

A VERIFICATION APPROACH FOR DYNAMICS OF
METAMODEL BASED CONCEPTUAL MODELS
OF THE MISSION SPACE

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

BY

UTKAN ERYILMAZ

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

MARCH 2011

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.

Prof. Dr. Semih BİLGEN
Supervisor

Examining Committee Members

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

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

Assist. Prof. Dr. Kayhan İMRE (Hacettepe U., CENG) _____

Assist. Prof. Dr. Altan KOÇYİĞİT (METU, II) _____

Assoc. Prof. Dr. Halit OĞUZTÜZÜN (METU, CENG) _____

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 : Utkan Eryılmaz

Signature : _____

ABSTRACT

A VERIFICATION APPROACH FOR DYNAMICS OF METAMODEL BASED CONCEPTUAL MODELS OF THE MISSION SPACE

Eryilmaz, Utkan

Ph.D., Department of Information Systems

Supervisor: Prof. Dr. Semih Bilgen

March 2011, 178 pages

Conceptual models were introduced in the simulation world in order to describe the problem domain in detail before any implementation is attempted. One of the recent approaches for conceptual modeling of the military mission space is the KAMA approach which provides a process description, a UML based notation, and a supporting tool for developing conceptual models. The prominence of the approach stems from availability of guidance and applications in real life case studies. Although the credibility of a conceptual model can be leveraged through use of a structured notation and tools, the verification and validation activities must be performed to arrive at more credible conceptual models. A conceptual model includes two categories of information: static and dynamic. The dynamic information describes the changes that occur over time. In this study, the dynamic characteristics of the conceptual models described in KAMA notation are explored and a verification approach based on these is proposed. The dynamical aspects of KAMA notation and example conceptual models provide the necessary information for characterization of the dynamical properties of conceptual models. Using these characteristics as a basis, an approach is formulated that consists of formal and semiformal techniques as well as supporting tools. For description of additional properties for dynamic verification, an extended form of KAMA is developed, called the KAMA-DV notation. The approach is applied on two different real-life case studies and its effectiveness is compared with earlier verification studies.

Keywords: Conceptual Modeling, Conceptual Model Dynamics, Verification.

ÖZ

METAMODEL TABANLI GÖREV UZAYI KAVRAMSAL MODELLERİNİN DİNAMİKLERİNE YÖNELİK BİR DOĞRULAMA YAKLAŞIMI

Eryılmaz, Utkan

Doktora, Bilişim Sistemleri

Tez Danışmanı: Prof. Dr. Semih Bilgen

Mart 2011, 178 pages

Kavramsal modeller simülasyon dünyasına problem uzayının detaylı bir şekilde tasviri için öne sürülmüşlerdir. Askeri görev uzayı için geliştirilmiş en yeni kavramsal model geliştirme yaklaşımlardan biri, süreç tanımı, UML tabanlı notasyon ve destekleyici aracı sağlayan KAMA yaklaşımıdır. KAMA yaklaşımının öne çıkması yaklaşımla ilgili rehber bilgi mevcut olması ve yaklaşımın gerçek dünyada vaka çalışmalarına uygulanmasından kaynaklanmaktadır. Kavramsal modelin güvenilirliği yapısal bir notasyon ve araçlarla yükseltilmesine rağmen, güvenilirliği artırmak için doğrulama ve geçerleme faaliyetlerinin gerçekleştirilmesi gereklidir. Bir kavramsal model static ve dinamik olmak üzere iki kategoride bilgi içerir. Dinamik bilgi zamanla meydana gelen değişiklikleri betimler. Bu çalışmada KAMA notasyonunda betimlenmiş kavramsal modellerin dinamik özellikleri incelenmiş ve bunları temel alan bir doğrulama yaklaşımı önerilmiştir. KAMA notasyonundaki dinamik bakış açıları ve örnek kavramsal modeller kavramsal modellerdeki dinamik özelliklerin karakterizasyonu için gerekli bilgiyi sağlarlar. Bu karakteristikler temel olarak kullanılarak, formel ve yarı formel yaklaşımları ve destekleyici araçları içeren bir yaklaşım tanımlanmıştır. Dinamik doğrulama için gerekli özelliklerin tanımlanması için KAMA'nın genişletilmiş bir biçimi geliştirilmiştir ve bu notasyona KAMA-DV adı verilmiştir. Yaklaşım gerçek yaşamdan iki farklı örnek üzerinde uygulanmış ve etkililiği önceki doğrulama çalışmalarıyla karşılaştırılmıştır.

Anahtar Kelimeler: Kavramsal modelleme, Kavramsal model dinamikleri, doğrulama.

ACKNOWLEDGMENTS

I want to express my gratitude to my supervisor Semih Bilgen for his guidance, support, understanding, and encouragement. With his support, I dared to deal with all the hardness I encountered during this study.

I also want to thank all the examining comitee members, Onur Demirörs, Halit Oğuztüzün, Altan Koçyiğit, and Kayhan İmre for their recommendations and suggestions.

Part of this work has been performed in the KAMA Conceptual Model Development Tool Project. I would like express my appreciation to members of the KAMA project group who participated in laying the groundwork for this study. I am most grateful to Alpay Karagöz and Ozgur Tanrıöver for their collabaration during this work especially as participants of the case study.

I also want to extend my thanks to Veysi İşler, my director at MODSIMMER, and other staff for supporting me during the study.

For the final years of this work, my friends in Ankara, Evren Deviren, Ulaş Beldek, and Çagatay Topal supported me morally and provided such friendly places to stay that I even did not feel as a guest. I also extend my thanks to all my friends for their encouragement.

I would like to thank to my family, my mother Nuray, my father İbrahim, my grandmother Mebrure, my uncle Hüseyin for supporting me without any doubt in pursuing my goals in life. My younger brother Baran made me smile even at the times I feel most stressful.

During last phases of this work, my wife Neslihan supported me and turned difficult times to enjoyable. I am grateful for her patience, understanding, and encouragement.

TABLE OF CONTENTS

ABSTRACT	iv
ÖZ.....	v
ACKNOWLEDGMENTS	vi
TABLE OF CONTENTS.....	vii
LIST OF ABBREVIATIONS AND ACRONMYS.....	x
LIST OF FIGURES	xi
LIST OF TABLES.....	xii

CHAPTER

1 INTRODUCTION	1
1.1 Verification and Validation of Conceptual Models.....	1
1.2 The Research Problem	4
1.3 The Proposed Approach.....	4
1.4 Contributions of the Thesis	6
1.5 Organization of the Thesis	6
2 CONCEPTUAL MODELS AND VERIFICATION.....	7
2.1 Interpretation of Conceptual Modeling for Simulations.....	7
2.1.1 Conceptual Model Definitions.....	7
2.1.2 Conceptual Model Scope.....	9
2.1.3 Conceptual Model Development Methods.....	10
2.1.4 Conceptual Model Verification.....	14
2.2 KAMA: A Metamodel based approach for Conceptual Modeling.....	19
2.2.1 Evaluation of KAMA as a Conceptual Modeling Approach.....	19
2.3 Verification Approaches for Dynamic Properties	20
2.3.1 Verification Methodologies for Metamodel Based Conceptual Models...	21
2.3.2 Verification of UML <i>Activity</i> Descriptions	22
2.3.3 Verification of Conceptual Business Process Descriptions.....	24
3 THE DEFINITION OF KAMA SEMANTICS.....	27
3.1 Sets of Model Elements	29
3.1.1 <i>Mission Space Fundamental Package</i> Elements	30
3.1.2 <i>Mission Space Sequencing Package</i> Elements	31
3.1.3 Elements of Other Packages	32

3.2	Attributes	36
3.3	Constraints	37
3.4	Derived Sets.....	43
3.5	The Relationship of KAMA Metamodel Definition with Dynamic Verification	44
4	CONCEPTUAL MODEL DYNAMICS.....	45
4.1	Conceptual Model Dynamics Characteristics and Limitations	45
4.1.1	Characteristics of Conceptual Model Dynamics.....	45
4.1.2	Limitations of Conceptual Model Dynamics.....	47
4.2	The Context of Dynamics in KAMA Models	48
4.2.1	Basic <i>Tasks</i>	48
4.2.2	The Hierarchy of <i>Tasks</i>	51
4.2.3	<i>SynchronizationPoint</i> Model Element.....	54
4.2.4	<i>DecisionPoint</i> Model Element.....	57
4.2.5	Cascaded <i>DecisionPoints</i> and <i>SynchronizationPoints</i>	57
4.2.6	<i>WorkProduct Package</i> Model Elements	57
4.2.7	<i>Role Package</i> Model Elements	58
4.2.8	<i>Objective Package</i> Model Elements	58
4.2.9	Other Static Relations.....	59
4.2.10	Relations with Objects (Instances of Entities).....	59
5	V&V REQUIREMENTS FOR CONCEPTUAL MODEL DYNAMICS.....	60
5.1	Conceptual Model Verification Requirements	60
5.1.1	Error Prone Parts of a Conceptual Model.....	61
5.2	Execution Dynamics for Model Elements	61
5.2.1	Basic Concepts Related to Execution of <i>Tasks</i>	61
5.2.2	Phases of <i>Task</i> Execution	65
5.2.3	Example of Basic Triggering during <i>Task</i> Execution	66
5.2.4	Variances in <i>Task</i> Executions Related to Model Elements	67
6	THE VERIFICATION APPROACH FOR CONCEPTUAL MODEL DYNAMICS.....	72
6.1	Model Based Verification Approach	72
6.2	KAMA-DV Metamodel	73
6.2.1	KAMA-DV Metamodel Organization.....	74
6.2.2	KAMA-DV Core Package.....	74
6.2.3	KAMA-DV Behaviour Package	80
6.3	General Outline of the Method.....	81
6.3.1	Step 1: Structuring the Model and Static Verification	83
6.3.2	Verification of the Static Constraints	83
6.3.3	Checking of Properties Related to Dynamics	87
6.3.4	Step 2: Enriching the Dynamic Semantics	89
6.3.5	Dynamic Analysis	90
6.4	Tool Support for the Verification	91

6.4.1	Selection of the Verification Tool.....	91
6.4.2	Models Involved in the Verification Approach	96
6.4.3	Adaptation of the Tool for Checking of Metamodel Properties	97
6.4.4	The Tool for checking the Dynamic Properties of the Model	98
7	EVALUATION OF THE VERIFICATION APPROACH	99
7.1	Selection of the Case Study Approach.....	99
7.2	Case Study 1: Conceptual Model of the Surveillance Mission	100
7.2.1	Case Study Design	100
7.2.2	Results of the Case Study	102
7.2.3	Evaluation of the Case Study.....	105
7.2.4	Improvements of the Approach Based on the Recommendations	106
7.3	Case Study 2: Engagement Task	107
7.3.1	Case Study Design	107
7.3.2	Results of the Case Study	108
7.3.3	Evaluation of the Case Study.....	111
8	DISCUSSION AND CONCLUSION.....	112
8.1	Contributions	112
8.2	Accomplished Work	113
8.2.1	Verification Approach Based on Formal Techniques for Metamodel Based Conceptual Models	113
8.2.2	UML <i>Activity</i> Verification and Conceptual Business Process Verification.....	117
8.2.3	Comparison of Related Approaches.....	118
8.3	Directions for Further Research.....	120
8.3.1	Better Syntax and Semantics Definitions for Conceptual Models.....	120
8.3.2	Execution of Conceptual Models	120
8.3.3	Tool Support Throughout the Verification Lifecycle	121
8.3.4	Application of Other Techniques	121
8.3.5	Quantitative Analysis of Issues Considering Model Metrics	122
	REFERENCES	123
	APPENDICES	
	A. SGKS SURVEILLANCE MISSION AND VERIFICATION RESULTS.....	131
	B. KAMA-DV METAMODEL DEFINITION IN EMF.....	171
	CURRICULUM VITAE.....	177

LIST OF ABBREVIATIONS AND ACRONYMS

BOM: Base Object Models
C4ISR: Command, Control, Communications, Computer, Intelligence, Surveillance, Reconnaissance.
CMMS: Conceptual Models of the Mission Space
DCMF: Defence conceptual Modeling Framework
DMSO: United States Department of Defense Modeling and Simulation Office
DoD: Department of Defense
DV: Dynamic Verification
EMF: Eclipse Modeling Framework
FDMS: Functional Descriptions of the Mission Space
FEDEP: Federation Development and Execution Process
HLA: High Level Architecture
IDE: Integrated Development Environment
IEEE: Institute of Electrical and Electronics Engineers
IEEE FEDEP: IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process
IEEE HLA: IEEE High Level Architecture
KAMA: KAMA Conceptual Modeling Approach
KAMA-DV: KAMA Conceptual Modeling Approach Extended for Dynamic Verification
MDA: Model Driven Architecture
MDT: Model Development Tools
MOF: Meta Object Facility
OCL: Object Constraint Language
OMG: Object Management Group
REVVA: Common Validation, Verification and Accreditation Framework for Simulation
UML: Unified Modeling Language
SGKS: Border Surveillance Control System
SISO: Simulation Interoperability Standards Organization
V&V: Verification and Validation
VV&A: Verification, Validation and Accreditation
VV&A RPG: Verification, Validation and Accreditation Recommended Practices Guide

LIST OF FIGURES

Figure 2-1. Relationship between Metamodels and Models	22
Figure 3-1. Dependencies between Mission Space Packages.....	30
Figure 4-1 Two <i>Task Flow Diagrams</i> with Different Semantics for <i>Task</i> Execution	49
Figure 4-2. A Delayed <i>Task</i> Execution	49
Figure 4-3 <i>Task Flow Diagram</i> Containing <i>SingleExecution Task</i> vs. a <i>Task</i> that is not <i>SingleExecution</i>	50
Figure 4-4 UML <i>activity</i> , <i>activityNode</i> and <i>action</i> Relations	52
Figure 4-5 <i>FinalTask</i> Differences in Execution.....	53
Figure 4-6 A <i>SynchronizationPoint</i> of Fork Type with Flows Synchronized	56
Figure 5-1 Execution States of Elements of <i>Task Flow Diagram</i>	64
Figure 5-2 The Creation, Starting and Finishing Time of a <i>Task</i> Execution.....	65
Figure 5-3 General Case of <i>Task</i> Execution	66
Figure 5-4 Execution of Successor <i>Task</i>	68
Figure 5-5 Execution of Fork Type <i>SynchronizationPoint</i>	68
Figure 5-6 Execution of Join Type <i>SynchronizationPoint</i>	69
Figure 5-7 <i>DecisionPoint</i> Execution	70
Figure 5-8 Execution of <i>Task</i> Hierarchy	71
Figure 6-1 Distinct Models Used for Verification	73
Figure 6-2. The Process of Dynamic Verification Shown as a <i>Task Flow Diagram</i>	82
Figure 6-3 Distinct Models Used for Verification	97
Figure 7-1 KAMA-DV Model Elements for Sequencing Expressed as an Ecore Diagram.....	103
Figure 7-2. The Equivalent EPC of “Advance in Air” <i>Task Flow Diagram</i> that is not sound	110

LIST OF TABLES

Table 2-1. Various Definitions of Conceptual Models	8
Table 2-2. The Definition of V&V from the Standpoint of Software and Simulation Development	15
Table 2-3. Verification and Validation Methods for Conceptual Model Listed in VV&A RPG	18
Table 2-4. The Elements Used in EPC Diagrams and their Counterparts.....	26
Table 3-1. UML Metamodel Elements Related to KAMA and KAMA Mission Space Package Model Elements.....	28
Table 3-2. KAMA Mission Space Model Elements, their Abbreviations and Containing Packages	35
Table 6-1. Number of Iterations Needed for Checking Fundamental Package Rules	84
Table 6-2. Number of Iterations Needed for Checking Sequencing Package Rules	85
Table 6-3. Number of Iterations Needed for <i>Workproducts Package</i> Rules	86
Table 6-4. Number of Iterations Needed for <i>Roles Package</i> Rules	86
Table 6-5. Number of Iterations Needed for <i>Objectives Package</i> Rules	86
Table 6-6. Number of Iterations Needed for <i>Mission Space Complete</i> Package Rules	87
Table 6-7. Tools Dealing with Analysis of Activity Charts and EPCs	94
Table 6-8. Other Tools for Simulation of UML Models	95
Table 6-9. Features and Plug-ins Used during Verification Process	96
Table 7-1. Model Statistics of SGKS Conceptual Model.....	101
Table 7-2. Statistics of the SGKS <i>Surveillance Mission</i>	102
Table 7-3. Number and Type of Errors Found in Various DHS <i>Tasks</i>	108
Table 7-4. The Corresponding Errors and Issues.....	109
Table 8-1. Errors Found by Inspection and Related Errors in KAMA-DV in Surveillance Mission	115
Table 8-2. The Correspondence of Consistency Checks with KAMA-DV	116
Table 8-3. Comparing of EPC, UML and KAMA in Terms of Aspects in Conceptual Models.....	119

CHAPTER 1

INTRODUCTION

As the use, scope and effect of complex simulation systems grow, higher levels of validity are required. It is easier to build systems from components, run the systems in distributed environments and analyze the results, but determining whether the simulation is correctly designed for the intended purpose is a hard task. Establishing validity is always difficult and costly; validity of a simulation is enhanced through iterative application of validation and verification activities during the simulation development process.

Typically for large scale systems, before a significant simulation instance is implemented, a conceptual model of the subject to be simulated is developed. The conceptual model serves to establish the fundamental components and relationships that will be modeled in the simulation. As such, it precedes the detailed design and implementation phases, and constitutes the foundation on which simulation instances are developed. Consequently, correctness of actual simulation exercises requires the correctness of the conceptual models based on which the simulations are developed. That is, conceptual model correctness is a necessary condition for the correctness of any simulation.

In this thesis we will deal with the verification of dynamic descriptions in conceptual models. As our aim is to contribute to the validity of simulations through increasing the validity of a fundamental artifact in simulation development, the conceptual model, we will briefly discuss what a conceptual model is and how it contributes to simulation development in the following section. In the second section we will describe the research problem and afterwards we will present our approach briefly.

1.1 Verification and Validation of Conceptual Models

The dictionary definition of simulation [3] is “the imitative representation of the functioning of one system or process by means of the functioning of another”. A computer model is formed that represents functioning of the real system where the specific aspects of the system are simulated at a specific level of detail. In complex simulations there can be several processes, entities behaving deterministically or randomly, and users interacting with the simulation. It is hard if not impossible to check every execution for correctness for such systems. So the strategy for developing a valid

simulation is generally to try as much as possible to find errors, to perform as much verification and validation activity as feasible during the development process, and build credibility in an iterative way into each work product before building the final executable simulation.

The development of simulation systems is not a simple task. Simulation software development activities involve many parties which assume different roles such as the user, sponsor, developer, verification agent, where each role may be assumed by more than one organization. Simulation software is typically not standalone any more; it may consist of a number of communicating components, such as federates participating in a federation [5]. Such components may either be reused to develop other simulations or all or parts of simulations can be integrated with others to form new simulations. So the development can also be shared among different parties based on components.

Among the actors involved in simulation, there usually is a gap in understanding; while the users emphasize the problem to be solved and the requirements of the simulation, the developers deal with how to solve the problem, that is, the design.

During development of a simulation application, the primary concern is the correct description of the domain elements represented, which is introduced as the conceptual model. It describes the domain thoroughly and is agreed by both developers and users. It is a document that describes not only the elements in the domain but also the behavior of these elements.

The conceptual model is defined by DoD Modeling Simulation Glossary [4] as “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. They are simulation-neutral views of those entities, actions, and interactions occurring in the real world”. Usually to attain a correct representation, which is called the conceptual model, that imitates the behavior of the real life process or system, an adequate representation of the system or process has to be developed that defines the system behavior in a common language, that could be understood by both the user and the developer. A conceptual model describes the real world components and their interactions therefore it tries to fill the gap between the requirements and design.

The conceptual model is the first detailed product that provides information on the domain. To determine the adequacy of conceptual models for the intended purpose, several properties have been defined for conceptual models, such as completeness, depth, level of detail, consistency, accuracy, performance [9].

The conceptual model is related to verification in two distinct ways as explained by Sargent [2], first, a conceptual model can be used to determine simulation capability, for instance to detect if simulation can be used in a case where no test data available. Secondly it is used for testing the capabilities of the model, by testing simulation system

against the descriptions in the model. Furthermore, it ensures that model components and their relations are consistent and complete.

The importance of conceptual modeling is apparent as it is the first, highest level abstraction that is agreed upon by simulation user and developer. The conceptual model shall thoroughly represent the domain elements and their actions and interactions. All the design and development activity is based on the conceptual model. So finding errors in the conceptual modeling phase is crucial as finding errors in both requirements and design in a software development project. Yet complete verification and validation during the conceptual phase is not done practically, where verification of the dynamics is usually the most neglected aspect.

The conceptual model is a bridge between the user and the developer, and software requirements and design. Approaches dealing with verification of only requirements or design are not adequate. Verification is not straightforward as the descriptions provide limited information on the dynamic definitions as the model is conceptual as implied by its name. It is usually presented in a scientific paper format and most of the information is not in a structured format. So even if there is need for increasing the validity of a conceptual model which affects the validity of a simulation, there is a lack of specific methods and tools for verification of conceptual models.

A recent work has defined a metamodel based notation, KAMA [29], for conceptual models. The metamodel based notation has its own merits. It provides a structural approach and it tries to support satisfaction of the desired properties of conceptual models. Even if KAMA notation is more structural, behavior definitions are not executable, and as such, are not suitable for analysis. Furthermore, there is a need to analyze the way the KAMA metamodel describes the dynamics, constrains the behavior and instantiation of the elements representing the behavior.

As KAMA notation focuses on modeling of conceptual information, therefore the abstract behavior descriptions related to precedence, synchronization, branching, and hierarchy relations between tasks are vague. More specific relations can be relations attached to entities, defining states of the entities and transitions. However the relations between the tasks which show similarity with UML activity descriptions are more characteristic of the conceptual models as the case studies using KAMA show. There is no verification method for dynamics that can be used for verification of such models. Such an approach will reduce the postponed effort spent for finding and diagnosing the errors of dynamic nature in simulation system that propagated through development process.

Related to this problem, several other problems exist. Conceptual modeling is done usually not for execution but for communication. A set of models exist where each model shall be verified concerning their essence without loss of information. So verification of dynamics in conceptual models must be done without the loss of any information in the conceptual model. That is, verification of conceptual model dynamics must depend upon the information represented in the conceptual model.

Conceptual models can not be interpreted easily and executed so that dynamic verification techniques can be easily applied. A detailed analysis is needed to understand the nature of information in these models and their interpretation for execution. Many elements that affect the dynamics are used in conceptual models yet the semantics of these elements are not defined precisely similar to operational semantics definition for computer programs. For instance in KAMA notation, views such as *mission space* and *task flow diagram* describe task precedence, hierarchy, synchronization and branching yet it is usually not clear if these descriptions are satisfactory for interpretation of dynamics.

Even if the information in a conceptual model can be used for verification, how do we define the criteria for verification? Another problem arises here: how the models should be characterized to satisfy certain criteria based on the nature of information available for dynamic verification and validation. The variability of information in conceptual models and the variability in application of the verification techniques complicate this task.

1.2 The Research Problem

The research problem, then, to be tackled in this study can be stated concisely as follows: developing a formal verification approach for dynamic properties in conceptual models utilizing the existing information gathered without the concern for precise behavior description. Two conflicting concerns exist, to preserve the informal semantics in conceptual models and increase the credibility through dynamic verification. So the approach shall not simply leave out the existing notation or dynamic implications.

This requires description of the semantics, examination of the variations in behavior descriptions enriching the semantics while adhering to the original conceptual model content and semantics, increasing the credibility of conceptual models through formal verification, which are developed adhering to metamodel based approach.

1.3 The Proposed Approach

To increase the validity of conceptual descriptions, we develop a dynamic verification methodology for metamodel based conceptual models. The dynamic verification approach also utilizes the relation between tasks and the other properties such as related roles, work products and objectives.

Conceptual models are early products for clarifying domain entities and simulation features. The capability of classical requirements engineering seems to be lacking for representing the elements to be simulated in adequate detail. On the other hand executable modeling techniques tend to be developer oriented and hard to be comprehended by the domain expert. These may be even either inapplicable or very costly to apply for large systems.

The primary objective in developing a conceptual model is not representing the dynamic behavior so that it can be used for execution. In fact dynamic behavior descriptions are usually not thorough in the conceptual modeling phase, the classifiers may not exist. Although a set of techniques addressing the same problem have been recommended for conceptual models which will be discussed in Chapter 2, no application to real case studies is available. In this thesis we aim to provide an approach for checking the dynamics of conceptual models. As part of this approach we also provide the criteria related to validity of conceptual models.

As a first step, the nature of dynamic behavior in models, emphasizing the relation between tasks and other elements is analyzed and characterized. By formalizing the characteristics, we try to verify the descriptions of precedence relations between tasks, model elements related to synchronization and decision, as well as decompositions. The techniques for checking dynamic descriptions are developed using formal approaches. Traditionally these techniques are used for checking more formal and executable models.

UML based notations exploit the benefit of being structured without alienating the domain expert therefore can provide an interim solution for representing conceptual models. As an example the KAMA approach consists of a process and accompanying metamodel oriented conceptual modeling framework based on MOF [26] and UML [27]. Relevance of UML is threefold for KAMA, utilization of similar diagrammatic notation, OCL for expression of constraints and UML foundation package as the core of KAMA notation.

The KAMA approach has been tested for many real life case studies and evidence is available for its usefulness and effectiveness for conceptual modeling. One of the compromises of the approach is that the domain model can not be executed so the dynamic features cannot be verified.

The dynamic verification approach proposed in the present study is developed and tested using the models developed in KAMA. This may also be done in similar UML based approaches which do not aim to precisely describe the execution of a simulation. We consider conceptual models represented exclusively with a UML based notation.

Model execution is defined by marking the dynamic elements, defining their execution lifecycles, and their interaction with other model elements. The uses of this approach are twofold: first is verification of the model and the second is providing a base for executable model formulation.

In this work we propose an approach for verification of dynamics of conceptual models of the mission space. In our approach we check the structural properties related to dynamics and analyze the dynamics of the diagrams for possible errors. This technique will allow early verification of dynamic descriptions of conceptual models of the mission space.

1.4 Contributions of the Thesis

The following major contributions are made through this work:

- KAMA semantics are formally defined and a notation is introduced for conceptual modeling of C4ISR systems.
- An enriched model called KAMA-DV is developed for behavioral descriptions. This model can be used for both model execution and analysis.
- Based on the KAMA-DV notation, structural properties related to dynamics are verified and dynamics are analyzed by tracing feasible execution paths.

The following further contributions can be relevant for other studies in the field:

- A process is proposed for validation of behavioral descriptions in conceptual models, including characterization of dynamics, structural checking and dynamic analysis.
- The soundness definition for EPCs is applied to conceptual model verification.
- The dynamic verification approach is applied on two case studies that deal with real world conceptual models.

1.5 Organization of the Thesis

In the second chapter we discuss the current literature on conceptual modeling, KAMA approach, verification approaches for conceptual models and other similar conceptual descriptions. In the third chapter we review KAMA mission space package definition, focusing on *task flow diagrams* including the mathematical description of the rules that the elements of this package obeys and the relevance of the KAMA mission space package definition for our verification approach. In Chapter 4, we list the properties and limitations of dynamic definitions of conceptual models and provide example *task flow diagrams* presenting the semantic definitions and variances. In Chapter 5, we list dynamic verification requirements for conceptual models enriched with model elements to enable verification of dynamics, and classes of conceptual models. In Chapter 6, we present the formal and semiformal approach for verification of the models and software tools that is used for enabling formal verification. In Chapter 7, we present two case studies where our approach is applied to real life conceptual models. Finally in Chapter 8, we compare our approach with the past work and conclude our study, suggesting directions and items for future work.

CHAPTER 2

CONCEPTUAL MODELS AND VERIFICATION

In this chapter we will review the approaches for conceptual modeling and verification. In the first section we will discuss past conceptual modeling approaches. In the second section we will investigate KAMA and various verification approaches. KAMA has a structured notation based on a metamodel derived from UML, it stands out providing a potential for application of novel dynamic verification techniques. As formulating the dynamic techniques is not straightforward and these techniques will be built upon considering the relevance and adequacy of past studies on dynamic verification of other conceptual descriptions, in the third section we will list the literature on similar techniques used for UML activity models and conceptual business process descriptions.

2.1 Interpretation of Conceptual Modeling for Simulations

In this section, the definition of conceptual models in modeling and simulation, the differences in scope of conceptual models, the approaches dealing with conceptual modeling, and methods for verification of conceptual models will be reviewed.

2.1.1 Conceptual Model Definitions

Conceptual model has been defined in different ways by various sources. The most cited and used definitions are provided in Table 2-1. Dale Pace who authored several articles on conceptual models, used the term simulation conceptual model for the early developed model describing domain elements and simulation features that can be both understandable by the developer and user [1]. DoD M&S Glossary[4] (very similar definitions exist in VV&A RPG[9], IEEE 1516.3 [6] and other sources) define conceptual model as “an abstraction from either the existing or a notional physical world that serves as a frame of reference for further simulation development by documenting simulation-independent views of important entities and their key actions and interactions”. According to VV&A RPG, IEEE 1516.3 and Brade [23] conceptual modeling stage follows the determination of intended use. Balci [20] states that conceptual modeling is the first stage in simulation development that is subject to evaluation (verification and validation) activities. Conceptual models are also classified as domain models (conceptual models of the mission space) and simulation conceptual models, former

having no information on the simulation functions and later include description of simulation elements [21]. Conceptual models can be expressed in various formats such as scientific paper, structured model or UML model [1][9][23].

Table 2-1. Various Definitions of Conceptual Models

Concept	Definition Source	Definition
Conceptual Models of the Mission Space	DoD Modeling and Simulation Glossary [4]	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. They are simulation-neutral views of those entities, actions, and interactions occurring in the real world.
Simulation Conceptual Model	DoD VV&A RPG v3.0 [9]	A simulation conceptual model is the simulation developer's way of translating modeling requirements (i. e., what is to be represented by the simulation) into a detailed design framework (i. e., how it is to be done), from which the software, hardware, networks (in the case of distributed simulation), and systems/equipment that will make up the simulation can be built
	SISO PDG / SG SCM FINAL REPORT	A simulation conceptual model is an abstraction from either the existing or a notional physical world that serves as a frame of reference for further simulation development by documenting simulation-independent views of important entities and their key actions and interactions. A simulation conceptual model describes what the simulation will represent, the assumptions limiting those representations, and other capabilities needed to satisfy the stakeholder's requirements. It bridges between these requirements, and simulation design.
	Brade [23]	The Conceptual Model describes the abstracted and idealized representation of the real system and holds all concepts of the model, i.e., its decomposition into interacting subsystems, the representation of properties of interest in the form of attributes, the degree of abstraction and idealization, and the rationale and reasoning that led to the chosen representation of the real system in the language of the model's application domain.
Federation Conceptual Model	IEEE Std 1516.3-2003 [7]	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. The federation conceptual model 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.

2.1.2 Conceptual Model Scope

Among conceptual models, two kinds of models can be identified, models that are dealing with the problem domain, leaving out the details on simulation and simulation conceptual models. In this subsection the characteristics of domain models, simulation conceptual models, and federation conceptual models are discussed to understand the nature of our dynamic verification problem.

Domain Conceptual Models: Domain models include abstract representations of the domain. They are supposed to be problem independent but a slightest effort to form representation of a real world object will be done based on implicit assumptions. To arrive at precise domain models, the domain shall be sufficiently restricted by operational constraints (legislations, organizational rules, etc.). Conceptual models of the mission space [6] or functional description of the mission space as the new name can be seen as an effort to form a representation of a complex real domain where the organizational constraints are loose and systems are evolving rapidly.

Simulation Conceptual Models: Simulation conceptual models are problem specific models. When dealing with physical reality formulating a problem greatly reduces complexity, leaving away the details of unnecessarily parts of the reality and enabling more focused and detailed look at the perceived part. Still a model therefore simulation of reality is not possible without any distortion. So when forming simulation conceptual models the assumptions about the concepts should be presented. In a simulation conceptual model the concepts of simulation shall be presented.

Yet another property of a simulation conceptual model is the inclusion of simulation control features such as pause and restart characteristics, data and control capabilities and the method to enter control messages [1]. These features do not have any real world counterpart and contribute achieving simulation objectives therefore their existence is also critical for simulation objectives and therefore validation of conceptual models. The relation between FDMS which stand for conceptual models of the mission space and conceptual model of the system (a more general term used for simulation conceptual models) is not clear.

Federation Conceptual Models: A third kind of conceptual model that can be encountered in the literature is the federation conceptual model. In fact federation conceptual model is the term for simulation conceptual model that is used in IEEE standard for federation development and execution process [7]. Despite the name has design flavor, considering simulation neutral clause in definition and the description and alignment of the activity in the development process, it is more requirements oriented then simulation conceptual model.

In this work, we primarily deal with domain conceptual models. Domain conceptual models emphasize the structural aspects, and are less concerned about behavioral aspects. The behavioral representations are more abstract in domain conceptual models, the defined behavior may belong to a set of different objects, the relations between behaviors may depend on more general context, and definition of behaviors may be partial.

2.1.3 Conceptual Model Development Methods

In this section we will discuss methods used for conceptual modeling. To qualify as a conceptual modeling method in our discussion, an approach has to specify the conceptual modeling process and an approach for representation of the model or at least guidance for development must be provided.

CMMS project is the first project dealing with solely conceptual modeling, explaining the need and infrastructure requirements [6]. Similarly VV&A RPG [9] provides the definition of conceptual model, guidance on the contents and format without any examples on details and examples. In IEEE 1516.3 standard [7], conceptual modeling is discussed as a step in development of federated simulations. Base object models (BOM) [10] approach provides a method for developing simulation conceptual models and design of a federation by providing definitions for reusable object models. The behavior representations of base object models are traceable to HLA objects and interactions. Other approaches provide also definition for conceptual modeling step, Brade's generalized process for V&V [23] and REVVA [14].

2.1.3.1 Conceptual Models of Mission Space (CMMS)

The work for the conceptual models of the mission space project was started in 1995 by US Defense modeling and simulation office (DMSO). The high cost of simulation development in US military was regarded as a problem and a solution based on using reusable and interoperable simulation components was proposed. CMMS project [6] was started for building the consistent world view these simulation components will be based on. These representations of real world will be agreed both by the user and developer, and shared and contributed by every project. Content differences of already developed conceptual models, the differences in understanding of the domain caused by different sources, terms, and format limited the reuse of conceptual models. The reuse of simulation components was also limited because of the difficulty of understanding existing conceptual descriptions of simulations.

The proposed solution was to develop an implementation independent conceptual model that serves as a base for development of the simulation. The CMMS technical framework will include definitions of terms, content, structure, process, and infrastructure for creation, distribution and management of models. This is only enabled by common syntax and semantics. The differences between agencies shall be resolved by using common syntax and semantics.

Another problem of CMMS was acquiring the existing models for further reuse, for which data interface format (DIF) was to be defined. The technical framework will involve functionality of transforming each existing format to DIF. If the conceptual models do not conform to the description of the CMMS, only the implementation independent components will be extracted as a conceptual model.

To support reuse and interoperation, formation of repository of models which will be shared among different agencies is desired. Conceptual models of existing simulations

will be transferred into the repository, by extracting only the information that is not specific to simulation implementation. The acquired knowledge from the existing efforts will be extracted by adequately authorized and extracted by using fully structured views.

The CMMS project was no longer mentioned during 2000s. Functional description of mission space (FDMS) is mentioned as the continuation of the CMMS framework.

2.1.3.2 VV&A Recommended Practices Guide (VV&A RPG) Special Topic: Conceptual Model Development and Validation

VV&A RPG [9] provides a special chapter for conceptual model development. By merging previous work from many authors and CMMS project and FDMS project are merged to form a general guidance on conceptual model development. The definition of conceptual model emphasizes the role as a bridge both between requirements and specifications, and developers and users.

Three key components of conceptual models are defined, simulation context representing the authoritative information on relevant entities, processes, data etc., simulation concept, the developers understanding of the simulation context. Simulation concept can further decomposed as mission space, the representations of real elements (simulation elements) and simulation space, the elements concerning to the functionality of simulation. These definitions are inherited from Pace.

VV&A RPG also lists the key items that should be present in a conceptual model which include the validation history in addition to key components. The process of development of the conceptual model and applications are listed. An important role of conceptual model related to validation is highlighted; it is used to determine the appropriateness of using the simulation in untested cases, which can occur because of the complexity and size of simulation data.

The VV&RPG underlines SME review as the normal form of conceptual model validation. The validation is performed to determine the adequacy simulation elements' representative capability for intended purpose and simulation concept for overall capability and the appropriateness of constraints and boundary conditions introduced on the simulation concept by the simulation context. Formal methods are mentioned as a way of validation for safety critical simulation applications.

2.1.3.3 Conceptual Modeling Step in FEDEP

IEEE Recommended Practice for HLA Federation Development and Execution Process (FEDEP) [7] is the recommended practice for development and execution of HLA compliant simulations. It is a high-level framework that can be tailored based on the specific development needs. One of the main step of the process “Perform conceptual analysis”, which is broken down into three steps, “Develop Scenario”, “Develop Federation Conceptual Model”, and “Develop Federation Requirements”.

The FEDEP activities “Develop Scenario” and “Develop Federation Conceptual Model” are both relevant to conceptual modeling of the domain. The scenario should include the description of major types and number of entities and functional descriptions of their capabilities, behaviors and relationships over time, and the relevant environmental conditions. While the federation conceptual model is the implementation independent federation objectives are transformed into a representation format to be used by federation design and development.

FEDEP [7] process itself leaves out details about VV&A to accompanying standard VV&A overlay for FEDEP. The VV&A overlay provides a set of VV&A activities for verification of conceptual models.

As both are high level frameworks and focus on the process, neither FEDEP nor VV&A overlay mention specific methods and provide guidance. In this work we provide verification approach for dynamics of conceptual models in a domain specific language KAMA.

2.1.3.4 Base Object Models (BOMs)

Base object models (BOMs)[10] have been introduced for conceptual modeling and federation design to further increase reuse of models and simulations. Base object model includes two pieces and their connections, a conceptual model, consisting of conceptual entities and events and their counterparts as HLA objects and interactions, attributes and parameters. In terms of VV&A RPG terminology, the conceptual description includes the mission space elements and the HLA object model form the simulation space.

Through connecting these pieces the simulation conceptual model developer can formulate a conceptual model from existing BOMs. The planned development includes a developer to develop conceptual model using higher level of abstraction from BOMs containing conceptual entities and events while corresponding simulation is designed and formed by reuse of existing HLA compliant simulations. BOM is to be supported with tools, so that all the BOMs will be stored in a library with upload, download, integration use history and search capability [10].

Upon inspection one can observe that very little is mentioned about verification and validation in BOM standard [10] and guidance documents [12]. In the BOM standard, it is stated that in the development process, BOM can be used to validate semantic composibility and verify the simulation and results without citation of any particular method.

BOMs are stored in XML format that conforms to BOM schema [10]. So any BOM syntax can be checked against this schema, which can be regarded as syntax check. BOMs are oriented for HLA development and typical use scenario involves a broad range of already developed BOMs are available during development so that by integrating pieces the user can easily bring together not only conceptual model elements but also

simulation components. However the feasibility of this approach is yet to be observed in practice. A recent approach concentrates on semantic composibility of BOMs [11].

2.1.3.5 Defence Conceptual Modeling Framework (DCMF)

DCMF was introduced by Swedish Defence Research Agency as the continuation of the CMMS project [13]. Conceptual modeling process is seen as a series of activities such as knowledge acquisition, knowledge representation, modeling, and use[13]. During knowledge modeling a structured ontology is developed that is equivalent of a conceptual model. According to DCMF specification[13], the ontology can be transformed into Activity diagrams, sequence diagrams, EPCs, Petri nets and BPMN diagrams during knowledge use. Also it is argued that the knowledge use phase can produce results that can be used to verify the simulation results.

Although DCMF underlines the need for VV&A and formalization in knowledge modeling in general and conceptual modeling in particular, there is not an explicit mention of methods and techniques for conceptual model verification and validation. DCMF relies on rules defined for objects and Business Process Modeling Notation (BPMN) for dynamic behavior description. A particular point relevant in DCMF specification is that there is a mention of checking instance BPMNs against conceptual BPMNs, although particular approach is not explicitly mentioned.

2.1.3.6 Other Approaches

In this section we will discuss the conceptual modeling phase in two related methodologies: Brade's V&V process [23] and REVVA [14].

Brade[23] presents a generalized process for simulation development that emphasizes verification and validation at each step. As a case study a sample tailoring of the process is also provided in the same work. Two kinds of models are identified: the conceptual model (also called the communicative model) and the formal model. The definition of conceptual model for Brade can be found in Table 2-1. The formal model represents all the behavior of the system in a formal notation. Whether it is possible to develop formal model or not for every case is not discussed by Brade, but for the case study provided it is stated that formulation and use of a formal model requires highly skilled effort.

The work also distinguishes between internal V&V which deals with V&V activities performed by considering the single intermediate work product, and transformation V&V performed using previously developed work products. For internal V&V activities of the conceptual model, the following methods are listed: form checking, consistency checking, unit dimension test, mental execution, comparison with other models. The enablers of documentation correctness and internal consistency are stated as checklists, forms, consistency matrices, modeling formalisms. According to Brade, with appropriate tools that use a central database the former three enablers may be supported. Modeling formalisms are supported by formal frameworks applied for formal and executable

models. Among these Brade cites propositional linear temporal logic (PLTL), Discrete Event Specification (DEVS), Simulation Language with eXtensibility (SLX).

Human review is performed at various phases in the sample application. In case of a conceptual model it is used for symbolic description review and mental execution. Each submodel is mentally executed to determine whether it behaves as desired and consistently with the available domain knowledge during conceptual phase. The results of the mental execution are reported by the domain expert performing the mental execution.

The emphasis of Brade's work is on the analysis of the formal model which encapsulates full model structure and behavior, can be pursued more conveniently with classical techniques such as syntax and semantic checking, control and data flow analysis, model checking and testing. The aim is to check the formal model which is expressed in a paradigm such as DEVS and later using it in the development of executable model preserving all syntax and semantics. As the case study by Brade shows, complete verification of formal model may be also impossible [23]. That happens even though Brade uses a case that is a part of a traffic modeling domain where the entity types and interactions are limited. In a more general case description of formal model may also be infeasible. It is also stated that most of the simulation development projects do not have a formal model.

