decision point must have a guard condition

context DecisionPoint *forall c: outgoing in outgoingConnections not (isEmpty(c.guardConditionList))*

Notation



A decision point is represented by a diamond symbol. The label of the decision point can be floating and can be placed anywhere close to the decision point symbol.



Decision point used to show branching.



Decision point used to show merging.

h. SynchronizationPoint

A synchronization point is a control node that splits a flow into concurrent flows, or joins multiple flows in order to synchronize them.

Generalizations

- ControlNode from Core

Attributes

- No additional attributes.

Constraints

[C-MS-h-1] A synchronization point having multiple incoming connections may have a single outgoing connection

context SynchronizationPoint $self.incomingConnections.size() > 1$
implies $self.outgoingConnections.size() = 1$


[C-MS-h-2] A synchronization point having multiple outgoing connections may have a single incoming connection

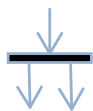
context SynchronizationPoint $self.outgoingConnections.size() > 1$
implies $self.incomingConnections.size() = 1$

[C-MS-h-3] The connections coming into and going out of a synchronization point must be task flows

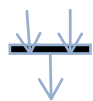
context SynchronizationPoint
 $self.incomingConnection.oclIsTypeOf(TaskFlow)$ *AND*
 $self.outgoingConnection.oclIsTypeOf(TaskFlow)$

Notation

 A synchronization point is represented by a horizontal or a vertical bar.



A synchronization point used to show a fork.



A synchronization point used to show a join.

i. InitialTask

An initial task is a control node at which flow starts when a task is invoked. A task may not have more than one initial task.

Generalizations

- ControlNode from Core

Attributes

- No additional attributes.

Constraints

[C-MS-i-1] An initial task has no incoming connections

context InitialTask *self.incomingConnections.size() = 0*

Notation



An initial task is represented by a filled circle.

j. FinalTask

A final task is a control node that stops all flows in a task flow diagram. A task may have more than one final task. The first one reached stops all flows in the task. The modeler may specify the unsuccessful final tasks by setting the `isSuccessful` attribute to false.

Generalizations

- ControlNode from Core

Attributes

- `isSuccessful`: Boolean value that is used to identify unsuccessful termination of a task.

Constraints

[C-MS-j-1] A final task has no outgoing connections.

context FinalTask *self.outgoingConnections.size() = 0*

Notation



A final task is represented by a circle surrounding a small solid filled circle.

k. ResponsibleFor

A responsible for relation represents the relation between a role and a task or mission. The role connected to a task or mission with this relation is responsible for the execution of the task or mission.

Generalizations

- Association from Relationships

Attributes


- No additional attributes.

Constraints

[C-MS-k-1] A responsible for relation has one role as source and one task or mission as target.

context ResponsibleFor *self.source.isTypeOf(Role)* AND
(*self.target.isTypeOf(Task)* OR *self.target.isTypeOf(Mission)*)

Notation

responsibleFor  A responsible for relation is shown with a dashed arrow labeled as responsibleFor.

l. Realizes

A realizes relation represents the relation between a role and a task or mission. The role connected to a task or mission with this relation realizes the execution of the task or mission.

Generalizations

- Association from Relationships

Attributes

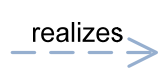
- No additional attributes.

Constraints

[C-MS-l-1] A realizes relation has one role as source and one task or mission as target.

context Realizes *self.source.isTypeOf(Role)* AND
(*self.target.isTypeOf(Task)* OR *self.target.isTypeOf(Mission)*)

Notation

 A realizes relation is shown with a dashed arrow labeled as realizes.

m. Extends

An extends relation specifies how and when the behavior defined in the extending mission (source) can be inserted into the behavior defined in the extended mission (target). This relationship specifies that the behavior of a mission may be extended by the behavior of another (usually supplementary) mission. The extension takes place at one or more specific extension points defined in the extended mission. The task flow diagram related with the mission should have a task that is specified as an extension point.

It should be noted that the extended mission is defined independently of the extending mission and is meaningful independently of the extending mission. On the other hand, the extending mission typically defines behavior that may not necessarily be meaningful by itself. The same extending mission can extend more than one mission. Furthermore, an extending mission may itself be extended.

When a mission extends another mission, the flow of the extended mission is stopped for a while, the flow of extending mission is executed and after it is finalized the extended mission flow continues from the extension point. The extended mission may have more than one extension points; therefore the extends relation should include the extension point information together with the extension condition. These information can be attached to the extends relation with a note model element.

Generalizations

- Association from Relationships

Attributes

- No additional attributes.

Constraints

[C-MS-m-1] An extends relation has one mission as source and one mission as target

context Extends *self.source.isTypeOf(Mission)* AND
(self.target.isTypeOf(Mission))

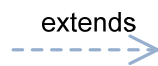
[C-MS-m-2] An extends relation cannot have the same mission both as source and as target

context Extends $Mission1 = self.source$ AND $Mission2 = self.target$
implies not (Mission1=Mission2)

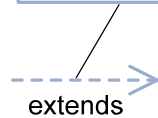
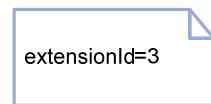
[C-MS-m-2] Extended mission must have a related task flow diagram with at least one task whose isExtensionPoint attribute set to true.

context Extends $self.target.subDiagram.includes$
(Task.isExtensionPoint = true)

Notation



An extends relation is shown with a dashed arrow labeled as extends.



Extension point information can be attached to the extends relation by a Note element

n. Includes

An includes relation defines that a mission contains the behavior defined in another mission. Includes is a directed relationship between two missions, implying that the behavior of the included mission (target) is inserted into the behavior of the including mission (source). The including mission may only depend on the result (value) of the included mission, which is obtained as a result of the execution of the included mission. The included mission is not optional, and is always required for the including mission to execute correctly

An includes relationship between two missions means that the behavior defined in the including mission is included in the behavior of the base mission. The includes relationship is intended to be used when there are common parts of the behavior of two or more missions. This common part is then extracted to a separate mission, to be included by all the base missions having this part in common. Execution of the included

mission is analogous to a subroutine call. All of the behavior of the included mission is executed at a single location in the included mission before execution of the including mission is resumed.

Generalizations

- Association from Relationships

Attributes

- No additional attributes.

Constraints

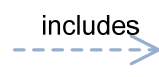
[C-MS-n-1] An includes relation has one mission as source and one mission as target

context Includes *self.source.isTypeOf(Mission) AND (self.target.isTypeOf(Mission))*

[C-MS-n-2] An includes relation cannot have the same mission both as source and as target

context Includes *Mission1 = self.source AND Mission2 = self.target implies not (Mission1=Mission2)*

Notation

 An includes relation is shown with a dashed arrow labeled as includes.

o. Achieves

An achieves relation is defined between a mission or task and an objective. A mission or task may have many Objectives. A mission or task will be counted as successful when it achieves all of its objectives.

Generalizations

- Association from Relationships

Attributes


- No additional attributes.

Constraints

[C-MS-o-1] An achieves relation has a mission or task as source and an objective as target

context Achieves (*self.source.isTypeOf(Task)* OR *self.source.isTypeOf(Mission)*) AND (*self.target.isTypeOf(Objective)*)

Notation

achieves  An achieves relation is shown with an arrow labeled as achieves.

p. TaskFlow

A task flow defines the flow of tasks in a task flow diagram. The sequence of flow implies the execution order of tasks; therefore the source of a task flow is executed before the target. Additional time constraints can be expressed by guard conditions and notes.

The node at the target of a task flow is executed only after the execution of the source node has been completed.

Generalizations

- ControlFlow from Core

Attributes

- No additional attributes.


Constraints

[C-MS-p-1] A task flow may have one of the {task, decision point, synchronization point and initial task} as source and one of the {task, decision point, synchronization point and final task} as target

context TaskFlow (*self.source.isTypeOf(Task)* OR *self.source.isTypeOf(DecisionPoint)* OR *self.source.isTypeOf(SynchronizationPoint)* OR *self.source.isTypeOf(InitialTask)*) AND (*self.target.isTypeOf(Task)* OR *self.target.isTypeOf(DecisionPoint)* OR *self.target.isTypeOf(SynchronizationPoint)* OR *self.target.isTypeOf(FinalTask)*)

[C-MS-p-2] Only one task flow may exist between the same source and target
context TaskFlow $T=$ *self.target* AND $S=$ *self.source* implies not ($S=T$)

Notation

 A task flow relation is shown with an arrow.

q. InputTo (Consumes)

An inputTo relation is used to represent the workproducts required for execution of a task. There may be many workproducts required as input for a task. InputTo is a directed relationship having a workproduct as source and a task as target.

Generalizations

- Association from Relationships

Attributes


- No additional attributes.

Constraints

[C-MS-q-1] An inputTo relation has one workproduct as source and one task as target

context InputTo $self.source.isTypeOf(WorkProduct)$ AND
 $(self.target.isTypeOf(Task))$

Notation

 An inputTo relation is shown with a dotted arrow labeled as inputTo.

r. Produces (OutputFrom)

A produces relation is used to represent the workproducts produced after or during execution of a task. There may be many workproducts produced as output of a task. Produces is a directed relationship having a task as source and a workproduct as target.

Generalizations

- Association from Relationships

Attributes


- No additional attributes.

Constraints

[C-MS-r-1] A produces relation has one task as source and one workproduct as target

context Produces *self.source.isTypeOf(Task) AND (self.target.isTypeOf(WorkProduct))*

Notation

produces  A produces relation is shown with a dotted arrow labeled as produces.

s. QuantifiedBy

A quantifiedBy relation is used to represent the relation between an Objective and a Measure. The performance of an objective is determined by evaluating the measures that are connected to the objective via quantifiedBy relationships.

Generalizations

- Association from Relationships

Attributes


- No additional attributes.

Constraints

[C-MS-s-1] A quantifiedBy relation has one Objective as source and one Measure as target

context QuantifiedBy *self.source.isTypeOf(Objective) AND (self.target.isTypeOf(Measure))*

Notation

quantifiedBy  A quantifiedBy relation is shown with an arrow labeled as quantifiedBy.

4.3.5. Diagrams

There are two types of diagrams within the scope of Mission Space package. These are Mission Space Diagram and Task Flow Diagram. The following sections describe these diagram types by providing examples.

a. Mission Space Diagram

Conceptual modeling activities begin with the definition of a mission space diagram that is aligned with the simulation objectives. Missions can be defined as high level tasks that are essential for implementation of the simulation system. A typical mission space diagram includes the missions, roles, objectives, measures and relationships among them. There may be many mission space diagrams representing a mission space. The diagram in Figure 19 shows all available model elements that could be used in a mission space diagram. Although a mission space diagram resembles a UML Use Case diagram in terms of the level of information represented and the terminology used, it has major structural differences as described in Section 4.3.3.

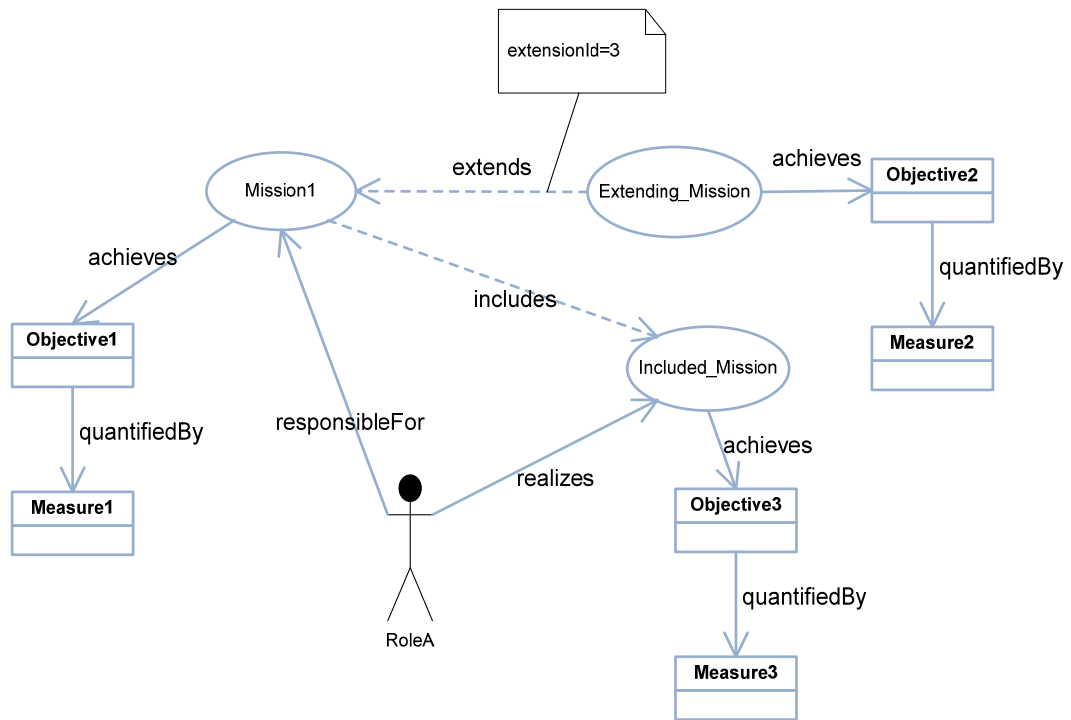


Figure 19: Mission Space Diagram

In this sample mission space diagram there are 3 missions. Mission1 includes the Included_Mission and is extended by the Extending_Mission. The semantics of the Included_Mission is similar to a procedure call. The Included_Mission is called at a specific point identified in the task flow diagram of the Mission1. After the execution of the Included_Mission, the control is passed back to Mission1 and the flow of Mission1 continues. The

Extending_Mission extends Mission1 at the extension point where the extensionId equals to three. As shown in the task flow diagram of Mission1 in Figure 20, Task5 is the point where the Included_Mission is called and Task3 is the extension point.

The objectives of each task are separately shown in the mission space diagram. Also the measures that are used to quantify these objectives and evaluate their performance are shown.

RoleA is responsible for Mission1 and also realizes the Included_Mission. There are not any semantic difference between responsible for and realizes relationships, however, they are used to represent two distinct operational responsibilities in the real world. In the military domain, generally it is essential to distinguish the ordering and performing people.

b. Task Flow Diagram

Each and every mission within a mission space is detailed using a task flow diagram. The task flow diagram is composed of tasks which also can be detailed using task flow diagrams. This recursive nature of task flow diagram enables the modeler to specify any level of detail as required. As task and mission share the same parents (Activity from Core package) in the KAMA metamodel, they have the same properties. However, task also inherits from the ActivityNode from the Core package, which makes it a superset of mission. In this context, task can be thought of as a low-level mission that can also include sub-tasks. A typical task flow diagram includes the tasks, roles, work products, decision points, synchronization points and the flow within these elements. The diagram in Figure 20 shows a sample task flow diagram that belongs to Mission1.

The task flow diagram is composed of 5 tasks, two of which include inner task flow diagrams as they are shown as rounded rectangles filled with blue color. The task begins with the InitialTask and flows to Task1, which is realized by RoleB. The execution of Task1 results in the production of Workproduct1. This work product may or may not be consumed within this particular task. Following Task1, there is a fork node which implies the parallel execution of Task2 and Task3. The semantics of KAMA notation does not specify the timing of these parallel tasks, they may begin at the same moment or there may be some delay between their executions. Task2 uses Workproduct2 as input and Task 3 uses Workproduct1 that is produced by Task1 as input. The Workproduct2 can be used as an input by any of the tasks in the task flow diagram of Task2. Task3 is specified as an extensionPoint and marked with extensionId 3. The Extending_Mission in Figure 19 extends Mission1 at this point.

Especially for the tasks with inner task flow diagrams, it is not mandatory to show the realizing Role. There may be many roles within this kind of tasks and it may be difficult to specify a common role for the task; however it may be appropriate to show the responsible role.

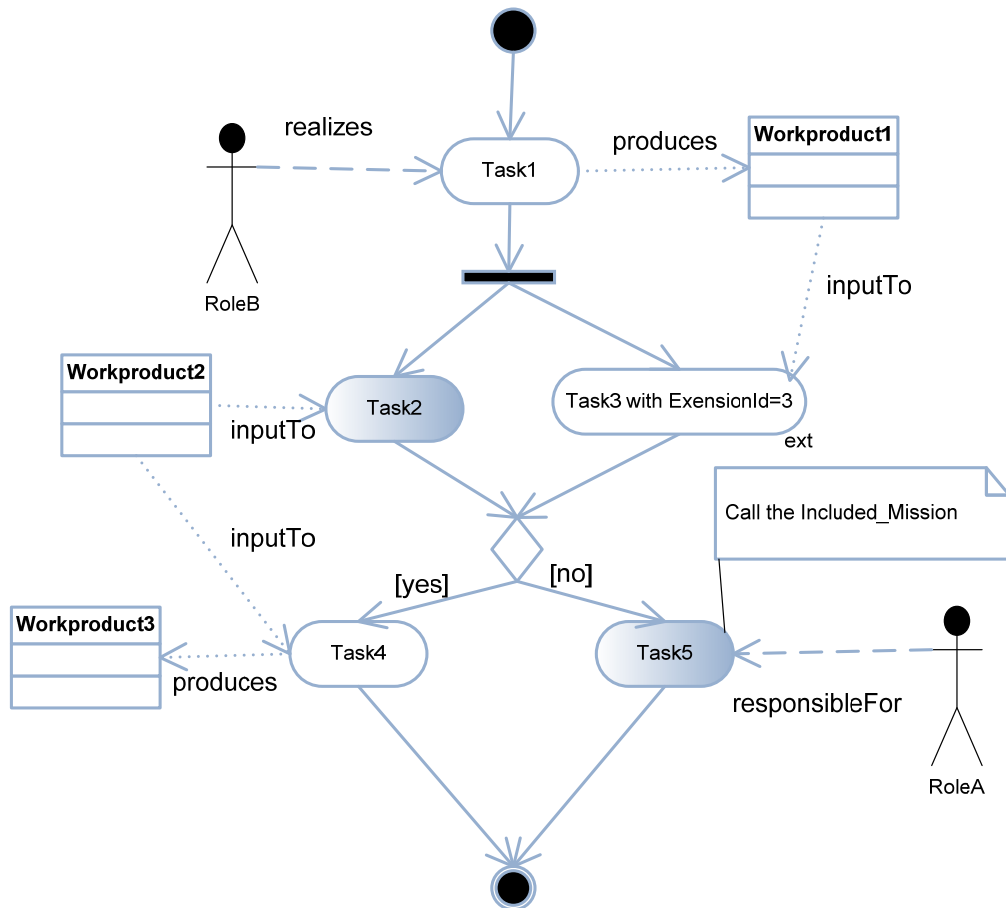


Figure 20: Task Flow Diagram

As one of the parallel tasks terminate, the control passes to the decision point. It is not necessary that both of these tasks terminate to evaluate the conditions at the decision point. If the modeler wants to wait for both of the tasks to terminate to proceed into the next step, she has to use a synchronization point as a join node. If the evaluation result at the decision point is “yes” the flow passes to Task4, if it is “no” then the flow passes to Task5. Task4 uses Workproduct2 as input and produces Workproduct3 as output. Task5 is the point where the Included_Mission that is shown in Figure 19 is called. Depending on the result at the decision point only one of these tasks are executed and after the execution is completed the flow passes to the FinalTask implying the termination of the Mission1.

4.4. Structure Package

4.4.1. Overview

The structure package is based on the Foundation package and includes the metamodel elements that are used to represent the static structure of a conceptual model. The structural elements are mostly based on UML classes but with additional constraints. The structural view of a conceptual

model is represented by 4 types of diagrams namely; Entity Ontology, Entity Relationship, Command Hierarchy and Organization Structure diagrams. The following sections explain the model elements required to develop these diagrams.

4.4.2. Abstract Syntax

The inheritance hierarchy of the Structure package metamodel elements is depicted in Figure 21.

The Structure metamodel elements are inherited from the Core and Relationships sub-packages in the Foundation package. In order to prevent name conflicts, the metamodel elements having the same name with their super classes are attached the word “KAMA” as prefix.

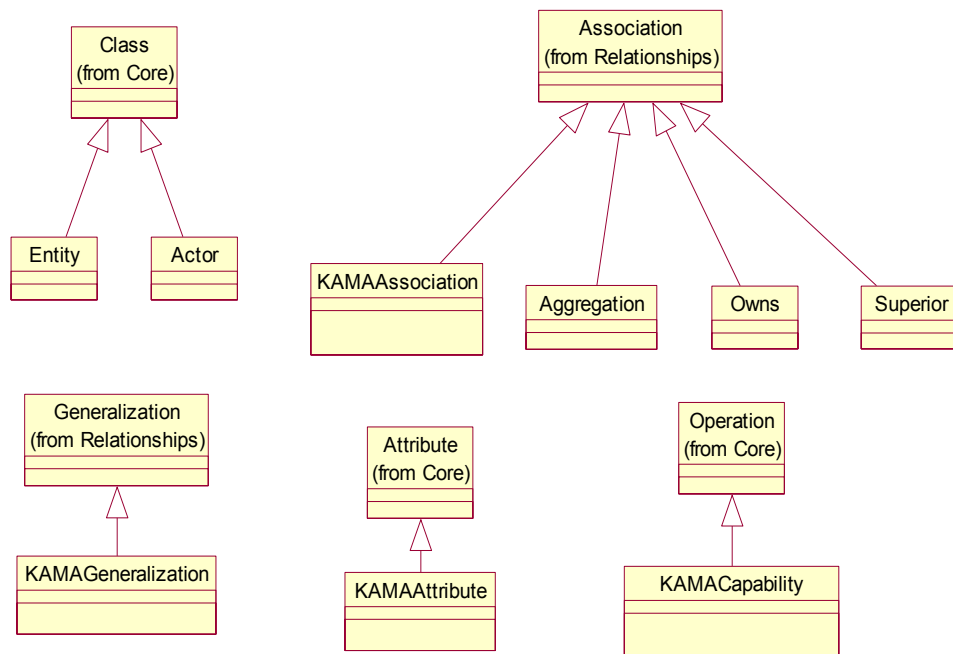


Figure 21: Structure Metamodel

4.4.3. Class Descriptions

a. Entity

An entity is used to represent any logical or physical entities in the mission space that should be included in the conceptual model. These could be any item with distinguishable features possessing necessary information to be stored, such as person, place, organization, material, object or concept. Entity is one of the most frequently used metamodel elements.

Generalizations

- Class from Core

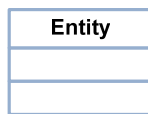
Attributes

- attributes: The list of attributes that are owned by an entity.
- capabilities: The list of capabilities that are owned by an entity.
- associationList: The list of incoming associations to an entity.

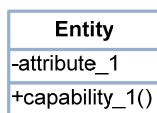
Constraints

[C-ST-a-1] No additional constraints.

Notation



An entity is shown with a rectangle with three compartments. The top compartment includes the name of the entity.



The middle compartment is used to display the attributes and the bottom compartment is used to display the capabilities of an entity.

b. Actor

An actor is an abstract class representing the command units in a mission space. Actor can represent a human being or any other active entity. Actors are derived from entities. Actors cannot perform tasks but they are required for defining roles that can perform these tasks. Actors are depicted in the Command Hierarchy diagrams.

Actor metamodel element is used for representing the more static nature of command hierarchy. This hierarchy generally does not change often and there may be some roles required by some missions that are not defined within this hierarchy. If these actor definitions are stored in a central repository, they can be reused among different simulation projects.

Generalizations

- Entity from Structure

Attributes

- roleList: List of roles that are owned by an actor.

Constraints

[C-ST-b-1] No additional constraints.

Notation



An actor is represented by a stick man.

c. KAMAAttribute

A KAMAAttribute represents the characteristic information about an Entity.

Generalizations

- Attribute from Core

Attributes

- No additional attributes.

Constraints

[C-ST-c-1] No additional constraints.

Notation

`name : type =
initialValue`

KAMAAttribute does not have a specific graphical notation. It is shown as a text string. The default syntax is as shown on the left column.

d. KAMACapability

A KAMACapability represents the functions that a Entity may perform. For example a tank entity may have the following capabilities; “move”, “fire”, “rotate the gun”, “climb”, etc. These capabilities are required for taking part in the tasks.

Generalizations

- Operation from Core

Attributes

- No additional attributes.

Constraints

[C-ST-d-1] No additional constraints.

Notation

`KAMACapability` does not have a specific graphical notation. It is shown as a text string. The default syntax is as shown.

e. KAMAGeneralization

KAMAGeneralization is a taxonomic relationship between a more general Entity or Workproduct and a more specific Entity or Workproduct. The specific Entity or Workproduct (known as child) inherits the attributes and capabilities of the general Entity or Workproduct (known as parent). KAMAGeneralization is a directed, transitive, anti-symmetric relationship. The direction of the arrow is from the child to the parent.

A Entity or Workproduct may not have itself as a parent or child. A Entity or Workproduct may have multiple parents. In this case the child inherits structure and behavior from its parents. If there are any conflicts among the parents, such as having the same capability with different behavior, it is the responsibility of the modeler to resolve the conflict by redefining the capability in the child.

Generalizations

- Generalization from Relationships

Attributes

- No additional attributes.

Constraints

[C-ST-e-1] A KAMAGeneralization may not have the source and target as the same Entity

context Generalization $T=self.target$ AND $S=self.source$ implies not $(S=T)$

[C-ST-e-2] A KAMAGeneralization may not be part of a cyclic graph

context Entity *not self.allParents()->includes(self)*

Notation



A generalization relation is shown with an arrow.

f. KAMAAssociation

A KAMAAssociation is the most basic relationship that can take place between entities and work products. The navigation path is shown with the direction of the arrow. An association line without any arrowheads implies that the association is bidirectional.

Generalizations

- Association from Relationships

Attributes

- No additional attributes.

Constraints

[C-ST-f-1] No additional constraints.

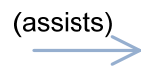
Notation



A bidirectional association is shown with a line.



A unidirectional association is shown with an arrow. The direction of the arrow shows the navigation path.



The name of the association can be shown above the association line within matched parentheses.

g. Aggregation

An aggregation relation is available among entities, work products and actors. Aggregation implies a whole-part relationship between the source and target model elements. The diamond like shape is close to the whole model element. An aggregation is used to show a strong association between two model elements. For example a plane cannot be considered as complete without its engine, therefore it is logical to define an aggregation relation between the engine entity and the plane entity. However, this

relationship can also be shown by embedding the attributes and capabilities of engine into the plane entity. It is the choice of the modeler to select among these options.

Generalizations

- Association from Relationships

Attributes

- No additional attributes.

Constraints

[C-ST-g-1] An aggregate may not be part of any of its parts in an aggregation relationship.

[C-ST-g-2] The source and target of an aggregation relationship may not be the same model element.

context Aggregation $T=self.target$ AND $S=self.source$ implies not $(S=T)$

[C-ST-g-3] The source and target of an aggregation relationship must be of the same kind. If the source is one of {entity, workproduct, actor}, the target must be { entity, workproduct, actor} respectively.

context Aggregation $self.source.oclIsTypeOf(Entity)$ implies $self.target.oclIsTypeOf (Entity)$ AND $self.source.oclIsTypeOf(WorkProduct)$ implies $self.target.oclIsTypeOf (WorkProduct)$ AND $self.source.oclIsTypeOf(Actor)$ implies $self.target.oclIsTypeOf (Actor)$

Notation



An aggregation is shown with a diamond shaped arrowhead.

The diamond is close to the whole.

h. Owns

An owns relationship is used to represent the relation between an actor and the roles that are owned by that actor. It is a specialized type of association that is available only between an actor and a role. The direction of the owns relationship is from the actor to the role.

Generalizations

- Association from Relationships

Attributes


- No additional attributes.

Constraints

[C-ST-h-1] An owns relation may exist between an actor (source) and a role (target).

context Owns *self.source.oclIsTypeOf(Actor) implies self.target.oclIsTypeOf(Role)*

Notation

 An owns relation is shown with an arrow labeled as owns.

i. SubordinateOf

A subordinateOf relation is used to represent the hierarchy between actors in the command hierarchy diagram. The direction of the relation is from the subordinate (inferior) actor to the superior actor. Although not being a strict rule, generally the actor that is responsible for a task is superior to the actor realizing that task.

Generalizations

- Association from Relationships

Attributes

- No additional attributes.

Constraints


[C-ST-i-1] A subordinateOf relation may exist between an actor (source) and another actor (target).

context SubordinateOf *self.source.oclIsTypeOf(Actor) implies self.target.oclIsTypeOf(Actor)*

[C-ST-i-2] A subordinateOf relation may not be part of a cyclic graph.

context SubordinateOf *T=self.target AND S=self.source implies not (S=T)*

Notation

subordinateOf  A subordinateOf relation is shown with an arrow labeled as subordinateOf.

4.4.4. Diagrams

There are four types of diagrams that are used to represent the structural aspects of the mission space. Entity Ontology and Entity Relationships diagrams are similar to each other syntactically and semantically, however there is a difference in their usage. Entity ontology diagrams are used to represent common entities that can be reused among the simulation systems and focus on the inheritance and aggregation relationships among the entities. It would be a best practice to store the entity ontology diagrams in a central repository and share among simulation development projects. Entity Relationships diagrams are used to depict the relationships between entities local to the mission space and also the relationship between work products can be shown. Both of these diagrams are derived from UML Class diagrams.

Command Hierarchy and Organization Structure diagrams used to show the roles and actors in the mission space. Command hierarchy diagram can be extensively defined and reused among the simulation projects. Organization structure diagram defines the roles that will take place in a task or mission. These two diagrams are also based on UML Class Diagrams with additional constraints.

a. Entity Ontology Diagram

As the modeler defines the missions and tasks in the mission space, she begins to identify the structural elements in the form of entities. An entity may be any item with distinguishable features and possesses necessary information to be stored, such as person, place, organization, material, object or concept. Entities generally map to domain classes in the object oriented software development paradigm. Most of these entities may be defined as classes during the simulation system design phase. It should be noted that, the focus of the modeler during the conceptual development phase is to identify the domain entities, their attributes and capabilities from an analysis point of view. However, the developers during the simulation system design phase focuses on developing a solution that will correspond to the conceptual model.

The inheritance and aggregation relations are depicted in the entity ontology diagrams. There may be a number of entity ontology diagrams within a mission space; however they should be connected through common elements. For example, the modeler may select to show the inheritance hierarchy of Child_Entity1 in a separate diagram other than the one shown in Figure

22. But the reader of the conceptual model will notice that these two diagrams are connected to each other through common entities.

The sample diagram in Figure 22, shows an inheritance hierarchy including an aggregate entity. It should be noted that although the inheritance relation is shown over attributes, it is also valid for operations. The Child_Entity inherits attributes and operations from both Parent_Entity1 and Parent_Entity2. Both these parent entities inherit from the same Ancestor_Entity. Parent_Entity2 is an aggregate of Part_Entity1 and Part_Entity2.

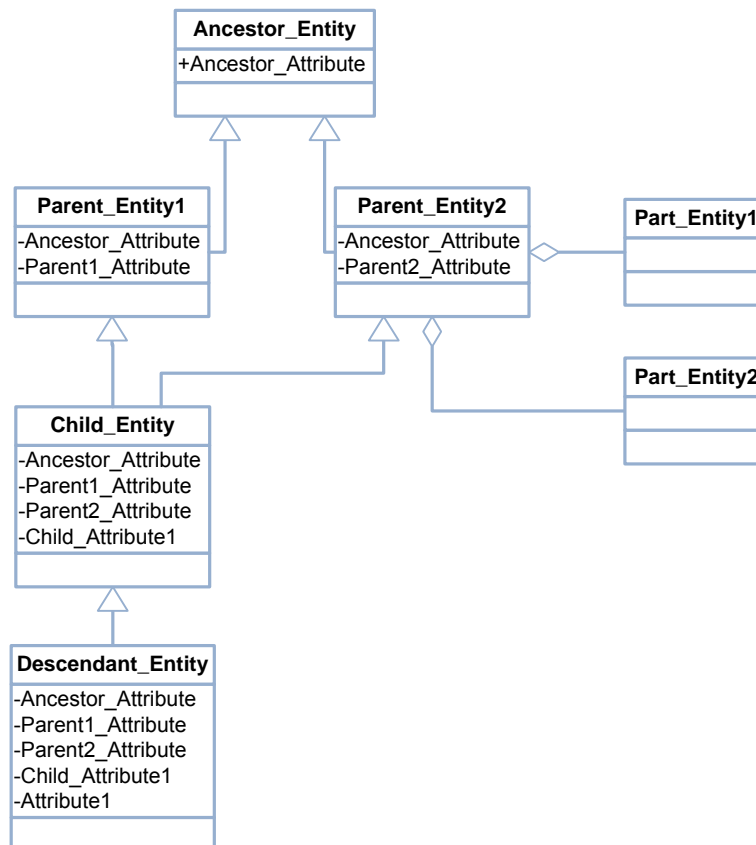


Figure 22: Entity Ontology Diagram

The inheritance hierarchy may not be cyclic, that is a parent may not inherit from a child or descendant; or an ancestor may not inherit from a parent, child or descendant.

Aggregation is a special kind of association, which implies a whole-part relationship between two entities. Parent_Entity2 is an aggregate entity that is composed of Part_Entity1 and Part_Entity2. It is semantically equivalent to represent these relationships by embedding the attributes and capabilities of part entities within the parent entity. However, if the part entities have inheritance or aggregation relationships with other entities, it is more suitable to represent them as distinct entities.

b. Entity Relationship Diagram

As we have pointed out before, although Entity Relationship diagrams are structurally similar to the Entity Ontology diagrams they serve different purposes. Entity relationship diagrams are used to represent the relationships between entities that are identified as specific to a mission space. Additionally, the relations between work products can be represented using entity relationship diagrams. A sample entity relationship diagram is shown in Figure 23.

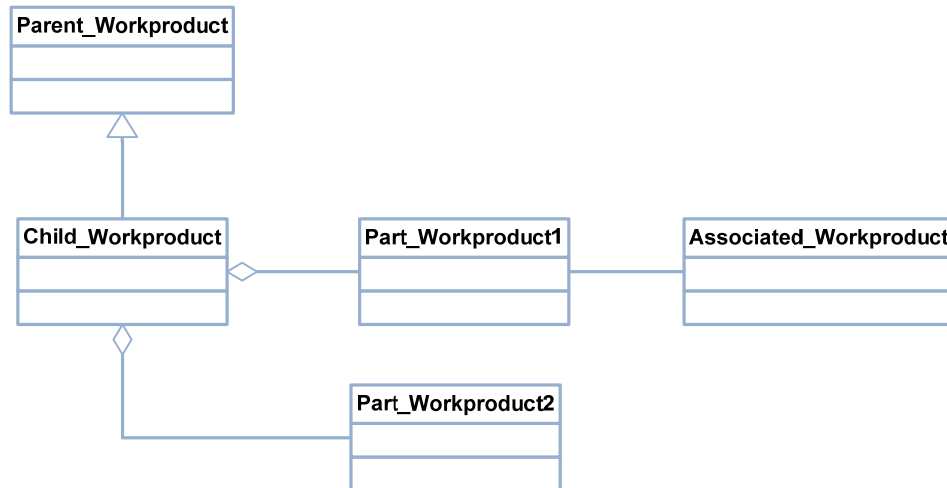


Figure 23: Entity Relationship Diagram

The available relationship types are generalization, aggregation and association. Association is the most basic relationship between two entities, representing any kind of relation such as one entity is affected by another's attributes, state changes or capabilities. Generalization and aggregation relationship have the same semantics as in the Entity Ontology diagrams.

c. Command Hierarchy Diagram

The hierarchy among the command units is depicted in the Command Hierarchy diagrams. This is a specialized UML Class diagram having actors, subordinateOf and aggregation relationships available. A sample command hierarchy diagram is shown in Figure 24.

Generally the command hierarchy of a domain is defined by a central authority and is subject to change rarely. Although it is a rare occasion, any changes must be reflected to every conceptual model related with this domain. Considering these facts, it will be a best practice to keep the command hierarchy diagram in a central repository and reuse among the simulation projects.

Command hierarchies may include aggregate actors such as groups and teams as well as individual actors. The sample diagram includes a Level1_Actor at the top of the hierarchy, which is superior to Level2_Actor and Level2_GroupActor. Level2_GroupActor is composed of GroupMember_Actor1 and GroupMember_Actor2. The command hierarchy may be represented

as multiple diagrams in order to reduce the complexity of the diagram; however, these diagrams should be connected to each other through common elements. For example Level1_Actor as the topmost element in this diagram may be at the bottom of another command hierarchy diagram. It will be better to utilize a visualization mechanism that enables the modeler view this kind of connections among various diagrams.

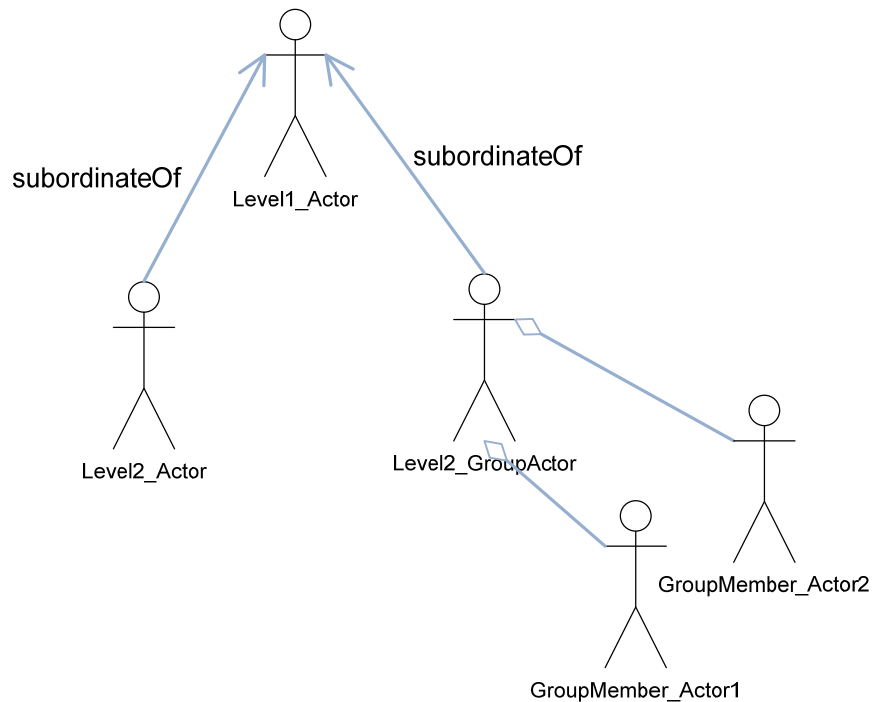


Figure 24: Command Hierarchy

The command hierarchy is used as a basis when defining the organization structure of the mission space.

d. Organization Structure Diagram

Organization structure diagram is used to depict the relationships between the actors in the command hierarchy and the roles that can take part in the task flow diagrams. Any role in an organization structure diagram should be owned by at least an actor; otherwise it will be regarded as incomplete and invalid.