REVVA [14] approach emphasizes acceptability criteria in simulation development and verification and validation. Similar to Brade, conceptual model is considered to be a base for formal model development. According to REVVA correctness and suitability of executable model depends on correctness and suitability of the underlying conceptual model. The correctness of the conceptual model has two major components: internal consistency of the conceptual model, and consistency with available real system knowledge. The most important property is appropriateness of the chosen abstraction and idealization within the context of the intended use. This is done by inspection of the conceptual model in order to determine that it satisfies the acceptability criteria [15].

2.1.4 Conceptual Model Verification

We will mention about the works that discussed the verification and validation concepts. From the software oriented perspective verification is a much more focused function that is concerned with a single phase of development, whereas the validation deals with the evaluation of the requirements from the satisfaction of requirements or intended use. In the simulation development, verification is defined based on the adequacy of the representation of the implementation of the conceptual model, which captures the requirements and the intended use and simulation validation is defined based on the adequacy of the model as a representation of real world from the perspective of the intended use.

Table 2-2. The Definition of V&V from the Standpoint of Software and Simulation Development

Concept	IEEE Std 1012-2004	DoD RPG v3
Verification	(A) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.	The process of determining that a model implementation and its associated data accurately represent the developer's conceptual description and specifications.
Validation	(A) The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. (B) The process of providing evidence that the software and its associated products satisfy system requirements allocated to software at the end of each life cycle activity, solve the right problem (e.g., correctly model physical laws, implement business rules, use the proper system assumptions), and satisfy intended use and user needs.	The process of determining the degree to which a model and its associated data provide an accurate representation of the real world from the perspective of the intended uses of the model.

The relation between the verification and validation and conceptual model is first mentioned by Sargent[2]. As verification deals with whether a simulation works according to its specification whereas validation deals with the working of the simulation for its intended purpose. As described by the Sargent[2] Triangle depicting the modeling process the conceptual model is validated against the real world, however simulation is verified based on the conceptual model. So the conceptual model acts as bridge between the real world and the model implementation and serves as a product which can be utilized for verification of implementations. A validated conceptual model coupled with verification activities will contribute to the validation of the implementation.

More broad view of verification and validation is introduced by Brade[23]. According to Brade simulation credibility can be built by a series of V&V activities each product in the phase can be evaluated by itself or in comparison to other products. According to this vies verification and validation can be done at each phase. Brade mentions the activities of verification and validation and do not discriminate two types of activities.

As there may be more than one product in development of the simulation, there may be more than one model based on the development process as in the case of Brade. So each model can be verified based on the products developed in the same phased, or validated concerning the requirements form the more general perspective, the overall user requirements, conceptual model, or intended use. So verification is an effort based on

increasing the self and comparative consistency, validation deals with contribution to the developing the right product.

Barlas [16] discusses the model verification and validation in general and application of them to system dynamics models. The validation means usefulness for some purpose. The usefulness of the purpose is also relevant which can hardly be determined and this is non-technical, informal and, qualitative process. Also validation is distributed and prolonged activity. Even in the case of formal validation of system dynamic models, the validation spans more than one stage, a high quality model can be formalized with less effort. Barlas also states that from the perspective of philosophy of science, there is not a purely objective/ formal way of “theory confirmation” even for the natural science”. More on the philosophical aspects of the discussion can be seen in Barlas and his references related to philosophy of science. The main two properties of validation as relativistic and holistic perspective theory confirmation, validation is a confidence building activity. The internal components of the model play crucial role, and informal and subjective aspects always exist. The formal quantitative tests always play important role as inputs to the process. However statistical significance tests are limited in determining the validity of a model by themselves.

Kleijnen [17] accepts the verification and validation definition by Law and Kelton, and assuming a perfect program is present, the verification can be performed. However a model is always limited in its expression of the reality so only can be good enough rather than perfect. So the author highlights that validation deals with the correctness of the conceptual model representation. In this work also methods for verification of simulation outputs such as testing, sensitivity analysis, risk analysis, and white box testing are discussed. A point that is relevant for this study is that, white box testing provides opportunity to compare the system with real life system as the transparency provides one to one correspondence between simulation and real data.

The Sandia report[18] also aims to define verification and validation concepts and survey applications of these in computational science and engineering and computational social science. Verification and validation has been applied more in disciplines where modeling and simulation software has been developed to automate and support human decision making. The standards, frameworks, regular approaches for verification and validation are not existent and studies in this field concentrate on application of one method for a particular problem. They also state that “the science of performing experimental-computational comparisons in “computational science” remains immature”. As a method they identify that includes development of the “Phenomenon Identification and Ranking Chart (PIR)”. In their particular application they provide a method for quantification of simulation errors compared to referent. The Sandia report states that the models are not necessarily computational; they can be physical, narrative, mathematical, logical, or graphical as in the case of UML models or conceptual models of KAMA. Also the report states that model verification is not separable from model building and validation and confidence is built iteratively. There is no verified and validated code, the confidence of

software depends on the intended purpose, on which most of the verification effort is based.

The C4ISR systems described in conceptual models are systems where system and decision makers are coupled. However during conceptual phase the intended purpose is to provide an agreed and reusable description of the mission space. From a wider perspective of simulation development, the verified conceptual model later can be used for definition of other products such as assessment standards and PIR. Our study aims to leverage confidence of conceptual models of C4ISR environment and ensure that the conceptual model represents the real world as much as possible.

Adak et. al.[19] presents a metamodel for HLA federations. The methodology involves checking of certain properties. Certain properties are represented in the metamodel to limit the incorrect behavior. The code generator provides capabilities to check the dynamic aspects based on the design. This work deals with later stages of development where federation design and implementation is performed.

For conceptual models that exclude simulation specific elements, and focus on the mission space the priority is describing the concepts of the domain. This priority forces the conceptual model to be understandable rather than formal, thus in practice the validity is based on SME review as explained in various sources [9] [22]. Conceptual model is the first product that defines the problem thoroughly and provides simulation constraints and assumptions, the internal verification and validation based on self-consistency is more important. SME review can focus on static or dynamic concepts and relations of the model. Usually SME will have to put significant effort to trace dynamic definitions as they are stated partially and organized as text document format.

VV&RPG [9] and Brade [23] list techniques for conceptual model verification and validation as both consider conceptual modeling a major phase of simulation development. However there is no concrete example on how these techniques can be applied in different types of conceptual models. Balcı [20] names three distinct evaluations for conceptual modeling stage: evaluation of conceptual model quality, conceptual modeling process, and M&S project characteristics.

Despite the need for more credible models that will led to more credible simulations, formal techniques have had limited use in conceptual modeling, and even rarer use in large scale models. This occurs as a result of limited scaling capacity of these methods and advanced level of expertise required [1]. Brade suggests a five step process, where modeling is divided into three steps: conceptual, formal and executable modeling [23]. Even with such division both development of the formal model, verification of the model, and the transformation of formal model to executable model are not straightforward and the transformation may even be infeasible as in the case of Brade [23].

The methods listed for conceptual model verification by VV&A RPG are shown in Table 2-3.

Table 2-3. Verification and Validation Methods for Conceptual Model Listed in VV&A RPG

Class	V&V Technique	KAMA Tool Support
Informal	Audit	Aids Audit
Static	Calling structure analysis	-
Static	Cause-effect graphing	-
Dynamic	Comparison test	-
Static	Control flow analysis	Supports checking of errors related to control flow
Static	Data dependency analysis	-
Static	Data flow analysis	Supports data flow analysis through checking of related errors
Informal	Desk check	-
Informal	Documentation check	Aids documentation check
Informal	Face validation	Aids face validation
Formal	Induction	-
Formal	Inductive Assertions	-
Formal	Inference	-
Informal	Inspection	Aids inspection
Formal	Lambda calculus	-
Formal	Logical Deduction	-
Static	Model Interface Analysis	-
Formal	Predicate calculus	-
Formal	Predicate transformation	-
Formal	Proof of correctness	-
Informal	Review	Aids review
Static	State transition diagram	Aids drawing of state diagram
Static	Structural analysis	Checking of structural errors.
Static	Traceability assessment	Supports traceability to original referent
Dynamic	User interface analysis	-
Informal	Walkthroughs	Aids walkthrough

According to our knowledge, experience reports on application of verification techniques to conceptual models are limited, there is hardly any study dealing with verification of the dynamics of these models.

More broadly the validity of models is treated for system dynamics models as explained by Barlas [16].

2.2 KAMA: A Metamodel based approach for Conceptual Modeling

KAMA conceptual modeling approach [29] consists of a process description[30] and a notation, as well as a tool [35] supporting both. The specific aspect is that the notation that is used by KAMA is expressed as a Meta Object Facility (MOF) based metamodel. KAMA is developed considering the main properties of conceptual models and past approaches which are not based on metamodel.

From the viewpoint of Model Driven Architecture (MDA) [25], a conceptual model resembles a computation independent model that provides information about the problem domain independent of any computational approach, which is called the mission space in military domain. Because of their nature, conceptual models are not executable. They shall be utilized as requirements to be used in formulating either more detailed simulation conceptual models or the simulation design.

A way of developing conceptual models by utilizing metamodels, KAMA approach [29], is to use a notation based on user concepts by extending model elements taken from a subset of UML [27]. It provides the SME the domain concepts for modeling, yet is structured enough to enable reuse of the model in later stages and online processing. This supports MDA [25] vision, aiming to transform software development to iterative development of models. Other advantages are multiple views provided to the user, which is mandatory for conceptual model development and a more general process definition accompanying the approach [30] on how to acquire and structure information.

The static verification techniques can be integrated into the model development environment and enforced during or after development of the model. Rules can be defined at various levels and consistency checking of these rules may provide user different kinds of errors [37] in different levels [34]. For KAMA environment possible static verification possibilities related to above and alternative techniques have been provided by Tanrıover and Bilgen [36][37]. A process for verification is defined that also utilizes these techniques and applied the process in a case study for a real conceptual model. They assert that formal techniques can be checked for given precise metamodel and a number of non-trivial errors can be discovered. Similar study is more focused on checking errors in activity diagrams. These studies motivate using comprehensive process definitions and novel techniques that may enable finding problems in early development.

2.2.1 Evaluation of KAMA as a Conceptual Modeling Approach

First reason we adapt KAMA for dynamic properties is that KAMA provides a novel approach for describing dynamic properties. As we will show in Chapter 4, model elements and diagrams can be interpreted in several different ways, and the defined

behavior is not executable. Also other elements that affect the behaviour exist such as *workProducts*. So the interpretation of behavior and checking is an interesting for KAMA behavior descriptions as they are conceptual rather than executable.

One of the drawbacks in conceptual modeling literature is the lack of real life case studies and models. There is very little number of studies that concerns real conceptual modeling. The importance of conceptual model is known for a decade, most of the simulations developed before do not have a conceptual model. The most used format as a scientific based format and FDMS and are more recent approaches. Even if such models are developed, the models are not easily accessible as they may contain sensitive information and commercial value. The major advantage of KAMA as a conceptual modeling language is the large number of case studies and experimental studies.

Use of KAMA in a number of case studies has been reported so far as Karagoz and Tanriover presents. Two among these case studies were industrial sized projects while other three have been experimental studies. The availability of such developed conceptual models has made KAMA the most suitable candidate for the application of our methodology.

It is reported that the researchers in the KAMA project team performed three experimental studies, which aimed to apply the KAMA notation in different settings. The subjects of the experimental studies were “Squad Fire and Battle Marksmanship Training Simulator”[31], “Small-scaled Military Unit Movement – Infiltration Task” [33], and “Synthetic Environment Simulation System” [32] respectively. All of the researchers found the KAMA notation easy to learn and apply. These works also stressed some drawbacks related to the need for representing relations among the work products, additional relationships between *role* and *task* model elements such as “assists”, distinctly representing the “equipment” required for a task, and additional mechanism for representing the geographical constraints.

2.3 Verification Approaches for Dynamic Properties

In this study we aim to examine the dynamic properties of conceptual models for modeling and simulation. We deal with models that are developed at the early stages in the simulation development process. The modeled concepts are real world missions, tasks, related roles, objectives, and work products. We have grouped relevant past research into three, which we will mention in order of relevancy. First, metamodel based conceptual modeling in general, and the KAMA approach in particular provides the main pillar of our research. Secondly, research dealing with semantic variations of UML activity models and verification of such models is relevant in the sense that the semantics of model elements for sequencing of *tasks* in KAMA show similarities with their UML counterparts. However, differences in size of models and semantic variations and the limitations of past work in that area justify a novel approach. Finally, verification of conceptual business process models, particularly the ones expressed by Enhanced Process Chains (EPCs) are relevant for our research. We provide an adaptation method for verification of KAMA mission space models using the research results of EPC

verification. In the next three subsections we will examine the relation of our research with these three related areas in detail.

2.3.1 Verification Methodologies for Metamodel Based Conceptual Models

The developed conceptual modeling notation of KAMA provides the modeling elements for these but lacks methods and tools for analysis and verification of the dynamics. Tanriover and Bilgen [36][38] provide an inspection approach for conceptual models developed in a domain specific notation. The approach includes checklists for interdiagram issues and for each diagram type. The checks are listed for *initialTasks*, *synchronizationPoints* (fork and join type), *decisionPoints* (fork and merge type), flow to the *finalTask*, loops and *workProducts* based on the soundness property of workflow nets. In this work we deal with both structure and dynamics of conceptual models and elaborate the diagrams describing dynamic behavior thoroughly. We define extended properties for some of the model elements and define structured conceptual models, and used the existing soundness definition for EPCs for verification of conceptual models. We use these definitions to develop our approach based on formal verification. The *task flow diagram* inspection steps and corresponding aspects that are dealt in this work are listed in Section 8.2.3.

In this work we focus on the mission space package of the KAMA metamodel, which includes elements to describe real world missions, tasks and their relations. The mission space is presented as a structural view in KAMA metamodel, however it provides information on sequencing, synchronization, and branching of tasks which can be used in understanding the behavior of the system. We describe an approach including a process and set of methods to be used in inspection of dynamics of these models. The inspection process will be iterative and will provide user with online and offline verification methods. For increasing the power of analysis we extend mission space models to cover intended but not explicitly specified behavior descriptions. Our mission space verification is based on this extended model. The relationship between metamodels is described in Figure 2-1. These extensions which do not exist in original KAMA metamodel, are defined based on dynamic properties and is used for the definition of KAMA Dynamic Verification (KAMA-DV) metamodel. We utilize approaches that were used for verification of Enhanced Process Chain Diagrams (EPCs) based on Petri nets during this process. As a result we aim to have a conceptual model that has consistent and correct behavioral information.

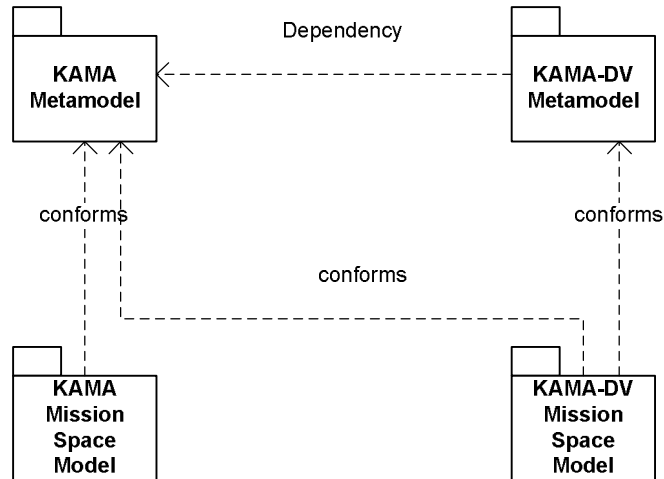


Figure 2-1. Relationship between Metamodels and Models

2.3.2 Verification of UML *Activity* Descriptions

As KAMA conceptual modeling notation extends some of the elements used in UML activity package [27], work on semantics and verification of UML activity package is relevant for our research. We will briefly discuss relevant works in this subsection.

Several reasons exist not to utilize UML activity package as is for conceptual modeling and dynamic verification. The first is that UML activity package have a number of complex constructs that have ability to model programming languages, exceptions, etc. that are not suitable for conceptual modeling. The studies aiming to define semantics for UML models focus on characteristic modeling elements of activity package namely *FundamentalActivities* and *IntermediateActivities*. A second drawback is that UML activity has Petri net based semantics and that is directly applicable for dynamics of conceptual models. The semantic variations exist in UML activity package model elements, only some of which are relevant for conceptual models.

Also UML activity package is an evolving modeling approach which has potential to be applied widely. For instance a potential application area described in the specification is business process modeling if the classifiers are not determined for models. The mission space package of KAMA notation has similarities with UML activity package in that respect. It has model elements mostly derived from UML activity and leaves out model elements such as *callOperationAction*, *message*, and other programming language related constructs. The modeling elements of KAMA represent traditional concepts used for defining conceptual models of the domain and are named similarly.

UML activity representation is considered to be a complex method to use properly because of semantic issues, variety of constructs and wide scope [44] [46]. It may be used for modeling for various purposes, ranging from workflow models where activities do not have any classifier, or operations of classes. Despite the broad scope, Fowler argues that

it is not frequently used in practice [40]. For instance the executable UML literature on developing executable UML models includes a dozen of books, where activity model is left out generally. It may be argued that the desire of the practitioners to arrive at more executable and easily analyzable models make activity less attractive. In our work we have utilized the modeling elements in KAMA that are derived from activity modeling. We will list works aiming to analyze activity models in the following paragraphs.

Drusisnky [45] provides examples of requirement specifications that can be formulated using activity diagrams. This work deals with conceptual diagrams expressed in UML 1.4. During activity scheduling he provides example constraints on multiple scheduling of the activities, the plurality of objects, execution times of loops, reentrance which are relevant for our case. For showing the different interpretations in activity diagrams, examples are discussed roughly. For each case additional requirements are advised, but no specific methodology to exist to inspect the dynamics. Our method tries to deal with similar specifications in two ways, a more clear and iterative behavior modeling and verification methodology.

The previous work on semantics for activity models is relevant for our research as we also aim to describe precise semantics for description of conceptual models in KAMA notation. The details of semantics of model elements are described by Bock in a series of papers. However other researchers aiming to simulate activity models still have difficulty in defining precise semantics [44]. Saarstedt [50] provided a method based on labeled transition system and Storrlé [46] used colored Petri nets to simulate the activity diagrams. For this purpose they also listed the variations encountered and their way of interpretation for semantically loose points [71][47]. Former used some assumptions to deal with some imprecise points in order provide automatic mechanism for simulation. Storrlé provides separate methods for simulation of control flow [48] and data flow [46]. Also these works deal with only a subset of activity model rather than whole. For instance none of these works provide a method for verification of elements in *completeActivity* package.

In the UML activity package [27], explicit relations with objectives, work products and roles do not exist. A metamodel based on UML, SPEM [28] provides couple of relations as observed in software development models. To our knowledge there is no work that specifically deals with verification of models expressed in SPEM.

There is also a relationship between *mission space diagram* and *task flow diagrams* similar to use case and activity diagrams. Smilaek et. al.[52] state that the behavioral interpretation of UML use case *extends* and *includes* associations are not clear. KAMA *mission space diagram* provides a more specific mechanism that affects the sequencing of tasks, therefore the dynamics of conceptual models. We have utilized the method to form complete *task flows diagrams* representing entire model using *mission space diagrams*. The included and extended missions are transformed to corresponding *task flow diagrams*. For this purpose we developed an alternative approach based on utilizing *extension points* and *includes* association.

All these differences make the use of UML activity execution approaches limited and unusable for verification of conceptual model dynamics which is provided by our novel approach. For conceptual models the timing, multiplicity and context relations are different from UML activity model. These are explicit in the KAMA metamodel specification and there exist a number of semantic variations and extensions as we will further discuss in Chapter 4. These aspects shall be parts of a dynamic verification approach that will be described in Chapter 6.

2.3.3 Verification of Conceptual Business Process Descriptions

Related areas relevant to research on conceptual model verification include business process modeling, workflow management systems, and verification of these systems using Petri nets. As we are talking about “conceptual” models, where the term conceptual describes the abstract nature of the model, works dealing with conceptual aspects business process modeling and their verification are most relevant to our study. On the other hand, workflow management systems use notations that are more execution oriented. There is a wide literature on methods, techniques and tools to define and analyze workflows. Most of this literature is barely relevant for us as they deal with mostly executable workflows that include more concrete and executable domain specific modeling elements.

Unlike the specific nature of workflow management systems, more conceptual Enhanced Process Chain (EPC) diagrams, the diagrams used to describe business processes in ARIS toolset [56], are dealt in the literature. A specific characteristic of works dealing with business process verification on conceptual level is that they treat semantic variations while others try to limit the description language, EPC, to arrive at easily analyzable descriptions without semantic variations. In parallel with the MDA vision, each model shall be adequate for its purpose and creating a complete model which captures all execution semantics formally is neither possible nor the ideal way in earlier stages of development. A conceptual model shall provide a coherent but abstract description, not pretending to be executable, but supports development. Having the semantic interpretation differences that will be listed in Chapter 4, approaches treating them are relevant for us.

The most common approach to verify EPCs is similar to works dealing with workflows [60]. The approach based on utilization of a special class of Petri nets, called workflow nets [53] is used for verification of business process models and workflow models. First the business process specification is transformed to a Petri net based specification, a workflow net. This Petri net based specification is checked for certain properties by creating the all execution sequences. Transformations and reductions are possible in this stage to deal with the state space explosion problem. Additional information on initial states, final states and invariants can be used during the process.

Researchers in business process and workflow verification have provided definitions for the soundness for these models. For verification of conceptual business process models,

relaxed soundness property is used that involves each process is executed in at least one valid execution of the process [57][64]. Satisfaction of these criteria is checked by using the semantic constraints of the nodes used for sequencing of processes in the models, namely OR, AND, XOR nodes that are used for splitting and joining the control flow.

There exist similarity between model elements used for sequencing in the UML activity package and EPCs, but considering the variations in their semantics, there is a need for research using models dealing with real applications. UML activity is examined in terms of adequacy for workflow modeling [54] and a similar work provides equivalency of some of the model elements in UML Activity and EPCs [63]. EPCs are used to describe conceptual business processes and have a number of semantic variations. The original EPC semantics is not described in detail and further works provide semantic definitions for some of the loose points. UML activity has much broader scope, including elements ranging more basic constructs to more complex. More basic constructs also have some semantic variances as discussed in previous subsection. As we will show in Chapter 4, the KAMA language has also semantic variances. In our work we have used the analysis techniques for EPC analysis for the case KAMA conceptual modeling diagrams. We also provide a comparison of aspects in conceptual models with those of EPC and UML 2 activity in next subsection.

2.3.4 Comparison of Approaches for Verification of Dynamic Properties of UML Based Conceptual Models

The equivalency of related elements of EPCs and UML Activity model and these nodes is summarized in Table 2-4 where previous mappings [55][61] are also included. In addition, we have appended the table with equivalent elements of the KAMA mission space package which are described in Chapter 3.

Table 2-4. The Elements Used in EPC Diagrams and their Counterparts

EPC	UML Activity	KAMA
OR-Split	N/A	SynchronizationPoint (Fork Type)
XOR-Split	DecisionNode	DecisionPoint
AND-Split	ForkNode	SynchronizationPoint (Fork Type)
OR-Join	N/A	SynchronizationPoint (Join Type)
XOR-Join	MergeNode	DecisionPoint (Merge)
AND-Join	JoinNode	SynchronizationPoint (Join Type)
Function	By Activity and Action Model Elements	Task Hierarchy and “ConsistOf” Relation
Cancellation Region Extensions	ActivityFinalNode	FinalTask

During comparison with UML activity, we only consider basic constructs, given in UML *FundamentalActivities* and *IntermediateActivities* packages, as they include more abstract modeling elements suited for conceptual modeling. If UML activity package is taken as a whole, for instance for the first aspect, *CallBehaviourAction* in an activity can provide a mechanism to trigger other activities during execution. For EPC, we used the definition provided by Mendling [61].

We do not limit the models to include only elements either of EPC diagrams or UML Activity Package. Even if such modeling can be possible for some of the cases, this will decrease the understandability and communicative power of conceptual model and will contradict with the properties of a conceptual model which will be discussed in Section 4.1. For the analysis of control flow of KAMA-DV models for some of the cases, equivalent EPC can be obtained by using the equivalent elements in Table 2-4. In the Chapter 6 and Chapter 7 we illustrate the soundness analysis applied in KAMA-DV.

CHAPTER 3

KAMA SEMANTICS

In this chapter we will define the KAMA elements and rules in terms of set theory. Later in Chapter 5, we will provide definitions for executable forms of KAMA elements based on these definitions, to verify dynamical aspects of KAMA models. Not only does the formalism introduced in this chapter aim to provide a basis for verification of dynamics as described in Chapter 5, but also it is expected to form a basis for any future work that may attempt at augmenting the KAMA approach.

In the following table the UML metamodel elements related to KAMA metamodel and KAMA mission space package elements are listed. The model element names are preserved as given in the KAMA metamodel to preserve consistency and traceability.

In what follows, we have used italics to indicate terms used in a formal sense as opposed to natural language usage. For instance, “mission” would indicate the ordinary meaning of the term whereas “*mission*” would indicate a specific KAMA model element. In general, we have defined sets (marked as S) for each kind of model element that is not derived from *association*. For *associations*, we define sets of ordered pairs (marked as O.P. in the table) which are uniquely identified by the source and target model elements. *Attributes* and *AssociationEnds* are also defined as attributes so they are assigned to *model elements* and *associations*, and therefore do not qualify as members of sets (marked as attribute in the table). These definitions are provided in the following subsection. Following these definitions, we have defined the *attributes* for elements of these sets in Section 3.2. In Section 3.3, we define the constraints provided in KAMA in textual and OCL form in terms of set theory. *Attributes* in terms of lists based on their *associations* exist for *mission*, *task*, *role* and *objective* model elements in KAMA metamodel, therefore we have included the definitions for such lists as also sets which are provided in Section 3.4.

Table 3-1. UML Metamodel Elements Related to KAMA and KAMA Mission Space Package Model Elements

KAMA related UML Metamodel Elements	Represented As	Included	KAMA Metamodel Mission Space Package Elements	Represented As	Included
element		-	model element	Set	✓
named element		-	mission	Set	✓
feature		-	task	Set	✓
namespace		-	role	Set	✓
classifier		-	objective	Set	✓
class		-	measure	Set	✓
operation		-	workProduct	Set	✓
attribute	Attribute	✓	decisionPoint	Set	✓
package		-	synchronizationPoint	Set	✓
generalization	Set of O.P [*]	✓	initialTask	Set	✓
dependency	Set of O.P [*]	✓	finalTask	Set	✓
association	Set of O.P [*]	✓	responsibleFor	Set	✓
associationEnd	Attribute	✓	realizes	Set of O.P [*]	✓
stateMachine		-	extends	Set of O.P [*]	✓
state		-	includes	Set of O.P [*]	✓
pseudoState		-	achieves	Set of O.P [*]	✓
vertex		-	taskFlow	Set of O.P [*]	✓
transition		-	inputTo	Set of O.P [*]	✓
activity		-	produces	Set of O.P [*]	✓
activityNode		-	quantifiedBy	Set of O.P [*]	✓

O.P: Ordered Pairs

In defining formal properties for model elements we have divided the model elements presented in a *task flow diagram* into five. The core of *task flow diagram* includes the model elements for modeling basic elements such as *mission* and *tasks*. The sequencing package has elements used for sequencing of tasks. The other three supporting packages are named based on their central elements as *role*, *workProduct* and *objective* packages. The *role* package includes the *role*, *actor* model elements, *responsibleFor* and *realizes* associations. The *workProduct* package includes *workProduct* model element, *inputTo* and *produces* associations. The *objective* package includes *objective* and *measure* elements, and *achieves* and *measures* associations. Finally the *mission space complete* package has rules for organizing the other elements and completely defining the *task* hierarchy.

3.1 Sets of Model Elements

When defining sets of model elements, we chose to use single letter for model elements that do not derive from association and double letter for model elements derived from association. For each association element we use the set of ordered pairs (x,y) in which the first element (x) represents the source element in the association and the second (y) denotes the target element.

In this context we have divided the KAMA mission space package into three subpackages. *Mission Space Fundamental Package* includes the *mission* and *task* elements. *Sequencing Package* includes the elements for sequencing of *tasks*. *Role Package* contains elements for defining roles and actors, *Workproducts Package* contains elements that are used to define *workProduct* associations, and *Objective Package* contains elements related with objectives of tasks and their measures.

While *mission* element of *Mission Space Fundamental* package is derived from *activity* of UML *FundamentalActivities* package, *task* element of same package is derived from both UML *activity* and *activityNode* of the same UML package. *Mission Space Sequencing* package *taskFlow* element is derived from *controlFlow*, elements *initialTask*, *finalTask*, *synchronizationPoint* and *decisionPoint* are derived from *controlNode* of UML *IntermediateActivities* package. The packages of mission space and related packages of UML are shown in Figure 3-1.

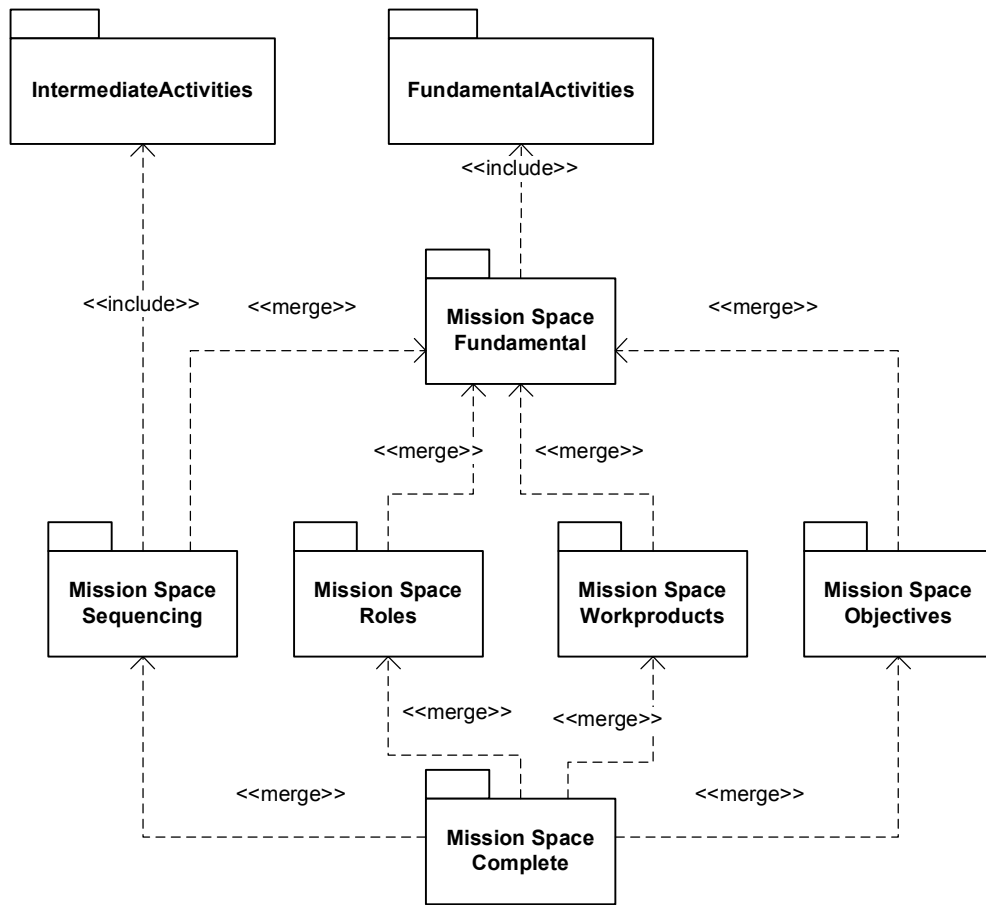


Figure 3-1. Dependencies between Mission Space Packages

3.1.1 *Mission Space Fundamental Package Elements*

E is the set of all model elements. In the current definition *attributes* and *associationEnds* are taken as *attributes* of model elements are not elements of the set.

Mission and *Task* are the basic elements for defining conceptual model dynamics.

M is the set of all *Mission* elements in a model.

T is the set of all *Task* elements in a model.

The *task* model elements are connected with associations of type *taskFlow*. Also *consistOf* relation between *tasks* exists for defining *task* hierarchy. These elements are included in the *Mission Space Sequencing* package.

AS is the set of all model elements that are derived from *association* metaclass in a model.

IN is the set of all *includes* associations. If there is an *association* derived from *includes* metaclass between $m_i \in M$ and $m_j \in M$ then $(m_i, m_j) \in IN$

GE is the set of all *generalizations*. If there is a *generalization* between $e_i \in E$ and $e_j \in E$ then $(e_i, e_j) \in GE$.

CO is the set of all *consistOf* associations between a *mission* and a *task* or two *tasks*. If there is an *association* derived from *consistOf* between $t_i \in M \cup T$ and $t_j \in T$ then $(t_i, t_j) \in CO$

3.1.2 Mission Space Sequencing Package Elements

In fundamental elements package we have the elements to define the scheduling of *tasks*.

TF is the set of all *taskFlow* elements in a model. If there is an *association* derived from *taskFlow* between $t_i \in T$ and $t_j \in T$ then $(t_i, t_j) \in TF$

Other than these core elements in *taskFlow* diagrams, other elements exist for defining more complex relationships between *tasks*.

I is the set of all *initialTask* elements in a model.

F is the set of all *finalTask* elements in a model.

D is the set of all *decisionPoint* elements in a model.

S is the set of all *synchronizationPoint* elements in a model.

N is the set of all of all model elements that are derived from *intermediateNode* elements in a model. The intermediate node is the node for describing causal relations between *tasks* which include *initialTask*, *finalTask*, *decisionPoint*, *synchronizationPoint* so $N = I \cup F \cup D \cup S$.

We extend the definition of CO , so that it can be defined between a *task* with an *intermediateNode* or a *taskFlow*. If there is an *association* derived from *consistOf* between $t_i \in M \cup T$ and $t_j \in T \cup N \cup TF$ then $(t_i, t_j) \in CO$. By this way we the *intermediateNodes* and *taskFlows* defined for sequencing become part of the description of the *task*.

E is the set of all elements in a conceptual model related with dynamics shown in Table 3-1.

3.1.3 Elements of Other Packages

3.1.3.1 Roles Package Elements

R is the set of all *role* model elements in a model.

A is the set of all *actor* model elements in a model.

RF is the set of all *responsibleFor* model elements in a model. If there is a *responsibleFor* association defined between $r_i \in R$ and $t_j \in T$ then $(r_i, t_j) \in RF$

RE is the set of all *realizes* model elements in a model. If there is a *realizes* association defined between $r_i \in R$ and $t_j \in T$ then $(r_i, t_j) \in RE$

OW is the set of all *owns* model elements in a model. If there is a *owns* association defined between $a_i \in T$ and $r_j \in R$ then $(a_i, r_j) \in OW$

3.1.3.2 Workproducts Package Elements

W is the set of all *workProducts* model elements in a model.

IT is the set of all *inputTo* associations in a model. If there is an *InputTo* association defined between $t_i \in T$ and $w_j \in W$ then $(w_j, t_i) \in IT$

PR is the set of all *produces* associations in a model. If there is an *Produces* association defined between $t_i \in T$ and $w_j \in W$ then $(t_i, w_j) \in PR$

3.1.3.3 Objectives Package Elements

O is the set of all *objective* model elements in a model.

U is the set of all *measure* model elements in a model.

AC is the set of all *achieves* associations in a model. If there is an *achieves* association defined between $t_i \in T$ and $o_j \in O$ then $(t_i, o_j) \in AC$

ME is the set of all *measures* associations in a model. If there is an *measures* association defined between $o_i \in O$ and $u_j \in U$ then $(o_i, u_j) \in ME$

3.1.3.4 Mission Space Complete

Following definitions are made to make diagrammatic display possible.

Let TD_i be the group of model elements that is part of i^{th} *taskFlow diagram* in a model. So the contents of a *taskFlow diagram* can be defined as an intersection of model elements presented in the diagram;

$$TD_i = (T_i \cup I_i \cup F_i \cup D_i \cup S_i \cup TF_i) \cup \\ (R_i \cup RE_i \cup RF_i) \cup (W_i \cup IT_i \cup PR_i) \cup (O_i \cup U_i \cup AC_i \cup ME_i) \\ \text{where } T_i \subset T, I_i \subset I, F_i \subset F, D_i \subset D, S_i \subset S, TF_i \subset TF, \\ R_i \subset R, RE_i \subset RE, RF_i \subset RF, W_i \subset W, IT_i \subset IT, PR_i \subset PR, \\ O_i \subset O, U_i \subset U, AC_i \subset AC, ME_i \subset ME)$$

A more precise definition of contents of *task flow diagram* can be done by using *consistOf* association.

$$\forall t_i \in T((t_i, t_k) \in CO \text{ implies } t_k \in TD_i) \\ \forall m_j \in M((m_j, t_k) \in CO \text{ implies } t_k \in TD_j)$$

For the other elements in a *task flow diagram*, we include the contents associated with included tasks.

$$\forall t_i \in T(((t_i, t_k) \in CO, ((t_k, x) \in (TF \cup PR \cup AC) \text{ or } \\ (t_k, x) \in (TF \cup RE \cup RF \cup IT)))) \text{ implies } x \in TD_i)$$

Finally we include the *measure* model elements associated with the *objective* model elements associated with *tasks*.

$$\forall t_i \in T(((t_i, t_k) \in CO, (t_k, m) \in AC, (m, u) \in ME) \text{ implies } u \in TD_i)$$

It must be noted that in this definition we assume that a *task* is strictly associated with a single *task flow diagram*. If a *task* has more than interpretation, that is used in more than one *task flow diagram* and each of these diagrams consists of the same set of *tasks* model elements (according to KAMA semantics) but different associations between them then we shall define the *taskFlow* accordingly. In this case all the associations are specific in the context of the single *task* which is presented in the specific diagram.

$$\forall t_i \in T(((t_i, t_k) \in CO, ((t_k, x) \in (TF_i \cup PR_i \cup AC_i) \text{ or } (x, t_k) \in (TF_i \cup RE_i \cup RF_i \cup IT_i)))) \text{ implies } x \in TD_i,$$

$$TF_i \subset TF, PR_i \subset PR, AC_i \subset AC, RE_i \subset RE, RF_i \subset RF, IT_i \subset IT)$$

In fact the definition of KAMA metamodel enforces only one *taskFlow* entering and leaving a *task*. In this case only other associations can vary. Although not defined in the metamodel a *task flow diagram* can have interpretations for different *missions* and *tasks*.

$$\forall t_i \in T(((t_i, t_k) \in CO, ((t_k, x) \in (TF \cup PR_i \cup AC_i) \text{ or } (x, t_k) \in (TF \cup RE_i \cup RF_i \cup IT_i)))) \text{ implies } x \in TD_i)$$

In case the assumption that a *task* may consist of different *tasks* in different *task flow diagrams* is to be accepted, a specific piece of information shall exist in the metamodel providing this information or the information in the *task flow diagrams* can be used for this purpose. Then the *TD_i*s defined as in the first definition using information on *task flow diagrams* instead of *consistOf* association. Each *task flow diagram* shows some of the model elements and represents a *task*. Although this is not the case in KAMA metamodel we will introduce this extension for KAMA-DV in Chapter 6. In the remaining parts of this Chapter, we will deal with elements explained in original KAMA metamodel and will not include definitions for *task flow diagram* and related associations.

3.1.3.5 Complete List of Elements

For the sake of completeness, the model elements are listed in Table 3-2. In addition to their package the abbreviation which is used to represent the set of elements in this chapter, they are also used in the codes for the rules for the corresponding element in Chapter 4. For the remaining chapters, in the mathematical expressions these represent the set of these model elements in the models, in other parts they represent model elements.

Table 3-2. KAMA Mission Space Model Elements, their Abbreviations and Containing Packages

Model Element	Ab.	Package	Model Element	Ab.	Package
mission	M	Fundamental	role	R	Roles
task	T	Fundamental	actor	A	Roles
association	AS	Fundamental	responsibleFor	RF	Roles
includes	IN	Fundamental	realizes	RE	Roles
generalization	GE	Fundamental	owns	OW	Roles
consistOf	CO	Fundamental	workProduct	W	Workproducts
Mission Space Diag.	MD	Fundamental	inputTo	IT	Workproducts
taskFlow	TF	Sequencing	produces	PR	Workproducts
initialTask	I	Sequencing	objective	O	Objectives
finalTask	F	Sequencing	measure	M	Objectives
decisionPoint	D	Sequencing	achieves	AC	Objectives
synchronizationPoint	S	Sequencing	measures	ME	Objectives
intermediateNode	N	Sequencing	Task Flow Diagram	TD	Complete

3.2 Attributes

All model elements have defining attributes of $Id \in \mathbb{Z}^+$ and $Name \in String$ that represent the unique identification code of the element and name of the element respectively. Each *model element* also have *description*, *assumptions*, *constraints* and *geographical information attributes*. However the *mission*, *task*, *objective*, *measure* and *finalNode* are defined by more attributes as follows:

Each $m_i \in M$ is a quadruple $(Id, Name, Pre, Pos)$ where $Pre \in String$ stands for the precondition, $Pos \in String$ stand for the post condition. Most general *String* type is used for conditions, as the syntax for them are not defined further.

Each $t_i \in T$ is a 6-tuple $(Id, Name, Pre, Pos, isExtensionPoint, extensionPointId)$ where $Pre \in String$ stands for the precondition, $Pos \in String$ stand for the post condition, $isExtensionPoint \in Boolean$ stands for the *isExtensionPoint* attribute, $extensionPointId \in \mathbb{Z}^+$ is the id of the extension point.

Each $o_i \in O$ is a triple $(Id, Name, Per)$ where $Per \in String$ stands for the *performance criterion*.

Each $u_i \in U$ is a triple $(Id, Name, Unit)$ where $Unit \in String$ stands for the unit of the *measure*.

Each $f_i \in F$ is a triple $(Id, Name, Iss)$ where Iss stands for whether the *finalNode* represents successful completion or not.

Each $tf_i \in TF$ has attributes $(Id, Name, guard)$ where $guard \in String$ stands for the guard expression.