A sample organization structure diagram is shown in Figure 25. ActorA owns the roles Role1 and Role2, which implies that ActorA plays two different roles in the task flow diagrams. Role2 is also owned by ActorB, meaning that both ActorA and ActorB can play the role Role2 in the task flow diagrams.

For example; in case of an infiltration mission the infiltration activities shall be performed by the infiltration team. Generally the command hierarchies do not include this kind of specific command

units, but a more general command unit such as specialized team may be defined as an actor in the command hierarchy. Then an organization structure diagram will include the specialized team actor owning the infiltration team as a role.

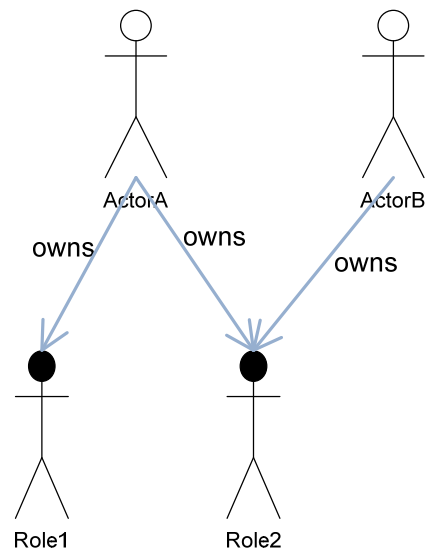


Figure 25: Organization Structure Diagram

4.5. Dynamic Behavior Package

4.5.1. Overview

The dynamic behavior package is based on the Foundation package and includes the metamodel elements that are used to represent the dynamic behavior of a conceptual model by focusing on the internal state changes of an entity. Any entity that has complex behavior or behaves reactively may be represented by a state machine.

4.5.2. Abstract Syntax

The metamodel elements in the dynamic behavior package are depicted in Figure 26. The dynamic behavior package uses all of the metamodel elements defined in the Behavior sub-package of the Foundation package. The constraints and semantic differences are given in Section 4.5.3.

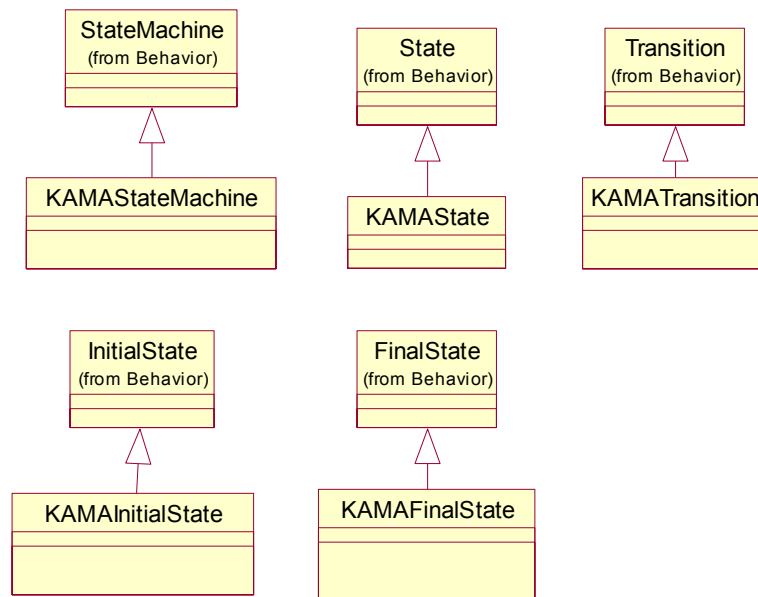


Figure 26: Dynamic Behavior Metamodel

4.5.3. Class Descriptions

a. KAMAStateMachine

KAMAStateMachine is used to express the behavior of a conceptual model entity. Behavior is modeled as a graph of state nodes interconnected by one or more joined transitions that are triggered by the dispatching of series of events. An entity with complex behavior may have a state machine related with it. KAMA state machines are simplified versions of UML state machines and do not represent any execution behavior in contrary to the UML state machines.

Generalizations

- StateMachine from Behavior

Attributes

- No additional attributes.

Constraints

[C-SM-a-1] No additional constraints.

Notation

An entity state diagram is used to represent a KAMA state machine. A sample diagram is provided in Figure 27.

b. KAMASState

KAMASState is used to represent a condition or situation during the life of an entity during which it satisfies some condition, performs some activity, or waits for some event. An entity may be in various states throughout its lifetime. Only simple states, which are the states without any sub-states, are allowed in KAMA. The composite states and submachine states that are available in UML cannot be used.

Generalizations

- State from Behavior

Attributes

- isSimple: Boolean, default value = true
- isSubmachineState: Boolean, default value = false

Constraints

[C-SM-b-1] A KAMASState must be a simple state

context KAMASState *isSimple* = *content.isEmpty()*

Notation



A KAMASState is shown as a rectangle with rounded corners, with the state name shown inside the rectangle.

c. KAMAInitialState

A KAMAInitialState shows the entrance point of a KAMASStateMachine. It is not a real state but rather used as a syntactic means to indicate where the control should go. Therefore it must include an outgoing transition to another state. A KAMAInitialState cannot have an outgoing transition with an event trigger. Since it is a dummy state, control should immediately leave it. A KAMAInitialState cannot have an incoming transition.

Generalizations

- InitialState from Behavior

Attributes

- No additional attributes.

Constraints

[C-SM-c-1] A KAMAInitialState may have only one outgoing transition.

context KAMAInitialState $(self.oclIsKindOf(KAMAInitialState))$
implies $((self.outgoing->size() = 1))$

[C-SM-c-2] A KAMAInitialState may not have an incoming transition.

context KAMAInitialState $(self.oclIsKindOf(KAMAInitialState))$
implies $((self.incoming->size() = 0))$

Notation



A KAMAInitialState is shown with an arrow leaving a filled circle.

d. KAMAFinalState

KAMAFinalState is a special kind of state representing the completion point of a KAMASStateMachine. It can have any number of incoming transitions but cannot have any outgoing transitions.

Generalizations

- FinalState from Behavior

Attributes

- No additional attributes.

Constraints

[C-SM-d-1] A KAMAFinalState may not have an outgoing transition.

context KAMAFinalState $(self.oclIsKindOf(KAMAFinalState))$ *implies*
 $((self.outgoing->size() = 0))$

Notation



A KAMAFinalState is represented by a circle surrounding a small solid filled circle.

e. **KAMATransition**

A KAMATransition is a relationship within a KAMAStateMachine between two states (source and target) indicating that an object in the first state will perform specified actions and enter the second state when a specified event occurs and specified guard conditions are satisfied. It is used to depict changes in the lifetime of a KAMA entity. A KAMATransition can have single source state and single target state.

A KAMATransition may include the triggering event, the guard condition and the action. The triggering event is the event that causes the state change, the state change is evaluated with respect to the guard condition and following that the specified action takes place. The triggering events and the actions should be aligned with the capabilities of the KAMA entity owning the KAMAStateMachine.

A KAMATransition may include a guard condition which is a Boolean expression used to determine the target state after being evaluated. If a KAMATransition has no guard condition, it is treated to have a guard condition with a true default value. If a KAMAState has multiple KAMATransitions, the guard conditions of these transitions should not conflict. Otherwise, the KAMAStateMachine will be inconsistent and ill formed.

The contextMission attribute is used to identify the mission in which the transition is valid. The modeler may like to ignore transitions among the states of an entity that are valid only for some mission contexts.

Generalizations

- Transition from Behavior

Attributes

- contextMission: Defines the mission in which the transition is accepted to be valid.

Constraints

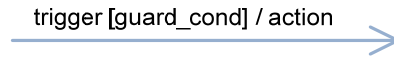
[C-SM-e-1] A KAMATransition can have one of {KAMAInitialState, KAMAState} as source and one of {KAMAFinalState, KAMAState} as target.

context KAMATransition (*self.source.isTypeOf(KAMAInitialState)* OR
self.source.isTypeOf(KAMAState)) AND
(self.source.isTypeOf(KAMAFinalState) OR
self.source.isTypeOf(KAMAState))

Notation



A KAMATransition is shown with an arrow.



A KAMATransition including a triggering event, a guard condition and an action.

4.5.4. Diagrams

a. Entity State Diagram

The dynamic behavior of an entity is depicted using the entity state diagrams. The modeler may optionally provide entity state diagrams for the entities having complex behavior. An entity state diagram may only be developed within the context of an entity defined in the entity ontology or entity relationship diagrams.

Entity state diagrams are used to represent the possible states of an entity, transitions among these states, the triggers initiating these transitions and events that are called after the transition. The entity state diagrams are simplified versions of UML State diagrams. Composite states and pseudo states other than the initial state are not allowed in KAMA. These metamodel elements increase the complexity of the entity state diagrams and therefore do not serve for our purposes.

It should be noted that the states and transitions are meaningful in the conceptual domain and should not reflect any information about the solution space that is the simulation system design. For example; the states defined for an airplane, gun, ship or submarine should be understandable by domain experts who are not experts in systems/software design. Therefore, it is very important to involve the domain experts in the validation of the conceptual models via joint reviews or walkthroughs.

The entity state diagram for Child_Entity (from Figure 22) is shown in Figure 27. The initial state flows to State1 which has multiple outgoing transitions. All of the transitions are triggered with the same trigger, trigger1. The guard condition specifies which transition should occur following the trigger. It is the responsibility of the modeler to develop a consistent entity state diagram, which does not include conflicting guard conditions. Depending on the guard condition, one of the transitions occurs and the related action is executed. If the transition having action1 as action is executed the transition flows to State2.

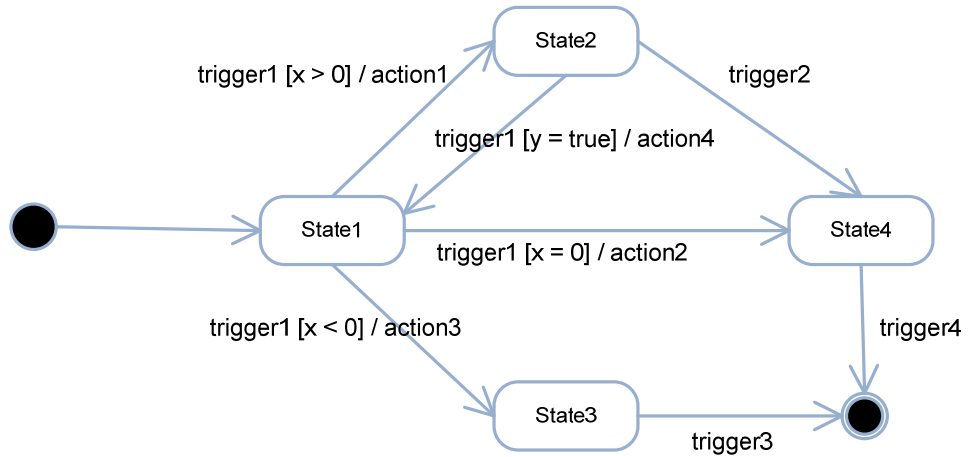


Figure 27: Entity State Diagram

The triggers may be formed by tasks in a task flow diagram or by the capabilities of an entity. Actions may also be related with the capabilities of entities. It will be a good practice to utilize a tool to verify that any action defined within an entity's state diagram can be mapped to the entity's capabilities.

State2 has two outgoing transitions depending on which trigger occurs. trigger1 takes the flow back to the State1 if the guard condition holds and trigger2 takes the flow to the State4 without checking any guard condition. The modeler should be careful not to cause any deadlocks by verifying the triggering events and guard conditions in this kind of cyclic transitions. An entity state diagram having deadlocks will be invalid.

Since there is not any guard condition defined for transitions leaving State3 and State4, the flow continues to the end state after the trigger3 and trigger4 occurs respectively.

CHAPTER 4

APPLICATION OF THE NOTATION

This chapter describes utilization of a multiple-case study to present the application of the KAMA notation. Two case studies were conducted to explore the applicability of the notation in real life settings. These case studies were performed by domain experts and our aim was to discover improvement opportunities for the KAMA notation and its applicability. Section 5.1 explains the research strategy followed and Section 5.2 describes the multiple-case study design parameters. Section 3.3 introduces the conceptual model development process that is utilized in the case studies. Sections 5.3 and 5.4 explain the conduct of the case studies and section 5.5 discusses the lessons learned and findings.

5.1. Research Strategy

As Yin has described case study as “an empirical inquiry that investigates a contemporary phenomenon within its real-life context especially when the boundaries between phenomenon and context are not clearly evident” in [74], we considered that case study would be an appropriate research strategy to investigate the applicability of the KAMA notation in real life settings. While examining a modeling notation, the phenomenon and the context may not be easily distinguishable and the researchers cannot manipulate the behaviors. These were the two most important arguments used to select case study as the appropriate research strategy together with our aim to evaluate the notation in natural settings.

The first major decision point in applying a case study is to determine whether it will be a single or multiple-case study. Single case studies are generally applicable when “the case represents a critical test of existing theory; the case is a rare or unique event or when the case serves as a revelatory purpose” [74]. Our case did not fit to any of the above situations and therefore was appropriate for applying a multiple-case study. Yin states that “multiple-case study should not be considered as a different methodology than single case-study” in [74]. A multiple-case study involves application of two or more single case studies with predefined objectives and the case study design should include replication logic. The replication logic used in the study is explained in Section 5.2.

5.2. Multiple-case study design

We selected two cases for our multiple-case study; one was a research center in the university and the other was a software and systems development company. The two cases aimed to address different research questions which are presented in Section 5.2.1. The first case study has been conducted right after the notation has been defined and aimed to evaluate the sufficiency of the notation for the development of a conceptual model. It was a retrospective case which aimed to re-develop a previously developed conceptual model using KAMA notation. It was also used to validate the tool that was developed to support the notation. The second case study aimed to evaluate the applicability of the notation in an industrial setting focusing on the development and use of the conceptual model. This case has been performed by modelers and domain experts with our assistance.

5.2.1. Research Questions

We determined the following research questions for the study:

- 1) What are the characteristics of a conceptual model?
- 2) How can we represent the characteristics of a conceptual model?
- 3) How sufficient is the KAMA notation for representing existing conceptual models that were developed as textual documents?
- 4) How effective is the KAMA notation for developing conceptual models of the mission spaces?

The first case study aimed to explore the notation, evaluate its sufficiency, identify weaknesses and propose improvement suggestions. Therefore, the first three questions were answered in the scope of the first case study. This study also helped us to determine the requirements of the tool which was developed as part of a research project. The feedbacks gathered from this study were reflected on both the notation and the tool.

In order to answer the first question, we examined the conceptual modeling literature, existing conceptual models and find out a common definition for what a conceptual model is. Then, we identified the essential elements of a conceptual model and designed a notation to represent these elements. The first case study focused on whether this notation was capable of re-developing an existing conceptual model. We identified three conceptual models defined for simulation development projects and selected the most comprehensive one among them for the case study. The selection criteria included the completeness of the model, the scope of the model and the implementation-independence of the model. One of the conceptual models has not been completed yet, therefore this option was eliminated. The remaining two models were complete and well-defined, however one of them was planned to be implemented as an agent oriented simulation and

included implementation details. We considered the modeling problems faced in this model, but selected the last model for our case study, which provided a complete, comprehensive and implementation independent conceptual model. In order to conduct the case study we applied the conceptual model development process for developing the conceptual models. This process definition is explained in detail in Section 3.3.

The second case study focused on the fourth question, which aims to explore and evaluate the effectiveness of the notation in an industrial setting. This study was performed by the domain experts and the modelers with our assistance. Modelers developed the conceptual model in an iterative fashion, this model was reviewed and validated by the domain experts and we monitored the execution of the process via joint review meetings. The feedbacks were gathered from this study through interviews with some of the participants and some of them are presented as future work in Section 6.2.

This thesis study proposes that KAMA is a sufficient notation for developing conceptual models including all of the essential features a conceptual model should possess and also provides a mechanism for increasing the usage of the conceptual models.

Only after the definition of a conceptual modeling notation, it is possible to evaluate the usage of it. There are two phases in the lifecycle of a model; “writing” which can be interpreted as the development phase of the conceptual model and “reading” which is the phase you try to understand and use what has been written. Our second case study aimed to evaluate both phases by examining the usage of the conceptual model by the domain experts.

5.3. Case Study 1

5.3.1. Background

The first case study was conducted as part of a research project that was realized by a consortium including two companies and the METU (Middle East Technical University). METU was in charge of research tasks and the companies performed the software development activities by establishing a professional software development team that includes an independent software quality assurance group. The number of people worked in the project, although varied among iterations was; between 5-7 persons in the software development team, between 2-3 people in the software quality assurance team and between 5-8 people in the research team. The research team included graduate students who performed some of the research tasks as part of their dissertation studies (i.e. MSc. or Ph.D. theses).

The objectives of the project related with our study were set to be;

- 1) Increasing the adaptability, reusability and interoperability of the M&S (Modeling and Simulation) applications through the use of common elements,

- 2) Supporting the acquisition phases of C4ISR systems through the use of conceptual models,
- 3) Increasing the quality and productivity of M&S developers through the use of validated and accredited information resources,
- 4) Increasing the quality of end products in M&S projects by supporting the validation of the conceptual models,
- 5) Providing usable and applicable solutions for conceptual model development related problems.

To accomplish these objectives the following high level tasks were defined by the project group;

- 1) Develop a conceptual modeling notation, define a common syntax and semantics
- 2) Develop a modeling tool that supports this notation,
- 3) Develop a common repository for storing reusable model elements
- 4) Validate the notation and the tool by developing the conceptual model of a C4ISR system.

The project was realized in 4 iterations. The first iteration included only the research related tasks and the remaining 3 iterations consisted of both research and development tasks. In the first iteration we investigated the literature to identify existing conceptual modeling definitions, notations and processes, sample conceptual models, available tools and any other related material. At the end of the iteration we came up with a notation for conceptual modeling and high-level software requirements for the tool that will support this notation. In the next 3 iterations the tool was developed in an incremental manner and the notation was validated at the end of each iteration. This is the point where our case study comes to the scene, with the objective of validating the notation, determining the weaknesses and proposing improvement suggestions both for the notation and the tool. Because of the iterative nature of the project, we implemented the case study in 3 iterations, focusing on different aspects at each iteration.

The research report [27], which was developed as an artifact of the first iteration included a literature survey on conceptual modeling for C4ISR systems, the definitions of the meta-model, the common syntax and semantics of the conceptual model, and explanations of the diagrams. We developed a process definition for conceptual modeling in order that the researchers could use during conduct of the case study. There were also a number of experimental studies performed based on the notation and the process definition included in the research report.

The scope of the case study was determined in the statement of work document. The conceptual model of a C4ISR system would be developed based on a 343 pages of System Requirements and Conceptual Model Document as the major information resource. The document included information on the method used for developing the conceptual model, which consisted of the following tasks;

- 1) Identify the high-level strategic and tactical missions,
- 2) Identify the existing conceptual models,
- 3) Analyze the missions,
- 4) Analyze the tasks,
- 5) Analyze the activities,
- 6) Analyze the actors and the entities,
- 7) Define the command and control hierarchy.

The conceptual model elements that are products of the above tasks were represented as structured textual explanations in the form of tables. Task definitions were also supported with drawings which resemble data flow diagrams. An example task definition diagram is depicted in Figure 28. A syntactic and semantic definition was not provided for these diagrams but they were used consistently throughout the document.

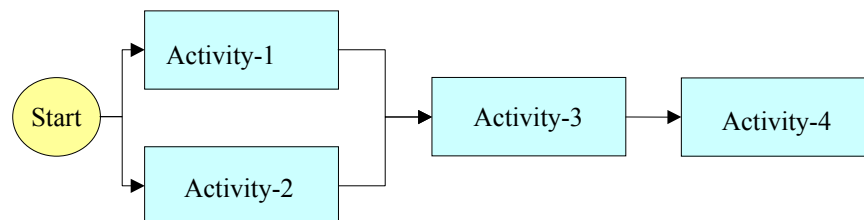


Figure 28: An example task definition diagram

The mission space consisted of 5 missions each of which was described with detailed task definitions. The following table is presented to give an idea about the scope of the conceptual model by providing the number of conceptual model elements.

Table 3: Conceptual Model Element Counts

Conceptual Model Element	Count
Missions	5
Tasks	54
Activities	94
Events	7
States	7
Entities	68
Data Items	29
Command and Control Units	12

5.3.2. Case Study Plan

a. Purpose and Scope

This section presents the plan for the execution of the KAMA notation in a C4ISR simulation system development project. The objective is to retrospectively develop the conceptual model of the mission space for the simulation system using the KAMA notation. A text based conceptual model document has been previously developed and it will be re-developed using the KAMA methodology and notation.

b. Case Study Organization

The case study roles are defined as follows:

Modeler: Responsible for developing the conceptual model using the KAMA notation and in accordance to the KAMA conceptual modeling process.

Reviewer: Responsible for performing the verification by reviewing the conceptual model that is developed by the modeler.

Domain Expert: Responsible for providing information when required and validating the conceptual model as planned.

c. Work Plan

The conceptual model is developed in 3 major iterations. The schedule for the project tasks and resource assignments and effort spent in person-hours is given in Table 4.

Table 4. Schedule Summary (Case Study 1)

ID	Phase	Start	Finish	Resource	Effort (person-hours)
1	Conceptual Model (CM) V1.0 Development	11.04.2006	21.04.2006	Modeler	54
2	Verification of CM V1.0	23.04.2006	25.04.2006	Reviewer	6
3	Conceptual Model (CM) V2.0 Development	17.12.2006	22.12.2006	Modeler	20
4	Verification of CM V2.0	24.12.2006	26.12.2006	Reviewer	6
5	Conceptual Model (CM) V3.0 Development	25.06.2007	28.06.2007	Modeler	18
6	Verification of CM V3.0	29.06.2007	02.07.2007	Reviewer	6
7	Validation of the CM	04.07.2007	06.07.2007	Domain Expert	12

d. Process and Outputs

The process defined in Section 3.3 of this thesis is used as a basis for the case study. We will perform a structured interview with the modelers in order to obtain feedback about the notation. These feedbacks will be used to improve the KAMA tool. Conceptual model is documented in the project's Research Report [27].

A training session is provided to the modeler, reviewer and the domain experts to introduce the basic concepts of conceptual modeling notation, process and usage of the KAMA tool

Verification of the conceptual model is performed informally at the end of each iteration by the two modelers who will cross-check the models developed by the other modeler. Formal verification is also performed at the end of third iteration by a separate researcher.

The expected outputs for the case study are as follows:

- 1) The Conceptual Model
- 2) The Verification Report
- 3) The Validation Report

e. Data Collection

Main sources of case study evidence are documentation, structured interviews and participant observation.

The effort spent in developing, verifying and validating the conceptual model is collected using the effort management tool. The conceptual model data is stored in the KAMA tool. The formal verification results are documented as technical reports.

f. Modeling Tool

The modeling is performed using the KAMA tool. Transfers among the computers is accomplished by XML files which are produced by the export utility of the KAMA tool. KAMA is used to store all of the developed model elements and diagrams.

g. Version Management

The model versions are managed by the KAMA tool. The modelers may create internal versions when they feel necessary. These versions are tracked by the file name. The file is named as <Model_Name>.XX where XX is a two digit integer, which is incremented one by one for each version.

h. Training/ Orientation

Training presentations are provided to the domain experts.

i. Case Study Questions

The modelers are interviewed using the following questions which are classified in two phases of the conceptual model; development and use. Development questions are expected to be answered by the modelers and the use questions by the domain experts; however both parties may provide answers to both question sets. The questions are not prepared for directly being asked to the modelers but posed to the investigator as suggested by Yin in [74]. These questions are used as reminders for the investigator regarding the information that needs to be collected and also used to keep the investigator in track as data collection proceeds.

Model Development:

- 1) Effectiveness: Is KAMA notation effective for developing conceptual models?
 - a. Is KAMA notation sufficient for representing conceptual models of the mission space?
 - b. Does KAMA notation capture essential features of operation and include all important aspects?
 - c. Does KAMA notation provide different viewpoints?
 - d. Do models developed using KAMA include enough detail?
 - e. Are KAMA models simple?
 - f. Did you make fewer errors during modeling?
- 2) Usability: Are KAMA models easy to develop?
 - a. Does diagrammatic notation make it easier to develop models?
 - b. Do KAMA models provide help for the subsequent simulation development phases?
 - c. Are KAMA models straightforward?
 - d. Are KAMA diagrams traceable to each other?
 - e. Is KAMA notation applicable to different types of simulation systems?
 - f. Are parts of the model re-usable?
 - g. Are you able to make efficient and creative use of existing data with KAMA?
 - h. Does KAMA notation help organize complexity?
- 3) Feasibility: Is it worth spending the effort for developing KAMA models?

Model Use:

- 1) Validity: Are KAMA models easier to validate?
 - a. How does KAMA help build internal and external validity?
 - b. Is it easier to detect defects?
 - c. Is it easier to understand what is modeled?
- 2) Usability: Are KAMA models easy to understand and interpret?
 - a. Do KAMA models provide help for the subsequent simulation development phases?

- b. Are KAMA models straightforward?
- c. Is KAMA notation simple to use?
- d. Are KAMA diagrams intuitive?

5.3.3. Conduct of the Case Study 1

The case study was implemented by 2 modelers, one of whom was experienced in object oriented software analysis and design, process modeling and the other in simulation systems development and modeling. Since the case project was confidential, the details are not provided in this thesis but can be found in [61]. The conceptual model is developed in three iterations in parallel to the project's development phases. In the first two iterations the diagrams are developed manually and the KAMA tool is used in the last iteration. The conduct of the case study is explained through the following steps.

a. Acquire training on the conceptual modeling notation and process

We performed short training sessions for the reviewer and the domain experts. Since the domain experts were also part of the project team, they had the opportunity to follow the development of the notation, process and the tool during the periodic progress meetings, technical meetings and presentations made by the research team. The reviewer was also a researcher in the project, studying on the verification and validation of the conceptual models, therefore we provided only a 3-hour mentoring session on how to use the KAMA tool.

b. Acquire knowledge about the conceptual model

As the first activity of this step, we identified the objectives of the system by looking at the high level requirements specified by the sponsor. The objectives of the system are defined as;

- Observing the movements within a specified boundary systematically or on requested time phases for security, control, attack and defense purposes,
- Gathering, evaluating, analyzing and interpreting data about enemy and potential enemy elements, about the geographical, meteorological structure of a defined boundary,
- Transfer the acquired information to national and international authorities and lead the military forces accordingly.

High level assumptions and constraints are defined as;

- The mission space focuses more on the surveillance and information gathering activities rather than intervention related activities.
- It is assumed that the surveillance and reconnaissance activities can be performed by both stable and mobilized units.

- The system guidelines of the existing information system are used as reference for determining the entities, actors and other mission space elements.
- Some of the vendors could not provide detailed information; therefore these information is extracted from international standard documents.

Fidelity requirements are specified as;

- Simulation models must execute in real-time and on the highest possible speed.
- The radar models must execute in real-time speeds.

We then identified the authoritative information resources. The customer has provided us the required documents for developing the conceptual model, which were the System Requirements and Conceptual Model Document and the Software Specifications Document. As we made progress we needed additional information, however due to security reasons we were not able to access some of these resources. Especially authoritative human resources were located on secure zones. We had to make assumptions on some decision points and recorded these as part of the conceptual model element's assumptions and constraints information. These assumptions can be found in the conceptual model documentation [27].

Since conceptual modeling is not a onetime activity but rather an iterative one, knowledge acquisition is also repeated throughout the modeling process and any changes in the information resources are recorded.

c. Analyze the information sources and search for similar conceptual models

After analyzing the information sources we identified the entities, missions and tasks as draft conceptual model elements. These were recorded in the first versions of the conceptual model. We figured out that activities and tasks that were specified in the source documents had the same attributes. The only difference was that activities were defined to be the leaf level tasks which are atomic and could not be decomposed into lower level tasks. According to KAMA notation, tasks are detailed using task flow diagrams and any task in a task flow diagram can be detailed using a task flow diagram recursively until the modeler reaches an appropriate level of detail. Therefore, we decided to handle tasks and activities both together as tasks.

d. Define conceptual model elements and develop conceptual model diagrams

We identified five missions namely; Surveillance, Reconnaissance, Intelligence, Intervention & Support and Frontier Infringement. The objectives of these missions and measures related with these objectives are depicted in Figure 29. The surveillance mission has a single objective, surveillance performance, which can be quantified by 4 measures. The measure model element includes the definition of the measurement unit and the performance criteria are defined in the objective model element. These measures are the length of the covered border, time elapsed to

cover, the rate of detect/identify/recognize, the density of coverage and cost. These measure have respectively; “kilometers”, “minutes”, “number of recognized threats / number of detected threats X 100”, “area of covered border / area of focused land X 100” and “money spent in YTL” as the measurement units. The performance of the mission is identified by checking the related measures. It should be noted that a mission will be successful when all of the success criteria related with all of its objectives comply with the specified success values. The surveillance performance objective will be counted as successful when the “length of the covered border is more than or equal to 12 km” AND “elapsed time value is smaller than 20 minutes” AND “the rate of detect/identify/recognize is more than 60” AND “the density of coverage is more than 80” AND “money spent in YTL is less than 10.000 YTL”.

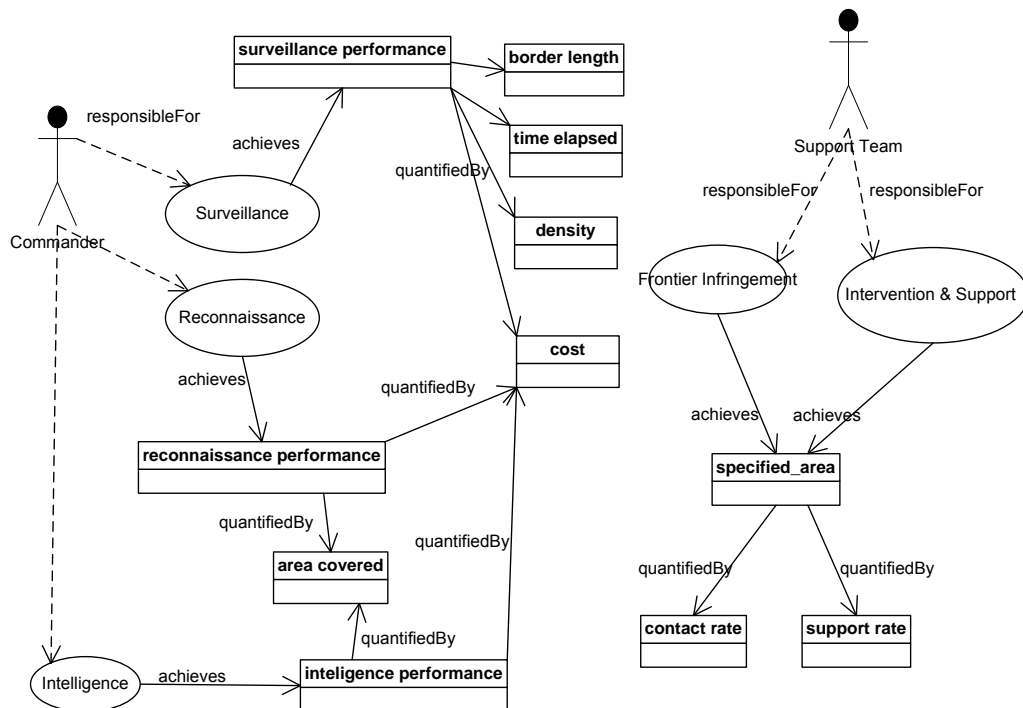


Figure 29: Mission Space Diagram for Case Study 1

Then we used task flow diagrams to present details of these missions and produced 50 task flow diagrams. We followed an iterative breadth-first approach in developing the task flow diagrams. We began with a mission and depicted the highest level task flow diagram, including the tasks and related work products. The objectives, measures and roles were missing in this first iteration. Then we detailed the tasks in this task flow diagram one by one. As the model tree expanded we began reusing some of the tasks and work products. This continued until we came to a point that we did no longer need a task flow diagram for any task. We repeated this procedure for each mission and produced 159 tasks, 50 start states, 50 end states, 32 synchronization bars, 94 work products and 13 decision points.

The sample task flow diagram presented in Figure 30 depicts the “Analyze Reconnaissance Information” task details. The synchronization bar right after the start state means that this task begins with parallel execution of three tasks. KAMA notation does not specify a specific start time for the execution of parallel tasks, therefore these tasks may begin at the same time or there may be some delay between their start times. The decision point after the task “Integrate Visual Intelligence Information” shows a branch which is dependent on the conditional query at the decision point. The guard condition for each branch must be labeled and these conditions must not be contradictory. Task flow diagrams are semi-formal descriptions and therefore KAMA notation does not handle conflicts that may arise at this point, neither the KAMA tool. The shaded tasks imply that the task is detailed with a task flow diagram.

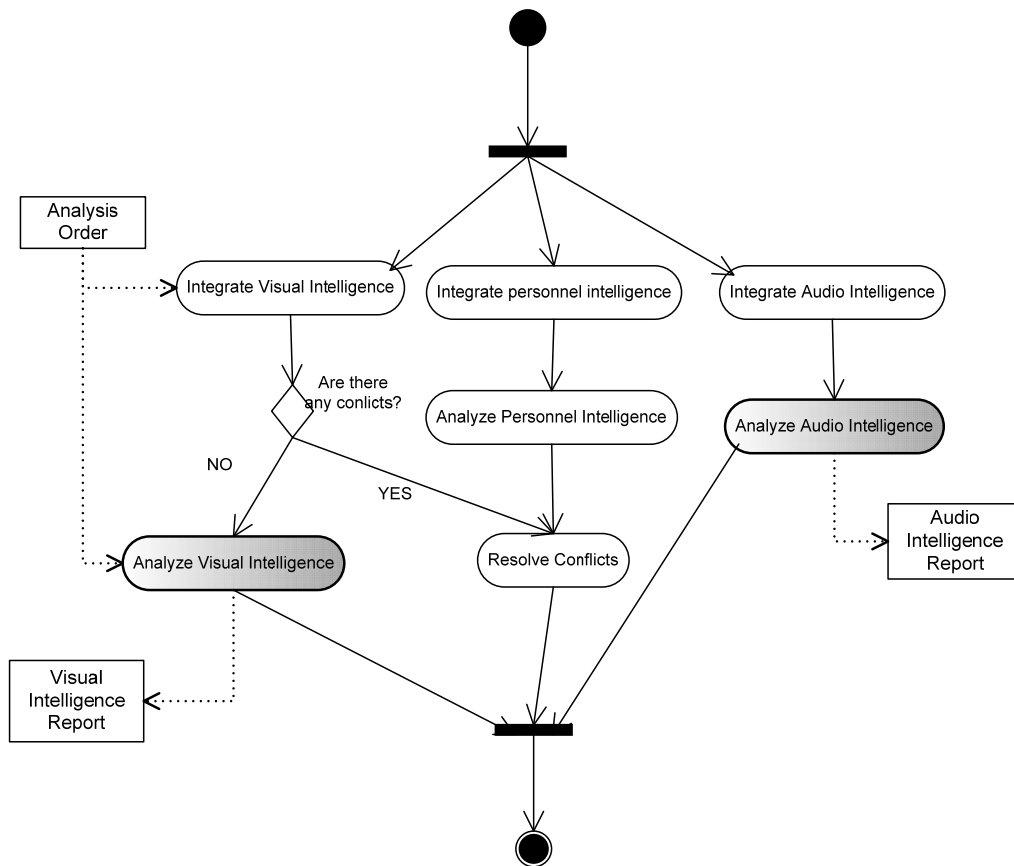


Figure 30: Sample Task Flow Diagram for Case Study 1

In the first iteration the modelers defined the tasks, the control flow among these tasks, the synchronization and decision points. The second iteration focuses on the roles and work products related with the tasks. There are two predefined relation types between task and work product; “produces” and “inputTo”. The task entitled “Analyze and Evaluate Visual Intelligence Information” produces the work product “Visual Intelligence”. The produce relation does not

necessarily have to produce the work product literally; it may also update an existing work product. In this diagram the task uses and updates the “Visual Intelligence” work product.

As we defined the task flow diagrams we introduced some work products and naturally there were some relationships among these work products. We used the entity relationship diagrams to depict these relations. This diagram is generally used to represent the aggregation relationships among the work products. The entity relationships diagram in Figure 31 shows the aggregation relationships between work products. The Intelligence Information is composed of three pieces; Visual Intelligence, Personnel Based Intelligence and Sign Intelligence which is also composed of Electronic Intelligence Data and Communication Intelligence Data. Many entity relationships diagrams can be developed for the work products in the conceptual model.

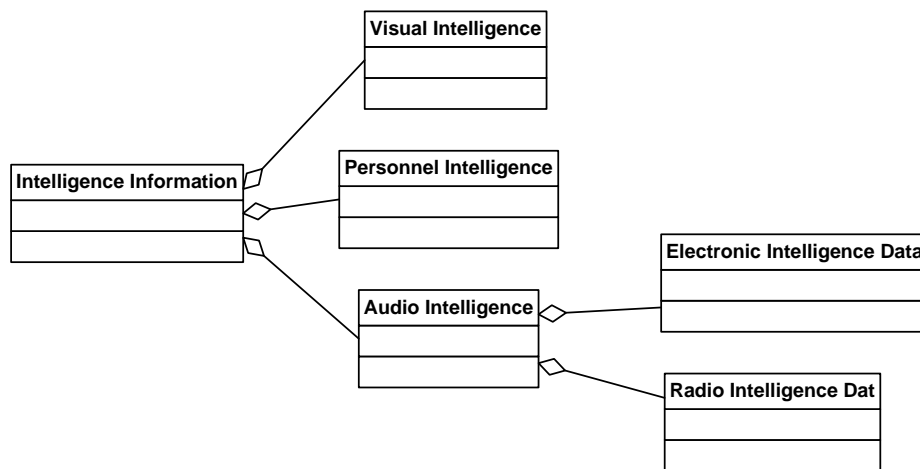


Figure 31: Sample Entity Relationship Diagram for Case Study 1

After defining the tasks we identified the entities in the mission space and depicted the relationship among these 82 entities using 8 different entity ontology diagrams. Representing the whole entity ontology within a single diagram would increase the number of possible defects and decrease the intelligibility of the model; therefore we used different diagrams to represent different logical groupings among the entities. In the first iteration we defined the entities briefly and provided the inheritance and aggregation relationships among them. Entity ontology diagram is based on abstraction hierarchy. The entity ontology diagram in Figure 32 displays this hierarchy with the details hidden. The “tank” entity inherits both attributes and capabilities from the “land vehicle” which also inherits from the “vehicle”. Although not shown in this diagram, when the model is checked, it could be seen that “vehicle” entity is located in the middle of another abstraction hierarchy.