Other than the attributes listed above, *mission*, *task*, *role* and *objective* have some lists of elements as attributes according to KAMA. The definitions of these lists are given in subsection 4.

3.3 Constraints

In definition of the constraints we have used the terminology of the KAMA metamodel definition. The textual constraint definition is provided before the constraint definition in mathematical notation.

For model elements *mission*, *task*, *role* and *objective* the list of related elements are included as attributes. In fact these lists can be formed in a given implementation of method or tool and need not to be attributes of the classes. For example each mission has a list of roles that is connected to it with *realizes* or *responsibleFor* associations. We define such a list as follows:

Let $m_i \in M$ be associated with a *roleList* (RL_i) according to KAMA metamodel. For the members of this list following rule applies;

For any $r_j \in R$, if only if $(r_j, m_i) \in RE$ or $(r_j, m_i) \in RF$ then $r_j \in RL_i$.

Such lists are used optionally as a shorthand notation in the definition of some of the constraints. All the lists that exist in the KAMA metamodel are defined in subsection 4.

a. Mission Constraints

1. “A *mission* should be related with at least one role.”

$$\forall m_i \in M (\exists r_j \in R ((r_j, m_i) \in RE \text{ or } (r_j, m_i) \in RF))$$

2. “A *mission* should be related with at least one objective”

$$\forall m_i \in M (\exists o_j \in O ((m_i, o_j) \in AC))$$

3. “A *mission* may not have a relation of type *include* or *generalize* to itself”

$$(\forall m_i \in M ((m_i, m_i) \notin IN)) \text{ and } (\forall m_i \in M ((m_i, m_i) \notin GN))$$

b. Task Constraints

1. “A *task* should be related with at least one *role*”

$$\forall t_i \in T(\exists r_j \in R, ((r_j, t_i) \in RE \text{ or } (r_j, t_i) \in RF))$$

2. “A *task* should be related with at least one *objective*”

$$\forall t_i \in T(\exists o_j \in O((t_i, o_j) \in AC))$$

3. “A *task* has one incoming and one outgoing *taskFlow*”

$$(\forall t_i \in T, \exists t_j (t_i, t_j) \in TF) \text{ and}$$

$$(\forall t_i \in T, \exists t_j (t_j, t_i) \in TF) \text{ and}$$

$$(\forall t_i, t_j \in T((t_i, t_j) \in TF \text{ implies } \neg \exists t_k \in T(k \neq j, (t_i, t_k) \in TF))) \text{ and}$$

$$(\forall t_i, t_j \in T((t_i, t_j) \in TF \text{ implies } \neg \exists t_k \in T(k \neq i, (t_k, t_j) \in TF)))$$

c. Role Constraint

1. “A *role* should be owned by at least one *actor*”

$$\forall r_i \in R(\exists a_j \in A((a_j, r_i) \in OW))$$

d. Objective Constraint

1. “An *objective* should be related with at least one *measure*”

We note that a *measure* and *objective* can be related by only *measures* association.

$$\forall o_i \in O(\exists u_j \in U((o_i, u_j) \in ME))$$

e. Measure Constraint

1. “A *measure* should be related with at least one *objective*”

We note that a *measure* and *objective* can be related by only one *measures* association.

$$\forall u_i \in U(\exists o_j \in O((o_j, u_i) \in ME))$$

f. *WorkProduct* Constraint

1. “A *workProduct* does not have any *capability*¹.”

By definition of *workProduct* this is straightforward. We can further check that there is not any association between *workProduct* and *capability* where B denotes the set of *capability*.

$$\forall w_i \in W (\neg \exists b_j \in B ((w_i, b_j) \in AS))$$

g. *DecisionPoint* Constraints

1. “The connections coming into and going out of a *decisionPoint* must be *taskFlows*.”

By definition of *TF* set, this constraint holds.

2. “The guard conditions on multiple outgoing connections from a *decisionPoint* must not be the same.”

$$\forall d_i \in D (((d_i, t_j) \in TF, (d_i, t_k) \in TF, i \neq k) \text{ implies } (d_i, t_j).guard \neq (d_i, t_k).guard)$$

3. “Every outgoing connection from a *decisionPoint* must have a guard condition”

$$\forall d_i \in D ((d_i, t_j) \in TF \text{ implies } (d_i, t_j).guard \neq \emptyset)$$

h. *SynchronizationPoint* Constraints

1. “A *synchronizationPoint* having multiple incoming connections must have a single outgoing connection”

$$\forall s_i \in S (((t_j, s_i), (t_k, s_i) \in TF, j \neq k) \text{ implies } (s_i, t_l) \in TF \text{ for only one } l).$$

2. “A *synchronizationPoint* having multiple outgoing connections must have a single incoming connection”

$$\forall s_i \in S (((s_i, t_j), (s_i, t_k) \in TF, j \neq k) \text{ implies } (t_l, s_i) \in TF \text{ for only one } l).$$

¹ Capability is a model element that is not a member of mission space package so not in our scope.

3. “The connections coming into and going out of a *synchronizationPoint* must be *taskFlows*”

By definition of other associations and *taskFlow* elements this will hold. For curiosity we can add the following two constraints, the first for ensuring that any association coming into a *synchronizationPoint* is a *taskFlow* and no association which is not a *taskFlow* coming into a *synchronizationPoint* exist, the second is similar to first handling connections going out (“-” denotes set difference);

$$\forall s_i \in S((t_j, s_i) \in AS \text{ implies } (t_j, s_i) \in TF) \text{ and}$$

$$\forall s_i \in S((t_j, s_i) \in AS \text{ implies } (t_j, s_i) \notin (AS - TF))$$

$$\forall s_i \in S((s_i, t_j) \in AS \text{ implies } (s_i, t_j) \in TF) \text{ and}$$

$$\forall s_i \in S((s_i, t_j) \in AS \text{ implies } (s_i, t_j) \notin (AS - TF))$$

i. InitialTask Constraint

1. “An *initialTask* has no incoming connections.”

$$\forall i_j \in I((x, i_j) \notin TF).$$

j. FinalTask Constraint

1. “A *finalTask* has no outgoing connections.”

$$\forall f_i \in F((f_i, x) \notin TF).$$

k. ResponsibleFor Constraint

1. “A *responsibleFor* relation has one *role* as source and one *task* or *mission* as target”

$$\forall (x, y) \in RF(x \in R, y \in (T \cup M))$$

l. Realizes Constraint

1. “A *realizes* relation has one *role* as source and one *task* or *mission* as target”

$$\forall (x, y) \in RE(x \in R, y \in (T \cup M))$$

m. Extends Constraints

1. “An *extends* relation has one *mission* as source and one *mission* as target”

$$\forall(x, y) \in EX(x \in M, y \in M)$$

2. “An *extends* relation cannot have the same mission both as source and as target”

$$\forall(x, y) \in EX(x \neq y)$$

3. “Extended *mission* must have a related *task flow diagram* with at least one task whose *isExtensionPoint* attribute set to true”

$$\forall(x, y) \in EX(\exists z(z \in T \text{ and } z \in TD(x) \text{ and } z.isExtensionPoint = I))$$

n. Includes Constraints

1. “An *includes* relation has one *mission* as source and one *mission* as target”

$$\forall(x, y) \in IC(x \in M, y \in M)$$

2. “An *includes* relation cannot have the same *mission* both as source and as target”

$$\forall(x, y) \in IC(x \neq y)$$

o. Achieves Constraint

1. “An *achieves* relation has a *mission* or *task* as source and an *objective* as target”

$$\forall(x, y) \in AC(x \in (T \cup M), y \in O)$$

p. TaskFlow Constraints

1. “A *taskFlow* may have one of the {*task*, *decisionPoint*, *synchronizationPoint* and *initialTask*} as source and one of the {*task*, *decisionPoint*, *synchronizationPoint* and *finalTask*} as target”

$$\forall(x, y) \in TF(x \in (T \cup D \cup S \cup I), y \in (T \cup D \cup S \cup F))$$

2. “Only one *taskFlow* may exist between the same source and target”

As distinct elements of *TF*, *taskFlow* is defined uniquely as a set of ordered pairs, it is valid by definition.

q. *InputTo* Constraint

1. “An *inputTo* relation has one *workProduct* as source and one task as target”

$$\forall(x, y) \in IN(x \in W, y \in T)$$

r. *Produces* Constraint

1. “A *produces* relation has one *task* as source and one *workProduct* as target”

$$\forall(x, y) \in PR(x \in T, y \in W)$$

s. *QuantifiedBy* Constraint

1. “A *quantifiedBy* relation has one *objective* as source and one *measure* as target”

$$\forall(x, y) \in QB(x \in O, y \in U)$$

3.4 Derived Sets

In this section we define the elements for sets defined in the KAMA metamodel.

a. Derived Sets of *Mission*

For each *mission*(m_i), *inputList* (IL), *outputList* (OL), *objectiveList* (BL), *roleList* (RL) and *measureList* (UL) are defined as follows;

$$IL_i = \{w \mid w \in W, (w, m_i) \in IT\}$$

$$OL_i = \{w \mid w \in W, (w, m_i) \in PR\}$$

$$BL_i = \{o \mid o \in W, (m_i, o) \in AC\}$$

$$RL_i = \{r \mid r \in R, ((r, m_i) \in RF \text{ or } (r, m_i) \in RE)\}$$

$$UL_i = \{u \mid u \in U, \exists o_j ((m_i, o_j) \in AC \text{ and } (o_j, u) \in ME)\}$$

b. Derived Sets of *Task*

For each *task*(t_i), *inputList* (IL), *outputList* (OL), *objectiveList* (BL), *roleList* (RL) and *measureList* (UL) are defined as follows;

$$IL_i = \{w \mid w \in W, (w, t_i) \in IT\}$$

$$OL_i = \{w \mid w \in W, (w, t_i) \in PR\}$$

$$BL_i = \{o \mid o \in W, (t_i, o) \in AC\}$$

$$RL_i = \{r \mid r \in R, ((r, t_i) \in RF \text{ or } (r, t_i) \in RE)\}$$

$$UL_i = \{u \mid u \in U, \exists o_j ((t_i, o_j) \in AC \text{ and } (o_j, u) \in ME)\}$$

c. Derived Sets of *Role*

For each *rol*(r_i), e , *taskList* (TL) and *ownerList* (WL) are defined as follows;

$$TL_i = \{x \mid x \in (T \cup M), ((r_i, x) \in RF \text{ or } (r_i, x) \in RE)\}$$

$$WL_i = \{a \mid a \in A, (a, r_i) \in OW\}$$

d. Derived Sets of *Objective*

For each *objective*(o_i), *measureList* (UL) is defined as follows;

$$UL_i = \{u \mid u \in U, (o_i, u) \in QB\}$$

3.5 The Relationship of KAMA Metamodel Definition with Dynamic Verification

The formal definition of model will be used in KAMA in three ways as explained below.

Some of the rules form the basis of the dynamic behavior. These include the semantic definition of *tasks*, *taskFlows*, *decisionPoints*, *synchronizationPoints*, and *task* hierarchy. However the definitions of KAMA are limited in terms of dynamic execution as we will show in Chapter 4. In Chapter 5 we will list the needed extensions for the model elements related to task execution. In Chapter 6, we will describe the detailed execution mechanism. This will require extensions, model elements, properties and constraints in addition to described in this section. This more detailed model for dynamic verification is named as KAMA-DV.

The special relations includes and extends that are described in mission space diagrams are further utilized for synthesis of *task flow diagrams*. After synthesis the dynamic analysis will deal with *task flow diagrams*. We will explain this mechanism in section 5.

Others which relate to more static aspects of the model will be utilized as they should be valid during execution and simulation. These rules are listed here and are used during checking the consistency of each instance during execution.

CHAPTER 4

CONCEPTUAL MODEL DYNAMICS

In this section we will discuss the characteristics of dynamics of conceptual models. In the first subsection we will discuss general characteristics. In the following one, first the limitations that prevent formal verification of conceptual model dynamics in the light of KAMA conceptual models and then the required extensions for the modeling constructs, *task flow diagrams* and the related model elements are discussed. In the third subsection, we will examine parts of *mission space models* to extract rules that will guide the conceptual model execution process. In this analysis we handle simpler modeling elements first, other elements later.

4.1 Conceptual Model Dynamics Characteristics and Limitations

4.1.1 Characteristics of Conceptual Model Dynamics

In this subsection we will highlight basic dynamic characteristics of conceptual models. Dynamic behavior in conceptual model is described in two different ways, behavior in context of a mission or behavior in context of an entity. Behavior in context of a mission is described by *mission space package elements*, behavior in context of an entity is described by *state flow diagrams*.

While the state flow diagrams described in KAMA metamodel are simplified version of state flow diagrams of UML, *task flow diagrams* include variations and extensions from their counterparts described in *activity package* of UML. We have provided detailed information on structural properties of *mission space package* of KAMA in the previous section. In this section we will highlight the important dynamic characteristics of *task flow diagrams* which are used for description of sequencing of tasks.

CM.A1: A set of missions, tasks and other elements are described using a diagrammatic approach.

The core of the mission space package is the *task flow diagram* and model elements of this diagram. *Actor* is a model element that is part of *mission space package*, but relevant in dynamic behavior. The *mission* and *task* model elements form the core of a KAMA conceptual model. One of the earliest attempts for conceptual modeling, the definition of

CMMS [6], also includes mission and tasks concepts. The *mission* and *tasks* definitions are also easier to define as their world counterparts exist such as mission and task manuals and procedures. *Task flow diagrams* and *taskFlow* associations are used to describe sequencing of *tasks*. Other than these *synchronizationPoint* is used to provide concurrent task executions by branching and *decisionPoint* selective execution of tasks and merging of executions. The *mission* and *task* model elements are primary elements having real counterparts while other elements are used for sequencing. We will further elaborate the behavioral aspects during execution in the next section before defining the execution semantics.

CM.A2: Decomposition is possible for intermediate form of behavior representing units, *tasks* for the case of KAMA conceptual modeling notation.

Decomposition is possible for *tasks*. In KAMA each *task* have a *taskList* which are the constituents of the *task*. There exists *consistOf* relation between the *task* and subtasks. The constituent *tasks* may be further decomposed to form a *task* hierarchy. Unlike CMMS, the KAMA does not include an element for representing action, the lowest level of *task*. If the conceptual model aims to develop entity level behaviors than actions can be defined for the tasks in the most lower level which are not decomposed. Also KAMA uses *initialTask* and *finalTask* constructs to mark the start and end of the flow relations between tasks similar to UML. Each decomposition of a *task* requires information on sequencing of constituent *tasks*.

CM.A3: The upper or lower form of behavior representing unit can be identified within the context.

As described earlier there is no definition for action in KAMA, the lowest level *task* that is not decomposed. So the lowest level is determined by the modeler considering the specific intent for the conceptual model. Although decomposition starts from a *mission*, *mission* can be equivalent to a *task*, if included by or extended in another mission. Both *mission* and *action* depends on the context both of which can be treated as *task*.

CM.A4: A *task* and corresponding *task flow diagram* can either be global or describe part of the overall behavior.

The *task flow diagram* can be global being equivalent to *mission* or related to a single entity. Moreover a *task* may be defined concerning all entities of an *entity* type or may be relates one instance of such an *entity*. So the context of a *task* varies greatly which also affects the other constituent tasks.

CM.A5: A *task* can abort several other *tasks*.

In a dynamic context, a *task* can affect other *tasks* even if the means for these effects are not defined during conceptual modeling. For instance a *task* may cause changing state of an entity which in turn aborts its *tasks* or a *task* may simply abort other tasks in order to continue. These relations are not explicitly defined in conceptual models as the effects of

tasks on *entities* are not defined in formal manner. Task description is generally in free text format.

CM.A6: A *task* may consume or may not consume its inputs. Also it may produce or update its outputs.

The relation of a *task* with the *workProducts* that represent inputs and outputs is open to interpretation. For instance a *task* may continuously get an information update (for example position of a platform) or order a weapon to launch. In first case the information update may still be available while for the second case the missile is consumed.

4.1.2 Limitations of Conceptual Model Dynamics

C.M.L1: The relationship between tasks is not perfectly available.

In conceptual models the relations between tasks are not defined fully and thoroughly. In textual conceptual models, the composition of tasks into missions and decomposition of tasks may not include the information on sequence of tasks. Even if such sequence information is provided it may be unstructured.

C.M.L2: Information depends on the context. The context is not defined precisely.

Information is related to context in conceptual models. A *mission* or *task* definition includes a reference to a *role* or *workProduct*, but information on the *workProduct* is limited. Most of the information that contributes to conceptual model are in free text format and come from different sources. It is hard to fully acquire this knowledge and specify it in a structured format.

C.M.L3: Information is not present for some parts of the conceptual model.

Some information may be lacking for the *mission* and *tasks*. The lacking information may be about the sequencing, the *roles*, the *workProducts* or the *objectives*. The information may be either unavailable during conceptual model development, or may not be precisely known because of the aim of the simulation system which may represent a system that do not exist.

C.M.L4: The information on the sequence of tasks may not be specified rigorously.

The information on sequence of *tasks* may be provided in a very basic way. For instance the synchronization of *tasks* and decisions is not defined so the *task* decomposition may not include details on the sequencing of tasks. The operational concept may have hidden properties that are complex to explain and be structured. Moreover the mission space may aim to represent a virtual scenario which may not have existed so far. So the related entities (*roles*, *workProducts*, etc.) are defined in a limited way.

C.M.L5: The information on which actions to be performed during a task execution is specified in textual form and may not be executable.

This is the main limitation of conceptual models. The task descriptions are given in an unstructured textual form depending on the context rarely conforming to a formal syntax. The textual representation is both difficult to process and interpret.

4.2 The Context of Dynamics in KAMA Models

In this section the context of KAMA models will be discussed. Differences of KAMA *mission space models* described in the form of *task flow diagrams* with diagrammatic techniques used to describe similar information based on UML, UML Activity Package will be underlined. UML Activity is selected as a reference for comparison because of the similarities between KAMA notation and the similarities in scope. We will mention the difference of KAMA *task* and UML *activity*. As both definitions are used for rather liberal modeling without context we base our analysis on concrete examples from conceptual models. Then we will describe the task execution in case of conceptual models.

After these we will describe the way a conceptual model can define general sequencing of tasks during execution and differences and similarities of this with semantics of UML *activity* models. Finally the properties and rules that drive execution of conceptual models are listed for the related elements. These properties and rules are categorized into two, mandatory rules that shall hold for execution of model elements and optional ones that may be helpful and meaningful for specific cases.

4.2.1 Basic Tasks

The sequencing and causal relations of *tasks* are described by *taskFlow* elements derived from control flow. An example describing the causal relationship between *tasks* is shown in Figure 4-1, which is taken from a conceptual model for describing the fixed wing helicopter operation. The causal relationship between *tasks* is simple for this case, after the termination of the first *task* “Gather at Take off Point”, the second *task* can start execution. For the second case however the causal relation is not so straightforward. During execution of the first *task*, “Continue in Mission Route” the “Observe output of sensors” *task* can be triggered in fixed or variable intervals. In fact while “Continue in mission route” has not been terminated, a specific execution of “Observe output of sensors” *task* can be executed and terminated. The first *task* continues its execution even after the successor *task* has started execution. So the executing *task* may trigger the execution of the successor *task* more than once.

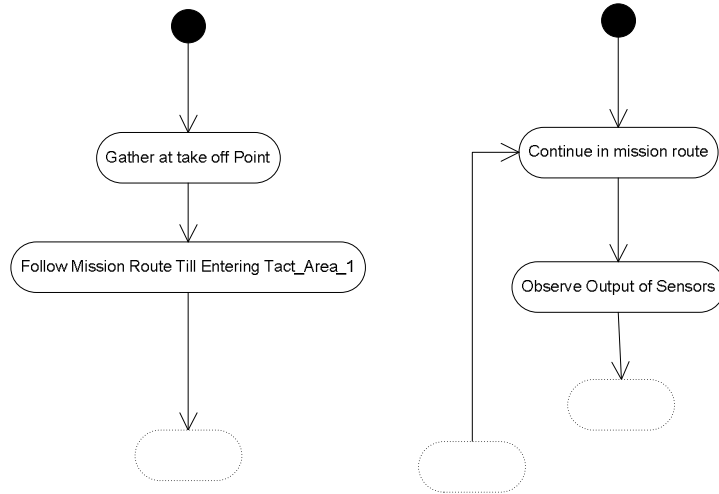


Figure 4-1 Two Task Flow Diagrams with Different Semantics for Task Execution

A similar issue can also exist, after the finishing of the predecessor *task*, the successor may not start. For this case, the reasons of not triggering execution of the successor *task* may be the termination status of the *task*, other elements between *task* executions (*synchronizationPoints* and *decisionPoints*) and unsatisfied guard condition of the *taskFlow* connecting two *tasks* and precondition of the successor *task*. The example of this type of occurrence can be seen Figure 4-2, where *Stop the Power Supply Task* may be delayed for some time after *Stop the Surveillance Equipment task* finishes.

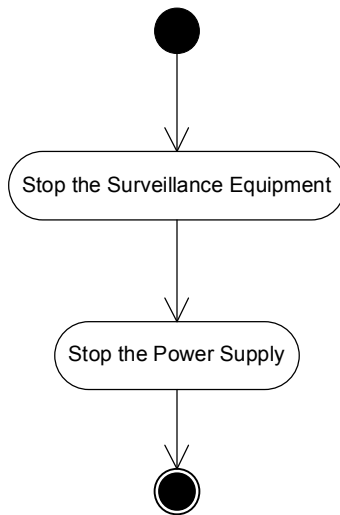


Figure 4-2. A Delayed Task Execution

If a *synchronizationPoint* of type join exists then the execution may be forced to wait for the other branch to terminate, this case will be explained in the following subsection. The case for unsuccessful completion of the *task* will also be explained in the next subsections.

Another important property of a *task* execution is whether a separate execution of *task* is created or not. While only one execution of *task* is possible for some of the *tasks*, a separate *task* execution may occur for the others. If a *task* execution is single then only one execution will exist at a time that handles all further executions. If not a separate execution will be created in each case. Two examples for each kind of *tasks* from conceptual models can be seen in Figure 4-3.

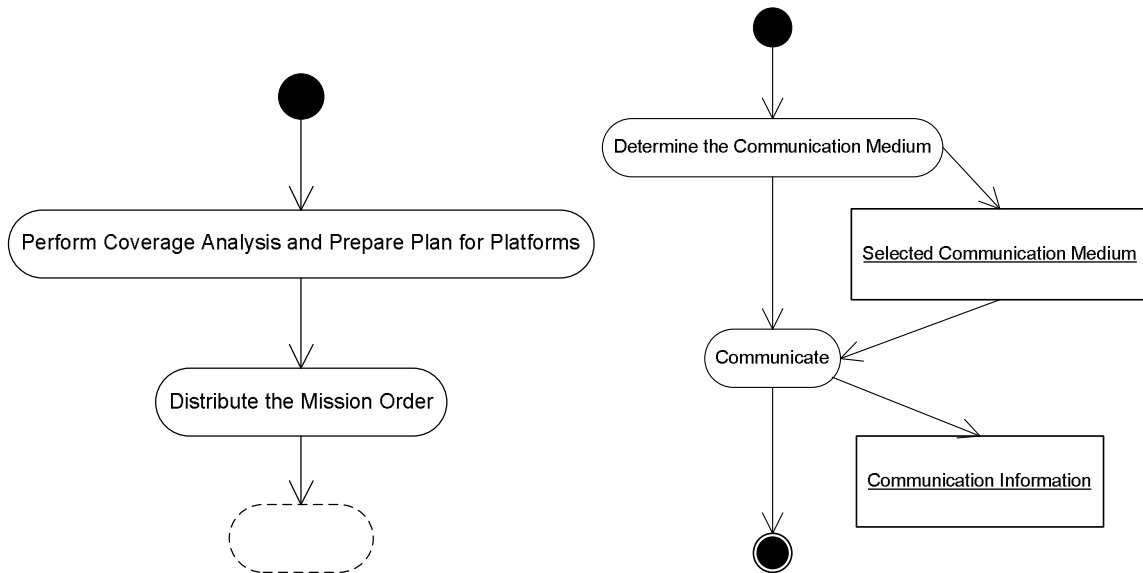


Figure 4-3 Task Flow Diagram Containing SingleExecution Task vs. a Task that is not SingleExecution

Both of the cases exist in conceptual models, creating a variation during execution of *tasks*. So this variation is one of the obstacles for automatic conceptual model execution. We have following rule related to *task* dynamics for *task* execution for the case, where no *consistOf* association, *synchronizationPoints*, and *decisionPoints* exist.

T-1: A *task* execution shall either be enabled during execution, in the time of termination or after termination of the predecessor *task*.

If a more restricted modeling approach was adopted during conceptual modeling, some of the variations may not be present. Then more restrictive optional rules can be used for management of *task* execution.

T-O-1: The *task* shall enable the successor *task* during the execution or after finishing execution or may not enable it.

T-O-2: The *task* shall enable the successor *task* at most once.

T-O-3: If the successor *task* is a *singleExecution task*, it shall not be enabled again if it is executing.

T-O-4: The enabled *task* shall start execution if all the preconditions are satisfied.

These rules are defined for the *mission space sequencing* package. The relation of *tasks* with other elements is explained in the following sections. As the examples show, for the simple case of relation of a predecessor and successor *task*, conceptual models show a lot of difference from semantics described by UML activity package. UML specification states that the semantics is based on Petri nets, so an activity is triggered if only if the all the inputs are available and consumes all the inputs at once and present outputs when another activity is ready to accept tokens. In case of conceptual models a *task* can trigger another *task* even if it had not finished execution and more than once.

4.2.2 The Hierarchy of Tasks

The hierarchy of *tasks* that occurs as a result of *consistOf* relation and *task flow diagrams* and introduces issues other than explained in the previous subsection. The *initialTask* and *finalTask* elements also used to mark start and end of other *tasks* contained *tasks*, which are connected with a *consistOf* relation with the upper level *task*. Each decomposition has own *task flow diagram* which contains *task* model elements that can be further decomposed.

KAMA decomposition of *tasks* differs from activity in UML 2.0. This difference can be seen from the metamodel definitions shown in Figure 4-4. In KAMA the hierarchy is maintained by *consistOf* relation between *tasks* while in UML the hierarchy is maintained by *activity*, *activityNode* and *action* model elements. So each activity is composed of nodes. As defined by the UML[27], “an *activityNode* is an abstract class for points in the flow of an activity connected by edges” so these points may be of *class nodes*, *controlNodes* and *objectNodes*. Another *activity* can be called by an *action* which is an *activityNode* of the *activity*. The behavior is not further decomposed in the *activity* but as described by the UML specification “a call behavior action may reference an activity definition, in which case the execution of the call action involves the execution of the referenced activity and its actions (similarly for all the invocation actions)”. So if activity is to be decomposed it is done by defining the *activityNodes* of the activity, then reference further *activity* elements by *call behavior action* of these *activityNodes*.

In KAMA a *task* is derived from both *activity* and *activityNode*. As such an activity that has *tasks* can be composed of other *tasks* by *consistOf* relation instead of *actions*. This is because of the initial aim of the KAMA *mission space package* description of the structure of the conceptual model.

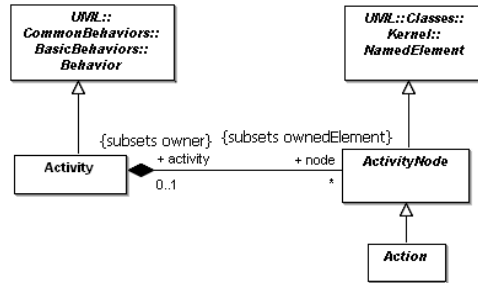


Figure 4-4 UML activity, activityNode and action Relations

The subtasks may be regarded as forming all the parent *task* or components of it. So an execution of subtasks may be started as soon as the execution of the parent *task* starts or during execution of the parent *task*. In the later case the execution of subtasks may start more than once during execution of the parent *task*.

The execution of upper or lower level *task* may determine the context. Context is determined by the upper level *task* which contains the other *tasks*. Based on this context *roles*, *workProducts*, *objectives* are effected which will be discussed in the following subsections.

The sequencing of contained *tasks* are described by a *task flow diagram*, the first node to be executed is the *initialTask*. The last node which does not have any outgoing connection is the *finalTask*.

In Figure 4-5, a *finalTask* execution that upon termination destroys all the other *tasks*' executions is shown. A counterexample can exist where a *finalTask* that will not abort execution of the other *tasks*. In UML 2.0[27] for clearly identifying the difference, two distinct elements named as *FlowFinalNode* and *ActivityFinalNode* exist.

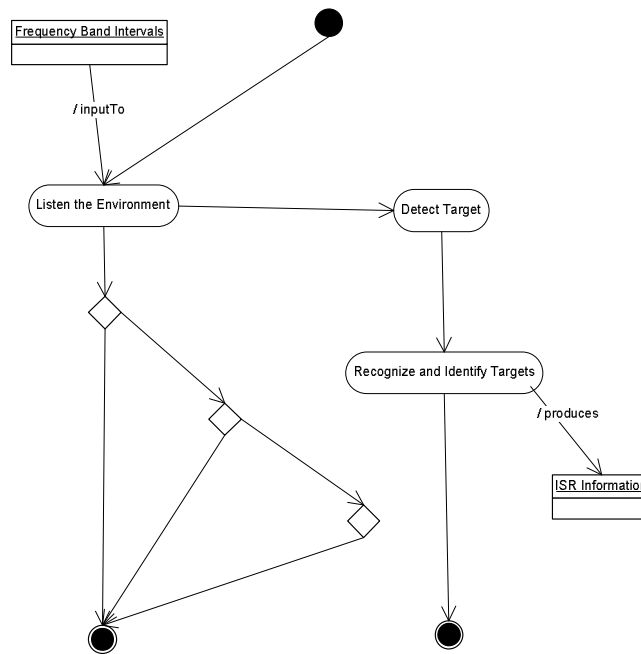


Figure 4-5 *FinalTask* Differences in Execution

Based on the above explanation, the rules concerning the *initialTask* and *finalTask* elements are as follows;

I-1 An *initialTask* shall be executed only during the execution of a *task* that contains the node.

F-1 A *finalTask* shall finish execution before the containing *task* finishes execution.

Further semantic variations were observed in conceptual models that confirm to rules explained as IN-1. The optional properties for *initialTasks* are listed as;

I-O-1: The *initialTask* execution shall be created when the continuing *task* execution is created.

I-O-2: The *initialTask* execution shall be created when the containing *task* execution is started.

Further semantic variations were observed in conceptual models that confirm to rule explained as FN-1. These optional properties are listed below where FN-O-1 stands for a similar case with UML 2.0 *finalNode*.

F-O-1: The *finalTask* execution shall cause termination of *tasks* that started execution by the same containing *task* execution.

F-O-2: The *finalTask* execution shall cause termination of containing *task*.

Other than these related to the execution of contained *tasks*, subtasks, the following rules exist;

CO-1: Subtask executions shall not extend execution interval of containing *task*.

Other optional properties for subtask executions are listed below.

CO-O-1: The subtask executions shall be unique. For each of the execution exactly one subtask execution can be created.

CO-O-2: The subtask executions shall not overlap, so during execution of parent *task* at most one *task* shall execute.

CO-O-3 The subtask executions shall be ordered. First started execution shall finish before the next execution.

CO-O-4: A subtask execution shall be executed only by one *task*. Subtasks can not be shared by more than one *task*.

4.2.3 SynchronizationPoint Model Element

SynchronizationPoints occur as two types, the fork type is used to create multiple *task* executions from a single execution, and join type is used to trigger a single execution from multiple incoming executions. Whether a fork node will execute if all the following branches accept, some of the branches accept or only one branch accepts is not explicitly explained in the KAMA definition.

Also the definition of fork node is present but it is not known whether the branches are matched, so that only matched executions are joined, is not discussed. Furtherly synchronized groups of *tasks* can be defined. The definition can be made by matching fork and join nodes. After matching the nodes, the execution can be a matched one or unmatched one.

After finishing a previous *task*, the fork type *synchronizationPoint* will create synchronization. During synchronized *task* executions, a *decisionPoint* or another fork

may provide an exit from the branch which shall later block joining. The synchronization nodes can also create loops which shall be checked for termination.

S-O-1: There shall be no leaving branch from synchronized branches. All the branches shall be connected with the join type node.

S-O-2: The executions shall be matched in a fork node shall only be later be joined by a join node.

S-O-2-1: Synchronized *task* executions shall start and finish at the same time.

S-O-2-2: Synchronized *task* executions shall overlap for some intervals.

S-O-2-3: At least two of the synchronized *task* executions shall overlap during execution.

S-O-2-4: The *tasks* on separate synchronization branches shall not overlap.

S-O-3: The synchronized branches can be connected based on predetermined conditions.

S-O-3-1: Synchronized *task* executions can be joined if only if all the branches completed.

S-O-3-2: Synchronized *task* executions can be joined even if some of branches are not active.

S-O-3-3: Synchronized *task* executions can be joined even if some of branches are not active.

S-O-4: Synchronized *task* executions order can depend on order of the other branches in the synchronization.

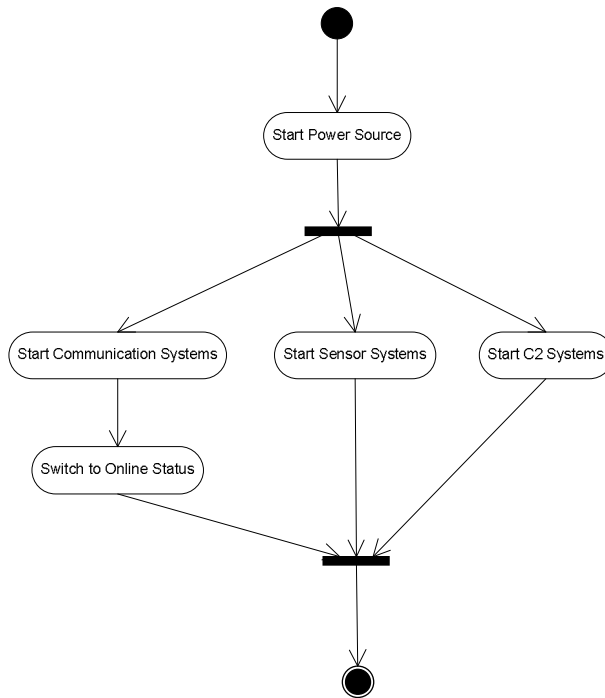


Figure 4-6 A SynchronizationPoint of Fork Type with Flows Synchronized

Another problem occurs in case which fork node is storing tokens in case at least one of the branches start execution but one of others could not if Petri net based semantics is used. For this reason we add an optional constraint.

S-O-5: Synchronization point of type fork will not execute if one of the succeeding branches failed to start execution.

S-O-6: Join node will execute if all branches terminate.

S-O-7: Join node will execute only a set of determined branches terminate.

S-O-8: If only a subset of the control flows to a join node is available than all the matched executions shall be terminated.

The synchronization nodes provide mechanism to enable execution of a set of *taskFlows*. The *task* execution during synchronization may not be explicit. In that case there is no need to match the fork and join type *synchronizationPoints*. After execution of *task* one of the enabled *tasks* may not start because of several reasons. In that case we have to continue its enabling or not. In the join case whether the information shall arrive synchronously or not is a problem. It shall be required that the information arrives at the same time or not.

4.2.4 *DecisionPoint* Model Element

When a decision node executes a single branch is chosen for execution. Then the node shall be marked according to the decision. As stated in KAMA metamodel the *guardConditions* of *taskFlows* from *decisionPoints* therefore the branches are mutually exclusive.

D-1: Only one of the branches shall be selected for each *decisionPoint* execution.

For some of the conceptual models there may be requirement on the execution of each branch as follows;

D-O-1: Each outgoing branch of *decisionPoint* shall be executable.

The *decisionPoint* can create a loop, whose termination may also be checked.

4.2.5 *Cascaded DecisionPoints* and *SynchronizationPoints*

Cascaded nodes will cause complexities during execution and their interpretation is complex. One of the cases to be avoided in conceptual models is cascading of the *intermediateNodes* which do not have direct real counterparts unlike *missions* or *tasks*.

IN-O-1: *intermediateNodes* shall not be cascaded.

IN-O-2: *synchronizationPoints* and *decisionPoints* shall not be cascaded.

4.2.6 *WorkProduct Package* Model Elements

The KAMA *workProduct* package includes *workProduct* model elements, *inputTo* and *produces* relations. Task descriptions can define the requirements of work products in the preconditions of *tasks*.(See Appendix A.4) These work products are modeled with *workProduct* model elements. The availability of *workProducts* may be a constraint for the *task* execution. The examples of such constraints are as follows;

WP-O-1: The executed *task* shall have inputs.

WP-O-1-1: An input will be consumed by the executed *task* and will not be available.

WP-I-1: The executed *task* shall provide outputs.

WP-I-1-1: A new output will be produced by the executed *task*.

The *workProduct* notation is also another concept in conceptual modeling whereby a single *workProduct* can be attached to more than one *task*.

4.2.7 *Role Package Model Elements*

We have two optional rules that effects execution of *tasks*. One of them is about assignment of *roles* to *tasks*, the other is assignment of roles to actors.

R-O-1-1: Each executing *task* shall be assigned to a *role* by using *realizes* relation.

R-O-1-2: Each executing *task* shall be assigned to a *role* by using *responsibleFor* relation.

R-O-1-3: Each executing *task* shall be assigned to a *role* either by using a *realizes* or *responsibleFor* relation

R-O-2-1: A *role* shall be assigned to an *actor*.

R-O-2-2: A *role* shall be owned by an *actor*.

Other variations may exist due to *task* hierarchy.

R-O-3-1: If a *role* is not assigned for a containing *task* than the *role* shall be assigned to contained *tasks*.

R-O-3-2 If a *role* is not assigned for a contained *task* than the *role* is the same as the containing *task*.

4.2.8 *Objective Package Model Elements*

For the objectives two basic requirements may exist for the executions. The objective elements are either shall have an objective or assigned the objective of higher level *task*. The objective shall have measure.

O-O-1: Each *task* execution shall have an *objective*.

O-O-2: Each executing *task* shall have an *objective* or have the containing *task* execution's *objective*.

O-O-3: A *task* execution's *objective* shall have at least one a *measure*.

If objectives in conceptual models are more thoroughly defined, more strict requirements can be attached so that *tasks* may only terminate if objectives are successfully attained.

4.2.9 Other Static Relations

The dynamical aspects of *task* execution are also constrained by static relations. The relations between the *actors* and *workProducts* shall be consistent with the dynamic descriptions.

Actors can be related with each other in command hierarchy diagrams. Usually containing *tasks* will be assigned to *roles* of *actors* which are superior to *roles* of *actors* of subtasks.

WorkProducts are related in entity ontology diagrams. There can be related *workProducts* in contained and containing *tasks* that shall be consistent with the information in entity ontology diagrams.

4.2.10 Relations with Objects (Instances of Entities)

Task execution affects individual objects. If more information is available on the objects states in during *task* execution and attribute values more through analysis can be performed. However since we deal with cases where the task descriptions are not defined in the context of entities, this topic is not in our primary focus.

CHAPTER 5

V&V REQUIREMENTS FOR CONCEPTUAL MODEL DYNAMICS

As seen in Chapter 3 and 4, the relationships between *tasks* are defined in such a way that automatic execution is not possible. In the context of conceptual models a *task* can be used to represent a wide range of behaviors, from the behavior of a single element to that of many elements constituting a large system. In this Chapter we will further try to clarify context of conceptual model dynamics. Section 5.1 below will provide the requirements for verification and validation of conceptual models and Section 5.2 will provide the framework for execution of model elements that can be used during dynamic verification process.

5.1 Conceptual Model Verification Requirements

Given characteristics of conceptual models in chapter 3 and the dynamic semantics of elements in section 4 makes possible of checking some properties, however given a dynamic description in a conceptual model it is impossible to determine the exact behavior. Rather than finding incorrect behavior, a more pragmatic and useful approach will be determining if the dynamic interpretation of the conceptual model implies behavior that is valid.

We will elaborate more about the execution of *tasks* in the following sections. However for the sake of completeness, we will categorize the requirements of dynamic behavior verification as follows:

- i. Any behavior shall not be allowed that include a *task* does not end execution, the deadlocks and a set of *tasks* executing infinitely many times, livelocks.
- ii. The capability to limit certain execution orders based on the information presented in the conceptual model as text. For instance even if two *tasks* are not depicted in the same *task flow diagram*, there may be a precedence relation between them.

- iii. In addition to precedence relation explained in the previous paragraph, for some of the *tasks* other constraints may exist such as two *task* executions sharing some time frame.
- iv. Other than these constraints related to sequencing, other constraints may exist between *tasks* based on the *roles* and *workProducts* which will further make the executions that do not obey them invalid.

5.1.1 Error Prone Parts of a Conceptual Model

While it will be generally agreed that most errors will occur in more complex parts of the model, finding the complex parts is not an easy task. To our knowledge, there is no work dealing with the complexity of simulation conceptual models or conceptual model parts.

The general approach is application of complexity metrics used for computer programs to models. However most the work related to metrics in modeling deals with complexity of the class diagrams [81] [82]. In a related work, activity diagrams describing the steps of functions of classes, is used to evaluate the number of function calls to distinct classes to determine complexity of the models. List of ideas on diagram complexity is provided in earlier works [83]. Also research at earlier stages [84] aiming to define and evaluate metrics comprehensively considering novel UML developments exist.

In the case of business models there are more relevant work for determination of complexity [61]. If sufficient information on determining model complexity is defined, it can be used to plan verification activities.

5.2 Execution Dynamics for Model Elements

In this section based on the findings of Chapter 4 and dynamic verification requirements specified in section 5.1, we describe the execution framework for conceptual models and for *mission*, *task*, *taskFlow*, *synchronizationPoint*, *decisionPoint*, *initialTask*, *finalTask*. After that we describe the mechanism for inspection of these elements dynamically.

5.2.1 Basic Concepts Related to Execution of *Tasks*

Based on observations on Chapter 4, we will explain a general execution framework first. Then for each model element in the context of *task flow diagram*, we will explain the dynamic aspects and variances in terms of our framework.

Considering these properties some basic assumptions shall be made for execution of *tasks*. First of the basic assumption is that *task* execution may be instantaneous or span time in

real world. As we are dealing with *tasks* that have some practical counterpart in real world, we adopt this assumption. Secondly we assume that the *taskFlow* association is a form of relationship that provides causality between *task* executions. So if one *task* execution enables other's execution, the second could not start before that *task*. These two assumptions form the basis of our *task* execution approach.