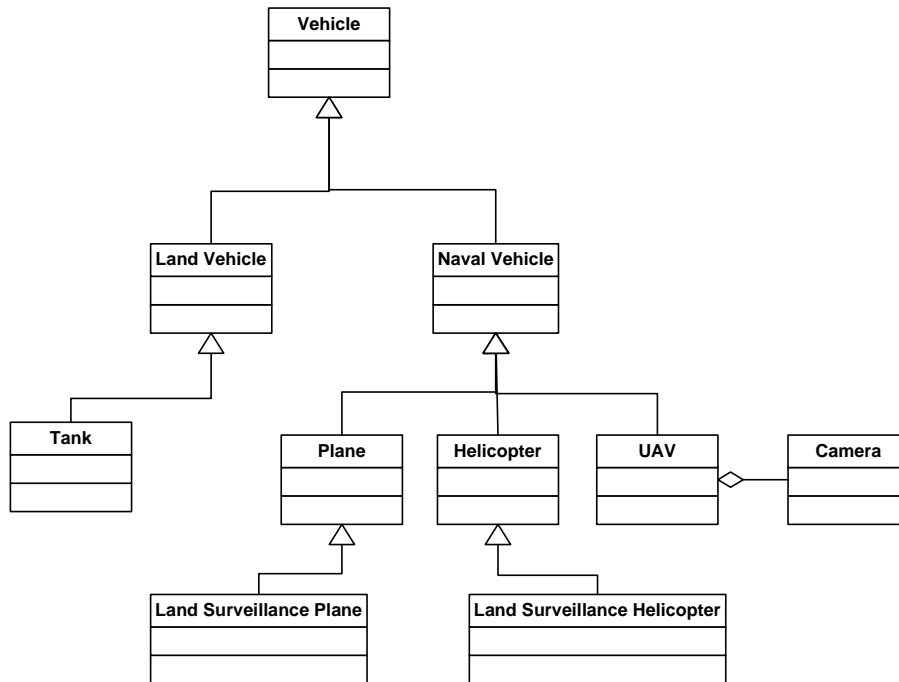


Figure 32: Sample Entity Ontology Diagram for Case Study 1

Each entity is characterized with its attributes and capabilities. The second iteration focused on defining these characteristics.

In order to define the roles that are responsible for tasks we needed to define the actors first. KAMA notation provides command hierarchy diagrams and organization structure diagrams for representing actors and roles respectively. We noticed that among the 17 actors identified in the mission space, only 8 of them took part in the task flow diagrams as roles. This might be an indicator that the conceptual model was incomplete. The command hierarchy diagram shown in Figure 33 includes some actors standing alone and not related with any other actor. This was one of the reasons that the command hierarchy should be reused from the common repository. The actors defined for this mission space might not include the whole set of actors and therefore there might be unrelated actors. In order to resolve this issue we assume that the common repository will include a complete command hierarchy diagram which would be used by the modelers.

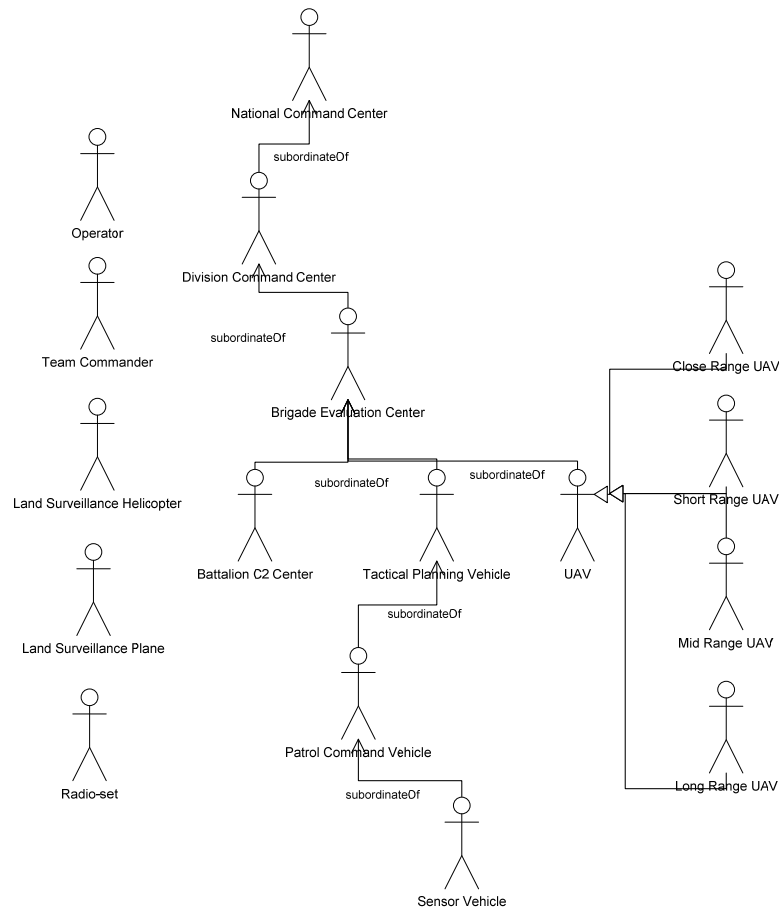


Figure 33: Sample Command Hierarchy Diagram for Case Study 1

There were 7 roles that took part in the task flow diagrams. These roles were owned by 8 actors as shown in the organization structure diagram in Figure 34. Each role should be in relation with a mission or task either by a “responsible for” or “realizes” relation type. Every role in the organization structure diagram should be related with an actor in the command hierarchy diagram. An actor may own many roles, or a role may be owned by many actors. While the former situation implies a real life actor wearing many hats, the latter shows an abstraction in the conceptual model by representing many real life actors by a single role. In the below diagram, the Operator actor owns 5 different roles and the Surveillance Mission Planning Vehicle role is owned by 6 different real life actors.

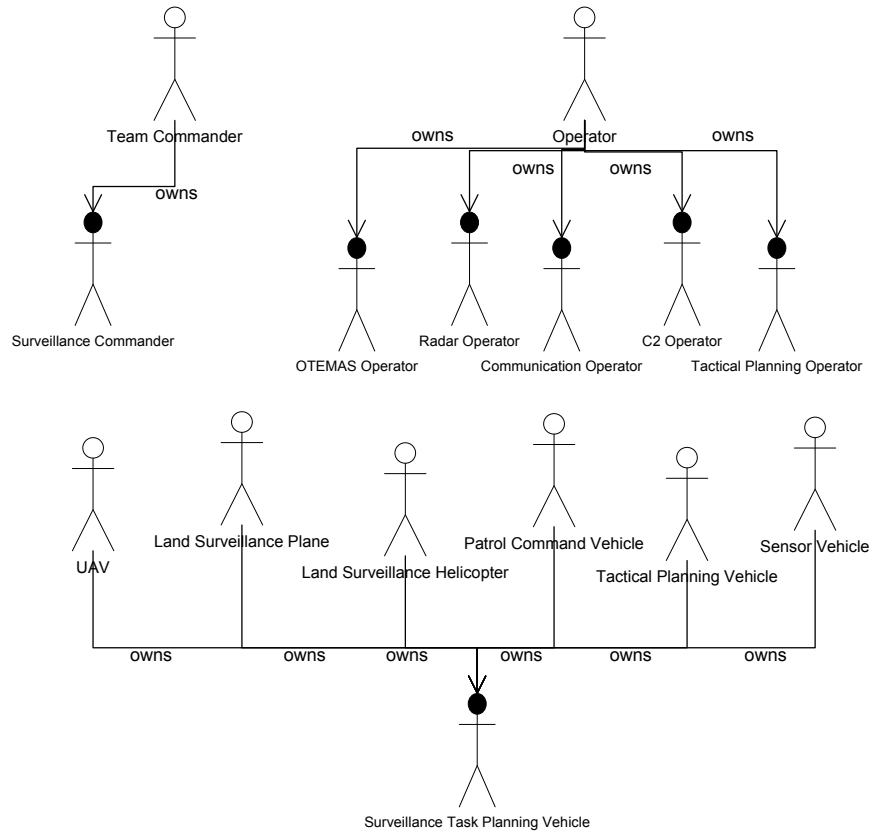


Figure 34: Sample Organization Structure Diagram for Case Study 1

As the next step; the access rights for the conceptual model and conceptual model elements are defined using the KAMA tool. Conceptual model as a whole is assigned a security level of “Restricted”.

e. Refine the conceptual model

The original conceptual model document included a different view than the behavioral view proposed by KAMA notation. The states were treated as modes of the mission space and the behavior of entities are described with respect to these modes. Therefore, the focus was on the modes of the whole system rather than distinct states of each entity. It was possible for us to identify behavior changes of 20 entities by looking at the system modes. However, as a result of this approach most of the entities had the same entity state diagram with the sample shown in Figure 35.

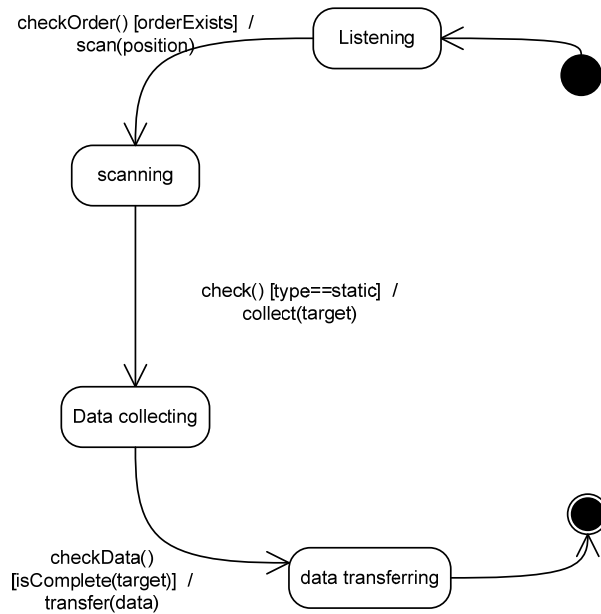


Figure 35: Sample Entity State Diagram for Case Study 1

f. Verify and validate the conceptual model

The verification of the model was accomplished in two steps. The first step was the execution of the rules, which were defined at the metamodel level. These rules consist of constraints on the relationships among conceptual model elements, attributes of the conceptual model elements, and diagrams that are defined using OCL in Section 4. These rules are then included in the requirements specification of the tool and implemented. The modeler can perform a verification of the conceptual model based on the defined rules using the KAMA tool.

The second step of the verification activity was carried out following the method that was defined in [68]. Tanriover proposed a 3-step inspection approach for verifying the conceptual models. The approach begins with a pre-inspection phase where the inspector identifies the diagram types used in the conceptual model, identifies unfamiliar types of notation elements and relations used in different types of diagrams, and determines the inspection strategy. The inspector then performs intra-diagram and inter-diagram inspections based on activities defined in [68].

The issues found during verification are documented and published as a technical report [67]. There were 85 issues defined as a result of this review. 10 of them were identified as major, 7 of them as moderate and 68 of them as minor. Major issues consisted of the following problems; 6 semantic deadlocks in the task flow diagrams, 1 missing model element in the diagram, and 3 improper usage of fork node which may result in dangling reference. Moderate issues included improvement suggestions both for the notation and the tool such as the need for a more intuitive synchronization point symbol and a different symbol for tasks that have sub-tasks. The review revealed that; semantic errors cannot be detected by the constraints defined in Section 4,

behavioral diagrams are more prone to errors than the structural diagrams, and graphical notations are important in understanding and examining the conceptual model. The improvement suggestions were included in the requirement specification document and implemented as permitted by the project resources and the schedule.

The validation of the conceptual model was performed by a researcher by checking the model with respect to the source documents. The researcher ensured that any information existing in the source documents is represented in the conceptual model. The parts of the source document that maps to a conceptual model element were marked by different colors and the validation process continued until there were no lines left unpainted. There was not any other documentation produced including the validation results.

g. Update the conceptual model with respect to the V&V findings

The conceptual model had 3 major and 36 minor revisions. The minor revisions were mostly caused by concurrent development and the integration of the model parts as well as informal internal reviews performed by the modelers. The reviews ended up in major revisions of the conceptual model. These revisions were kept under control using the KAMA tool.

5.3.4. Discussion and Findings for the Case Study 1

In the first case study we had the opportunity to explore the application of the notation, examine its drawbacks and enhance its structure. The initial phases of the case study helped us establish the requirements of the KAMA tool. We examined three conceptual models that were developed as free text documents and tried to develop the most comprehensive model using KAMA notation. The case study also assisted us in refining the conceptual model development process that could be used for subsequent case studies.

We observed that KAMA notation was adequate for representing all of the conceptual model elements as defined in the source conceptual model documents. The mapping between the conceptual model elements is shown in Table 5.

Table 5: Mapping

Conceptual Model Element (in the source documents)	KAMA Conceptual Model Element
Mission	Mission
Task	Task
Activity	Task
Event	Task

Conceptual Model Element (in the source documents)	KAMA Conceptual Model Element
State	State
Entity	Entity
Data Item	Entity
Command and Control Unit	Actor

The researchers in the project team performed 3 experimental studies [13] [20] [3], which aimed to apply the KAMA notation in different settings. The subjects of the experimental studies were “Squad Fire and Battle Marksmanship Training Simulator”, “Small-scaled Military Unit Movement – Infiltration Task”, and “Synthetic Environment Simulation System” respectively. We performed half-day orientation sessions with those researchers; they developed the conceptual models themselves, shared progress periodically with us and finally we examined the results together. We used these studies to check the validity of the KAMA notation by applying in various real life scenarios. The following list summarizes the issues extracted from the technical reports.

- All of the researchers found the KAMA notation easy to learn and apply. However, the researchers were all knowledgeable on the Unified Modeling Language, which might have formed a bias.
- The first study affirmed that although not crucial, UML knowledge was an advantage for learning and using the KAMA notation, pointed out that the difference in the level of detail of the conceptual models would affect the reusability of the models among the simulation projects, stressed the need for representing relations among the workproducts, additional relationships between a role and a task such as “assists”, distinctly represent the “equipment” required for a task, additional mechanism for representing the geographical constraints, establishing a bridge between the states of an entity and the task flows.
- The second study found KAMA diagrams easy to understand and helpful to detect mistakes, the different diagram types were used to effectively represent the different points of views, and pointed out the need for representing random/non-deterministic decisions in the task flow diagrams. These kind of probabilistic situations are represented by using decision points in the KAMA task flow diagrams, however, there is not a formal notation used for this purpose. In order to utilize a formal notation for representing the random events, there is a need to access the run-time values of the attributes of entities.

Since the KAMA framework does not aim to provide an executable model, this suggestion was rejected and proposed as a future work.

- The third study used the KAMA notation as is for developing the conceptual model of the mission space and extended the KAMA notation and process definition to represent the simulation space conceptual model.

Another experimental study was conducted to examine the cognitive aspects of the graphical notation used to represent the KAMA conceptual models [29]. The study used two groups of subjects for the experiment, one group of engineers and another group of domain experts. The problem resolution processes of these groups were examined by using the diagrams developed in KAMA notation. The groups were asked to find the errors that have been seeded in the diagrams. Both groups had difficulty in understanding computationally complex diagrams that included nested decision flows. Domain engineers found crowded diagrams difficult to understand. Both groups followed a linear flow and did not perform parallel controls, inter-diagram checks and return flows. None of the subjects used concurrent search techniques. The subjects suggested using different symbols or colors for representing the relations and the model elements.

The following changes to the notation were proposed as a result of the first case study. These issues were based on our participation in the modeling process. Command hierarchy diagrams should include additional relation types in order to effectively represent the real life situations. These relation types, which already existed in the entity relationships diagrams, are generalization, association and aggregation. The organization structure diagrams included the entity model elements, which were reported to increase the complexity of the model during the modeling process. The entity model element was removed from this diagram type.

After the first iteration of the conceptual model development process, the modelers felt necessary to represent the relationships among the workproduct model elements, which had a very similar structure with the entity model elements. Therefore, we decided to include the workproduct model elements in the entity relationship diagrams. The modelers also needed to represent the context information in the entity state diagrams. Entity state diagrams were used to show the states and transitions within an entity throughout its lifecycle; however an entity might have different behaviors within different missions. In order to represent this information, we decided to add the context data to the transitions among the states.

The modelers needed to attach comments to the model elements to provide free text information about the model elements. KAMA did not have such a utility; therefore we decided to add a “note” element to all of the diagrams, which would not have any semantics but only a graphical node with information on it. Also we had to provide a “note relation” that would be used to link notes to the other model elements.

As we completed the first case study we realized that performing verification and validation activities after the first release of the conceptual model would be an effective way to determine the defects earlier. Therefore, we decided to add a review step before the “refine the conceptual model” step. This might be a joint review with the domain experts, so that both modelers and domain experts would ensure that they speak in the same language. Formal review techniques employing checklists might be used for verification and structured walkthroughs might be used for validation purposes.

Verification results revealed that developing the conceptual models in a diagrammatic modeling language and using a tool that supports this language helped the modelers in describing relationships, navigating among the model elements, communicating the model and finding mistakes. We also observed that using intuitive graphical notations would increase the effectiveness of the verification process.

5.4. Case Study 2

5.4.1. Background

One of the objectives of this thesis was providing a suitable notation both for development and usage of the conceptual models. Therefore, we decided to conduct the second case study in an organization where we could have the modelers develop the model and domain experts use the model. The purpose of the case study was to explore and evaluate the sufficiency of the KAMA notation for developing and validating conceptual models of the mission spaces.

We conducted the case study in a software development company with 50+ software and systems developers. The company focuses on defense projects with major interest in simulation and information systems. Most of the software engineers have graduate degrees and are experienced in simulation systems development. Although they had experience in simulation development, they did not follow a defined conceptual modeling process in their previous modeling experience. The project has been estimated to last for 3 years, and an iteration will be released at the end of each year. A conceptual modeling phase has been planned for each iteration which is followed by a system requirements analysis phase. The conceptual modeling team consisted of 10 software engineers and the acquirer organization provided support to the team when required.

The conceptual modeling team consisted of 6 members from the team of a simulation development project, targeted for developing/enhancing a system for Turkish Military Forces. Two of them were senior team leaders with more than 12 years of experience and the rest of the team were senior software engineers who were specialized in modeling and simulation.

5.4.2. Case Study Plan

a. Purpose and Scope

This section presents the plan for the execution of the KAMA notation in a simulation system development project. The objective is to develop the conceptual model of the mission space for a simulation system using the KAMA notation. A study group will perform the conceptual modeling activities as defined in the KAMA notation under our supervision.

b. Case Study Organization

The case study roles are defined as follows:

Modeling Group: Responsible for developing the conceptual model using the KAMA notation and in accordance to the KAMA conceptual modeling process.

Moderator: Responsible for moderating the joint reviews.

Reviewer: Responsible for performing the verification activities.

Domain Expert: Responsible for providing information when required and validating the conceptual model as planned.

c. Work Plan

Table 6. Schedule Summary (Case Study 2)

ID	Phase	Start	Finish	Resource	Effort (person-hours)
1	Training Session on Conceptual Modeling	20.08.2007	24.08.2007	Modeler	29
2	Conceptual Model (CM) V1.0 Development	25.08.2007	04.09.2007	Modeler	320
3	Verification of CM V1.0	05.09.2007	10.09.2007	Reviewer	20
4	Validation of CM V1.0	12.09.2007	17.09.2007	Domain Expert	72
5	Conceptual Model (CM) V2.0 Development	19.09.2007	19.11.2007	Modeler	1000
6	Verification of CM V2.0	21.11.2007	23.11.2007	Reviewer	12
7	Validation of CM V2.0	25.11.2007	30.11.2007	Domain Expert	48
8	Conceptual Model (CM) V3.0 Development	02.12.2007	10.03.2008	Modeler	600
9	Verification of CM V3.0	10.03.2008	15.03.2008	Reviewer	16
10	Validation of CM V3.0	17.03.2008	22.03.2008	Domain Expert	68

The conceptual model will be developed in 3 major iterations. The schedule for the project tasks and resource assignments is given in Table 6.

d. Process and Outputs

The process defined in Section 3.3 of this thesis will be used as a basis for the case study. We will perform a structured interview with the modelers in order to obtain feedback about the notation. Conceptual model will be documented in the project's System Requirements and Conceptual Model document.

A training session will be provided to the modeling group and the domain experts to introduce the basic concepts of conceptual modeling notation, process and usage of the KAMA tool

Verification of the conceptual model will be performed informally at the end of each iteration by the reviewer. Validation will also be performed at the end of each iteration by a joint review with the domain experts.

Due to the iterative nature of the project the review and refinement of the conceptual model will also be performed iteratively.

The expected outputs for the case study are as follows:

- 1) The Conceptual Model
- 2) The Verification Meeting Minutes
- 3) The Validation Meeting Minutes

e. Data Collection

Main sources of case study evidence are documentation, structured interviews and participant observation.

The effort spent in developing, verifying and validating the conceptual model will be collected using the effort management tool. The conceptual model data will be documented in the System Requirements and Conceptual Model document. The verification and validation results will be documented as minutes of the meetings.

f. Modeling Tool

The modeling will be performed using a commercial UML modeling tool.

g. Version Management

The model versions will be managed by the configuration management infrastructure of the company. The modelers may create internal versions when they feel necessary. These versions will

be tracked by the file name. The file will be named as <Model_Name>.XX where XX is a two digit integer, which is incremented one by one for each version.

h. Training/ Orientation

Training presentations will be provided both to the modeling group and the domain experts. The training sessions will last for 2 days and include in-class exercises.

i. Case Study Questions

The modelers will be interviewed using the following questions which are classified in two phases of the conceptual model; development and use. Development questions are expected to be answered by the modelers and the use questions by the domain experts, however both parties may provide answers to both question sets. The questions are not prepared for directly being asked to the modelers but posed to the investigator as suggested by Yin in [74]. These questions are used as reminders for the investigator regarding the information that needs to be collected and also used to keep the investigator in track as data collection proceeds.

Model Development:

- 1) Effectiveness: Is KAMA notation effective for developing conceptual models?
 - a. Is KAMA notation sufficient for representing conceptual models of the mission space?
 - b. Does KAMA notation capture essential features of operation and include all important aspects?
 - c. Does KAMA notation provide different viewpoints?
 - d. Do KAMA models include enough detail?
 - e. Are KAMA models simple?
 - f. Did you make fewer errors during modeling?
 - g. Do KAMA models effectively reflect reality?
- 2) Usability: Are KAMA models easy to develop?
 - a. Does diagrammatic notation make it easier to develop models?
 - b. Do KAMA models provide help for the subsequent simulation development phases?
 - c. Are KAMA models straightforward?
 - d. Are KAMA diagrams traceable to each other?
 - e. Is KAMA notation applicable to different types of simulation systems?
 - f. Are parts of the model re-usable?
 - g. Are you able to make efficient and creative use of existing data with KAMA?
 - h. Does KAMA notation help organize complexity?
- 3) Feasibility: Is it worth spending the effort for developing KAMA models?

Model Use:

- 3) Validity: Are KAMA models easier to validate?
 - a. How does KAMA help build internal and external validity?
 - b. Is it easier to detect defects?
 - c. Is it easier to understand what is modeled?
- 4) Usability: Are KAMA models easy to understand and interpret?
 - a. Do KAMA models provide help for the subsequent simulation development phases?
 - b. Are KAMA models straightforward?
 - c. Is KAMA notation simple to use?
 - d. Are KAMA diagrams intuitive?

5.4.3. Conduct of the Case Study 2

The case study was conducted in accordance to the case study execution plan given in Section 5.4.2.

a. Acquire training on the conceptual modeling notation and process

The first step in the case study was establishing a 2-days training session for the conceptual modeling team. We introduced the objectives of conceptual modeling, its position in the simulation development lifecycle, the conceptual modeling process, conceptual modeling notation and also performed in-class exercises. Following this course, we organized a 1 day course for the project personnel in the acquirer organization including the same topics except for the in-class exercises. These training sessions were necessary in order to have both the supplier and the acquirer organizations talk in the same language.

b. Acquire knowledge about the conceptual model

The high level objectives of the simulation were;

- 1) to enhance the capabilities of the existing system
- 2) to provide interoperability with other systems
- 3) to support the use of hypothetical entities as well as real ones
- 4) to include water surface, underwater, air and land elements
- 5) to include any environmental conditions that can affect the simulation; such as atmospheric, meteorological, hydrographic, oceanographic and acoustic conditions.

The team also extracted the following constraints from the statement of work provided by the client;

- the system should be extensible to include additional entities and capabilities

- the system should provide integration capabilities with a selected set of available systems
- the system should provide three dimensional representation and analysis support.

As the next step, the conceptual modeling team identified the authoritative information sources related with the project. The executable copy of the existing system was one of the most important sources of information. The statement of work, which included the high level requirements of both the system and the conceptual model, was also an essential source. The team collected the documented sources, uniquely labeled and stored them into a local repository. Since there were security considerations specified in the contract, they established a limited access to the repository. These information resources are then referred from the conceptual model elements. Another major source of information was the domain experts and the conceptual modeling team planned training and observation sessions with those domain experts.

c. Analyze the information sources and search for similar conceptual models

The team analyzed the documented information sources and developed a draft document which included the high level conceptual model elements such as missions, entities, actors, roles, assumptions and constraints. Following the plan developed in the previous step, the team took a 3-week training from the domain experts. These training sessions aimed to provide basic knowledge about the domain and played a very important role in the conceptual model development activities. During these training sessions the conceptual modeling team had the chance to examine the existing system.

The team could not find any similar conceptual models, however the system requirements document of the existing system was used as an information source.

The team identified some of the simulation models that could be used in the system. These simulation models were not directly related with the conceptual model, however considering the potential effects on the system, the team decided to demonstrate these models to the client and obtained feedback from them in this step.

d. Define conceptual model elements and develop conceptual model diagrams

The team followed a diagram-first modeling process. All of the model elements were first introduced in diagrams and then recorded in the model element repository. As we suggested in our conceptual model development process in Section 3.3 they followed a concurrent process; they depicted the mission space, task flow, organizational structure and command hierarchy diagrams in one branch, and in another branch they built the entity ontology, entity state and entity relationships diagrams.

For the mission space model, the team followed a breadth-first approach. The missions that will make up the mission space were identified first, and then the relations among these missions were

defined. The team came up with 157 missions represented in 30 mission space diagrams. The topmost mission space diagram included 7 missions and in order to decompose the model, the team decided to develop mission space diagrams for these highest level missions.

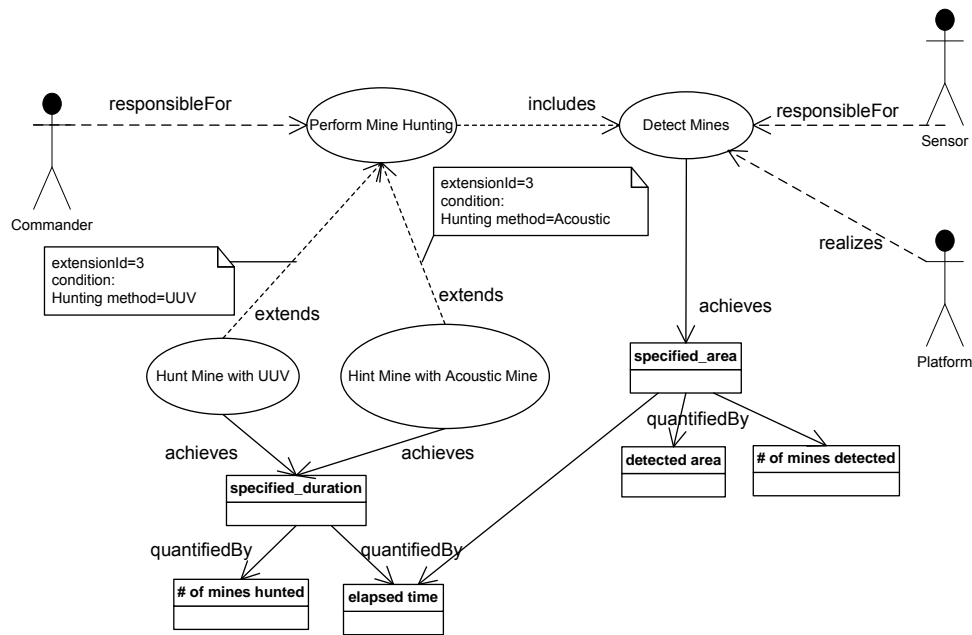


Figure 36: Sample Mission Space Diagram for Case 2

Then, each of these missions was detailed using task flow diagrams, and one level of detail had been established. After that, the team detailed each task in every task flow diagram until they reached to a sufficient level of detail. At the end of this phase every task has been detailed using task flow diagrams except for the leaf level tasks. The team represented the mission space with 425 tasks in 74 task flow diagrams. In the first iteration, the task flow diagrams did not include objectives, measures and the work products. These conceptual model elements were planned to be developed in the latter iterations, because the team decided to take commitment of the client first on the high level missions and tasks and then detail these diagrams. Sample task flow diagram is presented in Figure 37.

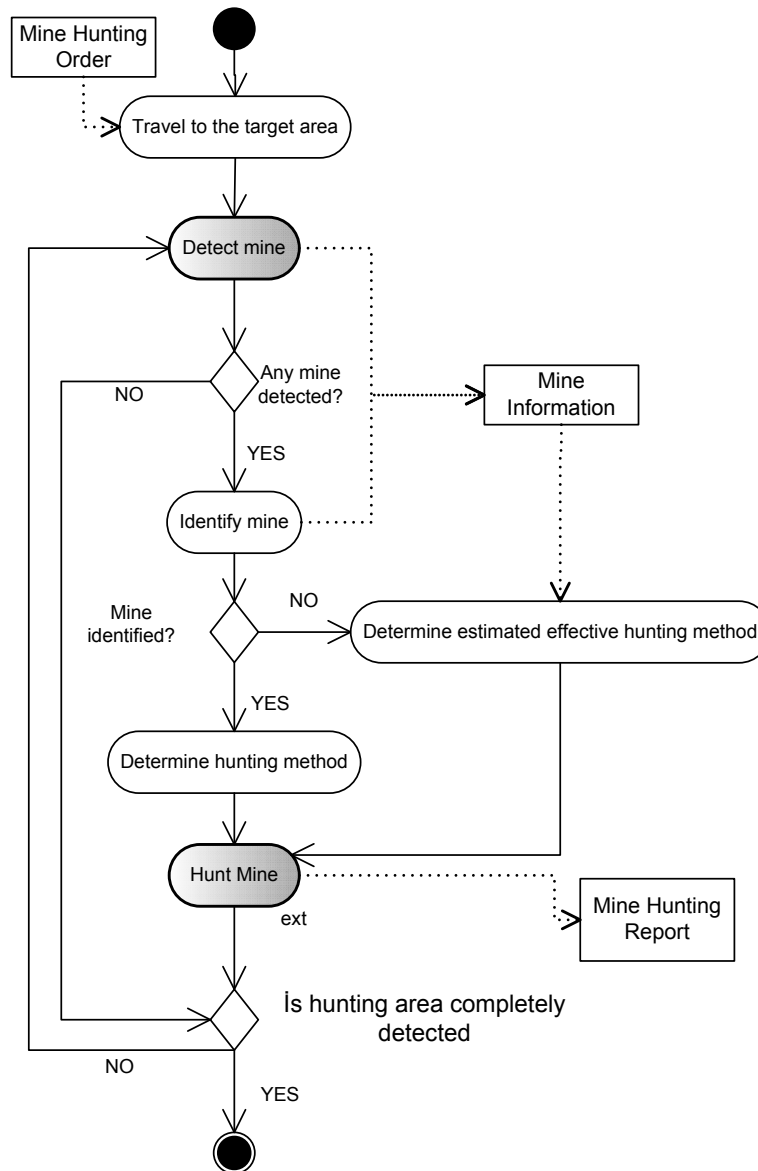


Figure 37: Sample Task Flow Diagram for Case 2

The entity ontology diagram was mainly based on the statement of work document which was provided by the client. Major entities, the relationships among these entities, key attributes and capabilities are defined first. Since the client had provided a detailed definition of entities, the conceptual modeling team was able to develop the detailed entity ontology diagram in the first iteration. There were a total of 88 entities that were grouped in 15 logical packages and represented in 16 entity ontology diagrams, which were logically connected to each other. Sample entity ontology diagram is presented in Figure 38.

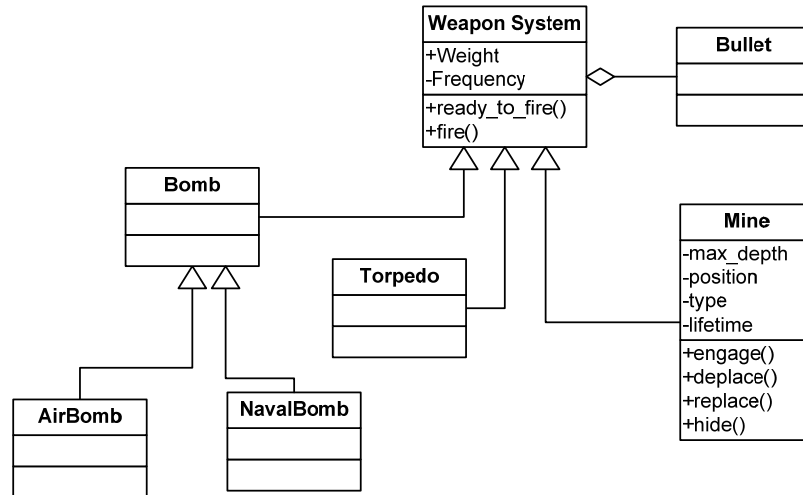


Figure 38: Sample Entity Ontology Diagram for Case 2

As the team could develop the entity ontology model in detail, they decided to develop the entity state diagrams that would represent the behavior of complex entities. The conceptual modeling team developed 26 entity state diagrams in the first iteration. These diagrams included additional model elements that are not specified in the KAMA notation. The results of this modification is discussed in the review of the conceptual model and documented in the following section. Figure 39 shows a sample entity state diagram for the lifecycle of a torpedo. All of the available states and transitions are depicted including also the guard conditions.

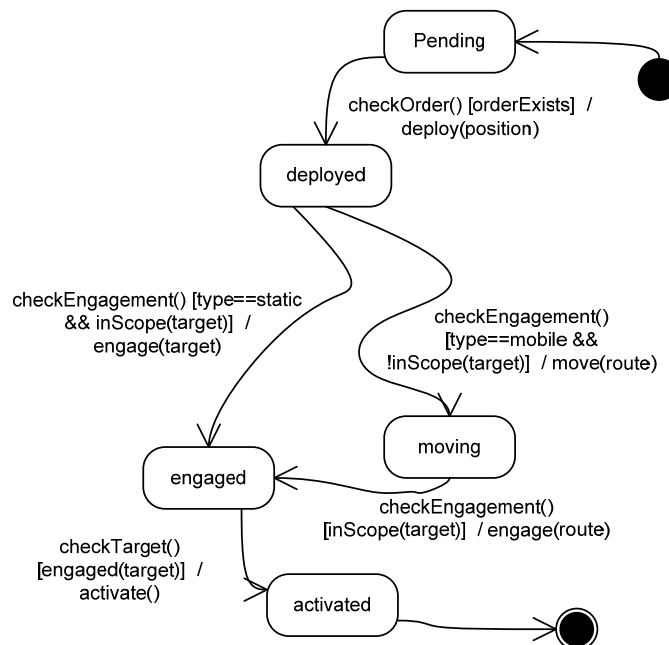


Figure 39: Sample Entity State Diagram for Case 2

e. Review the conceptual model

We reviewed the conceptual model and then shared our findings about the model with the conceptual modeling team through a review meeting. The preparations for the review took 2 days and the review meeting lasted for 4 hours. 6 members of the conceptual modeling team joined the meeting. The first review results indicated that the team did not use some of the modeling elements defined in KAMA notation. They did not need to use the entity ontology and entity relationship diagrams together because they did not make use of the central conceptual model repository. In that case they thought that these diagrams would have the same content, which was not true. The entity ontology diagram is basically used for representing the “has-a” and “is-a” relationships among the entities whereas the entity relationship diagram may include any kind of relationship among the entities. Entity relationship diagram is also used for depicting the relationships among the work products. Since the team had not yet utilized work products in the task flow diagrams, they did not need these diagrams.

f. Refine the conceptual model

The conceptual model is refined based on the review findings.

g. Verify and validate the conceptual model

Verification and validation of the conceptual model is performed iteratively. After the first baseline of the conceptual model, we performed a walkthrough together with the acquirer organization’s project personnel for validation purposes. It took five days to review the first iteration of the conceptual model, but both parties had great benefits. 7 members from the conceptual modeling team and 4 members from the acquirer organization joined the first day of the meeting and 4 members of the conceptual modeling team and 3 from the acquirer organization joined the rest of the meeting. There were 150 issues identified during this meeting. These issues can be classified into the following categories; detailed findings related with task flow diagrams, assumptions and constraints about the mission space, additional attributes and capabilities to the entities, assumptions about the random events, definition of roles and actors, system features and simulation models which were not directly related with the conceptual model.

12 (8%) of the issues were related with additional systems or software requirements and 13 (8.6%) of them were minor issues related with the conceptual model. 78 (52%) of the rest of the issues were moderate and 47 (31.4%) of them were major issues. Among the major issues were; scope changes, decisions on essential assumptions and constraints about the mission space, new missions identified, new task flow diagrams identified, need for additional information source, identification of autonomously executed missions. The moderate issues included changes in the task flows. Obviously these moderate and major issues would strongly affect the systems and software

requirements. Unless we had performed the joint review we would not have detected those mistakes and would have faced these problems later in the simulation development lifecycle.

h. Update the conceptual model with respect to the V&V findings

As the conceptual model had been verified and validated, the issues were reflected on the conceptual model document.

5.4.1. Discussion and Findings for the Case Study 2

Providing training sessions on the KAMA notation and the process helped the modelers and domain experts in developing and analyzing the conceptual model. The acquirer organization had a tendency not to be involved in the conceptual modeling process and had the supplier organization develop the model by its own resources. Therefore, they had not allocated resources for that phase. What we mentioned during the course was conflicting with this situation and after the training session the acquirer organization updated its plans and allocated about 3 person-months of effort for each iteration for supporting conceptual modeling activities. This support was especially in the form of providing domain knowledge and reviewing the conceptual model. The conceptual modeling effort has been planned in three iterations, each of which is followed by a one week lasting walkthrough with contribution of the acquirer organization. The project management planned for internal reviews in addition to these walkthroughs.

Developing the conceptual model in iterations and performing reviews after each iteration helped remove the defects earlier. Of the 150 issues identified during the joint reviews, 125 (83%) were moderate or major issues. Had the project group not detected those issues in the conceptual modeling phase, the cost of correction would have been higher. These joint reviews also helped establish a common language among the modelers and the domain experts. Both of the interviewees stated that the subject matter experts understood the mission space diagrams without further explanation, but needed oral explanations for understanding the other diagrams. They stated that if they had used a textual specification, “the subject matter experts would have been lost after the second paragraph”.

One of the participants stated that the use of KAMA framework affected the success of the project. In his previous simulation development project, the team had spent too much time for determining a common notation for representing the conceptual models. The KAMA notation provided not only a common language among the stakeholders of the project but also a “common way of thinking” for both the development team and the subject matter experts. Most of the developers were novice conceptual modelers, therefore following a well-defined conceptual modeling process enabled them focus more on the domain aspects rather than the modeling language and the modeling process. However, there appeared some improvement suggestions inevitably.