Other than these, inspection of KAMA metamodel points out that the special elements in *mission space package* like *initialTask*, *finalTask*, *synchronizationPoint*, and *decisionPoint* are described as intermediate constructs. For instance these elements lack attributes and therefore is not used to describe tasks, but to describe causal relationships between tasks. Therefore the execution of these elements does not take time. These elements are used for proper sequencing of the tasks. These sequencing is done by marking time points. However these elements can introduce delays. Delays can be as a result of *guards* of the *taskFlows* or introduced intentionally by the user.

During execution the main unit that the events are located is the order, instead of time. So after the assumptions of execution we will discuss how order is manipulated during the execution. An event that starts later is regarded to have a larger order. In case during execution an event can be given the same order as the previous. We discuss the specific cases in the next subsection. Elements such as *synchronizationPoint*, *decisionPoint*, *InitialTask* and *finalTask* elements acts as “*traffic switches*” (term is used in UML 2.0 standard to describe the *Intermediate Package of Activities*).

While we consider that the execution times of the tasks, we also take into consideration the fact that during mental execution the exact timing of the tasks may not be known precisely and fully. While we define the execution rules for the tasks as below we do not aim to provide a direct timing analysis for the execution tasks. Such a timing analysis requires much more information on timing than that is present in a conceptual model. Our primary interest is the order of the *tasks* during the execution. The order is important for two purposes:

1. The sequencing relations defined by *task flow diagrams* constraint the order. By this way execution is managed and controlled by the *task flow diagram*. An example execution described by the conceptual model can be produced by using the information in the conceptual model.
2. Satisfaction of constraints about the order of *tasks* can be checked. These constraints may be deadlocks, live locks and on abortion of *tasks*, execution times.

There are two distinct types of dynamic model elements in mission space behavior descriptions. First is the elements used for representing real counterparts, *mission* and *tasks*. Second type is used for sequencing of these, *taskFlows* for basic sequencing, *initialTasks* and *finalTasks* for organization of task hierarchy, *synchronizationPoints* for synchronization of *tasks*, *decisionPoints* used for branching and merging. While the first type of execution can consume time the second type can not, while both types can

introduce delays during the execution to describe the proper execution order equivalent to real order of the *mission* and *tasks*.

Mission elements are also treated as *tasks*. For *mission* elements that are related with an *includes* relation with an other element, the *task* of included *mission* is also available during execution. If a mission *extends* other mission, the extended *mission*'s elements are also embedded included after the extension point.

In Figure 5-1, the bar notation shows the creation of a particular element and execution of it. The corresponding to real world execution is shown with an arrow. The execution of *taskFlow* and other elements mark single points in execution. The points are used to determine the occurrence order of *tasks* during execution.


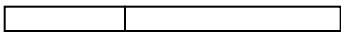

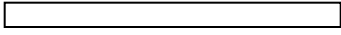

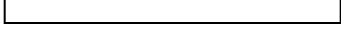

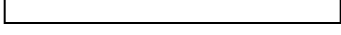

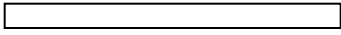

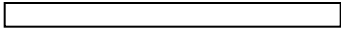
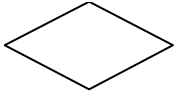
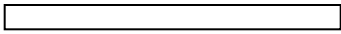
Name / Symbol	States
Task 	Created Started Finished <i>OR</i> Aborted 
TaskFlow 	Created Executed <i>OR</i> Aborted 
InitialTask 	Created Executed <i>OR</i> Aborted 
FinalTask 	Created Executed <i>OR</i> Aborted 
SynchronizationNode (Fork) 	Created Executed <i>OR</i> Aborted 
SynchronizationNode (Join) 	Created Executed <i>OR</i> Aborted 
DecisionNode 	Created Executed <i>OR</i> Aborted 

Figure 5-1 Execution States of Elements of *Task Flow Diagram*

5.2.2 Phases of *Task* Execution

The lifeline of *task* execution is shown in Figure 5-2. The *task* execution has three distinct points that mark enabling of the creation time of the *task* (e_{ij}), starting of the execution (s_{ij}), finishing of the execution either by termination or abortion of the *task* (f_{ij}).

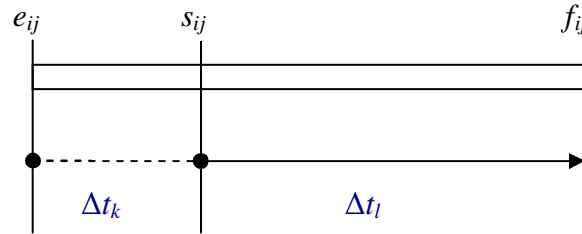


Figure 5-2 The Creation, Starting and Finishing Time of a *Task* Execution

Time elapsed between the creation and execution of a *task* $\Delta t_k = s_{ij} - e_{ij}$, is the waiting time of the *task*. The waiting time can occur because of *task* waiting for preconditions to be satisfied which may exist because of the extensions of the *task flow diagram* or because of the relations between *tasks* that are not depicted explicitly by the *task flow diagram*. The creation of the *task* has not a direct counterpart in the real world. It is used for proper scheduling of the *task* executions. It can be zero if the *task* does not wait for execution after creation.

The execution duration of the *task* $\Delta t_e = f_{ij} - s_{ij}$ represents the real execution time of the real *task*. If the *task* is an instantaneous *task* its duration may be equal to zero.

In our approach, the executions of *tasks* are described using order of the *tasks* rather than time. The order of a *task* is chosen for execution for the following reasons. First the order of execution is more important in conceptual modeling than the exact times. As conceptual model is an abstract representation of the reality, the definition of *tasks* and sequence of *task* executions are more central. Also related to this, the execution times are not precisely defined in a conceptual model generally. Yet the scale of timing may vary much in a conceptual model as well as in a simulation. So it will be hard to provide a real execution before the implementation of the simulation.

5.2.3 Example of Basic Triggering during *Task* Execution

In Figure 5-3, we have depicted two subsequent *task* executions conforming to scheme explained in previous subsection. In the following paragraphs, we will explain the relationships between the *tasks* based on the principles for *task* execution. Then we will describe the relations based on the order concept.

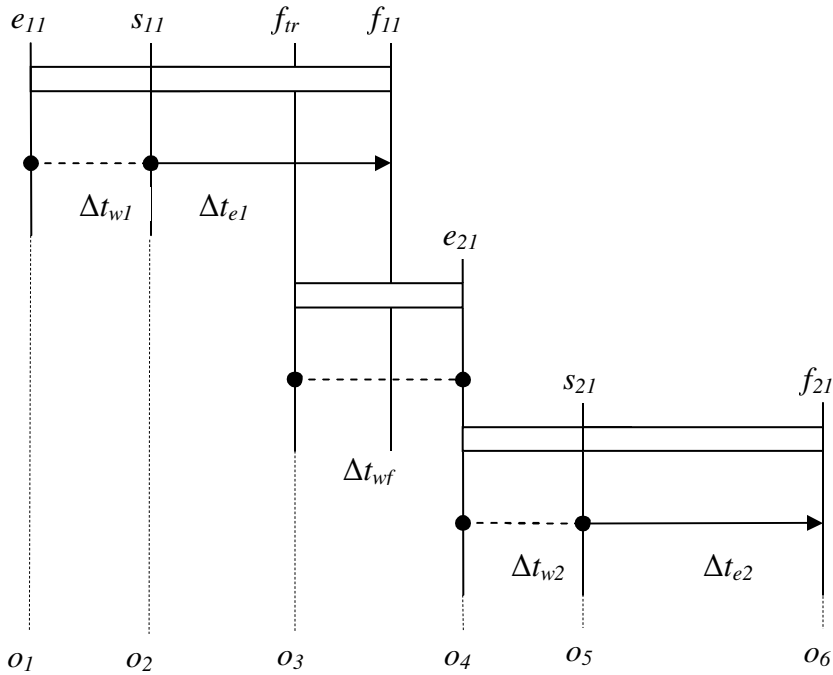


Figure 5-3 General Case of *Task* Execution

The first relation is preservation of causality, so the starting time of *task* 2 cannot be earlier than starting time of *task* 1 so $s_{11} \leq s_{21}$. The relation between orders is as follows, $s_{11} \leq s_{21}$ than $o_2 \leq o_5$.

The second property is the execution of *task* 2 shall be enabled during execution of *task* 1 so $s_{11} \leq t_{tr} \leq f_{11}$. This also results in the constraint that $s_{21} = \Delta t_{wf} + \Delta t_{w2}$. If there is a guard condition of the *taskFlow* than we assume that the flow will wait Δt_{wf} until it is satisfied. As we assume no unrecorded or unattributed time passes

Also we shall add here the constraints $e_{11} \leq s_{11} \leq t_{11}$ and $e_{21} \leq s_{21} \leq t_{21}$.

Given these two we shall add also another constraint $s_{11} \leq e_{21}$ for the execution approach.

Finally it shall be underlined that primary interest of the modeler is the start and finish orders of the *tasks*. The creation and execution orders of other elements are used for proper sequencing of *tasks*. So in the next subsections we will focus on relation between *task* execution orders.

5.2.4 Variances in *Task* Executions Related to Model Elements

5.2.4.1 Successor *Task* Execution

In Figure 5-4, we have depicted several cases of successor *task* execution. The *task* can either be executed during execution (Figure 5.4.i) or after the previous execution (Figure 5.4.iii), also it can execute for multiple times for both cases. In another case the execution of successor *task* can start with the finishing of the predecessor *task* (Figure 5.4.ii). In this case the execution may be one of the several executions that started earlier or later. In a more broad case the execution of the successor *task* may occur after the execution of predecessor *task* with some delay.

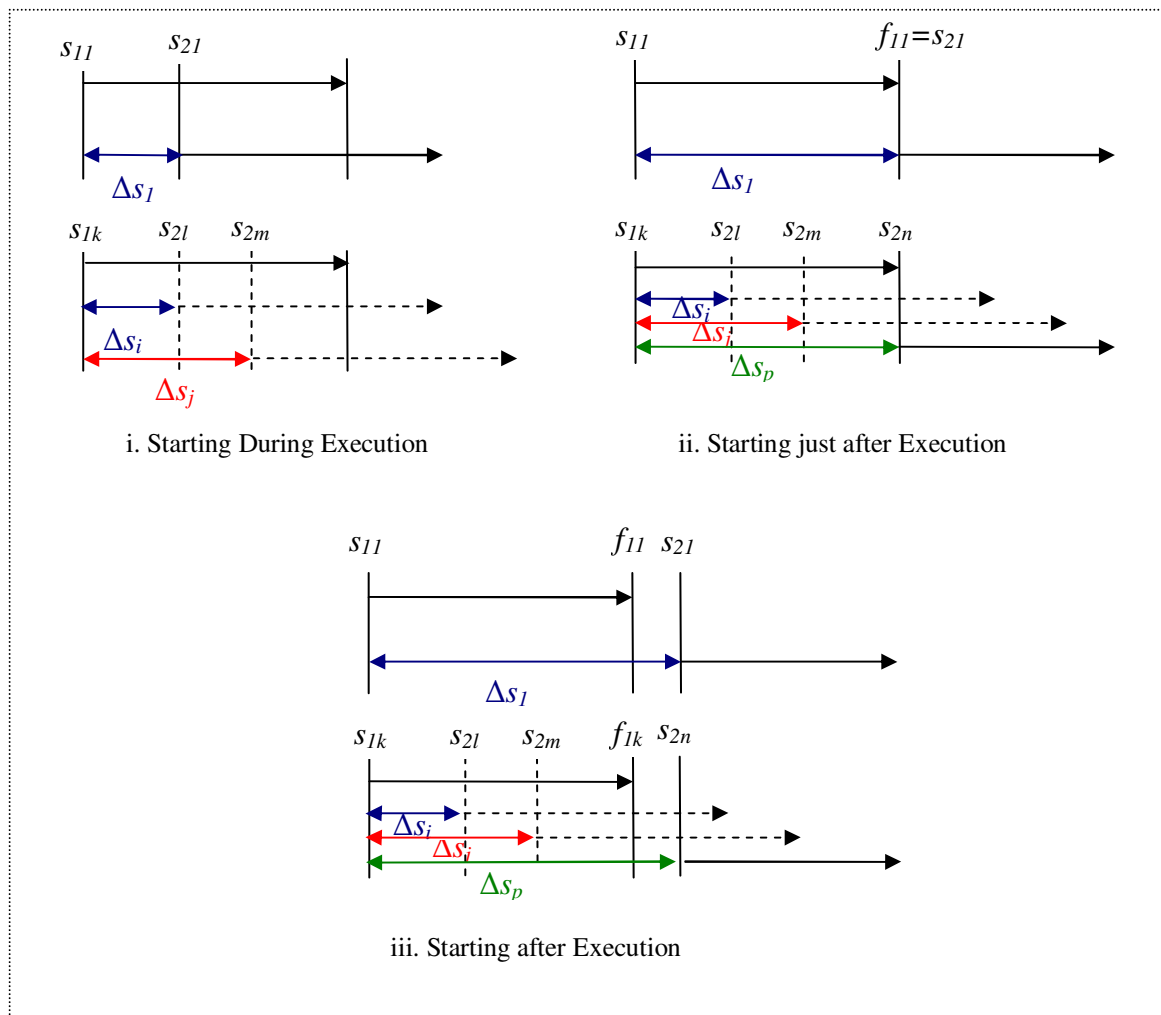


Figure 5-4 Execution of Successor Task

5.2.4.2 SynchronizationPoint (Fork Type) Execution

We present several examples of *task* executions in Figure 5-5. The synchronization can cause immediate start of the following *tasks* at the same execution point (Figure 5.5.i) or just after finishing of the *task* before the synchronization (Figure 5.5.ii). In a generalized case the synchronized *tasks* can start later because of the delay because of the *guardConditions* in *taskFlows* or by themselves after they are enabled (Figure 5.5.iii or 5.5.iv).

In Figure 5-5, different colors indicate unmatched executions. In another case the fork can mark executions so that only the matched executions can join later in the flow by an associated join type *SynchronizationPoint*.

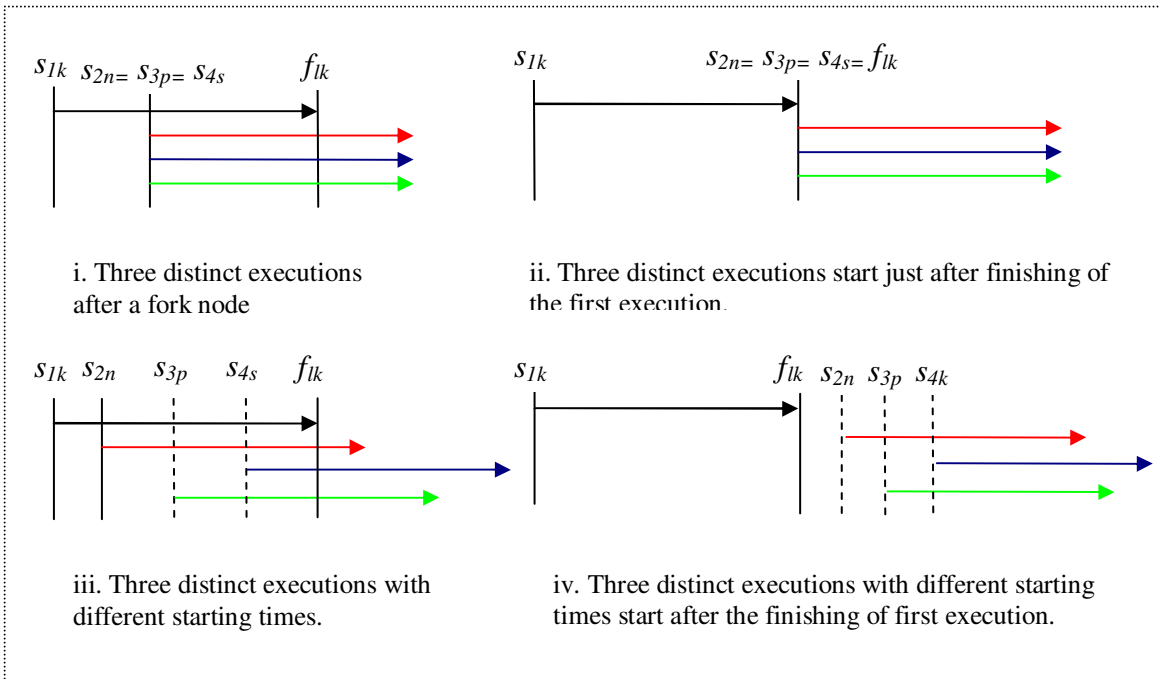


Figure 5-5 Execution of Fork Type SynchronizationPoint

5.2.4.3 SynchronizationPoint (Join Type) Execution

The synchronization point is also used to join the multiple branches to form a single branch. Then these executions that are joined finish at a single instant as shown in Figure 5-6, the former for the matched case, the latter for the unmatched case. Also a delay can

be introduced for the execution of joined executions so that the successor *task* can start after their execution finishes.

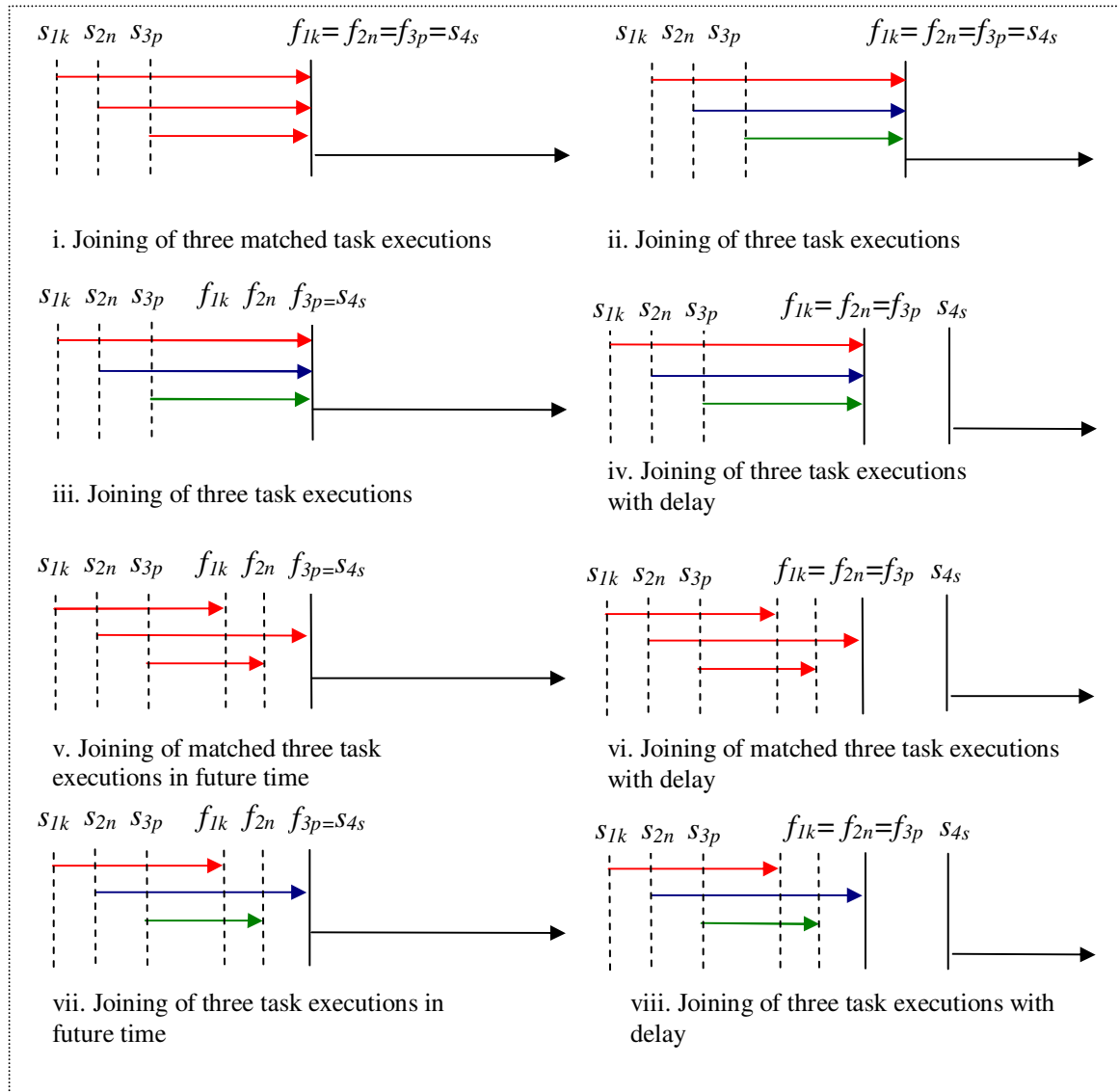


Figure 5-6 Execution of Join Type SynchronizationPoint

As an alternative the executions to be joined may not be required to finish at the same instant as shown in Figure 5.6.v and Figure 5.6.vi. This is also possible for the unmatched executions as shown in Figure 5-6.vii and Figure 5-6.viii.

5.2.4.4 DecisionPoint Execution

As the *decisionPoint* executes the time for decision is marked and the other *tasks* continue their execution. A *decisionPoint* execution is directly associated with the *task*

execution which triggers it. Only one decision at a time will be selected. As in the case of successor *task* execution the decision may start before finishing of the predecessor *task* or with a delay as shown in Figure 5-7.

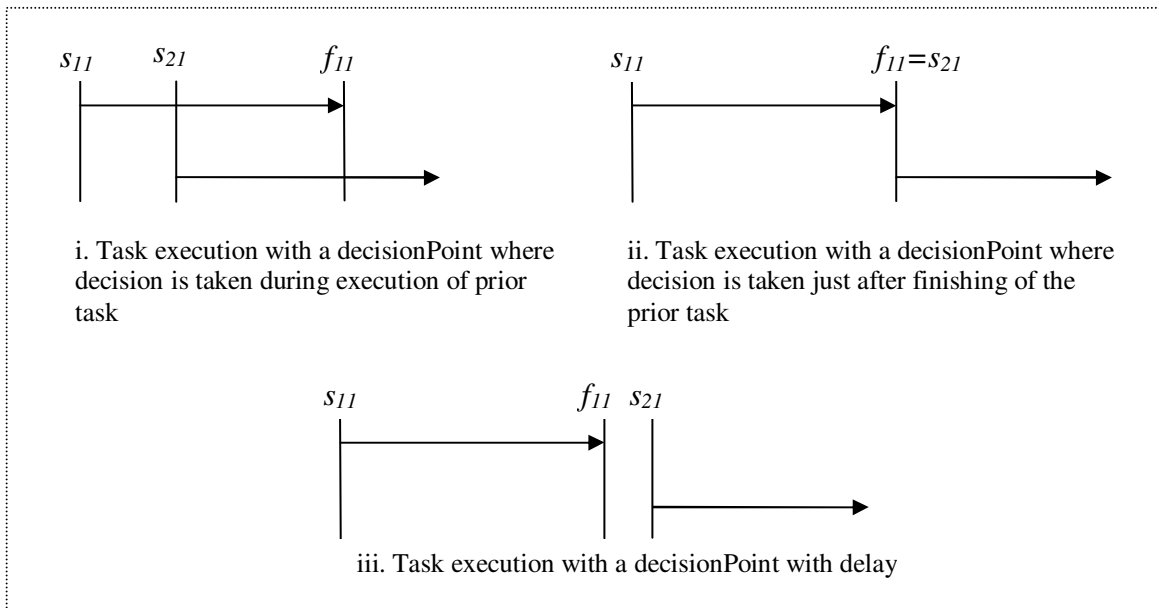


Figure 5-7 DecisionPoint Execution

5.2.4.5 Task Hierarchy Execution

The *task* can cause execution of the *tasks* which it includes during its execution. The execution of the contained *tasks* may be in parallel with the execution of the containing *task*. The *task* executions may start later, finish earlier, or both. Multiple executions of the subtasks can also occur, and these executions may overlap. These variations are shown in Figure 5-8. The overlapping executions may be ordered as subtask execution that starts earlier executing, finishing earlier or unordered.

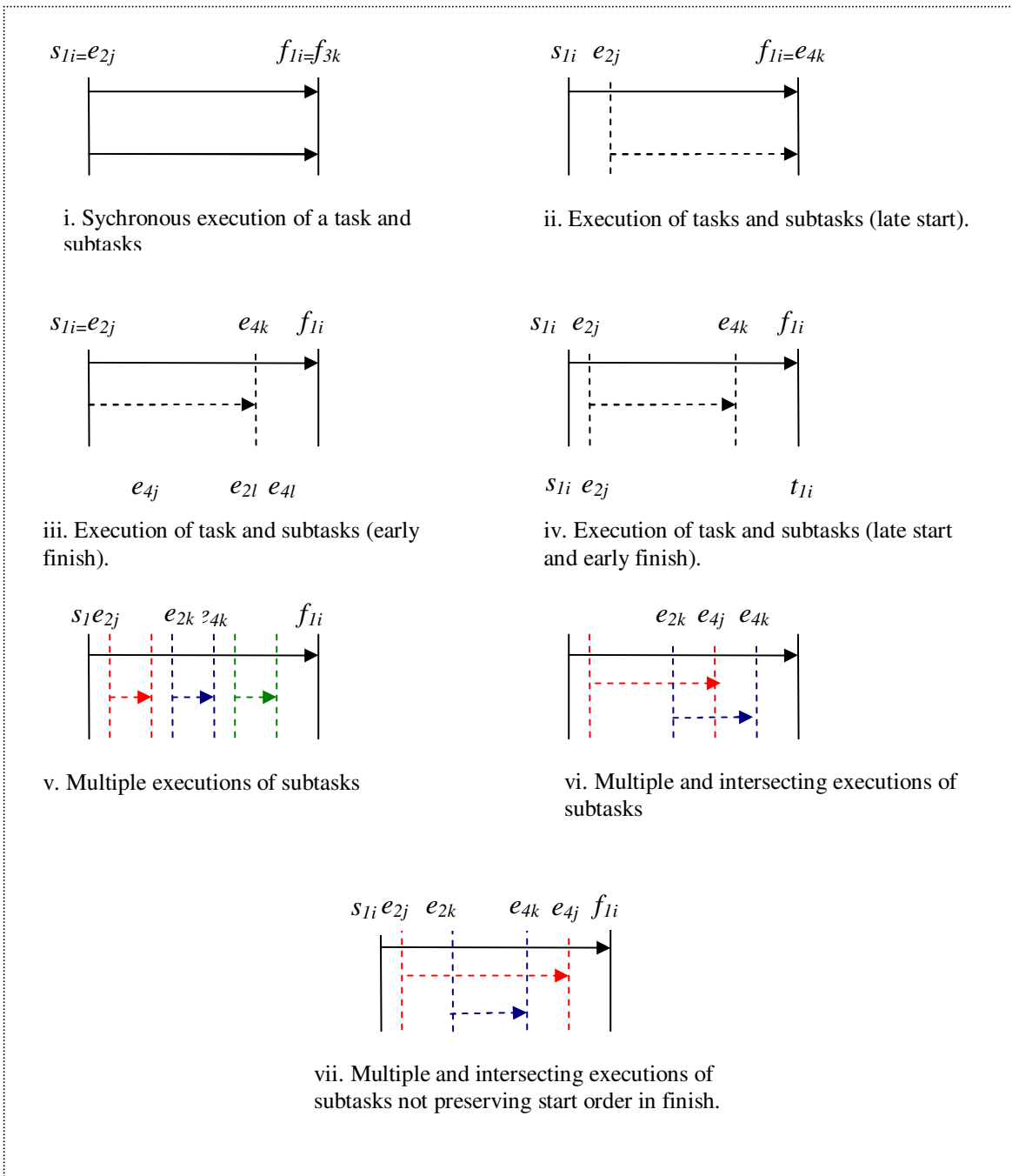


Figure 5-8 Execution of Task Hierarchy

CHAPTER 6

THE VERIFICATION APPROACH FOR CONCEPTUAL MODEL DYNAMICS

In this chapter the approach for dynamic verification of conceptual models will be explained. The aim of the approach is verification of dynamics of Conceptual models by exploiting the advantages of model based approach. As we have discussed in Chapter 2 in literature survey, the availability of conceptual models is the main limitation in developing the approach. Existing KAMA models which conform to KAMA metamodel do not have information so that they can be analyzed dynamically based on the execution rules given in Chapter 5. We have included the dynamic information requirements in the KAMA-DV metamodel we provide in this chapter. This will provide a baseline for the future studies that aim to form models that can be analyzed based on the execution rules while in this study we left out such an approach.

The method requires performing a set of activities. The general information of overall method is provided first, then each step is explained. Each step consists of application of semiformal or formal methods for verification.

In this chapter first the model based verification approach is explained. After this general explanation of model based approach, KAMA-DV metamodel which provides the basis for the approach is described. The additional model elements and properties of KAMA-DV metamodel, which differentiate it from KAMA metamodel expressed in Chapter 2, are based on the semantic variations expressed in Chapter 4 and execution approach described in Chapter 5.

Then we will outline the process of dynamic verification. In the last section, we will provide the description of tools that are used to realize the parts of the verification approach.

6.1 Model Based Verification Approach

As we aim to develop the verification using model driven approach [25], we have a set of models to work on during the verification process. The first model is KAMA model that

is developed based on KAMA metamodel. KAMA tool which includes a graphical editor and additional capabilities can be used for this activity.

Secondly we have the KAMA-DV Model based on the KAMA-DV Metamodel. This model includes all the model elements and rules expressed in KAMA and further dynamic extensions. These extensions are added for finding the errors and issues related to dynamics and support dynamic analysis.

The compliance of the KAMA-DV Model to its metamodel is the primary step of any verification effort dealing with dynamics. The second step is to analyze model to detect any errors stemming from dynamics. In Section 6.4, we will provide information on the tools that can support our approach and both explain how the necessary tool is used to check rules related to dynamic descriptions and how dynamic analysis can be performed.

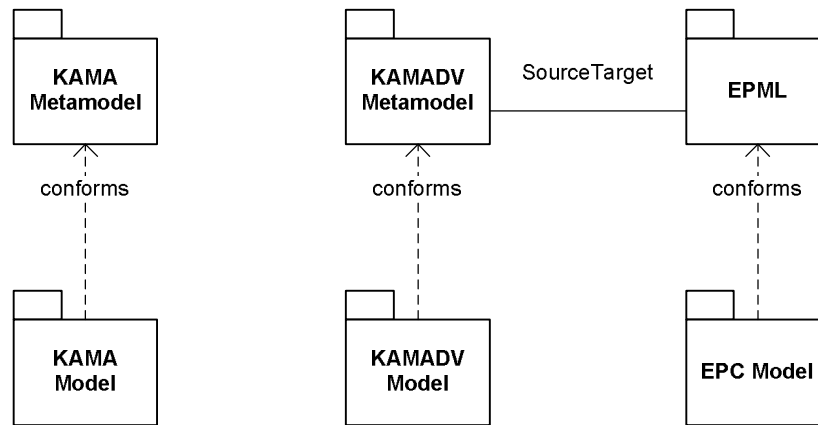


Figure 6-1 Distinct Models Used for Verification

6.2 KAMA-DV Metamodel

In this section, we briefly explain KAMA-DV Metamodel. KAMA-DV Metamodel is developed based on the semantic differences of model elements in KAMA models and dynamic requirements. In particular the KAMA-DV additions can be categorized in three:

First the KAMA approach is formally represented in Chapter 3, but the properties related to *task* hierarchy and *task flow diagrams* are left out as they are also left out in original KAMA. In KAMA-DV a particular addition is done to explicitly define semantics for the hierarchy by additional elements and rules, that are vaguely defined in KAMA. Some of these changes also affect the original KAMA metamodel constraints.

Secondly based on the observations in Chapter 4, additional properties and rules related to dynamics are added to represent these. Some of the constraints are added for better classification of errors and having a clear representation to ease application of dynamic verification.

Thirdly elements representing the dynamic instances of the various elements are also added. These elements have the properties that will represent the execution approach in Chapter 5. These are added to KAMA-DV Metamodel as a separate package.

In the following section we will briefly describe the model elements, properties and rules. Later we will define the properties and constraints formally in a way similar to Chapter 3.

6.2.1 KAMA-DV Metamodel Organization

KAMA-DV has two main packages, first contains the elements in KAMA and dynamic extensions, second contains the instances representing executed instances of these elements. The relations between the elements of these two packages is similar to relations between classes and objects in UML definition.

For some of the rules we add additional attributes to the KAMA metamodel elements. The information contained in these attributes will be used in dynamic verification and execution. These rules together with KAMA rules constitute the KAMA-DV metamodel. In this section first we will briefly discuss the KAMA-DV elements that are not explicitly included in the KAMA Metamodel. These are *TaskFlowDiagram* element, *consistOf* and *representedBy* associations. In this section we will detail the elements of KAMA-DV model.

6.2.2 KAMA-DV Core Package

KAMA-DV Core is the main package that includes all the information related to dynamics. It contains the KAMA Mission Space Package by merging it. The extensions of elements are explained in the following sections. Sets of ordered pairs for associations is defined, as *RB* for *RepresentedBy*, *HC* for *hascontext*, *IS* for *isShown* and *CN* for *context* property similar to Chapter 3.

6.2.2.1 Task

The semantics of *task* of KAMA-DV is similar to *task* of KAMA. As we will explain the same *task* may be associated with more than one *task* with *consistOf* association. When this happens the *task* is included in more than one *task flow diagram* and for each diagram it will have both incoming and outgoing *taskFlow* associations. So a *task* will have a *taskFlow* participation for each diagram.

We are required to represent the *task flow diagram* as distinct model element. Two relations between a *task* and *task flow diagram* elements may exist:

- i. A *task flow diagram* can show the subtasks of a *task*. Then the *task* is associated with *representedBy* relation with the *task flow diagram*.
- ii. A *task* can be shown in a diagram. This occurs when the *tasks* is a subtask of another *task* that is in *representedBy* association with the *task flow diagram*. Then we add the *isShown* association between the *task flow diagram* and the *task*.

We also include a property for the *tasks* that is not part of KAMA. These properties are needed for capturing the semantic differences explained in Chapter 4.

Properties

Context: This property is the set of pairs of *task flow diagrams* and *tasks*. *Task flow diagram* is one of the the diagrams this *task* participates *representedBy* association and *tasks* are parent *tasks* of this *task*.

Rules

1. A *task* participates in *consistOf* association as a containing task if it is shown in a *task flow diagram* which represents the containing task.

$$\forall x \in T ((z \in TD \text{ and } y \in TD \text{ and } (x, z) \in RB \text{ and } (y, z) \in SI) \text{ implies } (x, y) \in CO)$$

2. The *task* has a single incoming *taskFlow* and outgoing *taskFlow* for each *task flow diagram*.

For representing this constraint, we separate all related cases into two where a *task* does not consist of any subtask and consist of one or more subtasks. We further decompose these constraints into two as incoming and outgoing *taskFlows*.

$$\forall x \in T ((\forall y \in T ((x, y) \notin CO)) \text{ implies } (z \in T, (x, z) \in TF \text{ for only one } z)).$$

$$\forall x \in T ((\forall y \in T ((x, y) \notin CO)), \text{ implies } (z \in T, (z, x) \in TF \text{ for only one } z))$$

For the case where a *task* is represented by one or more *task flow diagram*, for every *task flow diagram*, *task* is associated with *shownIn*, there is only one incoming *taskFlow* and outgoing *taskFlow*.

$$\forall x \in T ((z \in TD, (x, z) \in SI)$$

implies $(y \in T, (y, z) \in SI, (x, y) \in TF \text{ for only one } y)$).

$$\forall x \in T ((z \in TD, (x, z) \in SI)$$

implies $(y \in T, (y, z) \in SI, (y, x) \in TF \text{ for only one } y)$).

3. The *task* can not participate in *consistOf* association with itself or a task that *consistOf* itself. So a *task* does not contain itself recursively.

$\forall t_i \in T, t_i \notin TC$ where TC_i is the set of closure of the *task* with respect to *consistOf* association. So closure set is defined as the *consistOf* tasks of the task and closure of them, $TC_i = \{t_j \mid (t_i, t_j) \in CO, \} \cup TC_j$.

4. If the task contains one or more tasks and participates in a *representedBy* association with a *task flow diagram* then it shall *consistOf* with at least one element of type *initialTask*.

$$\forall x \in T ((y \in T \text{ and } z \in TD \text{ and } (x, y) \in CO \text{ and } (x, z) \in RB) \text{ implies } (w \in I \text{ and } (x, w) \in CO))$$

5. If the task contains one or more tasks and participates in a *representedBy* association with a *task flow diagram* then it shall *consistOf* with at least one element of type *finalTask*.

$$\forall x \in T ((y \in T \text{ and } z \in TD \text{ and } (x, y) \in CO \text{ and } (x, z) \in RB) \text{ implies } (w \in F \text{ and } (x, w) \in CO))$$

6.2.2.2 Task Flow Diagram

KAMA-DV has the *task flow diagram* as a model element rather than a view. This is consistent with the aim of KAMA-DV since the exact aim is to develop a metamodel for verification of dynamic properties and execution and *task flow diagram* is the diagram describing the relations between *tasks*. A *task flow diagram* resembles a source code file as it describes the dynamics of the model. As explained in the definition of *task*, the precise definition of *task* as a part of other *tasks* requires *task flow diagram* definition.

In this respect we treat a *task flow diagram* similar to an activity associated with an activity node of UML. A *task flow diagram* stand for a single interpretation of the the

task based on a given context. So in this case a *task* itself may have more than one interpretation, where only one of them is valid for a given context. *Context* is associated *task* that contains the related *task*. Each *task flow diagram* can be associated no or multiple *contexts*. *Context* is used to resolve which diagram is used to define the behaviour of subtasks.

The *task flow diagram* bears no direct relation with the concept of a diagram presented by the UML standard or XMI diagram interchange format. For UML, diagram is a view of a model while XMI DI deals with the orientation of nodes and elements and their relationships, concrete syntax. In our case, *task flow diagram* provides a description of the relations between subtasks and other elements for different occurrences of a *task*.

Properties

1. *isUnique*: This attribute is used to describe the nature of the *tasks*.
2. *isOrdered*: This attribute is set if the *task* executions of the *task* are ordered. For ordered *task flow diagrams*, the executions of subtasks occurs in an order, the first strated *task* execution completes first.
3. *isOverlapping*: This attribute is set if the *task* executions of the *task* can overlap. If seperate executions do not use the same resource than they can overlap.

Rules

The *task flow diagram* shows *tasks* that are subsets of the *tasks* that the diagram represents.

1. If a *task flow diagram* is defined in the context of a *task*, that *task* shall be in *consistOf* relation with the *tasks* that *task flow diagram* represents.

$$\forall x \in TD (\forall y, z \in T ((z, x) \in RB), (y, x) \in CN_z) \text{ implies } (y, z) \in CO)$$

6.2.2.3 representedBy

The *representedBy* relation is used to describe that the a *task flow diagram* represents a *task*. One *task* can be represented by more than one *task flow diagram*. This relation only exist between a *task* and a *task flow diagram*.

Rules

1. A model element of type *task* and *task flow diagram* participates in each *representedBy* association.

$$\forall (x, y) \in RB (x \in T, y \in TD)$$

6.2.2.4 *isShown*

Although we have explicitly added the *representedBy* association for the representation of tasks, we can define *isShown* associations by using the others. *isShown* association shows that a *task* is shown in the diagram.

Rules

1. Element of type *task*, *taskFlow*, *initialTask*, *finalTasks*, *synchronizationPoints* or *decisionPoints* and element of type *task flow diagram* participates in each *isShown* association.

$$\forall(x, y) \in RB(x \in T \cup I \cup F \cup S \cup D, y \in TD)$$

2. A *task* can only be shown in a diagram if it is one of the subtasks of the *task* that this diagram represents.

$$\forall x, y \in T, z \in TD ((y, z) \in SI) \text{ implies } ((x, z) \in RB, (x, y) \in CO)$$

6.2.2.5 *hasContext*

If a *task flow diagram* is valid for explaining the dynamic behaviour of a subtask than is meaningful in the context of the *task*. A *task flow diagram* may have more than one context, as the behaviour it describes may be valid for more than one *task*.

Rules

1. The source element of *hasContext* is of type *task flow diagram* and target of type *task*.

$$\forall(x, y) \in HC(x \in TD, y \in T)$$

6.2.2.6 *SynchronizationPoint*

For *synchronizationPoint* additional attribute is added for identifying fork and join type *synchronizationPoints*. This is based on the multiplicity of *SynchronizationPoints* in KAMA model.

Other than these for specifying the precise model execution additional attributes are added for matched fork and join type *synchronizationPoints*.

Other attributes are defined for specifying the behaviour of the *synchronizationPoints*

In KAMA-DV *synchronizationPoint* has additional attributes for model execution.

Property

1. *isJoin*: This attribute is a derived attribute to explicitly define the type of the *synchronizationPoint*.
2. *MatchedNode*: This attribute holds the ID of the node that this node is matched with. A fork and join is matched if during execution all the *taskFlows* from a fork node to be matched by a join.
3. *isSynchronizedFork*: If the outgoingFlows are synchronized meaning succeeding *tasks* start at the same order, or if the incomingFlows are synchronized meaning preceding *tasks* end at the same order this attribute is set to true.
4. *forktriggeringType*: This attribute specifies the triggering type of the *synchronizationPoint* based on the timing of the succeeding *task* execution. The triggering type is of type triggering type.
5. *joinTriggeringType*: This attribute specifies the triggering type of the *synchronizationPoint* based on the timing of the succeeding *task* execution. The triggering type is of type triggering type.

6.2.2.7 DecisionPoint

Some of the *decisionPoints* are used to merge the flows previously diverted by *decisionPoints* that have more than one outgoing *taskFlows*.

Property

1. *isMerge*: If *decisionPoint* has a single incoming *taskFlow* and multiple outgoing *taskFlows* then this attribute is set to one.
2. *triggeringType*: This attribute is set to an enumerated value based on the triggering type of the *synchronizationPoint*.

6.2.2.8 InitialTask

The *initialTask* has an additional property for defining the exact nature of starting of the subtask. Based on this property further constraints are defined in execution.

6.2.2.9 *FinalTask*

The *finalTask* has an additional property for characterization of the behaviour when it is reached in execution.

Property

1. *isfinalFlow*: If the *finalTask* executions destroys all the active executions that is shown in the *task flow diagram* it is shown that this boolean attribute is set to true.

6.2.3 KAMA-DV Behaviour Package

In KAMA-DV behaviour package we include the executable versions of model elements. The executable versions of model elements are based on the execution framework and model element execution rules presented in Chapter 5. As explained in Chapter 5 the execution is different for *tasks* and *intermediateNodes*.

6.2.3.1 *TaskExecution*

TaskExecution is used for representing an execution of a *task*. Following properties exist for a *task* execution. *Task* execution is created during dynamic behaviour execution. The *task* execution has following attributes

Property

context: It determines the execution context of the *task*. It is determined by the *task flow diagram* the *task* starts execution and preceding node and *taskFlow*.

creationOrder: It is the order of creation of the *taskExecution*.

startingOrder: It is the order of starting of the *taskExecution*.

endingOrder: It is the order of ending of the *taskExecution*.

6.2.3.2 *IntermediateNodeExecution*

IntermediateNodeExecution is used for representing the execution of intermediate nodes. According to framework given in Chapter 5, these executions are created and triggered.

Property

nodeType: This attribute represents type of the node the intermediate node execution refers.

context: It determines the execution context of the *intermediateNode*. It is determined by the *task flow diagram* the *intermediateNode* starts execution and preceding node and *taskFlow*.

creationOrder: It is the order of creation of the *intermediateNodeExecution*.

triggerOrder: It is the order of triggering of the *intermediateNodeExecution*.

6.2.3.3 *dynamicInstanceOf*

This association exists between the structural elements and their dynamic instances. A dynamic instance is an instance of a corresponding structural element.

6.3 General Outline of the Method

The method for dynamic verification consists of 4 distinct steps, which are shown in a *task flow diagram* in Figure 6-2.

Verification tasks are executed in a sequential fashion; if problems are encountered in a task then the activities of previous tasks are performed. The high level steps can be interpreted as generic steps necessary for dynamic verification of a domain specific modeling notation. In the first step, the information is organized and verified for conformance to the static rules. Then based on the related information in the model, the dynamic information is enriched. This includes additional of KAMA-DV metamodel properties defined in Section 6.1. Some of the attributes can be derived from directly the metamodel, while others shall be added by the modeler by determining the exact semantics considering the semantic variations explained in Chapter 5.

After additions structural compliance to KAMA and KAMA-DV are checked in the model to determine errors and issues. If no structural errors remain, dynamic checking can be performed.

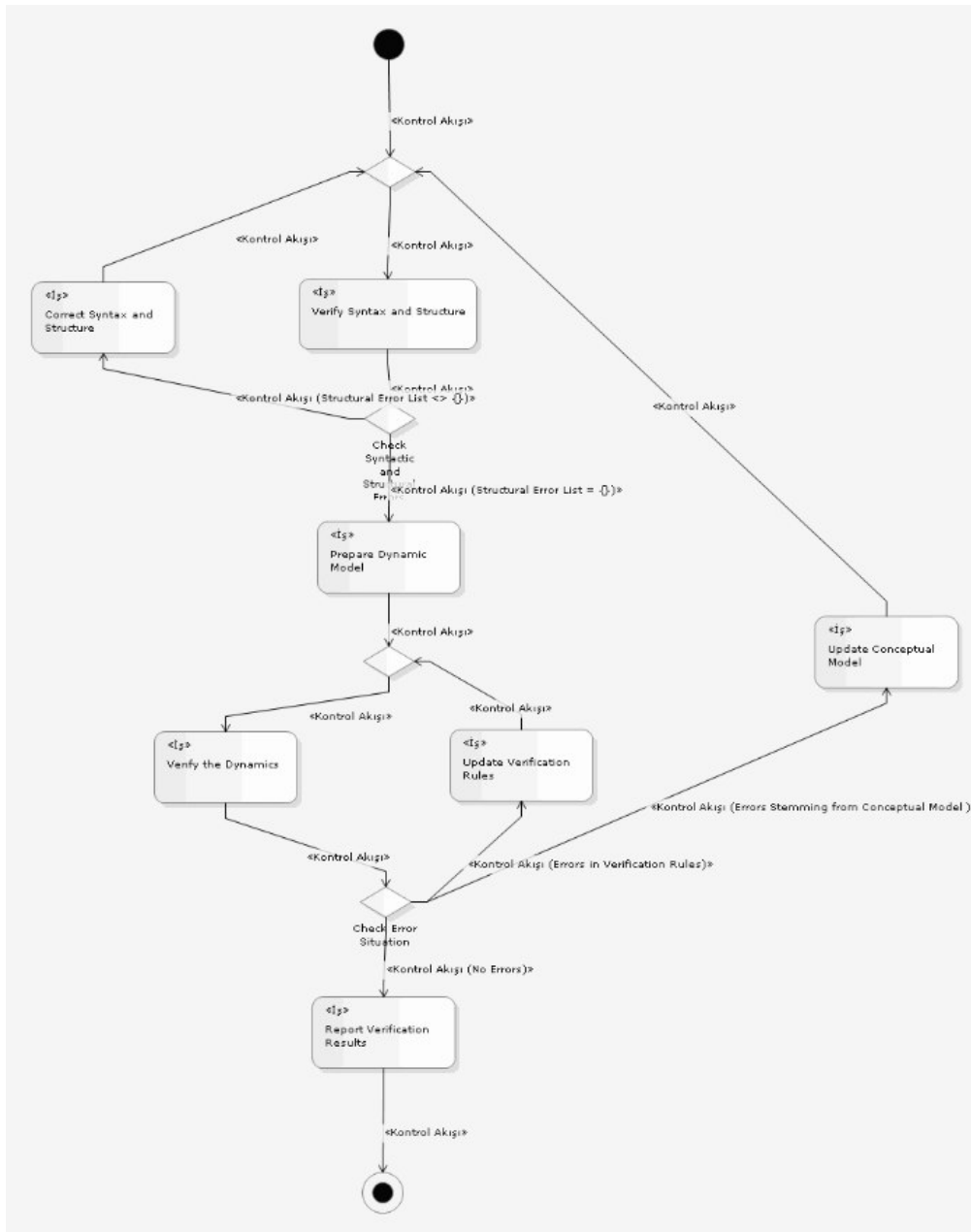


Figure 6-2. The Process of Dynamic Verification Shown as a *Task Flow Diagram*

6.3.1 Step 1: Structuring the Model and Static Verification

In this step the dynamic parts of the model (single or a group of diagrams) are represented as a KAMA task flow description as described in the previous section. The conceptual model may already be in a format similar to KAMA or some differences may exist. If there is a lack of information then referents and domain experts can be consulted for specifying dynamic information in the conceptual model.

After this step the conceptual model shall be checked for the syntax rules as described below in section 6.2. These checks can either be running algorithms or by formal checking of the model. In the next two subsections we will present the approach for verification of the static KAMA constraints and formal verification of properties related to dynamic analysis.

6.3.2 Verification of the Static Constraints

In this section we present an algorithm for checking each of the rules presented in Chapter 3. If the model elements and associations are stored in a database format assigning each model element metaclass a separate table, the number of iterations needed for checking each constraint can be calculated. The KAMA model for mission space description is already modularized as packages so the constraints are presented for each package with the needed number of iterations in the following subsections.

Integrity of elements can be checked independently for each package. Then checks can be applied to determine if the model obeys the constraints as a whole to address the errors that occur as a result of interaction of different packages. This mechanism has several advantages. For very large models, it will be convenient to process model information. Secondly, the constraints for each package can be understood more easily. Similar checks can be made for constraints that have similarity.

For KAMA models in the following subsections we list the rules that are defined from the KAMA metamodel. For traceability, we use the codes that are presented in Chapter 3 and the defined model element sets presented in Table 3-2.

6.3.2.1 Mission Space Fundamental Package Rules

Mission Space Fundamental Package constraints are presented in Table 6-1. The code represents the unique identifier of the rule based on its presentation given in KAMA Metamodel and KAMA semantics explained in Chapter 3. The first two letters of the code is taken from section number of these constraints in Chapter 3. The third letter is the letter where the corresponding rule is presented in KAMA metamodel. The fourth letter is further added to differentiate between the rules that apply to the same element. In the iteration column the model elements and associations that are iterated when checking a

rule is listed, and number of iterations represent the needed iterations. Iterations are calculated based on the number of sets of model elements, and for each set number of elements (e.g. n, m, k etc.) are multiplied.

Table 6-1. Number of Iterations Needed for Checking Fundamental Package Rules

Code	Rule	Iteration	No. of Iterations
3.3.a.1	$\forall m_i \in M (\exists r_j \in R ((r_j, m_i) \in RE \text{ or } (r_j, m_i) \in RF))$	M	n
3.3.a.2	$\forall m_i \in M (\exists o_j \in O ((m_i, o_j) \in AC))$	M	n
3.3.a.3A	$\forall m_i \in M, (m_i, m_i) \notin GN$	GN	n
3.3.a.3B	$\forall m_i \in M, (m_i, m_i) \notin IN$	IN	n
3.3.m.1	$\forall (x, y) \in EX (x \in M, y \in M)$	EX	n
3.3.m.2	$\forall (x, y) \in EX (x \neq y)$	EX	
3.3.m.3	$\forall (x, y) \in EX (\exists z (z \in T \text{ and } z \in TD(x) \text{ and } z.isExtensionPoint = 1))$	EX, T	nm
3.3.n.1	$\forall (x, y) \in IC (x \in M, y \in M)$	IC	n
3.3.n.2	$\forall (x, y) \in IC (x \neq y)$	IC	n

6.3.2.2 Mission Space Sequencing Package Rules

Mission Space Sequencing has the highest number of rules in KAMA Metamodel. In the same way as previous section, the number of iterations needed for checking the rules are presented in Table 6-2.

Table 6-2. Number of Iterations Needed for Checking Sequencing Package Rules

Code	Rule	Iteration	No of Iterations
3.3.b.3A	$(\forall t_i, t_j \in T((t_i, t_j) \in TF \text{ implies } \neg \exists t_k \in T(k \neq j, (t_i, t_k) \in TF)))$	TF	n
3.3.b.3B	$(\forall t_i, t_j \in T((t_i, t_j) \in TF \text{ implies } \neg \exists t_k \in T(k \neq i, (t_k, t_j) \in TF)))$	TF	n
3.3.g.1	“The connections coming into and going out of a <i>decisionPoint</i> must be <i>taskFlows</i> .”	AS	n
3.3.g.2	$\forall d_i \in D(((d_i, t_j) \in TF, (d_i, t_k) \in TF, i \neq k) \text{ implies } (d_i, t_j).guard \neq (d_i, t_k).guard)$	D, TF	nm
3.3.g.3	$\forall d_i \in D((d_i, t_j) \in TF \text{ implies } (d_i, t_j).guard \neq \emptyset)$	D, TF	nm
3.3.h.1	$\forall s_i \in S(((t_j, s_i), (t_k, s_i) \in TF, j \neq k) \text{ implies } (s_i, t_l) \in TF \text{ for only one } l).$	S, TF, TF	nmm
3.3.h.2	$\forall s_i \in S(((s_i, t_j), (s_i, t_k) \in TF, j \neq k) \text{ implies } (t_l, s_i) \in TF \text{ for only one } l).$	S, TF, TF	nmm
3.3.h.3A	$\forall s_i \in S((t_j, s_i) \in AS \text{ implies } (t_j, s_i) \in TF) \text{ and } \forall s_i \in S((t_j, s_i) \in AS \text{ implies } (t_j, s_i) \notin (AS - TF))$	S, AS, TF	nmk
3.3.h.3B	$\forall s_i \in S((s_i, t_j) \in AS \text{ implies } (s_i, t_j) \in TF) \text{ and } \forall s_i \in S((s_i, t_j) \in AS \text{ implies } (s_i, t_j) \notin (AS - TF))$	S, AS, TF	nmk
3.3.i.1	$\forall i_j \in I((x, i_j) \notin TF)$	I, TF	nm
3.3.j.1	$\forall f_i \in F((f_i, x) \notin TF)$	F, TF	nm
3.3.p.1	$\forall (x, y) \in TF (x \in (T \cup D \cup S \cup I), y \in (T \cup D \cup S \cup F))$	TF	n
3.3.p.2	“Only one <i>taskFlow</i> may exist between the same source and target”	TF	n

6.3.2.3 Rules for Other Packages

Here we list other rules for KAMA *Mission Space Packages* namely *Workproducts*, *Roles*, *Objectives* and *Complete Packages*. The rules for these elements show patterns as q.1, r.1, a.1, b.1, k.1., l.1, o.1 and s.1 are checked by iterating through a single database table. The second group is checked by iterating through two database tables.

Table 6-3. Number of Iterations Needed for *Workproducts Package Rules*

Code	Rule	Iteration	No of Iterations
3.3.f.1	$\forall w_i \in W (\neg \exists b_j \in B ((w_i, b_j) \in AS))$	W, AS	n
3.3.q.1	$\forall (x, y) \in IN (x \in W, y \in T)$	IN	n
3.3.r.1	$\forall (x, y) \in PR (x \in T, y \in W)$	PR	n

Table 6-4 Number of Iterations Needed for *Roles Package Rules*

Code	Rule	Iteration	No of Iterations
3.3.a.1	$\forall m_i \in M (\exists r_j \in R ((r_j, m_i) \in RE \text{ or } (r_j, m_i) \in RF))$	M	n
3.3.b.1	$\forall t_i \in T (\exists r_j \in R ((r_j, t_i) \in RE \text{ or } (r_j, t_i) \in RF))$	T	n
3.3.c.1	$\forall r_i \in R (\exists a_j \in A ((a_j, r_i) \in OW))$	R, OW	nm
3.3.k.1	$\forall (x, y) \in RF (x \in R, y \in (T \cup M))$	RF	n
3.3.l.1	$\forall (x, y) \in RE (x \in R, y \in (T \cup M))$	RE	n

Table 6-5 Number of Iterations Needed for *Objectives Package Rules*

Code	Rule	Iteration	No of Iterations
3.3.a.2	$\forall m_i \in M (\exists o_j \in O ((m_i, o_j) \in AC))$	M, AC	nm
3.3.b.2	$\forall t_i \in T (\exists o_j \in O ((t_i, o_j) \in AC))$	T, AC	nm
3.3.d.1	$\forall o_i \in O (\exists u_j \in U ((o_i, u_j) \in ME))$	O, ME	nm
3.3.e.1	$\forall u_i \in U (\exists o_j \in O ((o_j, u_i) \in ME))$	U, ME	nm
3.3.o.1	$\forall (x, y) \in AC (x \in (T \cup M), y \in O)$	AC	n
3.3.s.1	$\forall (x, y) \in QB (x \in O, y \in U)$	QB	n

Table 6-6 Number of Iterations Needed for *Mission Space Complete Package Rules*

Code	Rule	Iteration	No of Iterations
3.3.t.1	$\forall t_i \in T(((t_i, t_k) \in CO,$ $((t_k, x) \in (TF \cup PR_i \cup AC_i)$ <i>or</i> $(x, t_k) \in (TF \cup RE_i \cup RF_i \cup IT_i)))$ <i>implies</i> $x \in TD_i)$	T, CO, $(TF \cup PR_i \cup AC_i) \cup$ $(TF \cup RE_i \cup RF_i \cup IT_i)$	nmk

We only show the required iterations for explanatory purposes. In determining the number of required iterations for checks, we assumed that all the model elements are categorized, stored and accessible. If the models are stored in a format similar to XMI, and metamodeling and validation tools are available, these checks can be expressed in Object Constraint Language (OCL) and validation is feasible. As all these technologies evolve and there are limits on the functionality of metamodeling tools, we present the constraints for related elements of Mission Space Fundamental and Sequencing Packages in Appendix B.

6.3.3 Checking of Properties Related to Dynamics

There are some properties which are not explained in the models but are still relevant for dynamics of KAMA models. We will discuss these properties which can be categorized as derived properties, *task* hierarchy related properties, related to *taskFlow* properties, and *intermediateNode* related properties. When formulating these properties we do not use all the properties about dynamic execution described in Chapter 5. In this step, we provide a lightweight extension of KAMA Mission Space Package to include more information on dynamics. These properties are defined in Appendix B and used as part of KAMA-DV model.

6.3.3.1 Derived Properties

In KAMA the *decisionPoint* and *SynchronizationPoint* do not have an explicit attribute defining the type of the element in relation to multiplicity of incoming and outgoing *taskFlows*. As during dynamic verification it is not logical to test every time the nature of such a model element, a derived attribute can be added into the model element definition.

This information is inherent in the model definition based on the incoming and outgoing *taskFlows* and every *decisionPoint* can be classified as fork type if there is a single

incoming *taskFlow* and multiple outgoing *taskFlows*, and as join if vice versa. A single attribute of *isJoin* is added to this model element to define. For decision point the definition is same, a *decisionPoint* is either merge or split type and this information is stored in the derived attribute *isMerge*.

Other types of attributes also exist. For instance, *ModelID* attributes of model elements are used store the ID of model elements in the original conceptual model, whereas *Name* is used for storing the Name.

6.3.3.2 Properties Related to Task Hierarchy

In KAMA metamodel definition, we have explicitly added a *consistOf* relation to the *task* definition in Mission Space Fundamental Package (Section 3.1.1) and Mission Space Sequencing Package (Section 3.1.2) which is not mentioned in original KAMA Metamodel definition. We have also discussed the dynamic aspects of this relation in Section 4.2.2.

The *consistOf* relation has further description for the Mission Space Task Sequencing Package. This relation is needed for further analysis of the conceptual model for instance checking to prevent a *task consistOf* itself or a subtask.

Other properties related to the hierarchy may also be of interest during validation. According to KAMA Mission Space Model, *tasks* may have that have subtasks and corresponding *task flow diagram* shall participate in consist of relation with at least one *initialTask* and one *finalTask*. The semantics of *initialTask* and *finalTask* necessitates them to mark the start and end times of the diagram. Furthermore, in some of the cases dynamic analysis may require a point of entry (*initialTask*) and exit (*finalTask*).

6.3.3.3 Properties Related to IntermediateNodes

Other than these some of the constraints can be defined based on the nature of the models. In case of dynamic analysis, *tasks* are primary elements whereas *intermediateNodes* and *taskFlows* are used for sequencing according to our description given in Section 5.2. In this case loops formed by *task flow diagrams* that only contain such elements are prone to error as they do not contain *tasks*.

As we examine the *decisionPoint* and *synchronizationPoint* semantics in Chapter 5, the expected occurrence of these in pairs is expected for each *task* definition. Occurrence of more than one *decisionPoint* and *synchronizationPoint* may also indicate a problem as they merely stand for sequencing *tasks*. We also include necessary conditions for these in KAMA-DV Metamodel.

6.3.4 Step 2: Enriching the Dynamic Semantics

In this section we explain the merging of the diagrams into a single diagram. Through merging the modeler may aim to check the entire information given in the model. At the end of merging, the result is single diagram which is more concise and integrated to input into other graph based formalism for verification.

A set of diagrams can be verified one by one, or as a group by exploiting the *consistOf* association for *tasks*, *includes* and *extends* associations for *missions*, and *extensionPoint* tag value of *tasks*.

Only a group of *tasks* that form a true hierarchy can be merged to be analyzed. So there shall be no loop formed by *consistOf* association between presented *tasks*. We also limit our attention to case where each *task* has one *task flow diagram*, if not these can be resolved by using *context* information for the *task*.

6.3.4.1 Merging of diagrams based on *consistOf* association

The semantics of *consistOf* association is explained in Chapter 5. Below we provide a breadth first marking and merging algorithm for the diagrams.

- i. For the first iteration if there are composite *subtasks* of the top level *task*, these *task* elements will be deleted and replaced by their immediate *subtasks*.
- ii. For each of the first level composite *subtasks*, the associated *task flow diagrams* need to be processed by taking out *initialTask* and *finalTask* nodes, and associated *taskFlows*, marking *task* or *intermediateNode* after *initialTask* and before *finalTask* and copying them to the top level *task flow diagram*. Delete 1st level *subtask* and redefine the incoming *taskFlow* target as first marked node and outgoing *taskFlow* source as second marked node.
- iii. Repeat step 1 and 2 until no composite *tasks* exist in the diagram.

6.3.4.2 Merging of diagrams based on *extends* association

If more than one *mission* is to be merged which have *extends* association between each other, the *tasks* of the extending *mission* are appended in the *task* specified as *extensionPoint* in extended *task*. The *task* which is in an *extensionPoint* shall not participate in any *consistOf* association.

- i. Create an imaginary *task* named same as extending *mission*.
- ii. Add all the model elements of the *mission* to extension point, except the *initialTask* and *finalTask*.

6.3.4.3 Merging of diagrams based on *includes* association

Unlike *extends* association, information there is no information on how a *mission* is associated by *includes* with another mission and how their *tasks* are weaved into each other. A subtask can be specified in the hierarchy that the included mission's *task* and subtasks will be appended. If such a *task* can be determined, the included *mission* is added as an imaginary *task* corresponding to included *mission*. The description of merging involves the following steps:

- i. If there is no subtask of the including *mission*, create an imaginary *task* named same as including *mission*. Create a new *task flow diagram*, by adding an *initialTask* and *finalTask*. If only one *mission* is included just add all the model elements to this diagram.
- ii. If more than one *mission* with *subtasks* is included, then determine the relations of them by consulting the domain expert. They can be connected in parallel by *synchronizationPoints* or *decisionPoints*, or serial.
- iii. If the including mission have *subtasks* so a *task flow diagram*, add the *tasks* of the included *mission* to a specific *task* that can be regarded as *inclusionPoint* similar to *extensionPoint* by consulting the domain expert.

6.3.5 Dynamic Analysis

After syntax and structure checking the dynamic analysis can be performed. The first step of dynamic analysis is determination of the methods and tools for dynamic analysis based on the dynamic characteristics of conceptual models explained in Chapter 5.

Task flow diagrams can be converted to required formalism based on the desired analysis. This conversion can be done manually or by transformation languages. After this conversion the dynamic characteristics are examined and evaluated.

If all *task flow diagram* is described for a single entity and includes only *singleExecution tasks* by transforming the diagram into an EPC soundness and relaxed soundness analysis can be performed in a straightforward way. This transformation can be performed on a *task flow diagram* with no structural errors by replacing the similar EPC elements shown in Table 2-4. Two different kinds of property are checked as follows:

- i. If the corresponding EPC is not sound, some of the executions are not sound, meaning that there are execution sequences that do not complete and results with a deadlock.
- ii. If the corresponding EPC is even not relaxed sound, some of the *tasks* in the *task flow diagram* is not part of a sound execution, an execution in which *finalTask* terminate without a *task* under execution. This means that there are execution sequences that do not complete and results with a deadlock.

More detailed analysis can be made by interactively executing the EPC to infer the causes of being not sound or relaxed sound using tools like EPC tools [59].

6.4 Tool Support for the Verification

In this section, the tool to be used in implementing the verification approach proposed in this study will be described. As developing a tool from scratch requires a lot of effort and reuse is the primary concern for model based development, we decided to customize existing tools developed for modeling.

6.4.1 Selection of the Verification Tool

The tools considered in this section are listed according to their developer institution (commercial, academic, personal). The criteria for evaluation of the tools are determined as functionality, availability, modifiability, and level of support.

In terms of functionality, the tool shall support both the inspection and visualization of the conceptual model dynamics. Our work emphasizes exploration and inspection of conceptual model dynamics so the tool may support most of the functions related to visualization and user interaction. As KAMA diagrammatic elements have structural, behavioral and visual similarity with UML 2.0, the tools based on UML 2.0 are more appropriate for us in terms of functionality. However as we have also KAMA tool that has a graphical editor, the metamodel definition and verification aspects are more crucial in terms of functionality. More precisely following are the main functional requirements from the verification tool:

Support Model Manipulation

KAMA tool has limited power in interoperability as models developed by the tool can be imported and exported by only proprietary XML format. The verification tool shall have capability to store, query and update the models so that it can acquire models from KAMA, process the models and export them to other formats that other tools work with.

Support Metamodel Based Development

One of the main points is that the original KAMA metamodel is based on UML. As MOF and UML are de facto modeling notations used for software modeling and extend their reach to other modeling domains it is crucial for verification tool to support metamodel based development.

Structure and Semantic Checking

The tool shall have capability to check the syntax and structure of models so that they can be used in dynamic analysis conveniently. The rules expressed in KAMA metamodel are also to be checked on a model. In this sense the model shall have an environment for defining MOF based metamodels and OCL constraints. Any additional functionality like graphical editors, model editors, text editing aids and error highlighting is also a plus for the tool.

Model Transformation

In the process of verification the models may need to be merged, model elements may be extended with additional properties. While during syntax and structure checking an extended notation to KAMA Mission Space and Task Flow can be used, during dynamic analysis other types of notation may be used as in our case. So the tool shall have capability to transform models from the given formalism to another.

Dynamic Checking of Models

The tool shall have capability to check dynamics of models. This includes checking of soundness and existence of paths with no livelocks and deadlocks. Checking of constraints on utilization of *workProducts*, assignment of *roles* and *actors* may also be desired.

As an advanced functionality, the tools ability for state space analysis and step by step model execution is also highly appreciated for dynamic analysis. Other than functional criteria non functional aspects exist such as availability, modifiability and level of support.

Availability

In terms of availability the tools can be categorized into two the commercial tools and free tools. There are large number of tools that support metamodel based development and verification.

Commercial modeling tools are not inexpensive. Usually these tools come as a set of components, so that for the required functionality, the user has to obtain license for a handful of separate tools. Academic tools are available usually for free but the provided support is limited. In comparing availability, various levels can also exist (source code availability, application software availability, cost, licensing cost).

Modifiability

Both the dynamic views of KAMA and KAMA-DV has significant differences from UML Activity Package. The limitations of MOF and UML based tools stemming from the limitation of their capability and UML standard may necessitate need for modification of the modeling tool and environment provided by the tool substantially. In case of commercial tools this modification is not possible. An application programming interface (API) may be provided by some tool vendors which enables modification of the functionality as desired, one by one and compose it to form new functionality. Unfortunately the support of highly developed APIs is scarce in general and in the modeling domain in particular. In modifiability one can further define criteria such as source code availability, coherence of the tool, source code documentation, and capable API.

Level of Support

In terms of support we evaluate the software documentation, user community and availability of a point of contact (POC) for questions. Commercial case tools have a rich documentation and wide user community, and support personnel. But they are usually configured for a specific functionality.

6.4.1.1 Existing Tools and Their Functional Deficiencies

As we focus on MOF and UML based tools, we see a great number of tools. However most of the tools have drawbacks that make them unsuitable for the functionality we desire.

- i. Although UML development tools deal with most of the modeling only subset of them allows metamodel based development or profiling. So we can not describe the semantics of KAMA and KAMA-DV in these tools.
- ii. In formulating KAMA and KAMA-DV constraints we desire to use OCL, which only a limited number of tools support. OCL verification is not integrated with UML editors and Activity Package model elements. Among the OCL tools available only a couple of them provide an extensive API for developing and updating metamodel and models.
- iii. Tools generally do not support analysis and model execution for activity charts and EPCs. We will list the capable tools that provide analysis support and their limitations. Tools that provide functionality of execution of activity charts and EPCs, which have similarity with KAMA dynamic descriptions, are shown in Table 6-7. These tools only provide limited capability for model execution.

Table 6-7 Tools Dealing with Analysis of Activity Charts and EPCs

Tool	Lan.	Integrated Modeling Tool	Website / Reference	Formalism	Source code
ActiveChartsIDE	C#	Microsoft Visio 2003	http://activecharts.informatik.uni-ulm.de/ [50]	Activity Diagrams	N/A
ACTi	N/A	Use tool	N/A [51]	Activity Diagrams	N/A
IBM Model Debugger	Java	Integrated	http://www.research.ibm.com/haifa/projects/services/uml/vision.shtml [49]	Activity Diagrams	N/A
EPC Tools	Java	Own graphical Interface	http://www.cs.uni-paderborn.de/cs/kindler/research/EPCTools/	EPCs	Available

ActiveChartsIDE is tool developed for activity diagram animation. The limitation of the tool is the limitation of working with UML activity diagrams only and no extensions are possible. Among these tools ACTi, is not currently provided or supported. It has also limitation of being only a textual language for activity diagram specification. IBM model debugger is presented in a research paper but is not available for the larger community. Also there is limited information on how debugger is realized based on the UML2 activity package semantics. EPCtools [59] is used to analyze soundness of event process chains. The information on the application of the tool and its use make EPCtools a better option in analyzing *task flow diagrams*.

There are also a set of related tools which is used for defining models, checking of constraints that are relevant for us. These tools are listed in Table 6-8. However these tools have significantly different scope and it will be hardly represent the semantics of KAMA-DV metamodel in these tools. For instance none of the tools used for simulation of UML models given in Table 6-8 can be used for syntax and structure checking of activity diagrams.

Table 6-8 Other Tools for Simulation of UML Models

Tool	Language	Aim	Website / Reference
Use tool	Java	Animation and Verification of UML Class Models	http://www.db.informatik.uni-bremen.de/projects/USE/ [72]
Key tool	Java	Verification of Java Programs	http://www.key-project.org/
GENGED	Java	Simulation by graph transformation	http://user.cs.tu-berlin.de/~genged/index.html [78]
Omega	Java	Simulation of UML Models	http://www-omega.imag.fr/ [77]

Although a bunch of tools exist for these purposes only a couple of them has capabilities of having multiple tools for achieving all of them. Moreover the tools are developed for specific aims which require a lot of effort to tailor them for our verification purpose.

Based on the wide range of capabilities and support, Eclipse Modeling Frameworks is used as the environment for dynamic verification of models in this work. Core of the Eclipse Modeling Framework is the ecore metamodel which is similar to OMG Meta Object Facility (MOF) definition. Eclipse modeling environment also has a separate facility for UML2 based development and graphical model development of UML models.

Eclipse modeling framework provides an implementation of OCL. OCL is used to query and validate ecore based models and UML models. The constraints can be entered in terms of invariants in the ecore model by using an editor. Also a separate OCL console exists to enter queries and constraints.

There are many model development tools in the software market. As OCL is our preferred choice for expressing the metamodel rules, number of tools are less numerous. A bunch of other OCL tools exist but only a few of them provide extensive IDE like functionality [73]. As they have a large community of users contributing to documentation, testing and support, Eclipse OCL and Eclipse Modeling Tools (MDT) have advantages in the non-functional criteria. The components of the tool and their associated role in development of verification environment are shown in Table 6-9.

Table 6-9 Features and Plug-ins Used during Verification Process

Eclipse Modeling Framework Component	Version	Function
EMF Model Development Framework	2.6.0.v20100614-1136	Model based development environment
Ecoretools	0.10.0.v20100615-1639	Diagrammatic tools for metamodel development
EMF Model Validation Framework	1.4.0.v20100428-2315	Additions to Model to Validation, adding OCL constraints, OCL integration
OCL Extender		OCL parsing and interpretation
OCL Examples and Editor	3.0.0.v20100506-1704	OCLinEcore Editor
EPCtools	2.0.3	Dynamic analysis of EPC diagrams

6.4.2 Models Involved in the Verification Approach

In developing the approach we have selected the Eclipse Model Development Framework for the verification of static properties, enriching the model and transformation of the model to EPC model. For checking of the dynamic constraints, EPCtools plug-in was utilized.

Eclipse Modeling Framework is open source project. The environment provides tools to develop custom plug-ins. The forum for support exists where users and developers can ask questions and requests. The aim is to provide an integrated environment for model driven development. This includes model transformation tools, code generation tools, and tools for parsing and interpretation of OCL constraints.

ECPtools was selected for verification of the soundness as it supports defining the EPC model and checking for the soundness. EPCtools is a plug-in developed in JAVA so the models that are checked for consistency can be transformed to EPC and loaded to the model conveniently.

In the next three subsections we will discuss the static checking, transformation and dynamic checking approaches in a more detailed way.

As we aim to develop the verification using model driven development, we will have a set of models we will work with during the verification process. The first model is a

KAMA model that is developed based on KAMA metamodel. KAMA also provides a graphical tool for modeling.

For static and structural checking we use the eclipse modeling framework (EMF) Model Development Tools (MDT). MDT involves an editor for developing metamodels, OCL editor for adding invariants to model classes, facilities to create instances of models and checking models conformance to metamodel and invariants. Moreover Eclipse has facilities to transform the models to other models using a model transformation language which is also based on OCL querying language. This enables transformation of models to EPCtools models.

For dynamic analysis we use the EPCtools that is also an Eclipse project to define and validate EPC diagrams. EPCtools uses graphical editing Framework of Eclipse (GEF) to define models and stores models in EPML format which is the format for EPCs. By using Eclipse we have exploited benefit of working in the same environment and exploiting many different tools.

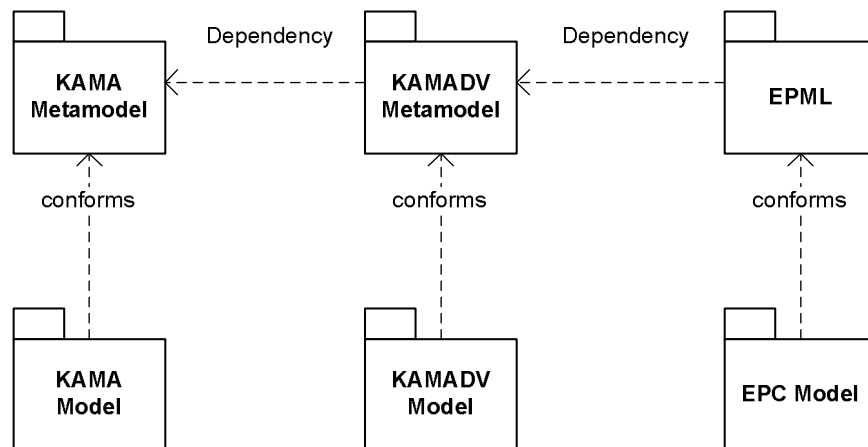


Figure 6-3 Distinct Models Used for Verification

6.4.3 Adaptation of the Tool for Checking of Metamodel Properties

For adaptation of the plug-in environment we have formed a profile for expressing the dynamic features in Java environment. Java profile development plug-in provides a visual interface for defining the profile. By using the interface one can define new stereotypes and tag values. After each model element is defined, the OCL statements are added for the elements. The corresponding profile is given in the next chapter in Figure 7-1 which correspond to metamodel elements provided in Chapter 3.

By using the *ecore_diagram* facility the profile defined in Eclipse is also shown to the user.

6.4.4 The Tool for checking the Dynamic Properties of the Model

We have used EPC tools for checking the dynamic properties of the model. For a better understanding of the EPCtools suite [59] one can refer the documentation. We will briefly discuss the working of the EPCtools suite in Java environment in the next Chapter.

CHAPTER 7

EVALUATION OF THE VERIFICATION APPROACH

In this chapter we focus on applying the KAMA-DV approach for verification of real life conceptual models. In order to develop our method and validate the approach we use the case study methodology. Before detailing the applications, in the first section, case study approach and its use in the information systems domain is discussed.

7.1 Selection of the Case Study Approach

The case study approach aims at empirically investigating a phenomenon within a real-life context. Related to using case study approach in information systems domain, Benbasat [66] states that it is appropriate when “research and theory are in their early formative stages”. It is a practice based approach where “importance of experiences of actors are of utmost importance” and “context of application is critical”. It is used in scarcely researched problems when other methods of research are not applicable, such as to study information systems in their natural setting for understanding the nature and complexity of problems. Unlike other approaches the level of control is low in case study research. Runeson and Höst [67] point out similar uses of the case study approach in the information systems domain. Various information sources, including first, second and third degree sources are utilized for theory development and testing for an area for which understanding is limited.

As summarized by Benbasat [66], the case study approach can be used in various ways, on the one extreme it can be just used to test a well formed theory in a positivist way. On the other hand, it can also be used just to interpret the application of a method. In another dimension, while for some research problems a single case study would suffice, multiple applications may be necessary for some others. For the latter case, the prior applications may be used to build up the theory and the latter ones to test it. The case study research has also been evaluated based on several different perspectives for validity and reliability [68].

As we have explained in Chapter 2, the methods for verifying conceptual models for simulation systems are limited in scope. Real examples of application of verification methods and experience reports are not abundant in the literature. Developing and verification of metamodel based conceptual models is also a novel concept and there are not established approaches and research practices. All these limit the level of understanding in the conceptual modeling area. Conceptual modeling is inherently

complex as explained in Chapter 2. The effort to verify a conceptual model is also high which indicates the necessity of repeated trials.

In the scope of the present study, considering the limitations of time and case availability, two case studies are performed: one for increasing the level of understanding and enhancing the approach, and the second for testing it. We present our case studies in an outline similar to the one provided by Runeson and Höst [67].

In the following sections, we will describe each case study, related real world conceptual models and the application. Our aim is to develop the approach and show that the approach can be used to aid conceptual model verification even if the conceptual models are not fully aligned and existing tools provide limited support for the approach.

7.2 Case Study 1: Conceptual Model of the Surveillance Mission

In the first case study we applied the technique to a conceptual model developed by two experts in KAMA notation. The initial version of KAMA-DV was developed using the mission space package of the KAMA metamodel. We aim to test the approach to see the feasibility of the approach, enhancing the KAMA-DV method and comparison with the results of the inspection approach performed on the same model.

7.2.1 Case Study Design

Case design includes the research questions and the case selection, subjects, data collection and analysis procedures.

In the first case study the aim is to answer two research questions:

What are structural and dynamic properties of the conceptual models that can be checked automatically?

How should the metamodel for conceptual modeling be expressed for automatic checking of structural and dynamic errors?

The motivation for these questions stem from the motivation to understand the consequences of using metamodel constraints expressed in KAMA for automatic verification. By this way we aim to understand the consequences of automatic verification by these constraints and develop KAMA-DV into a framework for automatic verification.

The original conceptual model was developed in textual format during a C4ISR simulation development project. Various entities in the conceptual model have templates which list their properties and relations to other elements. The conceptual model was later transformed to KAMA notation by the experts. The KAMA tool also has some limited

verification functionality, so the model was also checked for some of the syntactic properties during development.

The case study was performed in collaboration with a panel of two modeling experts in the form of a series of meetings. In the first meeting the KAMA metamodel constraints and corresponding KAMA-DV constraints were discussed. By this way KAMA-DV has been leveraged to a state where first testing in the context can be performed. Later in a meeting with the experts the results of the case study were discussed again. In this meeting the focus was on the errors found in the model with respect to the semantics defined in the KAMA-DV. The results were also compared with the results of a previous verification study based on the inspection approach [34].

The case study has also some limitations. As the conceptual model was originally developed in textual format and later represented in the KAMA notation, some of the aspects of the KAMA modeling notation were not applied rigorously. While there is a vast number of tasks and activities which are also modeled as *tasks*, other definitions are incomplete. For instance the roles were not defined precisely; only actors were defined as command and control units. These units are also very limited in nature as there are 12 command and control units. These definitions were also vague and effectively all the command and control units could perform most of the tasks. The work products were also not defined as separate items and their characteristics were not explicitly specified. As a result, in verification of the conceptual model we focused on the sequencing behavior of *tasks*. For the sake of brevity, we have selected the *surveillance mission* for conceptual model verification as it displays the characteristics of the other *tasks* and it has relatively more complete descriptions.

Table 7-1 Model Statistics of SGKS Conceptual Model

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

The original SGKS conceptual model document has a total of 153 missions, tasks and activities, all of which are modeled as *tasks* in KAMA. The *surveillance mission* consists of 91 of these *tasks*, which nearly accounts for 60 percent of the *tasks*. As discussed below, the model was not automatically transformed into the KAMA-DV model. As the Surveillance mission already represents a major part of SGKS behaviour description, and

additional effort is required to define other missions in KAMA-DV, the case study is performed using the part of SGKS that includes *surveillance mission* and subtasks, collectively named Surveillance Mission Conceptual model.

Another limitation stemming from the selection of SGKS is that it is rich in information about missions and *tasks*, while the information on *workProducts* and *roles* is vague. Most of the *tasks* are performed by many roles. In verification we limit our attention on the *tasks* and their relations. The *surveillance mission* encompasses all the command and control units and entities that are described in the SGKS document related to other *missions* and *tasks*.

7.2.2 Results of the Case Study

In this section, we will provide the results of the case study. First we will provide information on the automatic checking approach.

In the first meeting with the modeling experts KAMA metamodel elements and constraints related to them were discussed. The implementation of KAMA-DV metamodel in *ecore* was also discussed. During these meetings there were updates and corrections for the KAMA-DV constraints and their implementation. Some of these are also discussed further in the following subsection on the improvements of the metamodel.

Then the automatic verification is performed, and results from syntax and structure checking and soundness analysis are obtained. Finally during the process the experts are provided with initial feedback results and their opinion on the result of the process is used to further develop the verification process.

Syntax and Structure Checking

We have selected the surveillance mission as the mission to be verified in our conceptual model. The exact list of *tasks* that surveillance mission consists of is given in Appendix B. The related elements in KAMA Mission Space and KAMA Sequencing Package are redefined as an *ecore* metamodel using *ecore_diagram* facility of *ecoretools* as seen in Figure 7-1. Later this metamodel is opened using *OCLinEcore* editor and the constraints of KAMA Mission Space Sequencing Package and KAMA-DV metamodel are added.

Table 7-2 Statistics of the SGKS *Surveillance Mission*

Model Element	Number
<i>Tasks</i>	91
<i>taskFlows</i>	179
<i>IntermediateNodes</i>	77
<i>Task Flow Diagrams</i>	24

Although we aim to directly use the KAMA output format as an input to KAMA-DV as mentioned in Chapter 6.3.1, because of the limitations of the KAMA tool, the model content was manually created in the EMF environment. By using OCL, we have acquired list and number of each type of model element present in the model and these are compared with the original SGKS model.

Additional rules that exist in KAMA-DV metamodel are explained in Chapter 6 and their exact specifications are provided as OCL statements in Appendix B.2. The constraints are based on the specified formal properties given in Chapter 3 and Chapter 6. All of the constraints we listed for the sequencing package can be represented in OCL. While formulating these constraints some of the constraints given in the original metamodel [29] are modified to fit the syntax of the MDT OCL. We have provided the metamodel definition of KAMA-DV used for this case study in Appendix B.1.