One major improvement to the KAMA notation appeared while developing the task flow diagrams. Some of the tasks would be realized automatically by the system or autonomously by the entities in the system, whereas others would be realized by the roles which represent real life entities. This difference had a major effect on the system, simply about the intelligence that would be included in the system. In order to automatically realize the tasks the system must know every required step in advance, however, otherwise the system had to provide only the atomic features to the user and the user would perform the main task herself. In the former situation, the conceptual model would have to include all the knowledge required to execute the task. For example, a task in the latter situation would have to be detailed using a task flow diagram for the former situation. The conceptual modeling team utilized swim lane representation in the task flow diagrams to represent this difference. If a task is marked to be performed by an entity then this task will be performed by the system. If a task is marked to be performed by a real life actor, then the system will provide limited support to the realization of this task.

The conceptual modeling team developed a specification document that included all of the diagrams in the conceptual model and pseudo-code like descriptions of the relationships among these diagrams. These specifications played the role of an amalgamation by describing the relations between the task flow diagrams and the other diagrams by a particular focus on the states of the system. The interviewees acknowledged that the subject matter experts found this specification very useful, however they also admitted that they could not have developed this specification if there had not been the KAMA diagrams. Therefore, this specification may be proposed as an improvement to the KAMA framework.

Another improvement opportunity that was proposed was related with the inheritance mechanism used in the KAMA notation. As the KAMA notation has its base in the UML specification, the inheritance mechanism is also derived from the UML inheritance concepts. The attributes and capabilities of a parent entity can be inherited by a child entity. The interviewees stated that the inheritance mechanism may be extended to include the state-machines. Since the state machines are used to depict the dynamic behavior of an entity, it is meaningful to inherit this aspect of an entity to the children entities. This suggestion is handled in the future work section of the thesis.

We observed a major issue about the usage of the entity state diagrams. Some members of the conceptual modeling team, although not specified in the KAMA notation, used the concurrent state machines in their models and created composite states. They have also used some of the state machines as if they were task flows, which resulted in wrong interpretations. The reasons for using simplified state machines in KAMA has been explained in Section 4.5, to put it briefly, state machines are generally considered to be artifacts related with the solution domain and the domain experts may find it difficult to understand. However, a conceptual model should be a part of the problem domain and should mainly serve for the domain experts. But besides this, state machines

are powerful tools to represent event based changes in a model; therefore we proposed the use of a simplified state machine in the conceptual modeling process.

Both of the participants stated that the subject matter experts had difficulty in understanding especially the entity ontology, entity relationship and entity state diagrams. The subject matter experts had a tendency to accept these diagrams as a product of the simulation design phase. The conceptual modeling team performed walkthroughs for clarifying the information contained in these diagrams and they also had to simplify the entity state diagrams. This issue can be explained by the eagerness of the conceptual modelers to jump into the solution domain, which is a natural result of lack of modeling experience.

The acquirer organization had the chance to perform an early validation of the conceptual model; they identified some major issues which would cause catastrophic defects had they been found in later phases. The conceptual modeling team was satisfied with the model they had developed, because they observed that it was understood by the acquirer organization, and they also remembered that they had to keep the model simple.

5.5. Lessons Learned and Discussions

In order to explore the applicability of the KAMA framework, three experimental studies and two case studies have been performed. The experimental studies were conducted to assist the design of the method and the notation. The first case study was conducted within the scope of a research project where we had the chance to explore the adequacy of the conceptual modeling notation, validate the applicability of the method and identify the requirements of the tool. Based on our findings from the first case study, we improved the framework. The second case study was conducted at a software/systems development company which was in charge of developing a complex simulation system. In this study we aimed to evaluate the effectiveness of the KAMA framework for developing conceptual models of the mission spaces.

The first study was performed at a research centre in the Middle East Technical University by two modelers and a reviewer. One of the modelers was experienced in object oriented software analysis and design, process modeling and the other in simulation systems development and modeling. The reviewer was in charge of verifying the developed conceptual model. Both of the modelers and the reviewer were part of the research team, therefore they knew the framework well. It was a retrospective study that aimed to re-develop a text-based conceptual model using the KAMA framework. It is also used to validate the KAMA tool that was developed to support the notation within the scope of a research project.

The second study was performed by a team of modelers, reviewers, and subject matter experts all of whom have engineering background. However, they did not have any experience on using structured methods for developing conceptual models. The number of the participants varied

during this study. There were 6-10 modelers, 1-3 reviewers and 3-6 subject matter experts taking part in the development and validation of the conceptual model. We performed a two-day training workshop for the modelers and the subject matter experts. The conceptual model has been developed by the modelers and subject matter experts with our help.

Following the development of the conceptual models, we performed structured interviews with the modelers and reviewers who took part in the studies. A modeler, a reviewer and a user of the conceptual model were interviewed for the first case and two modelers were interviewed for the second case. 40 minutes were spent for each interview on the average and a total of ten hours were spent for analyzing and documenting the results. The files including the conceptual models, review records and the interview records were stored for further analysis. The first study was conducted using the tool developed as part of the KAMA framework. In the second study however, a commercial UML modeling tool was used due to specific project requirements. We provided the UML profiles that correspond to the elements of the KAMA notation to enable the UML modeling tool create KAMA diagrams.

The first case study was performed as part of a research project. In this project we looked for answers to the following research questions; 1) What are the major elements of a conceptual model? 2) How can we represent these elements? 3) How sufficient is the KAMA notation for representing existing conceptual models that were developed as textual documents?

In order to answer the first research question, we performed a literature survey and investigated existing conceptual models. A conceptual model should include both the static and dynamic properties of the mission space. We identified entity and mission as the major elements of the static and dynamic views respectively. Entity is very similar to the UML class possessing attributes and capabilities. The dynamic behavior of an entity is represented by a state machine, which shows all of the states an entity can be in and the transitions among these states. The actors, which can be derived from entities, are used to define roles that are one of the connectors that link the static and dynamic views. Another connector is the workproduct that is defined as part of the static view and used in the dynamic view. The dynamic view reveals a process based decomposition of the mission space, which is mainly formed by missions and tasks. Tasks are the building blocks of missions and contain the executing role, used inputs and produced outputs. Every mission has an objective, which is evaluated with respect to the measure connected to the objective. These major elements of a conceptual model arising from the results of our literature survey are confirmed by performing experimental studies that aim to regenerate existing text-based conceptual models.

The second research question is related with the representation technique that will be employed. The KAMA framework includes a notation that has been developed as a metamodel based diagrammatic modeling language for representing the conceptual models. The notation

specification contains the abstract syntax, concrete syntax and semantics definitions which are defined as textual specifications, diagrammatic symbols and a mixture of textual and OCL specifications respectively. As Akkök [1] has put forward, the diagrammatic modeling languages are more powerful from cognitive and visual viewpoints compared to the textual languages. We therefore preferred a diagrammatic conceptual modeling notation that is based on UML. The notation specification also includes the context and content of each KAMA diagram as well as the rules that constrain the modeler. Three experimental studies, which applied the notation in simulation systems development projects, have been performed for validating the notation. Another experimental study was conducted to examine the cognitive aspects of the graphical KAMA notation and a comparison with the textual representation of conceptual models [27].

In order to answer the third research question, we performed the first case study that included re-developing an existing text-based conceptual model using the KAMA framework. The objective of this study was to develop a KAMA model that included all information in the text-based conceptual model. The reviewer created a traceability matrix between the text-based conceptual model and the KAMA model. The validation results revealed that the KAMA notation was adequate for representing existing conceptual models. However, it should be noted that not all information residing in the original conceptual model was represented diagrammatically such as the geographical descriptions and the mathematical formulae. KAMA diagrams are used when they provide advantage over the textual descriptions. It is obvious that the geographical descriptions are best depicted using maps or three-dimensional drawings and the mathematical formulae are explained by mathematics equations. The first iteration of the first case study also helped us establish the requirements of the KAMA tool and refine the KAMA method definition. The use of the KAMA tool provided beneficiary features such as filtering of the diagram elements, extended search, moving model elements among the conceptual models, versioning of the diagrams, conceptual model elements and the conceptual models.

The second case study evaluated the effectiveness of the KAMA framework for developing conceptual models of the mission spaces for providing an answer to the fourth research question. The first advantage of using the KAMA framework appeared to be the increased involvement of the subject matter experts. Following a two-day seminar on conceptual modeling using the KAMA framework, the subject matter experts were actively involved in the conceptual modeling process through attending joint review meetings. The orderly definition of the KAMA method enabled them monitor the conceptual modeling process and perform validation activities as needed. As a consequence of this effective involvement, changes in the system requirements were identified early in the development life cycle. The missions and tasks defined in the conceptual model were used to establish traceability links to the operational requirements that were specified in the statement of work document. Furthermore, some of the entities were almost exactly transferred from the statement of work document.

The KAMA framework included the ability to represent both the dynamic and static features of the conceptual model, which was found to be valuable by the modelers. Roles, work products and entities played an important role in the integration of the different viewpoints provided by the framework. These integration points were noticeable through the links and the usage of common model elements among the diagrams. The recursive structure of the task flow diagrams enabled the modelers to define the details of the missions and tasks as required, with the advantage of not losing comprehensibility. We observed that the subject matter experts have a tendency to ignore some facts about the mission space that are assumed to be known by everyone, such as the roles that are responsible for a task. The use of task flow diagrams helped the modelers identify and correct such issues easily. The modelers mentioned that the diagrams also helped manage the complexity of the conceptual model. This is one of the difficulties of developing conceptual models using text based notations. Usability of a conceptual model is as important as the ease of development. One of the modelers stated that he found the method definition very helpful as a guide, since it was his first conceptual modeling experience.

Developing the conceptual model in iterations and performing walkthroughs after each iteration helped remove the defects earlier. Of the 150 issues identified during the walkthroughs, 125 (83%) were moderate or major issues. Had the project group not detected those issues in the conceptual modeling phase, the cost of correction would have been higher. These joint reviews also helped establish a common language among the modelers and the domain experts. Both the modelers and the subject matter experts agreed on the effectiveness of using diagrammatic representation for conceptual modeling over textual representation. However, a reviewer of the first case study stated that due to the semi-formal nature of the KAMA notation it had been difficult to perform formal validation techniques. The use of natural language for specifying the details of the missions and tasks was a negative issue in checking the internal consistency of the conceptual model. This semi-formal structure is a consequence of providing a common language for both the subject matter experts and the conceptual modelers. On the other hand, the KAMA framework enables to check syntactic and semantic constraints on the conceptual model through the use of rules at the metamodel level.

The case studies had the following limitations. The first case was a retrospective study; therefore the modelers could not validate the conceptual model with the subject matter experts. In the second study, the modelers did not use the KAMA tool, which might have caused difficulties in terms of usability. Although the commercial UML tool had to be interpreted in a way that supports the KAMA notation, it could not satisfy some of the operational features that were provided by the KAMA tool. All of the modelers in both of the case studies were knowledgeable in UML or similar software modeling languages. Even the subject matter experts in the second study did have software development knowledge. This might have formed a bias on the comprehensibility and usability of the KAMA method and the notation. The modelers did not have any experience on

developing conceptual models using structured methods; therefore it was not possible to have them compare the effectiveness of the KAMA method with other methods. One of the modelers in the first study stated that the KAMA framework provides a common language for the modelers and the subject matter experts at a high level; however, it may need to be enriched by first order logic or similar mechanisms for representing conditional rules formally.

CHAPTER 5

CONCLUSIONS

A conceptual modeling framework that includes a method definition, a notation and a supporting tool is essential for effective implementation of the conceptual analysis phase. The method definition should include the detailed process steps; the notation should provide an easy to use interface for both the conceptual modelers and the subject matter experts and the tool should support both the method and the notation. Our survey on the related literature revealed the lack of conceptual modeling frameworks that would provide effective mechanisms for the modelers and the subject matter experts. In the projects we have examined, we observed that the modelers had to define their own methods or use ad-hoc methods both of which resulted in inefficient representations of the mission space conceptual models that could not be reused and could not be effectively communicated. This thesis addressed issues arising from the field of mission space conceptual modeling by providing a framework for this purpose. The proposed framework can be generalized to all kinds of conceptual models such as simulation space conceptual model by modifying the metamodel definitions.

This chapter summarizes the results and contributions of this research and suggests future research directions based on the findings discovered during the thesis study.

6.1. Contributions

The major contribution of this research is the KAMA framework that is used for developing conceptual models of the mission space. We have created a framework for mission space conceptual modeling that included a method definition, a notation and a tool. The method defines a systematic way for developing conceptual models of the mission space and lets the subject matter experts and the modelers focus on the essential features of the mission space. The notation provides a well-defined syntax for the conceptual modelers and a diagrammatical representation that benefits the subject matter experts during the validation activities. The tool provides an efficient mechanism for developing conceptual models of the mission space by supporting the method and the notation. This framework helped conceptual modelers build a better understanding on the position of the conceptual analysis phase in the simulation development lifecycle.

Moreover, the framework offered all of the required tools for the novice modelers to develop conceptual models of the mission space. As stated in the findings of the second case study, the modelers did not spend time in inventing a new method for developing conceptual models and they spent most of their effort on analyzing and modeling the mission space. By following a defined method and using a specific notation, the modelers established a basis to propose improvement opportunities.

The method definition, which was very helpful especially for the novice modelers is another contribution of this study. The method defined the boundaries of the conceptual analysis phase by explicitly defining the series of activities, the inputs to and outputs from these activities, and the roles that are responsible for the execution of these activities. The modelers focused on the more important domain aspects rather than reinventing the conceptual modeling process. The method and the notation enabled the effective involvement of the subject matter experts in the development and validation of the conceptual model. In the second case study, the customer representatives who played the role of subject matter experts stated that they would like to spend little effort during the conceptual analysis phase. Following the training session on the KAMA framework, the customer decided to get more involved in the conceptual model development and validation activities and spent 188 person-hours for the validation activities. The conceptual modelers developed the conceptual models in a structured way and represented diagrammatically so that the subject matter experts could easily understand. The process definition helped manage the conceptual analysis phase by describing the responsibilities and dependencies among the tasks. The subject matter experts together with the conceptual modelers reviewed the diagrams that make up the conceptual model by means of joint walkthroughs. During these collaborative studies no issues have been raised about the flow of the review process. Both the modelers and the subject matter experts were knowledgeable about the method and notation therefore they walked through each diagram type on the most appropriate order. Our initial findings in the second case study revealed that this early validation activity would serve a great benefit in developing complex simulation systems. 125 of the 150 issues identified during the validation walkthroughs were denoted as moderate or major. Considering the cost of performing delayed changes in the simulation development lifecycle, the early detection of these issues decreased the total cost of defect removal activities.

A UML based domain specific conceptual modeling notation is a contribution of this study. The KAMA notation specification presents its relation with the UML and describes additional properties and constraints. The semantics of the notation is defined textually or as constraints using the Object Constraint Language where available. Since the KAMA notation is designed as an extension to the Meta Object Facility, it can also be easily extended or modified using similar extension rules that it is based on. This extendible structure of the notation makes it part of a suitable solution for different types of simulation systems. The KAMA notation is a domain

specific modeling language that enables developing conceptual models using the terminology of the subject matter experts. The notation is based on a metamodel and specifically designed for representing the conceptual models. Since the notation is domain specific, the subject matter experts did not have to cope with irrelevant topics. We have observed how easy subject matter experts can adapt to the notation specifically during the second case study. The second case study showed that the modeling notation is understandable by the subject matter experts after a two-days training session. The subject matter experts not only learned the modeling notation easily, but they also got actively involved in the validation of the conceptual model. The second case study revealed also the necessity to keep such a modeling notation simple. The conceptual modeling team in this case study developed complex state machines which were found to be complicated by the subject matter experts and eventually these diagrams had to be simplified. The notation covered both the dynamic and static views of a conceptual model and also the relations among these views, which were found to be useful by the conceptual modelers in the case studies.

The related studies that have their origins in this study are also contributions of this study. Aysolmaz [3] accomplished a master's thesis by extending the KAMA notation to include simulation space models. This study is a step to transform conceptual models into simulation design models. Tanriover [68] introduced a formal verification framework based on the KAMA notation and used the KAMA diagrams in his case studies. Kılıç [29] evaluated the cognitive aspects of the KAMA diagrams in order to reveal the perception differences among the non-engineer subject matter experts and the engineers who are experienced in modeling. He also performed experiments to identify the differences between text-based conceptual models and diagrammatic conceptual models in finding seeded defects.

Another contribution of the study is the accomplishment of the requirements elicitation and architectural design activities of the KAMA tool within the scope of a research project. However, due to the limitations of the development project, some of our ideas could not be implemented, which are presented as future work in Section 6.2. The KAMA tool provides guidance to the conceptual modelers by enabling the modelers develop, navigate, share, verify and maintain the conceptual models. A moderate simulation system may have hundreds of model elements and diagrams that may result in a difficult to manage conceptual model. The tool provides convenient methods for navigating and searching among the conceptual model elements. It uses the syntactic and semantic definitions of the notation for verifying the conceptual models. The modelers can publish the verified conceptual models into a common repository, where other simulation developers may benefit from previously developed conceptual models. Thus, the KAMA tool serves as a repository manager that supports reuse of the conceptual models as well as serving as a modeling editor.

The results of these case studies revealed that the KAMA framework can be successfully used for developing conceptual models. Despite the limitations of the notation and the tool, our

observations showed that both the subject matter experts and the modelers can achieve the expected benefits. The framework helped build a common language among the modelers and the subject matter experts, increasing the benefit of the conceptual analysis phase. It provided an environment and a mechanism for developing and representing the conceptual models.

6.2. Limitations and Future Work

Our case studies revealed some limitations and improvement opportunities about both the framework and the study. Some of these limitations have been discussed while the case studies are being explained; however, we provide a detailed view in this section.

Different types of simulation systems

KAMA framework has been tested on various types of simulation systems. Although our study aimed to cover a wide range, different types of simulation systems may require exclusive conceptual modeling needs. The metamodel based architecture of the KAMA notation may propose a solution to this requirement. As a future work, the metamodels may be designed according to the unique requirements of the domain of interest, such as simulation space. However, in such a case the modelers should thoroughly analyze the tradeoff between a better fitting metamodel and a more general metamodel that allows more flexibility and reusability. A further suggestion may be the usage of the framework not only for developing conceptual models of the mission space but also for developing any conceptual model. The metamodel may be extended to represent any domain, such as banking information systems, software processes, business processes, etc.

Limitations of the diagrams

The diagrams utilized in the KAMA notation are graphically similar to the UML diagrams. These diagrams aim to represent the knowledge stored in the model from different viewpoints, such as dynamic, static, local and central viewpoints. During the realization of the case studies, we observed that a diagram can become complicated as the number of model elements in a diagram increases. Kılıç [29] investigated the cognitive aspects of the KAMA diagrams and found that subject matter experts are more sensitive to complicated diagrams and therefore they face more comprehension problems. The KAMA notation prescribes the content of the each diagram type; however the tool may provide additional features for better utilization of the diagrams. Currently, a filtering mechanism and color schemes are utilized in the KAMA tool for assisting the modelers. The n-dimensional viewing feature helps the modelers observe the conceptual model from a wider viewpoint. This view also includes the relationships between model elements that are located in different diagrams, so that the modelers can see the dependencies among the diagrams. Future work may include alternative diagram types that represent conceptual models with different

abstraction levels. These diagrams may also be defined dynamically such that the modeler will design the content of the diagram.