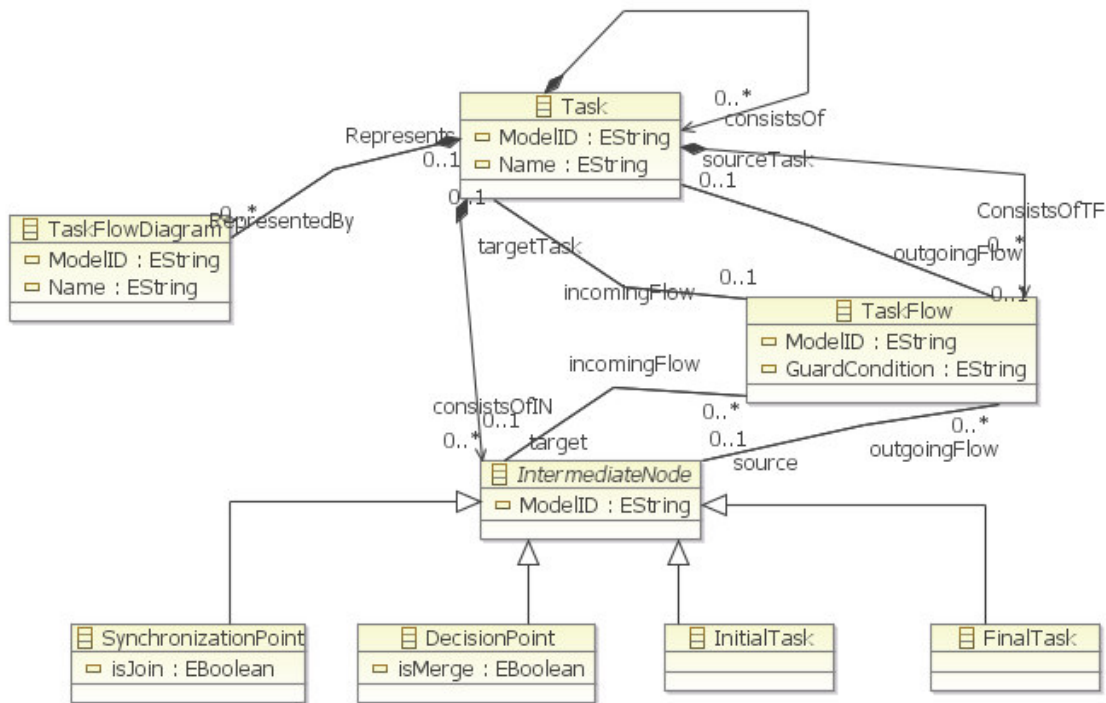


Figure 7-1 KAMA-DV Model Elements for Sequencing Expressed as an Ecore Diagram

After defining the model development based on the metamodel, the constraints related to KAMA and KAMA-DV are checked on the SGKS Surveillance Mission. The unsatisfied constraints are displayed as errors in popup window and error panel. The error pane enables to directly browse through the model elements in the context of which OCL constraint violation occurred.

A significant number of errors are detected in this process in KAMA Surveillance mission even though it had formerly been validated by the KAMA tool. The list of errors found after initial checking is presented in Appendix B. We will discuss the nature of these errors in the following paragraphs.

The multiplicity constraints defined for associations are enforced in the *ecore* model during model development, for instance only one incoming and outgoing *taskFlow* is allowed for a *task*. So if a *task* is associated with more than one *taskFlow* the prior participation is automatically cancelled. Hence only one end of *taskFlow* is present and this error is output by the environment.

MDT OCL has some limitations when checking the errors. In formulating the metamodel constraints we have modified original constraints to fit the MDT OCL environment.

The Errors Arising from Violation of KAMA Metamodel Constraints

There are errors that are found by the OCL validator that are not found by KAMA tool (See Appendix A.5). These include the multiplicity of incoming *taskFlows* and outgoing *taskFlows* of a *task* and existence of guard conditions for *taskFlows* outgoing from a *decisionPoint*, multiplicities for fork and join type *synchronizationPoints*.

The Errors Arising from Violation of KAMA-DV Metamodel Constraints

Additional constraints defined for KAMA *taskFlow* lead to easier interpretation of errors by providing the exact source of errors. The cumulative error list that includes both those stemming from KAMA and KAMA-DV constraints are given in Appendix A.6.

Among the errors *HangingTask* and *FlowDeadEnd* constraints enable us to identify the *tasks* in which no association with incoming *taskFlow* but have outgoing *taskFlow*, and vice versa.

Additional errors stemming from the properties listed in previous chapter are also found. These include the existence of *initialTasks* and *finalTasks*, sequential occurrence of *intermediateNodes*, loops formed by *taskFlow* associations without any *task* element. The model has a very high number of sequential *intermediateNodes*. There are also occurrences of loops without *tasks*.

All these errors are listed and discussed with the domain experts. The experts came up with recommendations that are used for the improvement of the approach, especially by modifying KAMA-DV metamodel and constraints.

As a matter of fact practical application lead to redefining of the rules in the metamodel to increase effectiveness of detection.

Finally the results were also compared with the results of the inspection case study by Tanriover [37]. A summary of errors found by KAMA-DV and corresponding errors found by inspection is given in Table 8-1. In the context of Surveillance mission 18 issues were found by manual inspection. The KAMA-DV indicated problems for all of these issues. The semantic issues are not found directly, at least one structural issue is found related to the semantic issue.

7.2.3 Evaluation of the Case Study

The case study allowed us to develop KAMA-DV further and provide new insights that are relevant for further study. These insights stem from application of the process and inspection of error reports, comparison with manual inspection results, and discussion of the reports with the experts.

In the initial discussion with the experts there was output from the experts on the KAMA-DV metamodel. Based on their feedback, the KAMA-DV is updated and first implementation of automatic verification is developed.

In the first case study, it was revealed that the rules presented in KAMA metamodel are not enough for an automatic verification in terms of locating the errors. There are two distinct kinds of deficiency: The first deficiency occurs as there are some underspecified issues in KAMA which KAMA-DV is based on in terms of structure and dynamics. The second deficiency occurs because of the resolution of KAMA constraints, that is the number of model elements and attributes the rule is related with. In the case of the first deficiency, *consistOf* relation is added to the metamodel for representing the *task* hierarchy observed in Surveillance metamodel and is used to relate different *tasks*. In the case of the second deficiency, the resolution of the rules is increased by partitioning them.

During the case study it is also observed that tool support is crucial for metamodel based development and verification. For instance there is limited interoperability based on XMI because of evolution of the standard, integration of UML and OCL tools, usability of graphical modeling environments and the limitation of OCL tools in terms of the OCL standard. One important point is that it is highly desirable for these tools to have capability for advanced configuration management and prioritization of checking of the rules.

One important limitation of the case study was that the surveillance model was not originally developed using KAMA metamodel but using templates in textual format. We limit error checking to the *taskFlow* elements, namely those that are related to sequencing which is the most detailed part of this case study.

7.2.4 Improvements of the Approach Based on the Recommendations

During the first case study, there were some recommendations about the approach in general and metamodel and rules in particular. In this subsection we discuss the suggestions about the metamodel and improvements made based on the first case study.

i. Representation of the constraints in a more concise way:

Once the metamodel is represented in Essential Meta Object Facility (EMOF) for checking the model, the rules are written in a most concise way. Also the lists that are originally attributes of some of the model elements are no longer represented as first class attributes as they can simply be formed using OCL queries.

The correspondence between the constraints and formal metamodel is increased by writing all constraints not using the lists, also the lists that are referred by some of the constraints in original KAMA are left out in KAMA-DV.

ii. Increasing the resolution of constraints to increase exact error detection

Constraints are divided to increase the resolution and better locating the errors. For instance when the constraints that involve both upper and lower multiplicity are used, it is very hard to understand the nature of errors.

For some of the recommendations we took no action in the scope of this work. We will discuss some of the suggestions about which we took no action in the scope of this thesis in the following paragraphs.

i. Expressing the constraints in First Order Logic (FOL)

The constraints can be expressed in FOL using a similar approach to Tanriover[37], but in the case of structural checking and dynamic checking such representation does not bring any benefits. If logic based approaches will be used such representation may be used.

ii. The Naming of consistOf Association

The considered names for this relation are *delegate* or *relay*, but there is no similar UML element. A more precise naming for *consistOf* is searched in KAMA-DV as if the subtasks are not related to the *task* with a strong relation like aggregation. In different *Task Flow diagrams*, *tasks* can be composed of different *tasks*, and these *tasks* can be utilized yet in other diagrams. The *delegate* connector type is of components language unit and has far more enriched semantics. Meaning of *relay* is also much different so they are left out for now and *consistOf* is used for describing the relation between *tasks* in KAMA-DV. For instance change of the name will lead to better understanding of the relation between model elements and error messages, but not the number and type of errors.

iii. Use of Alternative Stereotype Constructs for Model Elements

Another suggestion was to use the structured activity and related elements for some of the model elements as stereotypes. These are not used, as they include more specific features which are not included in the KAMA metamodel. However some of the constructs can be used if more refined conceptual models are available. For instance details of execution can be used to specify ConditionalNode, LoopNode, or SequenceNode elements. Moreover descriptions of multiple executions of *tasks* can be mapped to expansion regions.

7.3 Case Study 2: Engagement Task

In the second case study we aim to test the KAMA-DV approach as matured after the first case study. The second case study is performed on the engagement task of a Naval War Game simulation. The conceptual model was developed using KAMA approach and Enterprise architect tool.

7.3.1 Case Study Design

In this case study we aim to test the effectiveness of the KAMA-DV so the research questions are different from the ones of the first case study. There are two research questions: one about the effectiveness of the approach in finding structural errors, and the second on the capability of the method for finding dynamic errors.

- i. Can KAMA-DV approach be effectively used for finding the structural errors in conceptual models?
- ii. Is KAMA-DV capable of finding dynamic errors in conceptual models?

In contrast to the first case study, naval war game simulation was developed using the KAMA approach and cruise and engagement *task* and 8 subtasks were described using *task flow diagrams* defined in the metamodel. The engagement task is different from the definitions of surveillance missions and other missions in the first case study. Cruise subtask has detailed description of cruise in land, sea, and submarine. In total account 10 *task flow diagrams* were used in inspection, which include 22 % of all 45 *task flow diagrams*.

Subjects are two modeling experts similar to first case study. KAMA-DV approach is used first to find structural errors and then dynamic errors. After these issues are found, subjects are asked to review the errors found by the environment. Also the structural issues are compared with previous results that are found by manual inspection.

7.3.2 Results of the Case Study

The results of the case study were unexpected as the automated verification found syntax errors in the model which was verified before. In addition to syntax errors there were In comparison to the former study by Tanriover [35], some of the detected issues are signals for the errors indicated by inspection.

We have provided corresponding error occurrences for 5 diagrams in Table 7-3, which contains not only syntactic errors but also semantic ones. In engagement *task* we have encountered total of 83 errors and issues. Detail examination of the results indicated that there is either an error or an issue that corresponds to errors found in manual exception. There are very few cases where the errors found in manual inspection are not associated with an error or an issue found by the automatic approach. The results indicate that the errors are less related to KAMA syntax but more on the structural issues added by KAMA-DV and semantic issues.

Table 7-3. Number and Type of Errors Found in Various DHS Tasks

Task Name	Number of Elements	Number of Errors	Number of Issues	Number of Errors found in Manual Inspection
Cruise	21	5	5	1
Advance According to Cruise Plan	54	7	8	N/A
Advance in Air	43	1	5	2
Engagement	23	5	1	N/A
Engagement of Guided Ammunition	72	3	15	3

For comparison with manual inspection, we included a comparison of errors reported by Tanriover for the given *tasks* considering the *intradiagram* issues, issues found by singular inspection of *task flow diagrams*. In some of the number and types of errors found in various diagrams found by the KAMA approach are compared with inspection approach.

Table 7-4. The Corresponding Errors and Issues

Task Flow Diagram (See [37] Appendix D)	Manual Inspection	KAMA-DV
Advance in Air	Possible deadlock	Sequential <i>intermediateNodes</i>
Cruise	Deadlock	Multiplicity of incoming <i>taskFlow</i> of <i>synchronizationPoint</i> of type <i>Fork</i>
Scan for Mines	Deadlock	Multiplicity of incoming <i>taskFlow</i> of <i>synchronizationPoint</i> of type <i>Fork</i>
Engagement of Guided Ammunition	-Dangling <i>tasks</i> occurrence -Deadlock Decision nodes -Multiplicity of <i>synchronizationPoint</i> of type fork	- <i>finalTask</i> Multiplicity -Sequential <i>decisionPoints</i> - Multiplicity of incoming <i>taskFlow</i> of <i>synchronizationPoint</i> of type <i>Fork</i> -Sequential <i>intermediateNodes</i>

Dynamic Analysis

In the very few cases where structural errors are limited and simple, these errors can be corrected to perform dynamic analysis. Dynamic analysis is performed by transforming the *task flow diagram* to an equivalent EPC and analyzing using EPC tools [59].

For some of the diagrams we have corrected the structural errors and applied dynamic analysis. For the 10 of 5 diagrams the number of issues was limited so we applied dynamic analysis in the case of these diagrams. For the case of 3 diagrams that show structural similarity, the diagrams are not sound. The EPC corresponding to “Advance in Air” *task* is shown in Figure 7-2 in which two of the distinct parallel branches prevents occurrence of valid execution sequence. By experimenting with the tool shown in Appendix A.5, the exact branches that prevent execution are determined. We have also discovered that the manual inspection also reported the same issue as a semantic error. We have identified that even if these errors can not be attributed to violation of syntax or structural rules, they can be detectable by soundness analysis.

In a few cases we have observed very little number of syntax errors found by KAMA-DV. In general the number errors and issues for diagrams are considerable in spite of the former verification efforts. Even the KAMA metamodel is used during the development of this model, the limited support of the model development tool for defining and

checking constraints and limited effectiveness of manual inspection finding structural errors entirely causes error prone models.

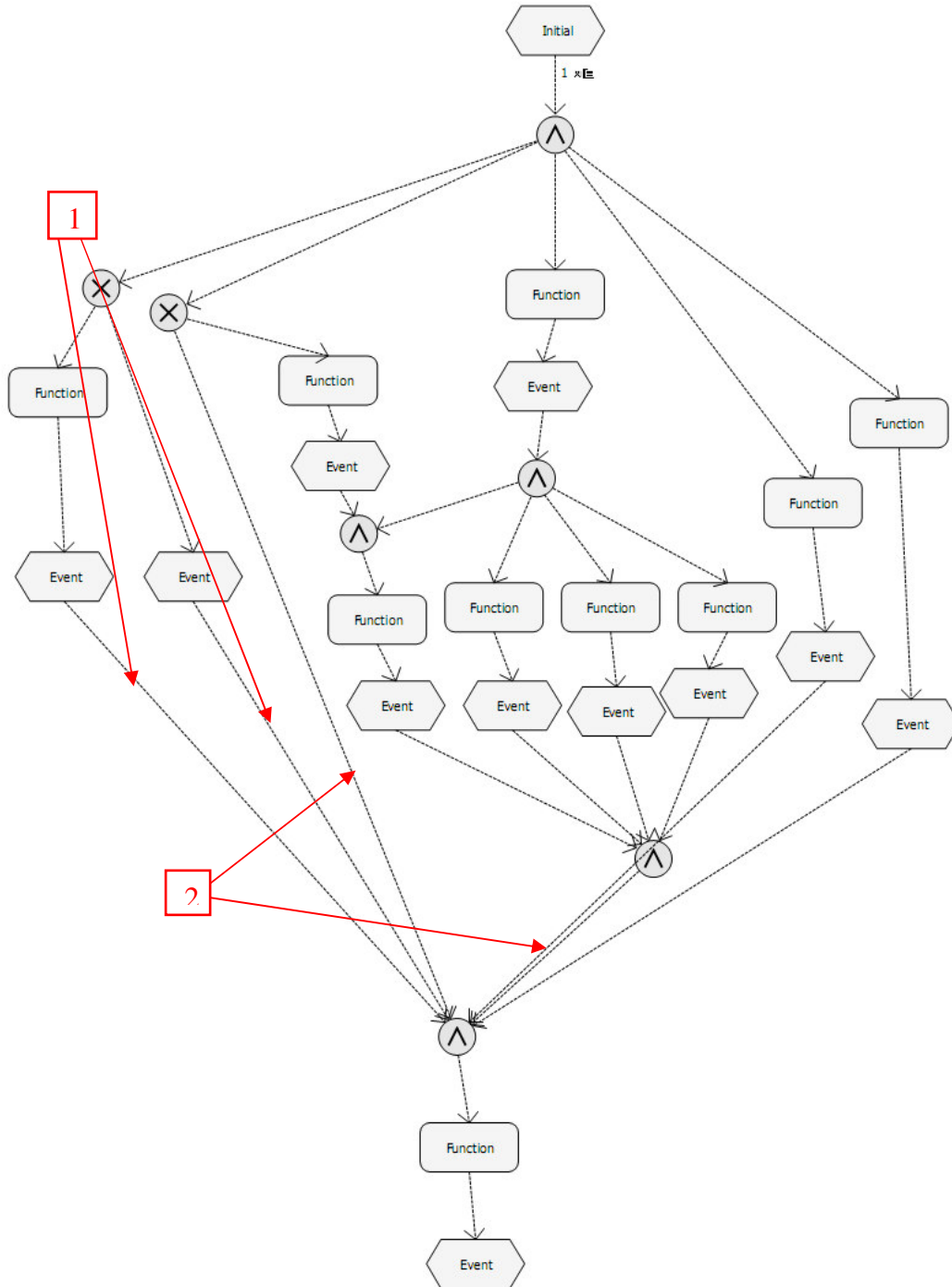


Figure 7-2. The Equivalent EPC of “Advance in Air” Task Flow Diagram that is not sound

7.3.3 Evaluation of the Case Study

Results of the case study results show that structural and dynamic verification can locate the dynamic errors within the scope of the DHS conceptual model. The errors found in the model are clearly errors but issues may either signal an error or not. Before dynamic verification the models need to be strictly error free for dynamic method to be applied. The results show that the approach is effective in finding errors that could not be found by modeling tool or manual inspection.

Some limitations of the case study deserve being mentioned. One of them is the unavailability of domain experts during validation so some errors were corrected based on consent of the modeling experts that were present in model development. In case of a high number of errors being present which may be attributed to high number of model elements in a diagram, the diagram has simply been excluded from dynamic analysis.

Finally we would like discuss the validity of the two case studies. The performed approach is based on the criteria discussed with model experts which contributed to construct validity. The internal validity of the approach is higher as for both domain conceptual models similar process is followed and results correlate.

However the conceptual models of simulation systems may vary a lot based on the application domain. Conceptual models may include a large number of different constraints (for instance mathematical models, time information etc.) that have not been tested in our studies. Also during the case studies the severity of errors were discussed with the experts but their implications if they are left in the model have not been observed during the actual simulation projects. These issues limit the external validity of the case studies carried out.

CHAPTER 8

DISCUSSION AND CONCLUSION

In this thesis, we have aimed to verify the dynamic properties of conceptual models for modeling and simulation. We have dealt with models that are developed at the early stages in the simulation development process. The modeled concepts are real world missions, tasks, related roles, objectives, and work products which are represented by corresponding model elements described by the KAMA-DV metamodel.

We have grouped relevant past research into three, which are mentioned in order of relevancy: First, metamodel based conceptual modeling in general, and the KAMA approach in particular provides the main pillar of our research. Secondly, research dealing with semantic variations of UML activity models and verification of such models is relevant in the sense that the semantics of model elements for sequencing of *tasks* in KAMA show similarities with their UML counterparts. However, differences in size of models and semantic variations and the limitations of past work in that area justify a novel approach. Finally, verification of conceptual business process models, particularly the ones expressed by Enhanced Process Chains (EPCs) are relevant for our research. In the next section we will examine the relation of our research with these three related areas in detail.

8.1 Contributions

In this work we have defined a formal semantics for KAMA metamodel. The defined semantics for conceptual models focus on definition of KAMA metamodel explained in Chapter 3 and extension explained in Chapter 6. Further more the KAMA-DV approach is realized by tools supporting model driven development, Eclipse modelling environment.

We also explored the dynamics of KAMA model elements more thoroughly and advised used of an execution framework. The execution framework deals with some variances that are encountered in conceptual modes that are not present in similar frameworks. The realization of such a model execution approach is left as a further study topic as explained in Section 8.3.

We used the KAMA metamodel definition in verification of two real life conceptual models. conceptual model. Although KAMA metamodel verification definition consists of rules that are not very complex, during verification we have found large number of errors in previously checked models. The case study results indicated that for these cases the KAMA-DV approach is applicable and effective in finding the errors in dynamics of conceptual models. KAMA-DV issues relate to syntax and semantic errors found by manual inspection.

At the final stage of verification, dynamic analysis is performed for the cases where supporting tools are adequate. Only by using the syntax and structural checks, the models that can be dynamically verified can be obtained. The dynamic analysis is applied based on the soundness definition for EPCs. The results suggested that for some of the task flow diagrams this can be used to find deadlocks in the diagrams. Moreover the corresponding tools can be used to simulate the diagrams indirectly by simulating the corresponding EPC.

However the soundness analysis and simulation can be used if number of errors are low in a diagram and appropriate conditions hold. There are two main reasons for this. First is that only for certain kind of diagrams the EPC analysis is applicable because of the semantic variations. Secondly the high number of structural errors has limited the number of diagrams that can be analyzed. Our work has layed ground for syntactically correct models, so more structured models that can be analyzed will be available in the future.

Below, after briefly comparing our approach to the related work, we highlight the further research topics related to our approach. Some of the research topics deal with bottlenecks of our approach, that is, the areas where our approach has limited capability and others deal with extending the scope of the work.

8.2 Accomplished Work

8.2.1 Verification Approach Based on Formal Techniques for Metamodel Based Conceptual Models

In this thesis, we have focused on the dynamic descriptions in metamodel based conceptual models, specifically mission space package of the KAMA modeling language, which includes elements to describe missions, tasks and their relations. The mission space is presented as a structural view in the KAMA metamodel, however it provides information on sequencing, synchronization, and branching of tasks which can be used in understanding the behavior of the system. We describe an approach including a process and set of methods to be used in inspection of dynamics of these models. The inspection process will be iterative and will provide the user with online and offline verification methods. For increasing the power of analysis we extend mission space models to cover intended but not explicitly specified behavior descriptions. While keeping the original conceptual model intact, our mission space execution is based on this extended model. These extensions which do not exist in the original KAMA metamodel, are defined based on the dynamic properties explained in Chapter 3 and 4. We utilize approaches that were used for verification of Enhanced Process Chain Diagrams (EPCs) based on Petri nets

during this process. As a result we aim to have a conceptual model consisting of consistent and correct behavioral information.

The developed conceptual modeling notation of KAMA provides the modeling elements for these but lacks methods and tools for analysis and verification of the dynamics. Tanriover and Bilgen[36] provide an inspection approach for conceptual models developed in a domain specific notation. The approach includes checklists for interdiagram issues and for each diagram type. The checks are listed for *initialTasks*, *synchronizationPoints* (fork and join type), complex structures (fork and *decisionPoints*), flow to the *finalTask*, loops and *workProducts* based on the soundness property of workflow nets. In this work we only deal with the dynamics of conceptual models and elaborate their dynamic behavior thoroughly. We define extended properties for some of the model elements and structured conceptual models, soundness and relative soundness for conceptual models. We use these definitions to develop our approach based on formal and interactive verification.

Checks on metamodel based conceptual models are classified by Tanriover as intra diagram and inter diagram checks. A detailed step by step inspection method is provided for each diagram for intra diagram checks. More than one type of diagram are considered for inter diagram checks.

In our approach, we formally define consistency checks in terms of set theory. Some of these properties not only allow a structured check but also enable checking of the dynamic model. The semantic variations in *task flow diagrams* also extend beyond the semantics of classical Petri nets and activity models which is further elaborated in this thesis especially in Chapter 4 and 5. At last based on this extended approach, we have defined KAMA-DV metamodel that includes semantic properties related to dynamics.

Our contribution includes specification of conceptual model dynamics in a formal language, characterization of the dynamics of the conceptual model, and using formal methods to verify the conceptual models.

Table 8-1. Errors Found by Inspection and Related Errors in KAMA-DV in Surveillance Mission

Issue Found	Error Found by Inspection	Error Found By KAMA-DV	Number of Similar Errors
2.1 Perform Coverage Analysis and Planning	Only one outgoing <i>taskFlow</i> from <i>decisionPoint</i>	See error 53 in Appendix A.6.	2
2.2.3 Detect the Targets	No Outgoing <i>taskFlow</i> for 5 <i>tasks</i>	See error 80, 81, 82, 83, 84 in Appendix A.6.	3
2.2.1 Recognize and Identify	More than one incoming <i>taskFlow</i> to fork node after <i>initialTask</i>	See error 1,6 in Appendix A.6.	8
2.2.1 Recognize and Identify	A potential for deadlock as the fork node after <i>initialTask</i> is connected to <i>decisionPoint</i> before <i>finalTask</i>	See error 12 in Appendix A.6 .	2
2.1.2 Perform Pre-Analysis of Coverage	“Determine Priority of Point and Path” <i>task</i> should be modeled as two distinct <i>tasks</i>	None	1

By using automatic checks, we have found the issues that cover the errors related to sequencing *tasks* except one type of semantic error found by inspection approach related to sequence of *tasks* in Surveillance *mission*. Issues that are related to inspection results are summarized in Table 8-1. In fact, by using the automated approach we have identified more issues related to *task flow diagrams* using KAMA-DV and each issue is singled out with the related elements. On the other hand, for some undetected semantic errors, the issues related to syntax can be used as a signal. In Table 8-2, we show the correspondence of the checklist formulated by Tanriover [36] with our approach.

Table 8-2. The Correspondence of Consistency Checks with KAMA-DV

Consistency Check		KAMA-DV
1. Check for syntactic errors such as dangling nodes, initial nodes with more than one outgoing transitions.		Constraints are formulated in set theory. Also the syntax checks are extended to include dynamic model.
2. Identify decision nodes		Not only the decision nodes are identified but also their nature, being structured or not, the matched merge node, and matching condition.
	2.1. Check if all flows outgoing from the decision nodes have guards	Syntax check that is defined in Chapter 3.
	2.2. Check the constraints on the guards to make sure that they do not overlap (overlapping such as constraint on one guard is $x \geq 0$ and on the other $x < 0$)	Syntax check that is defined in Chapter 3.
	2.3 Check if the guards define a complete set (such as $x \geq 0$ and $x < 0$)	Syntax check that is defined in Chapter 3.
	2.3.1 Identify overlapping and incomplete conditions	Syntax check. How the guard is defined.
3. Identify Fork Nodes		Not only the fork nodes are identified but also their nature, being structured or not, the matched join node, and matching condition.
	3.1 Check if the fork node has only one entrance, if not make sure that a task-flow is not missed before the flow is joined.	Syntax check that is defined in Chapter 3.
	3.2. Check if all the flows from the fork node are joined by a (same) join node (non-structurally joined nodes or fork nodes may indicate concurrency problems)	Structural and matched flows are identified. DV is used find candidate violations.
	3.2.1. If not, run the localized flows (flows coming out of the fork node) with UML's activity diagram (Petri nets like) control flow semantics	DV is used find candidate violations.
	3.2.2. Identify livelocks and their causes.	DV is used find candidate violations.
4. Identify join nodes		Not only the join nodes are identified but also their nature, being structured or not, the matched fork node, and matching condition.
	4.1 Check if join nodes have only one exit transitions.	Syntax check that is defined in Chapter 3.
	4.2 If not, it is possible that the join node is placed too early; there is possibility that there is still a need for a parallel flow.	Not relevant as 4.1 is not allowed by syntax.
	4.3. Trace incoming transitions of the join nodes to make sure that all may eventually be activated.	DV is used find candidate violations.
	4.4. If not, identify causes of deadlock	DV is used find candidate violations.
5. If the task-flow is complex (includes more than one fork node or composite decision nodes) trace each flow from the start to end.		Soundness criteria based on characteristics of task flows.
	5.1. Make sure that every task may execute.	DV is used find candidate violations.
	5.2. Identify dead tasks.	DV is used find candidate violations.

Table 8.2 (cont.). The Correspondence of Consistency Checks with KAMA-DV

Consistency Check		KAMA-DV
6. Trace the flows reaching the final nodes		DV is used find candidate traces.
	6.1. Make sure that they do not originate from a fork node.	DV or syntax check can be used find candidate violations.
	6.2. If they do, there is a possibility that some activities will terminate abruptly, try to identify such activities.	DV or syntax check can be used to find candidate violations.
7. Identify loops by tracing through transitions.		DV can be used to identify loops.
	7.1. Run the localized loop with UML's activity diagram (Petri nets like) control flow semantics.	DV can be used to find candidate violations.
	7.2. Identify livelocks and their causes.	Livelocks can be identified using soundness analysis.
8. Identify activities with <input> and <output> entities (An entity may be attached to a task according to the definition of KAMA notation)		Inputs and outputs can also be analyzed using DV.
	8.1. Make sure that if tasks use outputs of one another, they also follow the implied sequence in the control flow because a produced entity may be an input for another task, causing the task to never start or to prevent parallel flow.	Static check is applied before dynamic analysis. Dynamic analysis is performed using information on properties of workproducts such as consuming.
	8.2. Identify deadlocks and redundancy.	Deadlocks and livelocks can be identified using soundness analysis.

8.2.2 UML Activity Verification and Conceptual Business Process Verification

We have adopted the models used for verifying Conceptual Business Process Descriptions for conceptual models. Semantic variances in business process models can be handled by Petri net based models so that the analysis is still possible.

In our work we have researched the adequacy of methods to analyze EPCs in terms of conceptual models. For instance EPC based methods are applicable for single diagram cases considering semantic variations. In order to analyze *task flow diagrams*, we have also extended the method to check EPCs for relaxed soundness to synchronization points of join type, decision points of merge type. For verifying a set of *task flow diagrams* we have to combine them into one which introduces additional problems. During this combination we have utilized information from mission space diagrams, *consistOf* relation, and *finalTasks*.

Definitions for soundness are adapted for conceptual models, which have different model elements and semantics than EPCs.

8.2.3 Comparison of Related Approaches

In Table 8-3, we have listed the dynamic aspects observed in conceptual models. The references to instances of these aspects in section 4.2 are also provided. For comparison we also include the equivalents of these aspects in EPC and UML activity language.

During comparison with UML activity package [27], we only consider basic constructs, given in UML *FundamentalActivities* and *IntermediateActivities* packages, as they include more abstract modeling elements suited for conceptual modeling. If UML activity package is taken as a whole, for instance for the first aspect, *CallBehaviourAction* in an activity can provide a mechanism to trigger other activities during execution. For EPC, we used the definition provided by Mendling [61].

We do not limit the models to include only elements either of EPC diagrams or UML Activity Package. Even if such modeling can be possible for some of the cases, this will decrease the understandability and communicative power of conceptual model and will contradict with the properties of a conceptual model given in section 4.1.

We have provided a general framework to check conceptual modeling diagrams one by one considering their semantic differences and extensions. The semantic differences include the triggering mechanism, multiple instantiation, and multiple final nodes. Furthermore, we have developed a method to verify a set of diagrams using information in *mission space diagrams* and *consistOf* relation. For application in specific models we have utilized existing tools on soundness analysis.

Table 8-3. Comparing of EPC, UML and KAMA in Terms of Aspects in Conceptual Models

Reference	Aspect	EPC	UML Activity	KAMA
T-1	Triggering of next task before task completion	Not possible.	Not possible (Exception streaming input and output pin)	Possible
T-1	Triggering of successor task with delay	Not possible	Not possible.	Possible
T-1	Multiple instances of tasks	Not possible	Possible	Possible
T-1	Multiple triggering of successor task	Not possible	Not possible	Possible
IN-1 (default case)	More than one initial node	Possible	Not possible	Possible
FN-1 (default case)	More than one final node	Possible	Not possible	Possible
T-1	Cancellation of an active task	Cancellation region extensions.	Possible through use of actions	Possible
T-1, FN-O-1	Cancellation of group of tasks	Cancellation region extensions	Possible through use of actions	Possible, no restriction exists
FN-O-1, FN-O-2	Termination of group of tasks	Completion states	ActivityFinalNode	FinalTask
FN-1 (default case)	Partial completion of a task group	Not permitted	Possible through use of ActivityFinalFlow	Possible, no restriction exists.
S-O-4	Synchronization of active parallel branches	OR-join	Possible through use of actions	Possible, no restriction exists
S-O-5	Interaction of synchronized branches	Possible, no restriction exists	Possible through use of actions	Possible, no restriction exists

8.3 Directions for Further Research

In this section we will ponder upon further research topics related to this study. The novelty of the approach stems from both the metamodel based development in conceptual modeling and availability of limited support in terms of techniques and tools. First we will list the areas related to provide better syntax and semantics for conceptual models and development of techniques related to dynamic verification of these. Then we will list more general research topics related to model based development that also concern this study.

8.3.1 Better Syntax and Semantics Definitions for Conceptual Models

One of the limitations of our research was lack of well defined approaches for developing conceptual models. As of now only a couple of approaches exist for conceptual model development, among which KAMA [29] and BOM [10] are prevalent. We have examined and provided the limitations of these approaches in Chapter 2.

In the course of this study, although we have built our dynamic verification approach upon KAMA approach, we have observed some semantic and syntax issues related to KAMA. These issues are well expected as the modeling concepts can only be clearer as in the case of state modeling concepts of UML over time and by experience. Still a lot of work needs to be done in conceptual modeling to arrive better notations and metamodels. Semantic definition is difficult as the scope of conceptual model is multiple and development and verification activities are interwoven.

Throughout the study we have observed that there is limited supply of real life of models that conform to specific development methods like BOM or other UML based approaches such as KAMA. New applications of conceptual models will reveal novel problems related to model development and verification as we have observed in the course of this study. By both definition and through examples these approaches can be enhanced.

In this work for defining KAMA-DV we have utilized the dynamic definitions of KAMA. A possible future work is application of this method to other metamodel based conceptual modeling approaches. Alternatively in the course of a future study, conceptual model syntax and semantics can be developed with initial concern of verification and all the information can be supported by built-in verification components.

8.3.2 Execution of Conceptual Models

Throughout this study we have worked on the structural and dynamic verification of conceptual models. We have observed that there is a great deal of variability in conceptual model semantics compared similar semantic descriptions explained in the previous section. A detailed analysis of conceptual models can be made using a detailed

framework that supports execution of this semantics. The analysis may provide a basis for forming modified and detailed descriptions for dynamic properties such as soundness.

Another topic to be addressed is how to build the necessary verification capability. Some of the alternatives for realizing the capability can be obtained by using more executable frameworks like executable UML [39], timed UML [77], and other UML based methods for the analysis. As there are semantic differences, adaptations of these are further research topics to be addressed.

As such research will need a great deal of initial effort, it is better to focus on determining essentials of behavior verification first and developing methods focusing on these later. For the former detecting essential behavior errors in simulation software and prioritizing behavior verification requirements is a key ingredient.

8.3.3 Tool Support Throughout the Verification Lifecycle

In our case, structural and dynamic verification is supported by the KAMA tool [35] and Eclipse Model Development tools [58]. Many obstacles related to tool support in terms of model definition, editing, visualization and verification still exist in Model Driven Development. The support for both of these tools has some limitations in terms of model driven development. As tools supporting methods have limitations in terms of support for modeling standards and usability, tool development is an area of research with high potential.

A major concern is the interoperability of the tools for better exploitation of functional capability of each tool. Moreover, the interoperability of tools shall be built such that a continuous traceability from conceptual models to executable models is possible. By this way the development process will be more streamlined, transparent and traceable for modelers, system designers, developers, and subject model experts. In this sense, aligning of methods and increasing interoperability of tools are further research topics.

8.3.4 Application of Other Techniques

The focus of this thesis has been on core verification activities that are related to sequential relations between tasks. Based on the specific type of simulation project, conceptual model verification issues may be different. Although there is significant potential of application of logic based approaches on the analysis of dependencies between *tasks*, resource perspective (*actors* or *workProducts* may be regarded as resource) and data flow analysis, and some research exists for various kinds of system and software models, for conceptual models the available research on logic based approaches is rare if at all existent. So other verification approaches can be coupled with dynamic descriptions for thorough analysis of conceptual models.

8.3.5 Quantitative Analysis of Issues Considering Model Metrics

One obvious extension of the present study would be to alleviate the threats especially to external validity as discussed in Section 7.3.3. This would involve multiple implementations of our approach on real life conceptual modeling work followed by actual simulation exercises. Following a number of such studies, confidence in the validity of our approach would definitely be enhanced, possibly following further adjustments at various levels of detail.

Furthermore, in the presented case studies, we have not worked thoroughly on the distribution of the errors in a model. On one hand, there are indications of severe errors effecting the simulation to be developed, and on the other, simply aesthetic problems related to diagram visualization. Given the limitation of available effort for model verification, it is desirable to use the effort effectively. An important aid may be provided by severity metrics and patterns that can be related to errors in the model. Such work can use previous work on model metric definitions and their utilization for UML [82][84]. Also, EPCs [61] provide a starting point for quantitative analysis of conceptual model errors.

REFERENCES

- [1] Pace, D.K., (2000). Simulation Conceptual Model Development, Proceedings of the Spring 2000 Simulation Interoperability Workshop, Orlando, FL, CD. March 26-31.
- [2] Sargent, R.G., (1991) "Simulation Model Verification and Validation" , Proceedings of the 1991 Winter Simulation Conference.
- [3] simulation. (2010). In Merriam-Webster Online Dictionary. Retrieved March 25, 2010, from <http://www.merriam-webster.com/dictionary/simulation>
- [4] "DoD Glossary of Modeling and Simulation (M&S) Terms", DoD 5000.59-M, 29 August 1995.
- [5] Fujimoto, R.M., (1999). Parallel and Distributed Simulation Systems, 1st Edition, John Wiley & Sons, Inc. New York, NY, USA.
- [6] Sheehan J., Prosser T., Conley H., Stone G., Yentz K., and Morrow J., (1998). Conceptual Models of the Mission Space (CMMS): Basic Concepts, Advanced Techniques, and Pragmatic Examples," 98 Spring Simulation Interoperability Workshop Papers, March 1998, Volume 2, pp. 744-751.
- [7] IEEE Std. 1516.3-2003: IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP), 23 April 2003.
- [8] IEEE Std. 1516.4-2007: IEEE Recommended Practice for Verification, Validation, and Accreditation of a Federation—An Overlay to the High Level Architecture Federation Development and Execution Process, 20 December 2007.
- [9] VV&A Recommended Practices Guide Build 3.0, Special Topic: Conceptual Model Development and Validation, <http://vva.msco.mil/>, 15/09/2009.
- [10] SISO-STD-003-2006: Base Object Model (BOM) Template Specification (approved 8 May 06)
- [11] Moradi, F., Ayani, R., Mokarizadeh, S., Shahmirzadi, A. G. H., Tan, G., (2007). A Rule-based Approach to Syntactic and Semantic Composition of BOMs, Proceedings of the 11th IEEE Symposium on Distributed Simulation and Real-Time Applications, 2007.

- [12] SISO-STD-003.1-2006: Guide for BOM Use and Implementation (approved 8 May 06)
- [13] Mojtahed, V., Lozano, M. G., Svan, P., Andersson, B., and Kabilan V., (2005). DCMF-Defence Conceptual Modeling Framework, Systems Technology Methodology Report Number FOI-R--1754--SE, ISSN 1650-1942, FOI-Swedish Defence Research Agency, November 2005.
- [14] THALES JP 11.20-W3300: Common Validation, Verification and Accreditation Framework for Simulation (REVVA) Guidelines for VV&A Techniques.30 August 2004.
- [15] THALES JP 11.20-W3300: Common Validation, Verification and Accreditation Framework for Simulation (REVVA) VV&A Criteria Definition (CRIT) V1.1, 10 November 2004.
- [16] Barlas, Y., (1996). Formal Aspects of Model Validity and Validation in System Dynamics, System Dynamics Review.
- [17] Kleijnen, J.P.C., (1995). Verification and Validation of Simulation Models, European Journal of Operational Research.
- [18] Slepoy, A., Mitchell, S. A., Backus, G. A., McNamara, L. A., Trucano, T. G., (2008). R&D for Computational Cognitive and Social Models: Foundations for Model Evaluation through Verification and Validation (final LDRD report), http://www.osti.gov/bridge/product.biblio.jsp?query_id=0&page=0&osti_id=945901, last accessed on April 2010.
- [19] Topçu, O., Adak, M., Oğuztüzün, H., (2008). A Metamodel for Federation Architectures, ACM Transactions on Modeling and Computer Simulation, Volume 18 , Issue 3, July 2008.
- [20] Balci, O., (2003). Verification, Validation, and Certification of Modeling and Simulation Applications, Proceedings of the 2003 Winter Simulation Conference, pp 150-158.
- [21] 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 2001 Simulation Interoperability Workshop, Orlando, FL, CD, 2001.
- [22] Pace, D. K. and Sheehan J.. (2002). Subject Matter Expert (SME) / Peer Use in M&S V&V, V&V State of the Art: Proceedings of Foundations '02, a Workshop on Model and Simulation Verification and Validation for the 21st Century, Laurel, MD, October 22-24, 2002.

- [23] Brade, D.A., (2003), Generalized Process for the Verification and Validation of Models and Simulation Results, Dissertation, Fakultät für Informatik, Universität der Bundeswehr München. Neubiberg, 2004.
- [24] Brade D., (2003), Conceptual Modeling Meets Formal Specification, Proceedings of Spring Simulation Interoperability Workshop.
- [25] MDA Guide Version 1.0.1, Object Management Group (OMG), 12.06.2003.
- [26] Meta Object Facility (MOF) Core Specification Version 2.0, Object Management Group (OMG), January 2006, <http://www.omg.org/spec/MOF/2.0/PDF>
- [27] OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2, <http://www.omg.org/docs/formal/07-11-02.pdf>
- [28] OMG, Software & Systems Process Engineering Meta-Model, v2.0, <http://www.omg.org/cgi-bin/doc?formal/2008-04-01>
- [29] Karagöz, N.A, (2008). A Framework for Developing Conceptual Models of the Mission Space For Simulation Systems, PhD Thesis, Department of Informatics, Middle East Technical University.
- [30] Karagöz, N. A., Demirörs, O., Gencel, Ç., Ündeğer, Ç., (2006). Mission Space Conceptual Model Development in the Simulation Systems: A Process Definition, Proceedings of ODTÜ Savunma Teknolojileri Kongresi Ankara, 29-30 June 2006.
- [31] Civelek, M. (2006). Modeling A Sample Mission Space of TAF by Using KAMA-C4ISR MOS Notation, Technical Report, 2005-2006/2-21, Informatics Institute, Middle East Technical University.
- [32] 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.
- [33] 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.
- [34] Eryılmaz, U., Bilgen, S., Molyer, O., (2006). Verification and Validation Methods for Conceptual Modeling of the Mission Space, Proceedings of Savunma Teknolojileri Kongresi Ankara, 29-30 June 2006.
- [35] Eryılmaz, U., Karagöz, N. A., (2009). KAMA: A Tool for Developing Conceptual Models For C4ISR Simulations, Proceedings of 2009 European Simulation Interoperability Workshop, Istanbul, 2009.

- [36] Tanrıöver, Ö., Bilgen, S., (2007). An Inspection Approach for Conceptual Models in Notations Derived from UML: A Case Study, Proceedings of the ISCIS 2007, Ankara, Turkey, CD. November 7-9, 2007, ieeexplore.ieee.org.
- [37] Tanrıöver, Ö., (2008). A Verification Method for Simulation Conceptual Models in UML”, Ph.D. Thesis, Middle East Technical University, Ankara, September 2008.
- [38] Tanrıöver, Ö., Bilgen, S., (2010). Ch. 15. UML-Based Conceptual Models and V&V, in *Conceptual Modeling for Discrete-Event Simulation*, Edited by S. Robinson, R. Brooks, K. Kotiadis, D.-J. Van Der Zee, Taylor & Francis.
- [39] Mellor, S. J., Balcer, M. J., (2002). *Executable UML: A Foundation for Model-Driven Architecture*, Addison-Wesley Professional, The Addison-Wesley Object Technology Series.
- [40] Fowler, M., (2003). *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd edition, Addison Wesley, September 2003.
- [41] Luz, M.P., da Silva, A.R., (2004). *Executing UML Models*, WISME 2004.
- [42] Unhelkar, B., (2005). *Verification and Validation for Quality of UML 2.0 Models*, Wiley Series in Systems Engineering and Management, 2005.
- [43] Baker, P., Loh, S., Weil, F., (2005). Model-Driven Engineering in a Large Industrial Context — Motorola Case Study, Proceedings of Model Driven Engineering Languages and Systems 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005.
- [44] Schattkowsky, T., Forster, A., (2007). On the Pitfalls of UML 2 Activity Modeling, Proceedings of the International Workshop on Modeling in Software Engineering, p.8, May 20-26, 2007.
- [45] Drusinsky, D., (2008). From UML Activity Diagrams to Specification Requirements, Third IEEE International Conference on System of Systems Engineering.
- [46] Störrle, H., (2005). Semantics and Verification of Data Flow in UML 2.0 Activities, *Electronic Notes in Theoretical Computer Science* 127(4), pp. 35-52, April, 2005.
- [47] Störrle, H. and Hausmann, J., (2005). Towards a Formal Semantics of UML 2.0 Activities, *In Proceedings German Software Engineering Conference*, vol. P-64 of LNI, pp. 117-128, 2005.
- [48] Störrle, H., (2004). Semantics of Control-Flow in UML 2.0 Activities, *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*, pp. 235-242, September 26-29, 2004.

- [49] Dotan, D. and Kirshin, A. (2007). Debugging and testing behavioral UML models, in 22nd ACM SIGPLAN Conference on Object Oriented Programming Systems and Applications (OOPSLA), pages 838-839. ACM, 2007.
- [50] Sarstedt, S., Kohlmeyer J., Raschke, A., Schneiderhan, M., and Gessenharter, D., (2005). ActiveChartsIDE, Poster at ECMDA 2005, Nov 2005.
- [51] Crane, M. L. and Dingel, J., (2008). Towards a UML Virtual Machine: Implementing an Interpreter for UML 2 Actions and Activities, *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, pp. 96 – 110, October 27-30, 2008, Ontario, Canada.
- [52] Smialek, M., Bojarski, J., Nowakowski, W., Straszak, T., (2005). Scenario Construction Tool Based on Extended UML Metamodel, Proceedings of Model Driven Engineering Languages and Systems 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005.
- [53] van der Aalst, W.M.P., (1997). Verification of work-flow nets, in 18th International Conference on the Application and Theory of Petri Nets (ICATPN '97), volume 1248 of LNCS, pages 407-426. Springer, 1997.
- [54] Russell, N, van der Aalst, W.M.P., ter Hofstede, A.H.M., Wohed, P., (2006). On the Suitability of UML 2.0 Activity Diagrams for Business Process Modeling, Third Asia-Pacific Conference on Conceptual Modeling (APCCM 2006), Hobart, Australia. Conferences in Research and Practice in Information Technology, Vol. 53., Markus Stumptner, Sven Hartmann and Yasushi Kiyoki, Ed.
- [55] Staines, T.S., (2008). Intuitive Mapping of UML 2 Activity Diagrams into Fundamental Modeling Concept Petri Net Diagrams and Colored Petri Nets, in Engineering of Computer Based Systems, 2008. ECBS '08: Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp. 191–200, Wash-ington, DC, USA, 2008.
- [56] Davis, R., Brabander, E, (2007). Introduction to ARIS Platform, in ARIS Design Platform, Springer London.
- [57] van Dongen, B.F., van der Aalst, W.M.P., and Verbeek, H.M.W., (2005), Verification of EPCs: Using Reduction Rules and Petri Nets, O. Pastor and J. Falcao e Cunha (Eds.): CAiSE 2005, LNCS 3520, pp. 372–386, 2005.
- [58] Eclipse Model Development Tools Homepage,
<http://www.eclipse.org/modeling/mdt/>, last accessed on February 2011.

- [59] EPC Tools Homepage, <http://www2.cs.unipaderborn.de/cs/kindler/research/EPCTools/>, last accessed on March 2011.
- [60] Mendling, J., Verbeek, H.M.W., van Dongen, B.F., van der Aalst, W.M.P., Neumann, G., (2008). Detection and prediction of errors in EPCs of the SAP reference model, *Data & Knowledge Engineering* 64 pp. 312–329.
- [61] Mendling, J., (2008). Event-Driven Process Chains (EPC), J. Mendling (Ed.): *Metrics for Process Models*, LNBIP 6, pp. 17–57.
- [62] Mendling, J., (2008). Verification of EPC Soundness, J. Mendling (Ed.): *Metrics for Process Models*, LNBIP 6, pp. 59–102.
- [63] Dehnert, J., van der Aalst, W.M.P., (2004). Bridging the Gap between Business Models and Workflow Specifications, *International Journal of Cooperative Information Systems* 13(3), 289–332, 2004.
- [64] Wynn, M., T., (2009). Business process verification – finally a reality!, *Business Process Management Journal* Vol. 15 No. 1, pp. 74-92.
- [65] Kristensen, L.M., Mechlenborg, P., Zhang, L., Mitchell, B., and Gallasch, G.E., (2007). Model-based Development of a Course of Action Scheduling Tool, In *International Journal on Software Tools for Technology Transfer (STTT)* Springer-Verlag.
- [66] Benbasat, I., Goldstein D.K., Mead, M., (1987). The Case Research Strategy in Studies of Information Systems, *MIS Q* 11(3):369–386.
- [67] Runeson, P., Höst, M., (2009). Guidelines for Conducting and Reporting Case Study Research in Software Engineering, *Empirical Software Engineering* 14:131–164.
- [68] Gibbert, M., Ruigrok, W., Wicki, B., (2008). What Passes As A Rigorous Case Study?, *Strategic Management. Journal*, 29: 1465–1474
- [69] Appelbe, W., Stasko, J., and Kraemer, E.,(1993). Applying Program Visualization Techniques to Aid Parallel and Distributed Program Development, Georgia Institute of Technology, College of Computing, Technical Report GIT-GVU-91-08.
- [70] Kirshin, A., Dotan, D., Hartman, A., (2006). UML Simulator Based on a Generic Model Execution Engine, LNCS 4364, *Models in Software Engineering Workshops and Symposia at MoDELS 2006*, Genoa, Italy, October 1-6, 2006.
- [71] Sarstedt, S., Gessenharter, D., Kohlmeyer, J., Raschke, A., Schneiderhan, M., (2005). ActiveChartsIDE: An Integrated Software Development Environment Comprising a Component for Simulating UML 2 Activity Charts, *Proceedings of the*

2005 European Simulation and Modeling Conference (ESM'05), pages 66–73, October 2005.

[72] Richters, M. and Gogolla, M., (2000). Validating UML Models and OCL Constraints, UML 2000, LNCS 1939, pp. 265–277, 2000.

[73] Chiorean, D., Petrascu, V., Petrascu, D., (2008). How My Favorite Tool Supporting OCL Must Look Like, Electronic Communications of the EASST, Volume 15, 2008, Proceedings of the 8th International Workshop on OCL Concepts and Tools(OCL2008) at Models 2008.

[74] Westergaard, M., Lassen, K.B., (2006). The Britney Suite Animation Tool, Petri Nets and Other Models of Concurrency ICATPN 2006, LNCS 4024, pp. 431–440, 2006.

[75] Computer Tool for Colored Petri Nets Web page, <http://wiki.dimi.au.dk.cpntools/cpntools.wiki>, last accessed on January 2008.

[76] Letelier, P., Sanchez, P., (2003). Validation of UML Classes through Animation, Advanced Conceptual Modeling Techniques, LNCS 2784/2003.

[77] Ober, I., Graf, S., Ober, I., (2006). Validating Timed UML Models by Simulation and Verification, International Journal on Software Tools for Technology (2006) 8(2): 128–145.

[78] Ermel, C., Hölscher, K., Kuske, S., Ziemann P., (2005). Animated Simulation of Integrated UML Behavioral Models based on Graph Transformation, in M. Erwig and A. Schürr, editors, Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), Dallas, Texas, USA, September 2005.

[79] Taleghani, A., Atlee, J.M., (2006). Semantic Variations Among UML StateMachines, LNCS 4364, Models in Software Engineering Workshops and Symposia at MoDELS 2006, Genoa, Italy, October 1-6, 2006.

[80] Harel, D., Kugler, H., (2004). The Rhapsody Semantics of Statecharts (or, On the Executable Core of the UML), in Proc. of 3rd Int. Workshop on Integration of Software Specification Techniques for Applications in Engineering, volume 3147, pages 325–354. LNCS, Springer-Verlag, 2004.

[81] Genero, M., Piattini, M., Calero, C., (2005). A Survey of Metrics for UML Class Diagrams, Journal of Object Technology, Vol. 4, No. 9, November-December 2005, pp. 59-92.

[82] Bernárdez, B., Durán, A., Genero M., (2004). Empirical Evaluation and Review of a Metrics–Based Approach for Use Case Verification, Journal of Research and Practice in Information Technology, Vol. 36, No. 4, November 2004

[83] Marchesi, M., (1998). OOA Metrics for the Unified Modelling Language, Proceedings of the 2nd EUROMICRO Conference on Software Maintenance and Reengineering.

[84] Baroni, A.L., (2005). Quantitative Assessment of UML Dynamic Models, Proceedings of ESEC-FSE'05, September 5–9, 2005, Lisbon, Portugal.

APPENDICES

APPENDIX A: SGKS SURVEILLANCE MISSION AND VERIFICATION RESULTS

In this appendix we present information on the Surveillance mission of the SGKS conceptual model. In first section we provide the information on the tasks and their traceability to referent, SGKS conceptual model document. Then we list the diagrams, preconditions, postconditions, and roles. Finally we present the diagrams that we used in the case study.

1. Tasks and Activities of Surveillance Mission and Traceability to Original Referent

In this section we list the tasks the Surveillance Mission *consistsOf*. The english translation, diagram number, the reference number in SGKS document and type in SGKS document are also included.

Original Name (In Turkish)	Translation	Presence in Diagrams	Reference to SGKS Document	Type in SGKS Document
Gözetleme	Surveillance	1	3.3.1.1	Mission
Sensör Kapsama Analizini ve Planlamasını Yap	Perform Coverage Analysis and Planning	2	3.4.1.1	Task
Görev emri Dağıt	Distribute Mission Order	2	3.4.1.2	Task
Görev Yerlerine İntikal	Move to Mission Location	2	3.4.1.4	Task
Gözetleme Görev Hazırlığını Yap	Prepare for the Mission	2	3.4.1.9	Task
Görev Bölgesini Gözetle	Perform Surveillance of the area	2	3.4.1.10	Task
Görev Devir teslimini Gerçekleştir	Handover the mission	2	3.4.1.13	Task

Görevi sonlandır	End the mission	2	3.4.1.7	Task
Görev Başarımını Değerlendir	Evaluate mission performance	2	3.4.1.8	Task
Bölgeyi Tara	Scan the area	2.1	3.4.1.11	Task
Dost Birliklerin Yeirini Tespit ve Takip et	Detect the locations of Friend Forces	2.1	3.4.1.21	Task
Tespit/Teşhis/Tanıma verisini aktar	Transfer the Information on Detection, Identification and Recognition	2.1	3.4.1.42	Task
Veritabanını Güncelle	Update the Database	2.1	3.4.1.43	Task
Dinleme	Intercept	2.1.1	3.4.1.33	Task
Hedef tespiti yap	Detect the Targets	2.1.1	3.4.1.44	Task
Hedef Teşhisi ve Tanıma	Recognize and Identify	2.1.1	3.4.1.45	Task
Adım Tarama	Scan by step	2.1.1.1	3.5.1.52	Activity
Bant Tarama	Scan by band	2.1.1.1	3.5.1.53	Activity
Dinleme Bilgisini Gir	Input Interception Information	2.1.1.1	3.5.1.54	Activity
Dinleme Bilgisini Gir Altışleri	Subtasks of Input Interception Information	2.1.1.1.1	3.5.1.54 (S)	Activity Steps
Personelle Hedef Teşhisi ve Tanıma	Recognize and Identify Targets using Personnel	2.1.1.2	3.5.1.47	Activity
Radarla Hedef Teşhisi	Recognize using Radar	2.1.1.2	3.5.1.48	Activity
Gündüz Kamerasıyla Hedef Teşhisi ve Tanıma	Recognize and Identify Targets using Day Vision Camera	2.1.1.2	3.5.1.49	Activity
Termal Kamerayla hedef Teşhisi ve Tanıma	Recognize and Identify Targets using Thermal Camera	2.1.1.2	3.5.1.50	Activity
Hedef Tespiti Yap Altışleri	Subtasks of Detect the Targets	2.1.1.3	3.5.1.38 3.5.1.39 3.5.1.40 3.5.1.41 3.5.1.42	Activity
Haritada Veri Gösterimi	Data Visualization on Map	2.1.2	3.5.1.76	Activity
Yeni Veri Girişi	Input New Data	2.1.2	3.5.1.77	Activity
Veri Sıralama	Sequence Data	2.1.2	3.5.1.78	Activity

Veri Sorgulama	Query Data	2.1.2	3.5.1.79	Activity
Veri Silme	Delete Data	2.1.2	3.5.1.80	Activity
Kapsama Ön analizini Yap	Perform Pre-Analysis of Coverage	2.2	3.5.1.1	Activity
Platformların Görev planlamasını Yap	Perform Mission Planning for Platforms	2.2	3.5.1.2	Activity
Platformların konuşlanma nokta ve güzergahlarını tetkik et	Examine the deployment points and routes of the platforms	2.2	3.5.1.3	Activity
Platformların konuşlanma nokta ve güzergahlarını tekrar düzenle	Update deployment points and routes of the platforms	2.2	3.5.1.4	Activity
Kapsama alanını hesapla	Compute the Coverage Ratio	2.2.1	3.5.1.1 (S)	Activity Step
Nokta ve güzergah önceliklendir	Evaluate the priority of points and paths	2.2.1	3.5.1.1 (S)	Activity Step
Maliyet etkinlik en uygunlama	Optimize cost efficiency	2.2.2	3.5.1.2 (S)	Activity Step
Kullanım zamanlama ve sıklık planlaması	Usage Plan and Frequency Planning	2.2.2	3.5.1.2	Activity Step
Noktaya veya Güzergaha ulaşımı sağla	Supply Transportation to Point or route	2.2.3	3.5.1.3	Activity Step
Noktayı ve Güzergahı kontrol et	Control the point or route	2.2.3	3.5.1.3	Activity Step
Görev Emrini Oluştur	Prepare Mission Order	2.3	3.5.1.5	Activity
Haberleşme	Communication	2.3	3.4.1.36	Task
Emir Alındığını Teyit Et	Acknowledge Order Reival	2.3	3.5.1.6	Activity
Haberleşme Ortamını Belirle	Decide Communication Medium	2.3.1	3.5.1.57	Activity
İletimi Sağla	Perform Transmission	2.3.1	3.4.1.38	Task
Ses İletimini Sağla	Perform Transmission of Voice	2.3.1.1	3.4.1.39	Task
Görüntü İletimini Sağla	Perform Transmission of Images	2.3.1.1	3.4.1.40	Task

Veri İletimini Sağla	Perform Transmission of Data	2.3.1.1	3.4.1.41	Task
Ses İletimini Sağla Altışleri	Subtasks of Perform Transmission of Voice	2.3.1.1.1	3.5.1.59 3.5.1.60 3.5.1.61 3.5.1.62 3.5.1.63 3.5.1.67 3.5.1.69 3.5.1.70 3.5.1.72	Activity
Modemleri Kilitle	Interlock Modems	2.3.1.1.1.1	3.5.1.67 (S)	Activity step
Veri Kodlama Moduna Geç	Switch to Data Encryption Mode	2.3.1.1.1.1	3.5.1.67 (S)	Activity step
Serbest Mesaj Aktarımını Seç	Select Free Format Message Transmission	2.3.1.1.1.1	3.5.1.67 (S)	Activity step
Veri Gönderilecek Platformunu Seç	Select Data Receiving Platform	2.3.1.1.1.1	3.5.1.67 (S)	Activity step
Veri Mesajını Gönder	Send Data	2.3.1.1.1.1	3.5.1.67 (S)	Activity step
Veri Alındı Teyidi	Data Transmission Acknowledgement	2.3.1.1.1.1	3.5.1.67 (S)	Activity step
Görüntü Kodlama Moduna Geç	Switch to Image Coding mode	2.3.1.1.1.1	3.5.1.67 (S)	Activity step
Görüntü Matrisinden Aktarılacak Görüntüyü Seç	Select the Image from the Image Matrix	2.3.1.1.1.1	3.5.1.67 (S)	Activity step
Seçilen Görüntüyü uydu Göndermecine İrtibatlandır	Match the Image to Satellite Transmitter	2.3.1.1.1.1	3.5.1.67 (S)	Activity step
Görüntü Alındı Teyidi	Image Transmission Acknowledgement	2.3.1.1.1.1	3.5.1.67 (S)	Activity step

Görüntü İletimini Sağla Altişleri	Subtasks of Perform Transmission of Images	2.3.1.1.2	3.5.1.45 3.5.1.46 3.5.1.64 3.5.1.67 3.5.1.68 3.5.1.69 3.5.1.70 3.5.1.71 3.5.1.72 3.5.1.75	Activity
Veri İletimini Sağla Altişleri	Subtasks of Perform Transmission of Data	2.3.1.1.3	3.5.1.60 3.5.1.62 3.5.1.65 3.5.1.66 3.5.1.67 3.5.1.69 3.5.1.70 3.5.1.71 3.5.1.72	Activity
Başarım Ölçütü Değerlerini Hesapla	Calculate Performance Criteria	2.4	3.5.1.20	Activity
Başarım Ölçüt Değerleriyle Etkinlik Değerlerini Karşılaştır	Compare Performance Criteria and Effectiveness Values	2.4	3.5.1.21	Activity
Görev Maliyet/Etkinlik Değerlendirmesini Hazırla	Prepare mission cost effectiveness evaluation	2.4	3.5.1.22	Activity
Besleme Kaynağını Çalıştır	Start the power source	2.5	3.5.1.10	Activity
Haberleşme Sistemlerini Hazırla	Prepare Communication Systems	2.5	3.5.1.11	Activity
Sensör Sistemlerini Hazırla	Prepare Sensor Systems	2.5	3.5.1.12	Activity
K2 Sistemlerini Hazırla	Prepare C2 systems	2.5	3.5.1.13	Activity
Çevrime Gir	Open connection	2.5	3.5.1.58	Activity
Haberleş	Communicate	2.6	3.4.1.36	Task
Platformların Ulaşım Hazırlığını Yap	Prepare Platforms for Transportation	2.6	3.5.1.7	Activity

Görev Yerlerine Ulaş	Proceed to Mission Location	2.6	3.5.1.8	Activity
İntikal Teyidini Yap	Acknowledgement of Deployment	2.6	3.5.1.9	Activity
Performans Kriterlerini Belirle	Determine Performance Criteria	2.6.1	3.5.1.8 (S)	Activity Step
Yol Tanımlamasını Yap	Prepare Path Plan	2.6.1	3.5.1.8 (S)	Activity Step
Belirlenen Yolu Takip Et	Follow the Path Plan	2.6.1	3.5.1.8 (S)	Activity Step
Gözetleme Yap	Perform Surveillance	2.6.1	3.5.1.8 (S)	Activity Step
Gömülü Sensörleri Topla	Collect Embedded Sensors	2.6.2	3.5.1.7 (S)	Activity Step
Antenleri Topla	Collect Antennas	2.6.2	3.5.1.7 (S)	Activity Step
Platformları Çalışır Hale Getir	Start the Platforms	2.6.2	3.5.1.7 (S)	Activity Step
Durumu Raporla	Report The State	2.7	3.5.1.14	Activity
Son Durumu Doğrula	Confirm Last State	2.7	3.5.1.15	Activity
Gözetleme Durdur	End the Surveillance	2.8	3.4.1.16	Activity
Besleme Kaynağını Durdur	Shut down the Power Source	2.8	3.5.1.16	Activity
Sensör Sistemlerini Durdur	Shut down the Sensor Systems	2.8.1	3.5.1.17	Activity
Komuta Kontrol Sistemlerini Durdur	Shut down the C2 Systems	2.8.1	3.5.1.18	Activity
Haberleşme Sistemlerini Durdur	Shut down the Communication systems	2.8.1	3.5.1.19	Activity

2. Diagrams of the Surveillance Mission

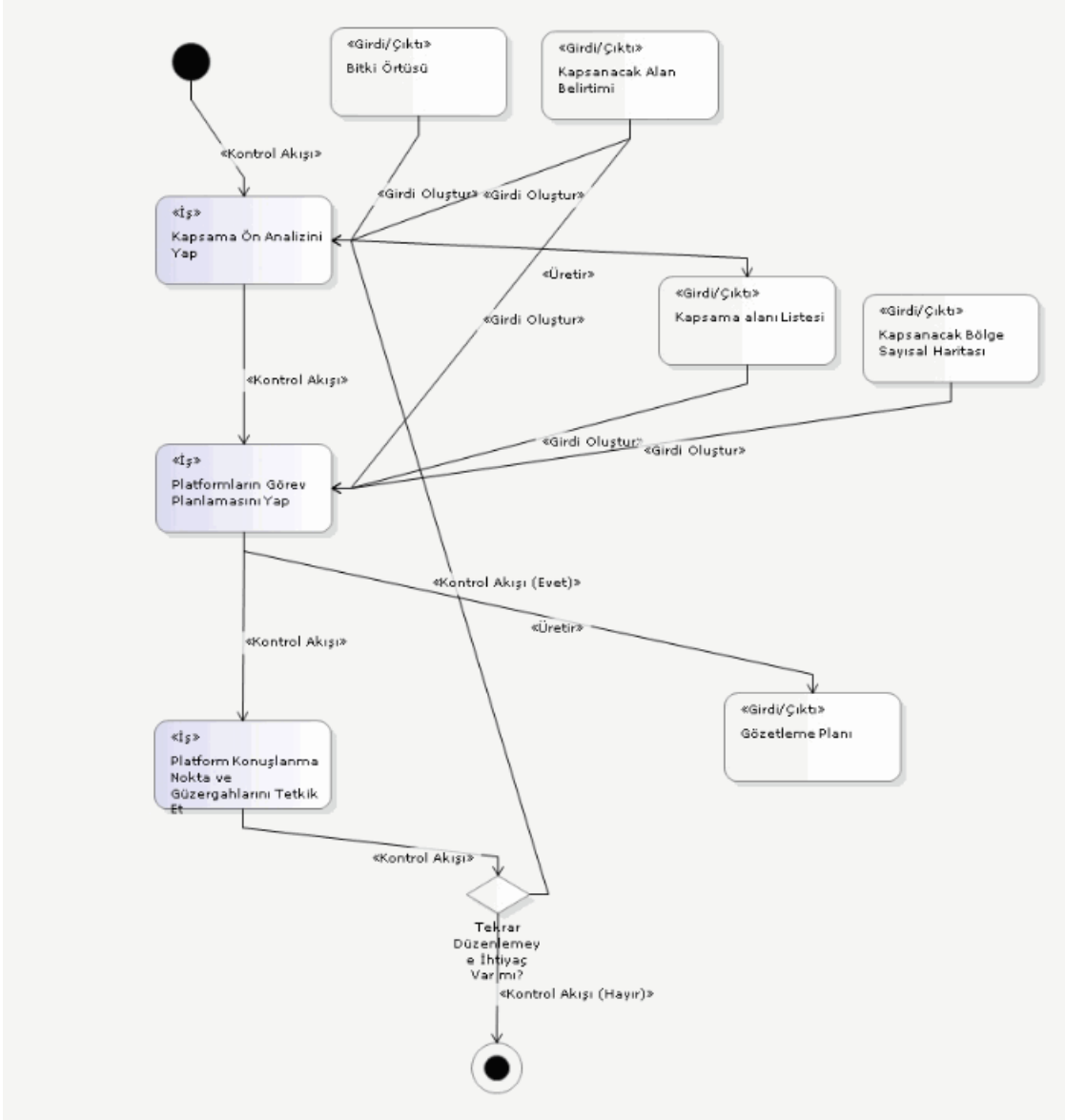
In this section the diagrams of surveillance mission and all the subtasks is listed first. Then the diagrams drawn by KAMA tool are given.

SGKS Diagrams	SGKS Document Reference	Task	Notes
2	3.3.1.1	Surveillance	Communication excluded
2.1	3.4.1.10	Perform Surveillance of the Area	
2.1.1	3.4.1.11	Scan the area	
2.1.1.1	3.4.1.33	Intercept	

SGKS Diagrams	SGKS Document Reference	Task	Notes
2.1.1.2	3.4.1.44	Detect the Targets	The subtasks are based on the type of the resource used
2.1.1.3	3.4.1.45	Recognize and Identify	The subtasks are based on the type of the resource used
2.1.2	3.4.1.43	Update the Database	
2.2	3.4.1.1	Perform Coverage Analysis and Planning	
2.2.1	3.5.1.1	Perform Pre-Analysis of Coverage	Consists of Sequential Tasks (Activity steps in SGKS document)
2.2.2	3.5.1.3	Examine the deployment points and routes of the platforms	Consists of Sequential Tasks (Activity steps in SGKS document)
2.2.3	3.5.1.4	Update deployment points and routes of the platforms	Consists of Sequential Tasks (Activity steps in SGKS document)
2.3	3.4.1.2	Distribute Mission Order	
2.3.1	3.4.1.36	Communication	
2.3.1.1	3.4.1.38	Perform Transmission	
2.3.1.1.1	3.4.1.39	Perform Transmission of Voice	The subtasks are based on the type of the equipment used
2.3.1.1.2	3.4.1.40	Perform Transmission of Images	The subtasks are based on the type of the equipment used
2.3.1.1.3	3.4.1.41	Perform Transmission of Data	The subtasks are based on the type of the equipment used
2.4	3.4.1.8	Evaluate mission performance	
2.5	3.4.1.9	Prepare for the Mission	
2.6	3.4.1.4	Move to Mission Location	
2.6.1	3.5.1.8	Proceed to Mission Location	Consists of Sequential Tasks (Activity steps in SGKS document)
2.6.2	3.5.1.7	Prepare Platforms for Transportation	Consists of Sequential Tasks (Activity steps in SGKS document)
2.7	3.4.1.13	Handover the mission	
2.8	3.4.1.7	End the mission	
2.8.1	3.4.1.16	End the Surveillance	

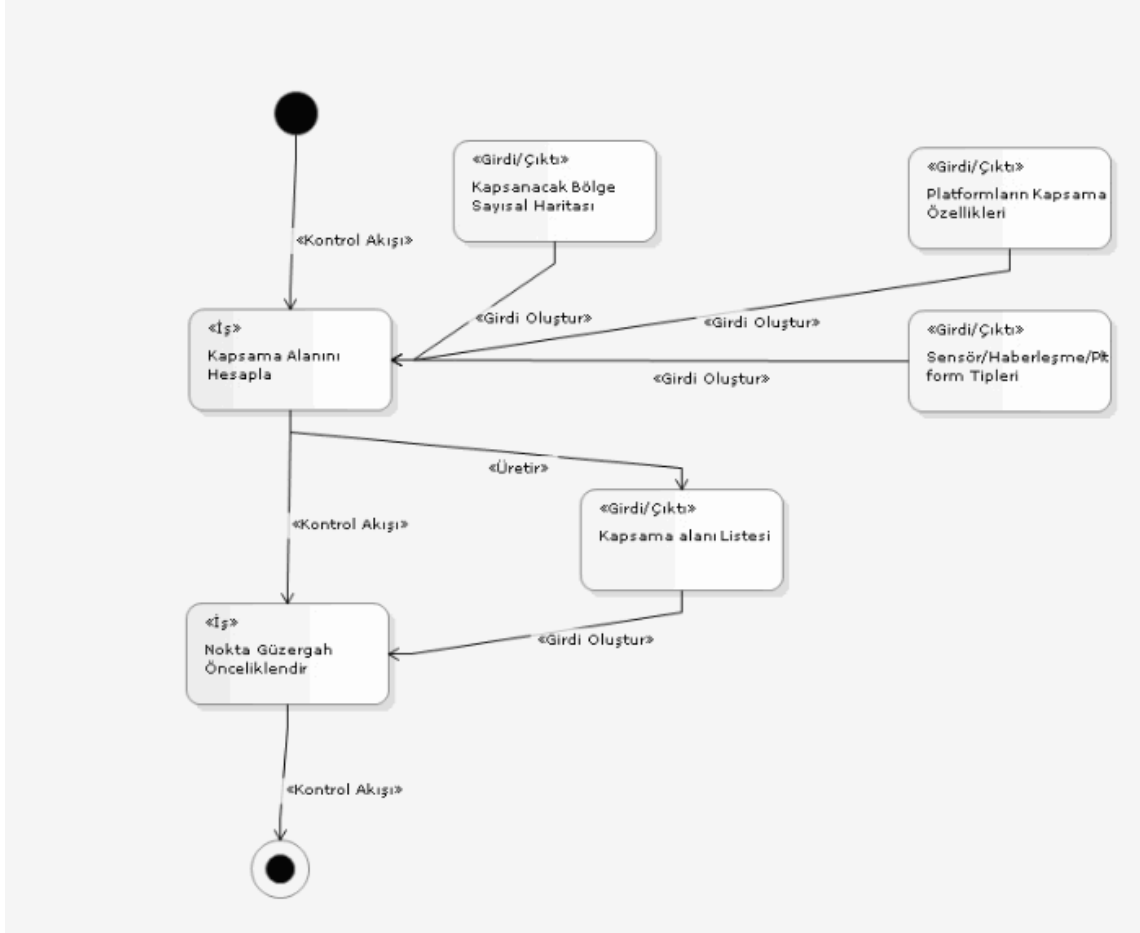
Id: 2.1

Name: Perform Coverage Analysis and Planning (Sensör Kapsama Analiz ve Görev Planlamasını Yap)



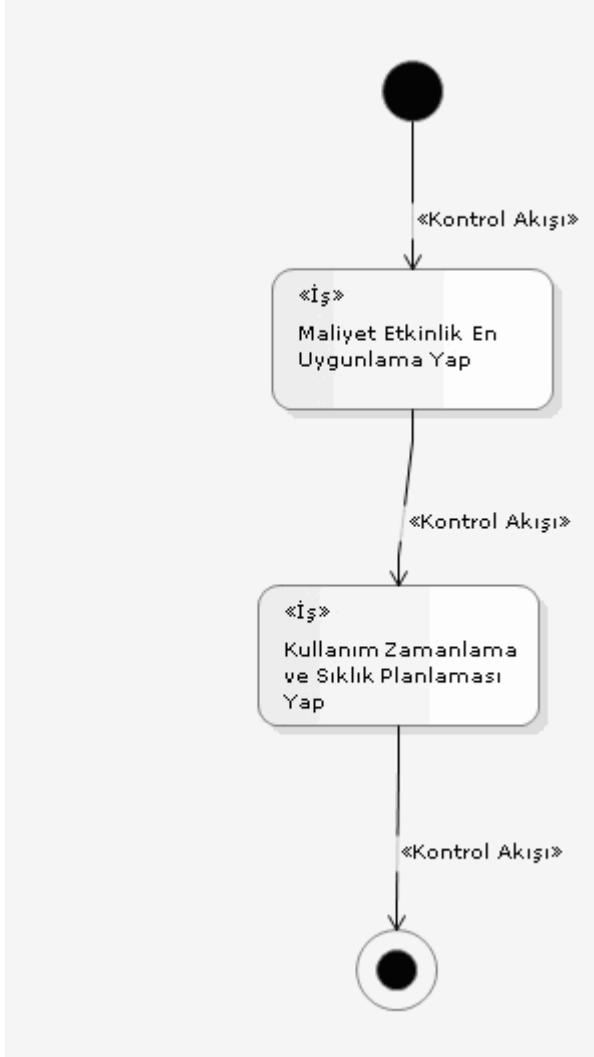
Id: 2.2.1

Name: Perform Pre-Analysis of Coverage (Kapsama Ön Analizini Yap)



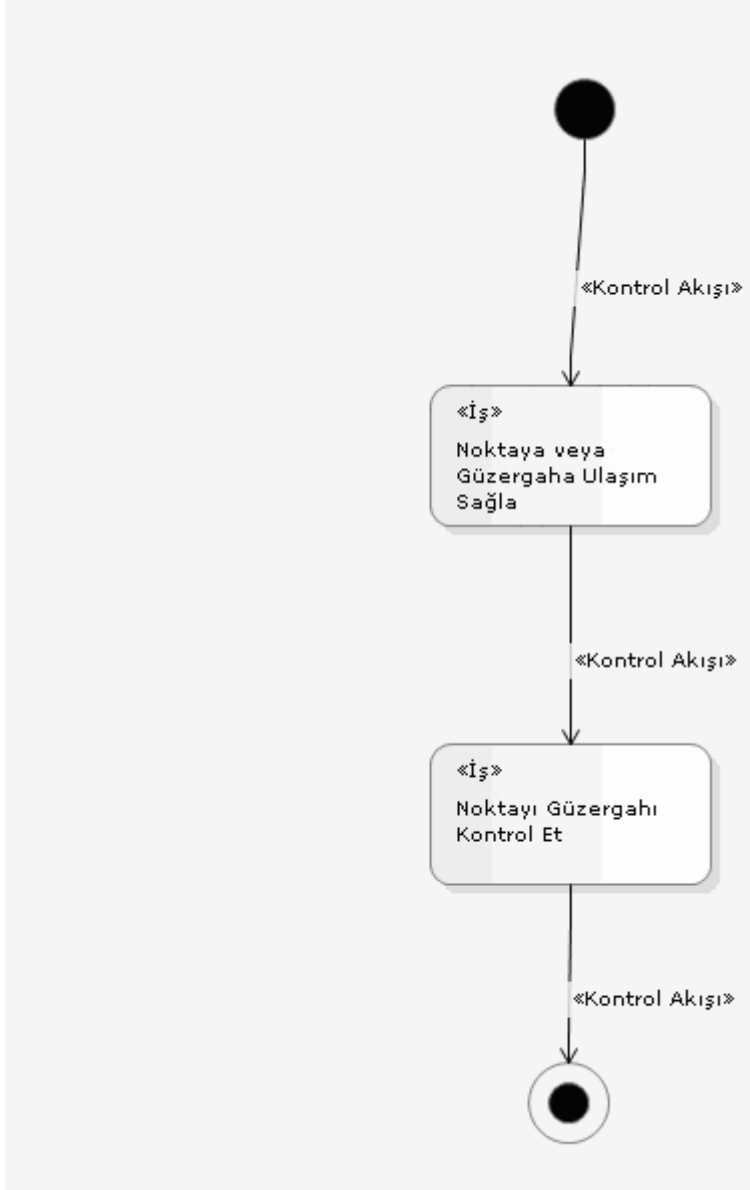
Id: 2.2.2

Name: Examine the deployment points and routes of the platforms (Platformların Görev Planlamasını Yap)



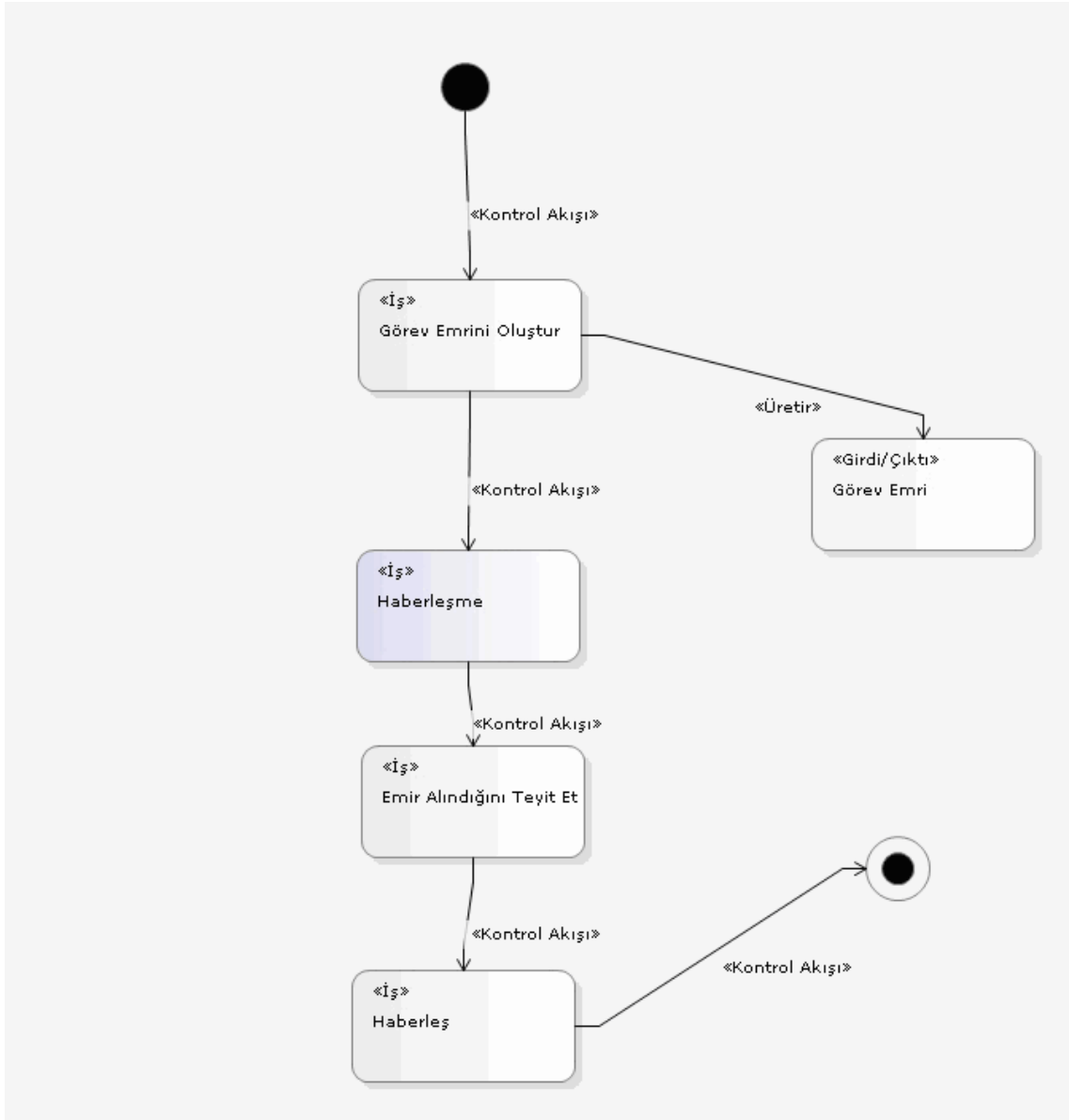
Id: 2.2.3

Name: Update deployment points and routes of the platforms (Platform Konuşlanma Nokta ve Güzergahlarını Tetkik Et)