Transition to the design phase

KAMA notation has been based on UML for a few reasons. First of all UML provides a sound infrastructure for defining modeling languages either by extending UML via profiles or defining a modeling language based on the Meta Object Facility. Another reason is the widespread usage of the UML in the software and systems development domains. We know that the conceptual model will be used as an input to the simulation design phase. It is of great benefit to the developers if the conceptual model can be transformed into simulation design, even partly. Using a common ancestor for the conceptual modeling language and the simulation design language will ease this process. The scope of our study did not include this transition; however Aysolmaz [3] extended the KAMA notation to include simulation space models. BOM [64] is another means for combining the conceptual model with the design model. An efficient transition process will increase the value of the conceptual models; therefore further research should be performed on this topic. Future work includes a transformation mapping between KAMA models and the BOM models.

Verification of the conceptual models

The KAMA tool verifies the conceptual models using its rule base to some extent. The well-defined syntax and semantics of the KAMA notation enables formal verification techniques to be applied. Tanriover [68] defined a verification method that can be applied to KAMA diagrams. This method is based on the execution of rules on the metamodel-level definitions. These rules are constraints that aim to decrease the defects injected by the modeler while developing the conceptual models. The metamodel-level rules can be defined in advance by the meta-model designer; however they provide only general checks. In order to accomplish a better verification of the conceptual models, these rules can be defined and applied at the model level. Model-level rules are more close to the domain of interest and are required to be defined by the modeler. As a future work, the tool may allow the user define new verification rules through a well-defined rule definition language. The verification rules may also be grouped in order to form conceptual modeling patterns. These patterns may represent best practices that the modelers should follow or worst practices that they should avoid.

Limitation of the tool and its criticality

The tool support is an essential part of our framework. A tool that is specifically designed and developed to support the conceptual modeling method has been developed within the scope of a research project. In the first case study and two experimental studies, we utilized the tool and observed that the tool increases the efficiency of conceptual modeling by providing unique features such as; reusing conceptual models from the common repository, filtering diagrams,

performing context-sensitive search, n-dimensional model viewing, model element versioning, custom report generation, conceptual model verification and custom properties management for the conceptual model elements.

In the second study, the modeling team did not use the KAMA tool. They used a commercial UML tool by utilizing the UML profiles for KAMA. They were not able to perform some of the features provided by the KAMA tool, which are explained in the above paragraph. The KAMA conceptual models can be generated using a UML tool, however the modelers will lack the above features. Besides, some syntactic and semantic rules are checked by the KAMA tool either simultaneously during the modeling process or manually after the model has been baselined. In case the KAMA tool is not used, these controls have to be performed by the conceptual modeler. Future work includes extending the KAMA tool to edit metamodels, allow merging of the conceptual models, perform more intelligent search, define and dynamically integrate the semantic rules and evaluate the probabilistic decision points by utilizing a well-defined syntax.

Interoperability among the conceptual models

There are many simulation systems that accomplish diverse objectives and need to operate in collaboration. The researchers in the simulation domain have been working on the interoperability issues and the HLA is proposed as an IEEE recommended practice [24]. HLA deals with interoperability at the technical level by providing a common language and common interfaces; however, interoperability at higher levels is still a problem. Tolk et.al [69] and US DoD [70] and NATO [40] proposed models for interoperability. These models define levels for checking the interoperability among simulation systems, information systems and conceptual models. The levels vary from isolated systems (no data exchange) to distributed systems (structured data exchange) and then to universal systems (seamless sharing of information). The KAMA framework supports interoperability by providing a common syntax and semantics for developing conceptual models. Further research should be performed to maintain interoperability when the metamodel is modified or a new metamodel is defined. The interoperability of the conceptual models may be checked according to the abovementioned interoperability models prior to the integration of the simulation systems so that the simulation developers predict the challenges awaiting them.

Reusability among the conceptual models

One of the objectives of our study was establishing a common platform that can be utilized for the reuse of the conceptual models. The common repository stores all of the published and accredited conceptual models, model elements and the diagrams that can be shared among various simulation systems development projects. The modelers may search the repository, view the found model element, conceptual model or the diagram and download and use in the local conceptual model. However, there are some problems regarding the reuse of these repository elements, finding the

suitable element and the level of interoperability among these elements. KAMA tool provides a search mechanism for searching the repository, but this search is based on simple string match. A conceptual model element that has the same name as the one we are searching for may have a different content or vice versa. Even though the names and contents match, the level of detail may be different so that the modeler may not reuse the conceptual model element. A ship in the conceptual model of a strategic level simulation system will probably have different attributes and behavior than the conceptual model of a tactical level simulation system. Thus, the reusability is very much related with the interoperability levels of the conceptual models. Future work may include intelligent search mechanisms that look for these interoperability levels as well as employing context aware search algorithms.

Case study method

The case study method used for the validation of the study also has its own limitations. The validity of the framework is limited with the number of case studies and the experimental studies. The participants' knowledge on the modeling languages may have caused a bias on the evaluation of the method and the notation. The conceptual models developed within the context of the case studies have not yet been used as an input to the subsequent simulation development phases; therefore we could not evaluate the exact benefits of the conceptual models to these phases. The case studies and the experimental studies addressed only a limited number of subsets of the simulation systems such as command and control simulation systems, training simulator, strategic level war-game simulation system.

In the first case study the KAMA tool could only be used as a modeling editor, omitting the common repository features. As this repository gets mature, the reusability of the conceptual models may be validated. In order to obtain further evidence about the validity of the framework, additional case studies on different types of simulation systems should be performed. Employing subject matter experts who are not knowledgeable on the modeling notations would reveal the actual benefits of the framework.

REFERENCES

1. Akkök, N., “Towards the Principles of Designing Diagrammatic Modeling Languages: Some Visual, Cognitive and Foundational Aspects”, PhD Dissertation, Department of Informatics, University of Oslo, 2004
2. Alan, A., Prikster, B. (1998) Principles of Simulation Modeling, Handbook of Simulation, New York: John Wiley & Sons.
3. Aysolmaz, B. (2007) Conceptual Model Of A Synthetic Environment Simulation System Developed Using Extended Kama Methodology, Technical Report, 2006-2007/2-17, Informatics Institute, Middle East Technical University
4. Balci, O. (1994) Validation, Verification, and Testing Techniques Throughout the Life Cycle of a Simulation Study. *Annals of Operations Research*, 53, 121-173.
5. Balci, O., Nance, R.E. (1985) Formulated problem verification as an explicit requirement of model credibility. *Simulation* 45 (2) 76–86.
6. Banks, J., Carson, J.S. & Nelson, B.L. (1996) *Discrete Event Simulation, 2nd ed.*, New Jersey: Prentice Hall.
7. Borah, J. (2002) Conceptual Modeling – The Missing Link of Simulation Development, *Proceedings of the Spring Simulation Interoperability Workshop*
8. Borah, J. (2007) Informal Simulation Conceptual Modeling -- Insights from Ongoing Projects *Proceedings of the Spring Simulation Interoperability Workshop*
9. Brassé, M., Mevassvik, O.M. & Skoglund, T. (2003) Federation Composition Process and Tool Support in EUCLID RTP 11.13 *Proceedings of the Fall Simulation Interoperability Workshop*
10. Business Process Modeling Notation, <http://www.bpmn.org>, Last visit on: 12.01.2007
11. Chapman, R.M. (2000) Mission-Oriented Conceptual Modeling Framework for Distributed Mission Training, *Proceedings of the Fall Simulation Interoperability Workshop*
12. Chrissis, M.B., Konrad, M., Shrum, S. (2003) CMMI(R): Guidelines for Process Integration and Product Improvement (The SEI Series in Software Engineering), Boston: Addison-Wesley
13. Civelek, M. (2006) Modeling A Sample Mission Space Of Taf By Using Kama-C4ISRMOS Notation, Technical Report, 2005-2006/2-21, Informatics Institute, Middle East Technical University

14. Davis, P. K. (1992) Generalizing Concepts and Methods of Verification, Validation and Accreditation (VV&A) for Military Simulations, Paper R-4249-ACQ, *RAND*, Santa Monica, CA
15. Defense Modeling and Simulation Office (DMSO) (1997) Conceptual Models of the Mission Space (CMMS) Technical Framework, USD/A&T-DMSO-CMMS-0002 Revision 0.2.1, <http://dmsomil/briefs/entereff/doc/cmmstf.doc>
16. Fishwick, P.A. (1995) *Simulation Model Design and Execution: Building Digital Worlds*. Upper Saddle River, NJ: Prentice-Hall.
17. Ford, K. (2004) The Euclid RTP 11.13 SE Development & Exploitation Process (SEDEP) *Proceedings of the Spring Simulation Interoperability Workshop*
18. Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley
19. Globe J. (2007) Using SysML to Create a Simulation Conceptual Model of a Basic ISR Survivability Test Thread *Proceedings of the Spring Simulation Interoperability Workshop*
20. Güleç, S. (2006) Conceptual Model Development with KAMA-C4ISR MOS Notation - An Analyst's Approach, Technical Report, 2006-2007/1-11, Informatics Institute, Middle East Technical University
21. Haddix, F. (1998) Mission Space, Federation, and Other Conceptual Models, Paper 98S-SIW-162, *Proceedings of the Spring Simulation Interoperability Workshop*
22. <http://www.euclid1113.com/>, Euclid RTP 11.13 SEDEP, Last visit on: October 2007
23. IDEF1X Data Modeling Method, <http://www.idef.com/IDEF1X.html>, Last visit on: 12.01.2007
24. IEEE Computer Society (2003) IEEE 1516.3 Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP)
25. IEEE Computer Society (2000) IEEE Std. 1516-2000 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) — Framework and Rules
26. Johnson, T. H. (2002) “The Conceptual Model of the Mission Space and Data Engineering Toolset”, Technical Report
27. Karagöz, N.A., Eryılmaz, U., Yıldız, A., Demirors, O., Bilgen, S. (2005) KAMA Project Research Report, Center of Modeling and Simulation, Middle East Technical University
28. Keuning, M. & Lemmers, A. (1997) RTP 11.13 Gets To Grips with SE Specifications *Proceedings of the European Simulation Interoperability Workshop*
29. Kılıç, Ö. (2007) Cognitive Aspects of Conceptual Modeling Diagrams : An Experimental Study, MSc. Thesis, Department of Cognitive Science, Informatics Institute, Middle East Technical University
30. Lacy, L.W., Randolph, W., Harris, B., Youngblood, S., Sheehan, J., Might, R. & Metz, M. (2001) Developing a Consensus Perspective on Conceptual Models for Simulation Systems, *Proceedings of the Spring Simulation Interoperability Workshop*

31. Law, A.M. & Kelton, W.D. (1999) *Simulation, Modeling and Analysis*, 3rd Edition, McGraw-Hill.
32. Lemmers, A. & Jokipii, M. (2003) SEST: SE Specifications Tool-set, *Proceedings of the Fall Simulation Interoperability Workshop*
33. Lundgren, M., Lozano, M.G. & Mojtahed, V. (2004) CMMS under the Magnifying Glass – An Approach to Deal with Substantive Interoperability”, 04F-SIW-0101, *Proceedings of the Fall Simulation Interoperability Workshop*
34. Manfred R. (1999) Fidelity Requirements Specification: A Process Oriented View, *Proceedings of the Fall Simulation Interoperability Workshop*
35. McGuire, F. (1998) Simulation of Healthcare, Handbook of Simulation, New York: John Wiley & Sons.
36. Merriam Webster Online Dictionary, <http://www.merriam-webster.com/dictionary/simulation>, Last visit on: 11.11.2007
37. Mojtahed, V., Lozano, M.G., Svan, P., Andersson, B., Kabilan, V. (2005) DCMF, Defense Conceptual Modeling Framework, Technical Report
38. Musselman, K. J. (1998) Guidelines for Success, Handbook of Simulation, New York: John Wiley & Sons.
39. Nance, R.E. (1994) The Conical Methodology and the Evolution of Simulation Model Development. *Annals of Operations Research*, 53, 1-45.
40. NATO (2003) Allied Data Publication 34 (ADatP-34): NATO C3 Technical Architecture (NC3TA), Version 4.0, <http://www.nato.int/docu/standard.htm>
41. Norman, V. & Banks, J. (1998) Managing the Simulation Project. Handbook of Simulation, New York: John Wiley & Sons.
42. OMG, Common Warehouse Metamodel Specification, version 1.0, 02.02.2001
43. OMG, UML Infrastructure Document, version 2.0, 05.07.2005
44. OMG, UML Superstructure Document, version 2.0, 04.07.2005
45. Pace, D.K. (1999) Conceptual Model Descriptions, Paper 99S-SIW-025, *Proceedings of the Spring Simulation Interoperability Workshop*
46. Pace, D.K. (1999) Use of Subject Matter Experts (SMEs) in Simulation Evaluation, *Proceedings of the Fall Simulation Interoperability Workshop*.
47. Pace, D.K. (2000) Simulation Conceptual Model Development, *Proceedings of the Spring Simulation Interoperability Workshop*
48. Pace, D.K., (1999) Development and Documentation of a Simulation Conceptual Model,” *Proceedings of the Fall Simulation Interoperability Workshop*
49. Pidd, M. (2004) *Computer Simulation in Management Science*, 5th ed. Chichester, UK: Wiley

50. Richter, H., Marz, L. (2000) Toward a standard process: The use of UML for designing simulation models. J.A. Joines, R.R. Barton, K. Kang, P.A. Fishwick (Eds). *Proceedings of the Winter Simulation Conference*. IEEE, Piscataway, NJ, pp. 394–398.
51. Robinson, S. (2002) Modes of Simulation Practice: Approaches to Business and Military Simulation. *Simulation Practice and Theory*, 10, 513-523.
52. Robinson, S. (2004) *Simulation: The Practice of Model Development and Use*. Chichester, UK: Wiley.
53. Robinson, S. (2005) Distributed Simulation and Simulation Practice. *Simulation: Transactions of the Society for Modeling and Computer Simulation*, 81 (1), 5-13.
54. Robinson, S. (2006) Conceptual Modeling For Simulation: Issues And Research Requirements, *Proceedings of the Winter Simulation Conference*
55. Robinson, S. (2007a) Conceptual modeling for simulation Part I: Definition and requirements. *J. Opl. Res. Soc.* 59 (3) 278-290.
56. Robinson, S. (2007b) Conceptual modeling for simulation Part II: A framework for conceptual modeling. *J. Opl. Res. Soc.* 59 (3) 291-304.
57. Rothenberg, J., (1991) Knowledge-Based Simulation at the RAND Corporation, *Knowledge-Based Simulation: Methodology and Application*, Fishwick and Modjeski (eds.), Springer-Verlag
58. Rumbaugh, J., Jacobson, I. & Booch, G. (1998) *The Unified Modeling Language Reference Manual*, Addison Wesley
59. Ryan, J. & Heavey, C. (2006) Requirements Gathering for Simulation. *Proceedings of the Third Operational Research Society Simulation Workshop*. The Operational Research Society, Birmingham, UK, 175-184.
60. Sargent, R. G., (1987) An Overview of Verification and Validation of Simulation Models, *Proceedings of the Winter Simulation Conference*
61. SGKS-C4ISR-MOS 106 (2004) Sistem İsterleri ve Kavramsal Modeli Dokümanı Sürüm 1.1.2, TÜBİTAK-MAM-BTE
62. Shannon, R. (1975) *Systems Simulation: The art and science*, New Jersey: Prentice-Hall
63. Sheehan, J. (1998) Conceptual Models of the Mission Space (CMMS): Basic Concepts, Advanced Techniques, and Pragmatic Examples, *Proceedings of the Spring Simulation Interoperability Workshop*
64. SISO (Simulation Interoperability and Standardization Organization) (2006) Base Object Model (BOM) Template Specification, SISO-STD-003-2006
65. SISO (Simulation Interoperability and Standardization Organization) (2006) Base Object Model (BOM) Guidance Specification, SISO-STD-003.1-2006
66. Sudnikovich, W. P., Pullen, J. M., Kleiner, M. S. & Carey, S. A. (2004) Extensible Battle Management Language as a Transformation Enabler, *Simulation*

67. Tanriover, O. (2008) Inspection of Conceptual Models Develeoped in Kama-C4ISRMOS Notation: Two Case Studies, Informatics Institute , METU/II-TR-2008-1
68. Tanriover, O. & Bilgen S. (2007) An Inspection Approach for Conceptual Models for the Mission Space Developed in Domain Specific Notations of Uml, *Proceedings of the Fall Simulation Interoperability Workshop*
69. Tolk, A. & Muguira J. A., (2003) The Levels of Conceptual Interoperability Model, *Proceedings of the Fall Simulation Interoperability Workshop*
70. US DoD OSD (C3I) (1998) Levels of Information Systems Interoperability, C4ISR Architectures Working Group, Director for Architecture and Interoperability: <http://www.c3i.osd.mil/org/cio/i3/>
71. van der Zee, D.J. (2006) Building Communicative Models – A Job Oriented Approach to Manufacturing Simulation. *Proceedings of the Third Operational Research Society Simulation Workshop*. The Operational Research Society, Birmingham, UK, 185-194.
72. Widman, L.E., Loparo, K.A., & Nelson, N.R. (eds.) (1989) *Artificial Intelligence Simulation & Modeling*, New York: John Wiley & Sons
73. Willemain, T.R. (1995) Model formulation: What experts think about and when, *Opns. Res.* 43 916–932.
74. Yin, R.K. (1994) *Case Study Research: Design and Methods*. Applied Social Research Methods Series, Vol.5, 2nd Ed., Sage Publications, Inc.
75. Zeigler, B.P., Praehoffer, H. & Kim, T.G. (2000) *Theory of Modeling and Simulation*, 2nd ed., San Diego: Academic Press

VITA

N. Alpay KARAGÖZ was born in Burdur, Turkey in 1977. He received his bachelor degree in Computer Engineering from METU (Middle East Technical University) in 1999. In 2001, he had his MSc. degree in Information Systems from the Informatics Institute, METU. During 1999 and 2001, he worked as a research assistant in the Informatics Institute. Since 2001, he is a partner of Bilgi Grubu Ltd. and he has been working as a researcher, consultant and trainer.

He took part in many projects as a software process improvement consultant. He offered “Object Oriented Software Development” and “Personal Software Process” courses in the Software Management program of the Informatics Institute, METU. He took part in 4 research projects as a researcher; these included developing a tool for assisting individual software process improvement initiatives, developing a process improvement asset library based on CMMi, developing a tool that supports the conceptual modeling activities, extending the conceptual modeling tool to support modeling any domain by utilizing a metamodel based approach.

His research interests include; process improvement, business process modeling, conceptual modeling, software quality engineering and object oriented software development.

Selected Publications:

1. Karagöz, N.A., Demirors, O., Gencel, Ç. (2007). Acquiring Innovative Software Systems: Experiences from the field. *EuroMicro Conference*, Luebeck.
2. Karagöz, N.A., & Demirors, O. (2007). Developing Conceptual Models of the Mission Space (CMMS) – A Metamodel Based Approach. *Simulation Interoperability Workshop Spring07*, Norfolk VA.
3. Karagöz, N.A., & Demirors, O. (2006). Uml Tabanlı Bir Metamodelleme Altyapısı ile Görev Uzayı Kavramsal Modeli Geliştirme. *SAVTEK06*, Ankara.
4. Karagöz, N.A., Demirors, O., Ündeğer, Ç., Gencel, Ç. (2006). Simülasyon Sistemlerinde Görev Uzayı Kavramsal Modeli Geliştirme: Bir Süreç Tanımı. *SAVTEK06*, Ankara.
5. Karagöz, N.A., Demirors, O., (2005) Simülasyon Sistemleri İçin Kavramsal Model Geliştirmeye Model Tabanlı Bir Yaklaşım. *USMOS05*, Ankara.