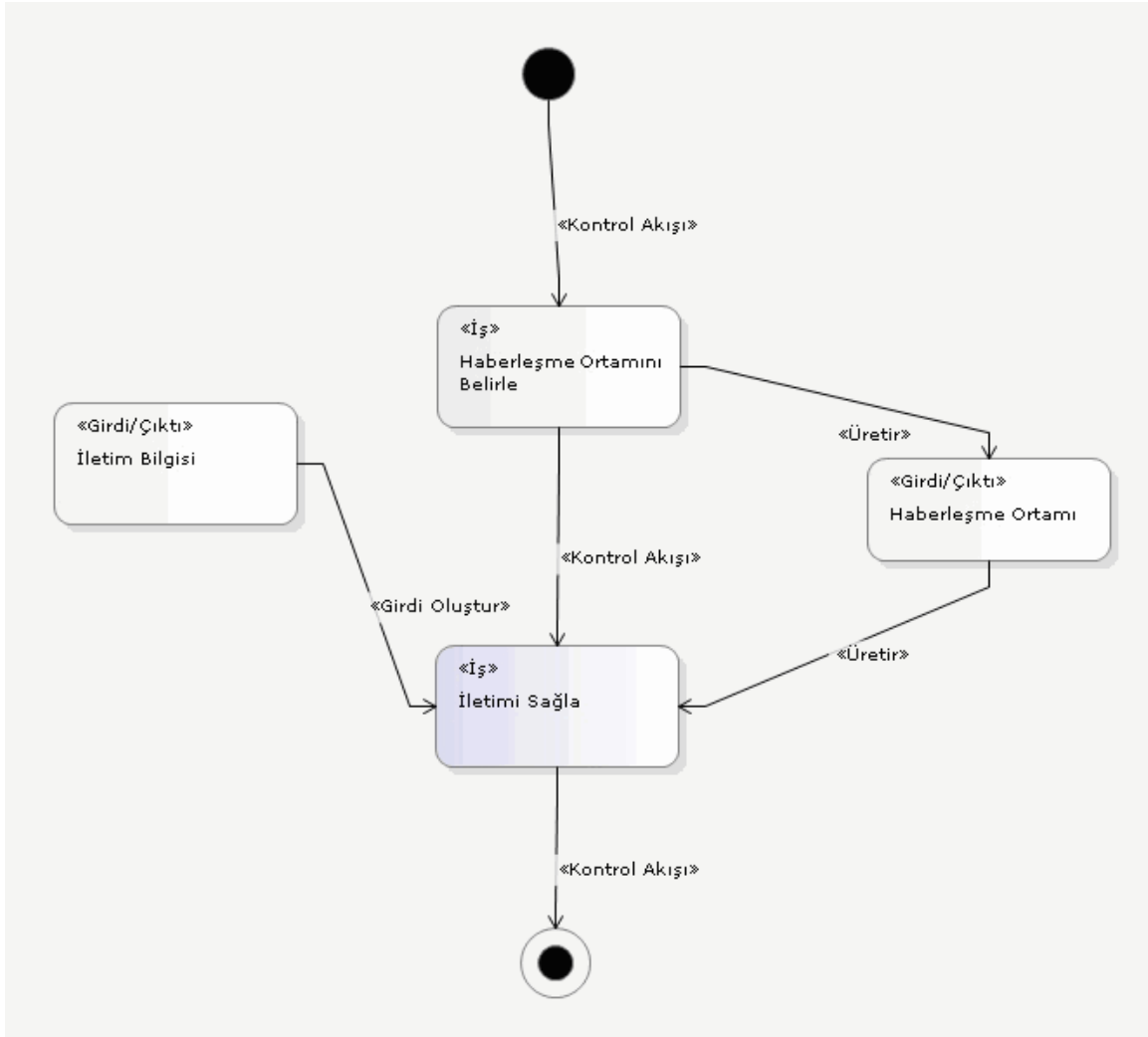
Id: 2.3

Name: Distribute Mission Order (Görev Emri Dağıt)



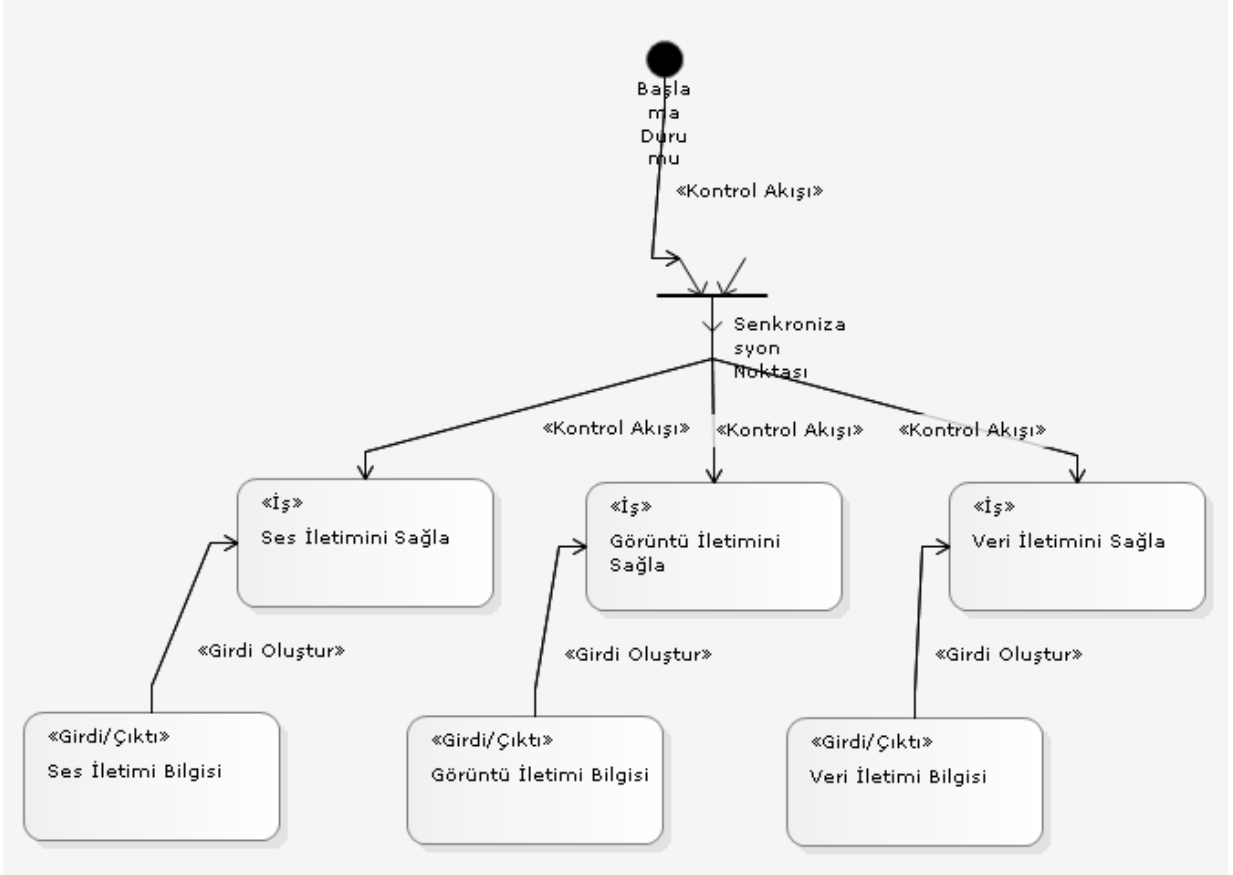
Id: 2.3.1

Name: Communication (Haberleşme)



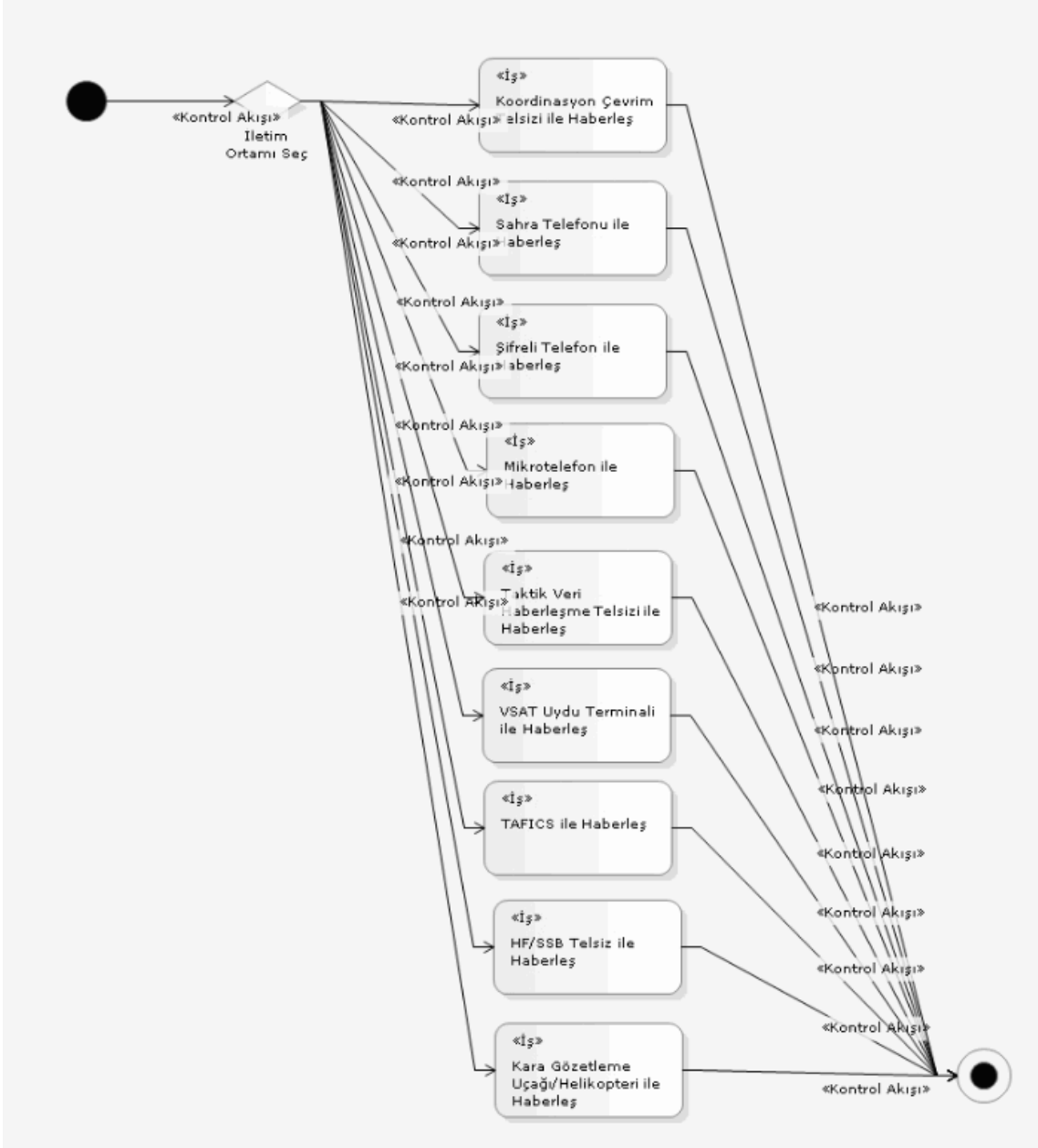
Id: 2.3.1.1

Name: Perform Transmission (İletimi Sağla)



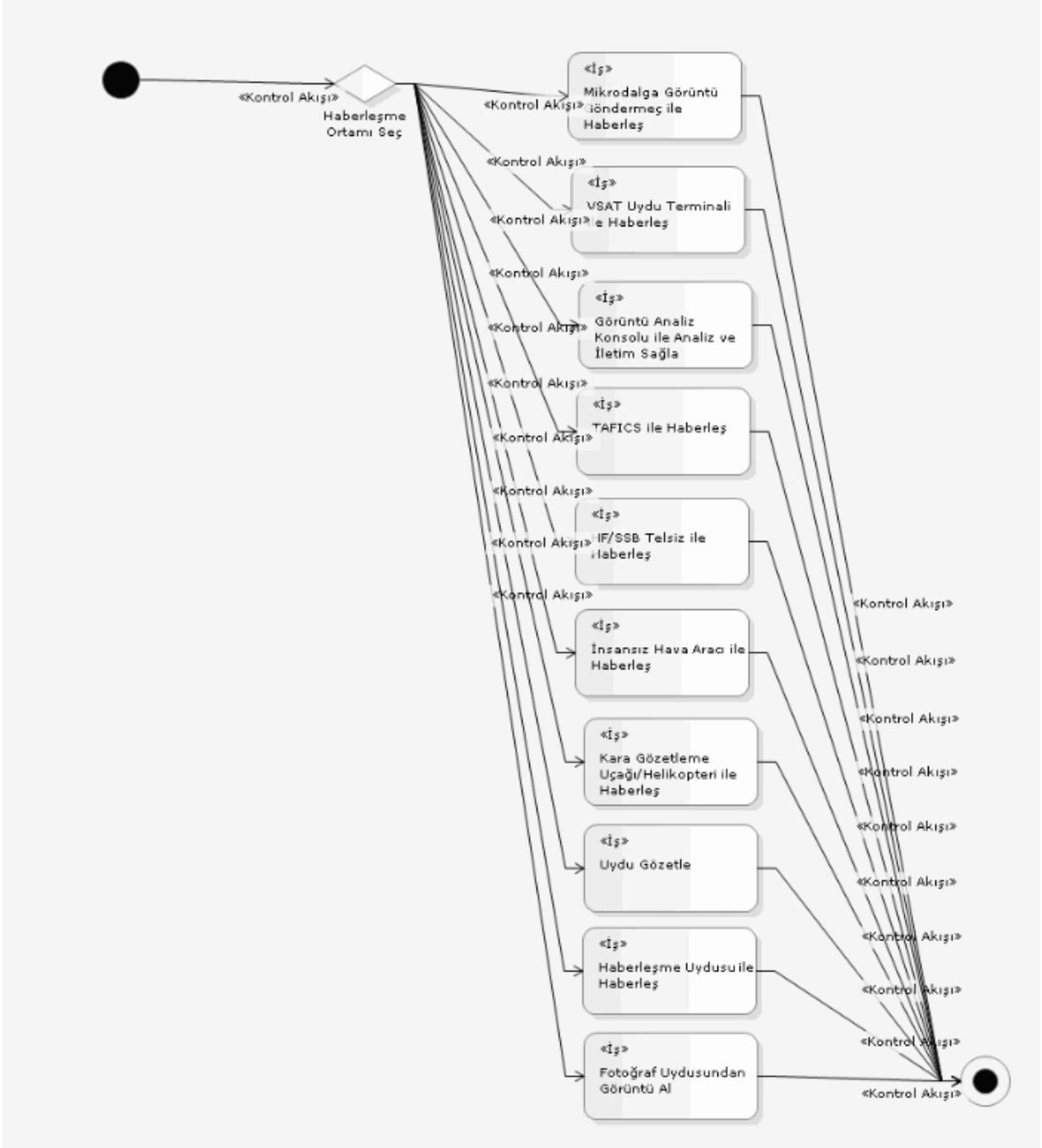
Id: 2.3.1.1.1

Name: Perform Transmission of Voice (Ses İletimini Sağla)



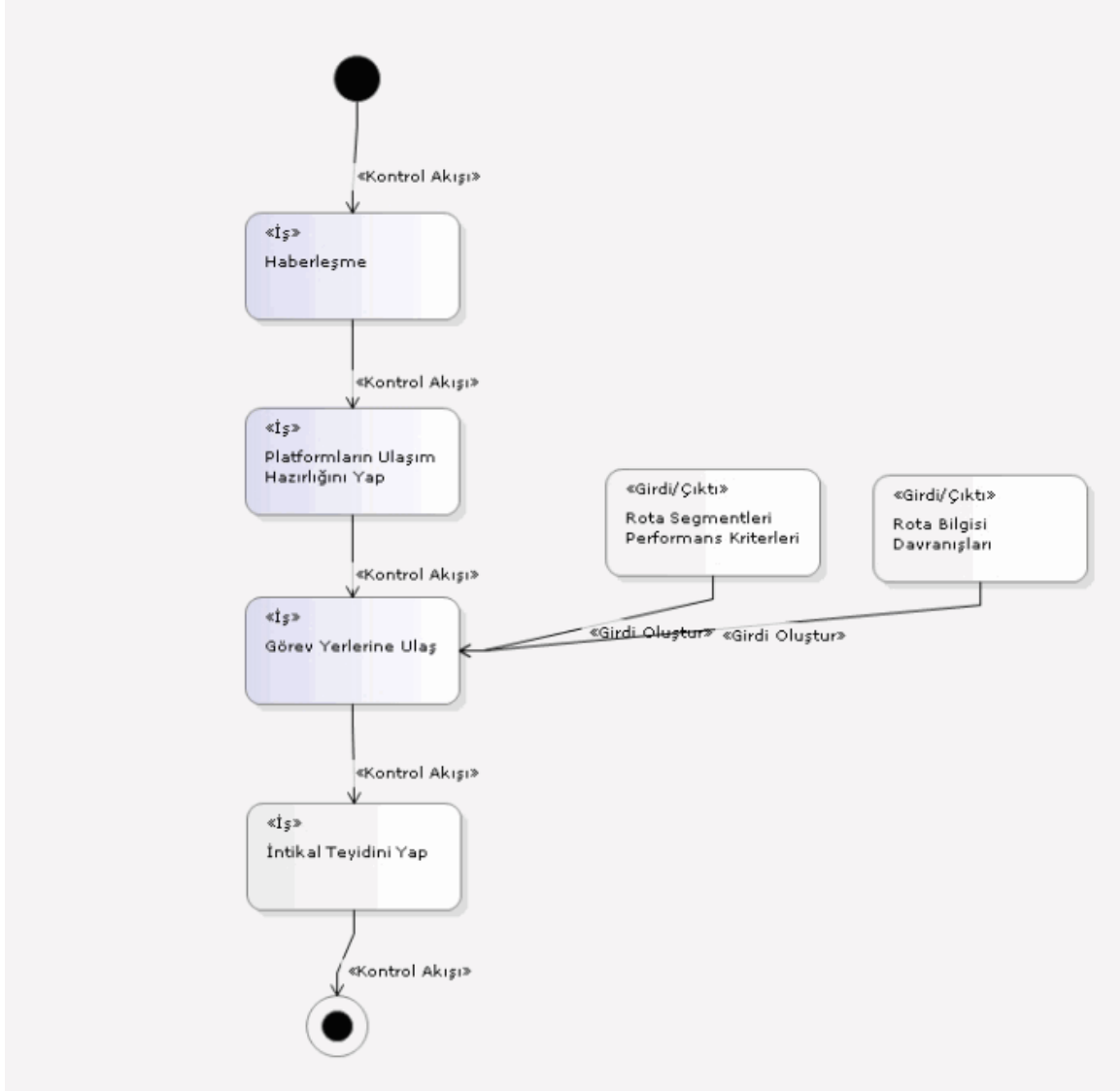
Id: 2.3.1.1.2

Name: Perform Transmission of Image (Görüntü İletimini Sağla)



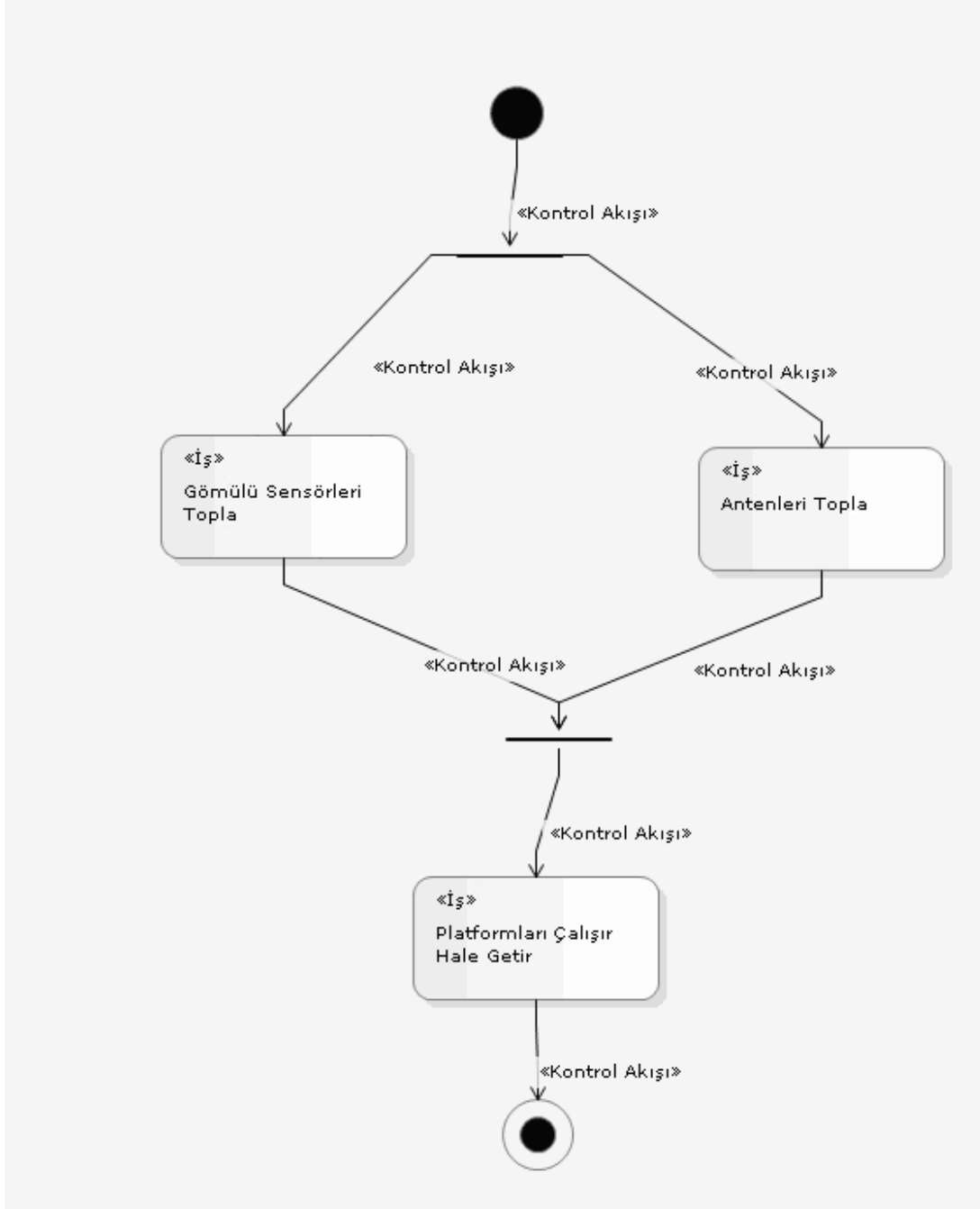
Id: 2.6

Name: Move to Mission Location (Görev Yerlerine İntikal Et)



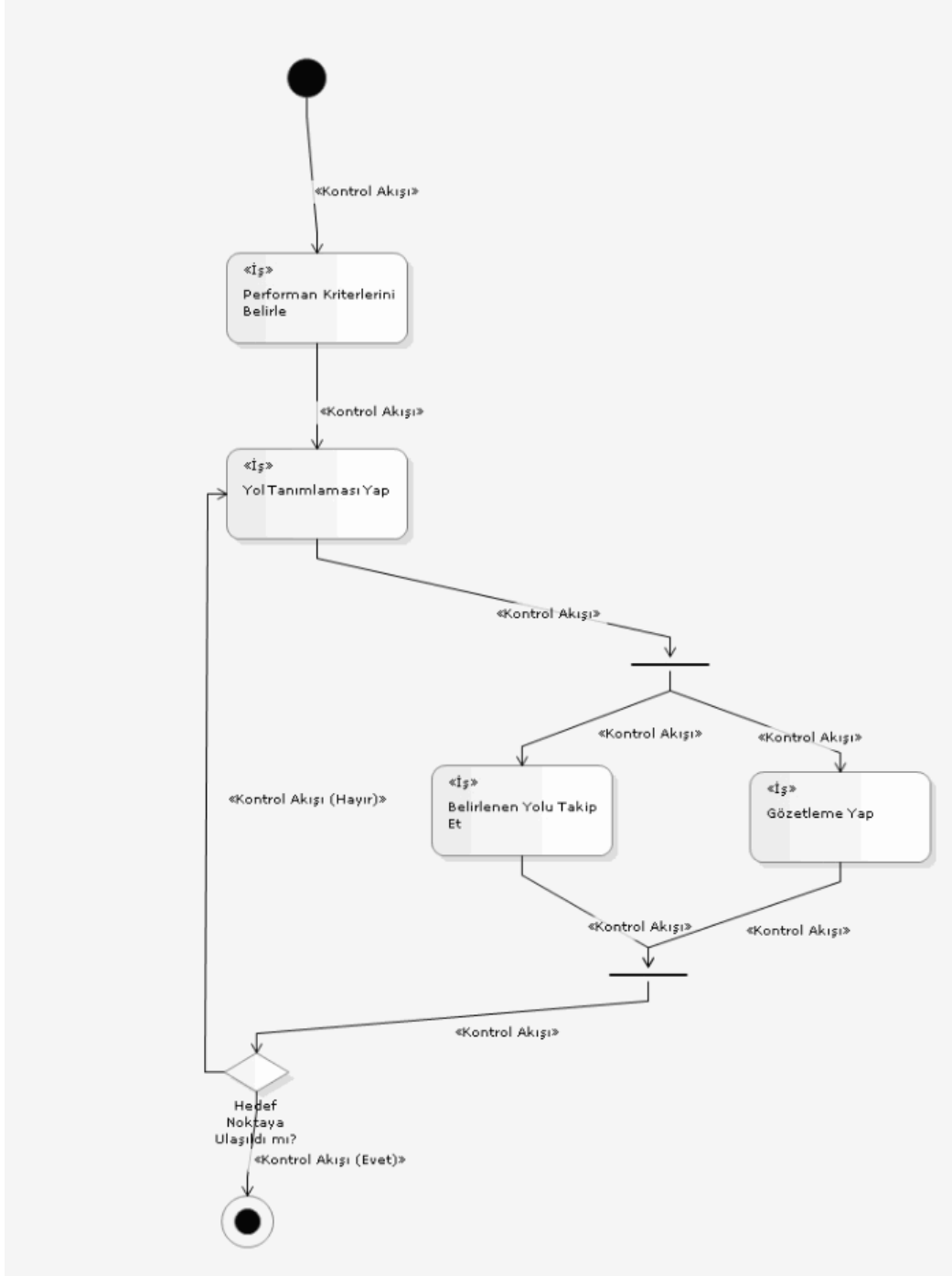
Id: 2.6.2

Name: Prepare Platforms for Transportation (Platformların Ulaşım Hazırlığını Yap)



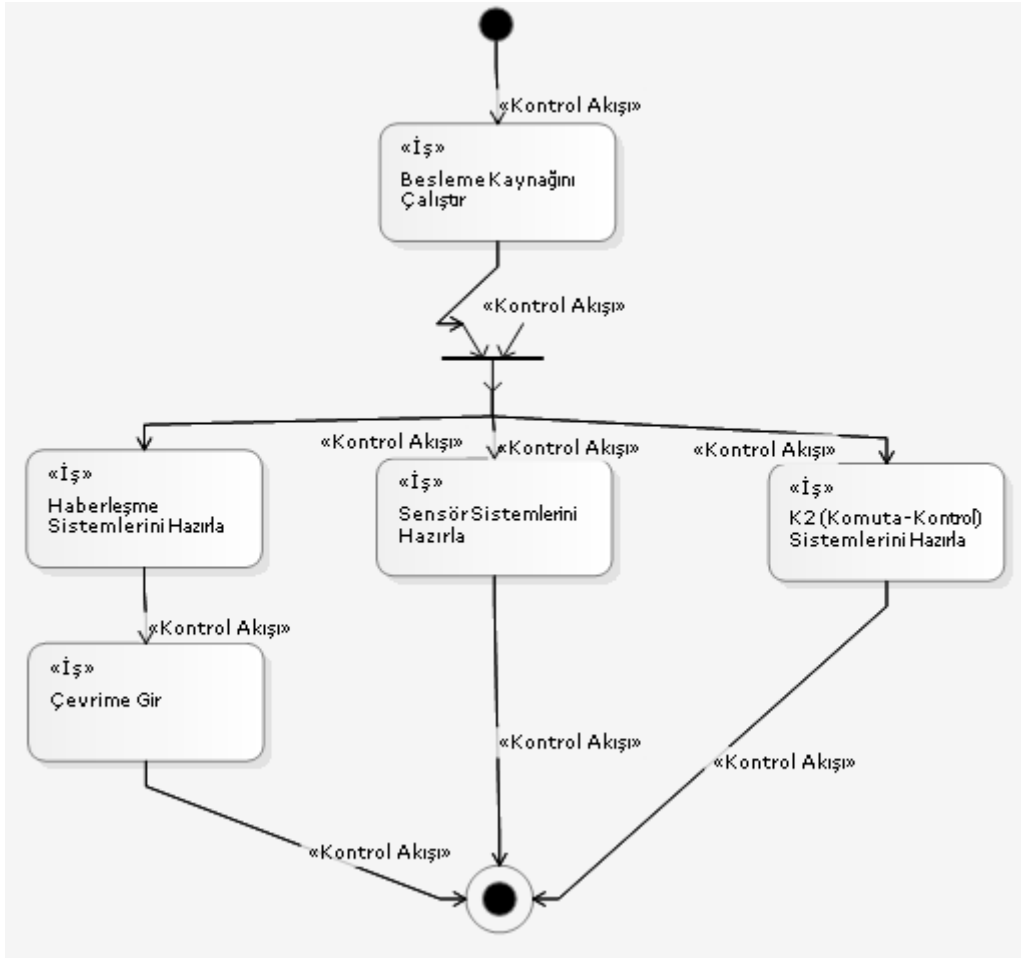
Id: 2.6.1

Name: Proceed to Mission Location (Görev Yerlerine Ulaş)



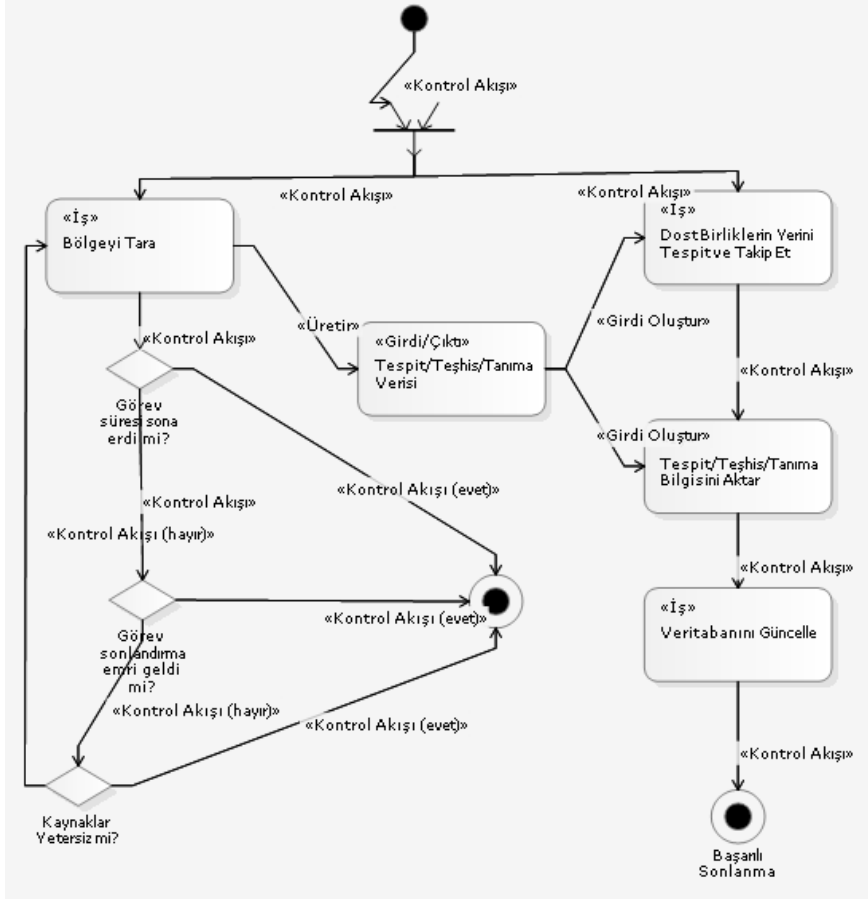
Id: 2.5

Name: Prepare for the Mission (Gözetleme Görev Hazırlığını Yap)



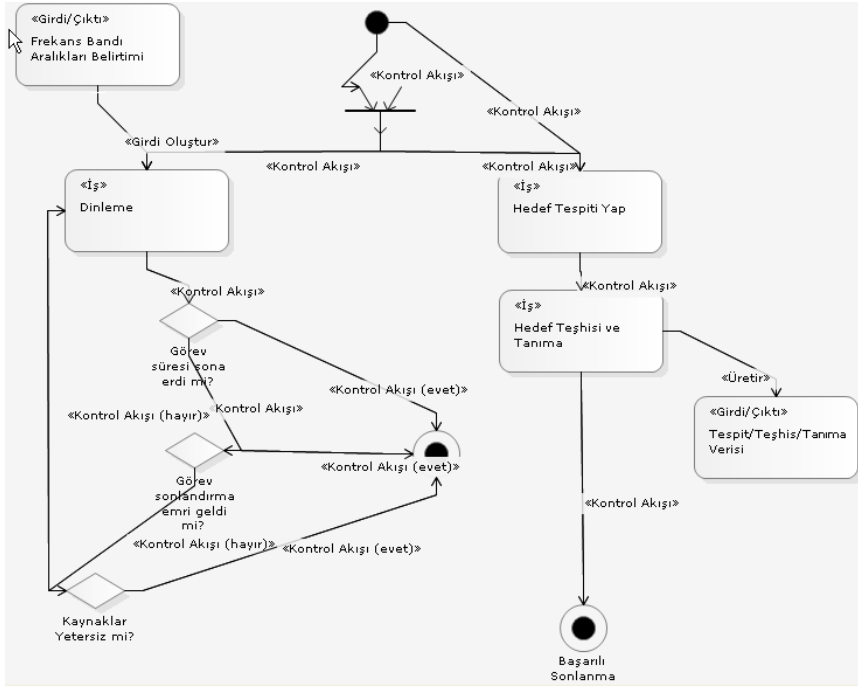
Id: 2.1

Name: Perform Surveillance of the Area (Görev Bölgesini Gözetle)



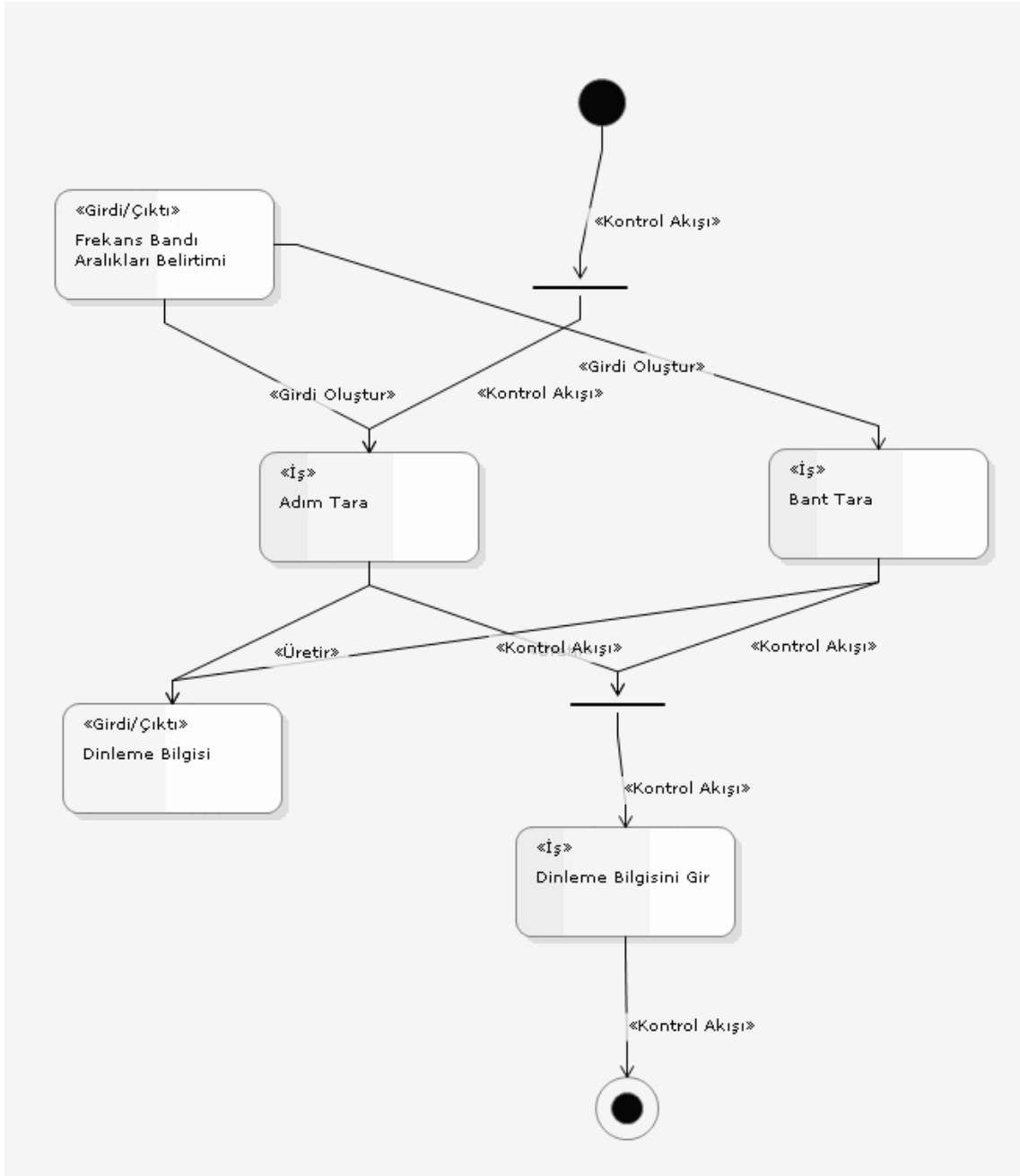
Id: 2.1.1

Name: Scan the Area (Planlı Bölge Tara)



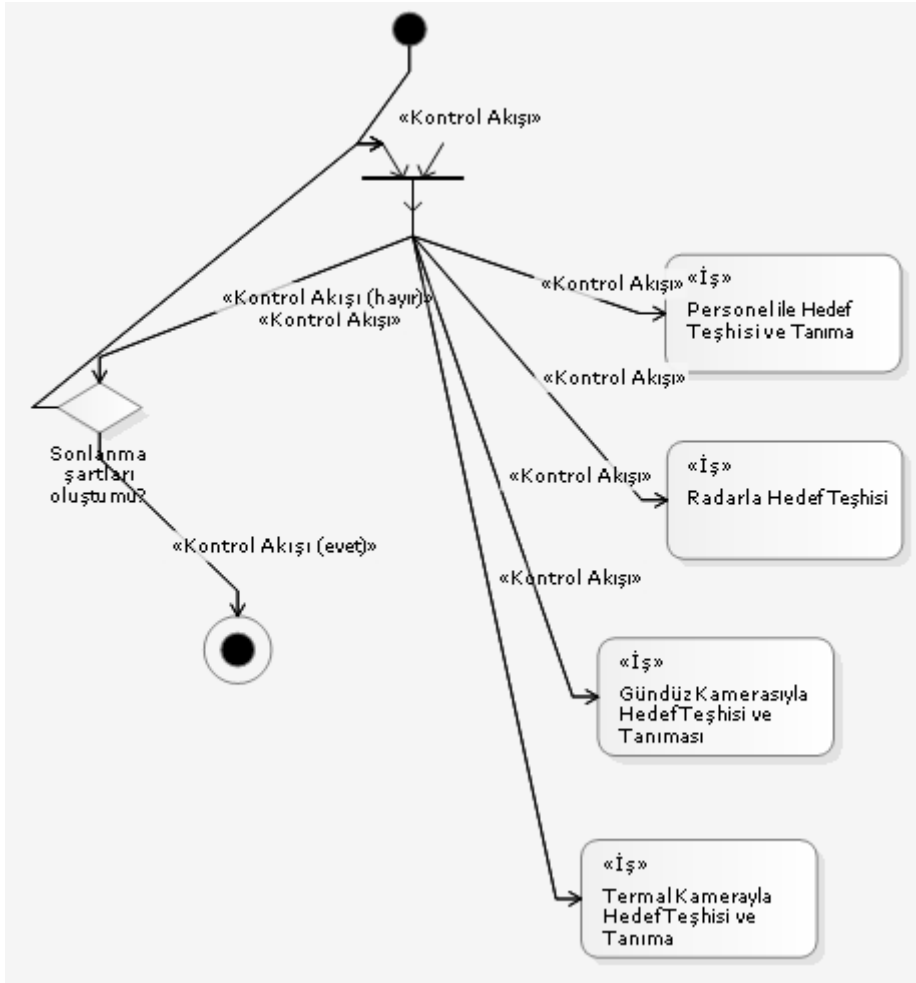
Id: 2.1.1.1

Name: Intercept (Dinle)



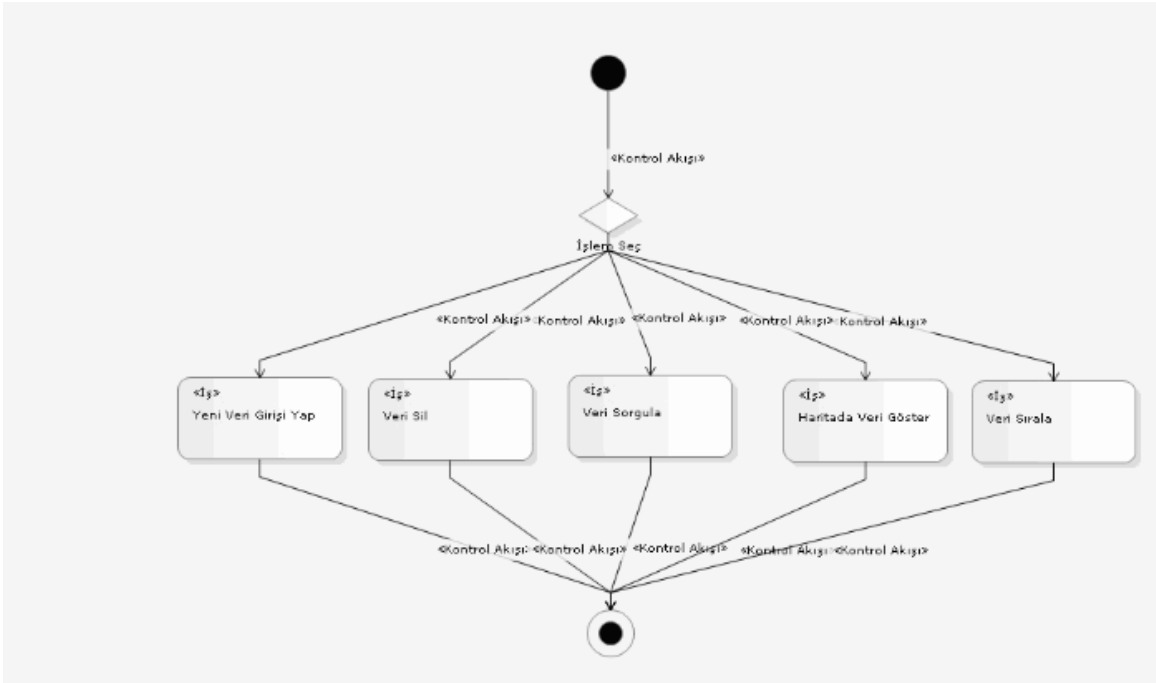
ID: 2.1.1.3

Name: Recognize and Identify (Hedef Teşhisi ve Tanıma)



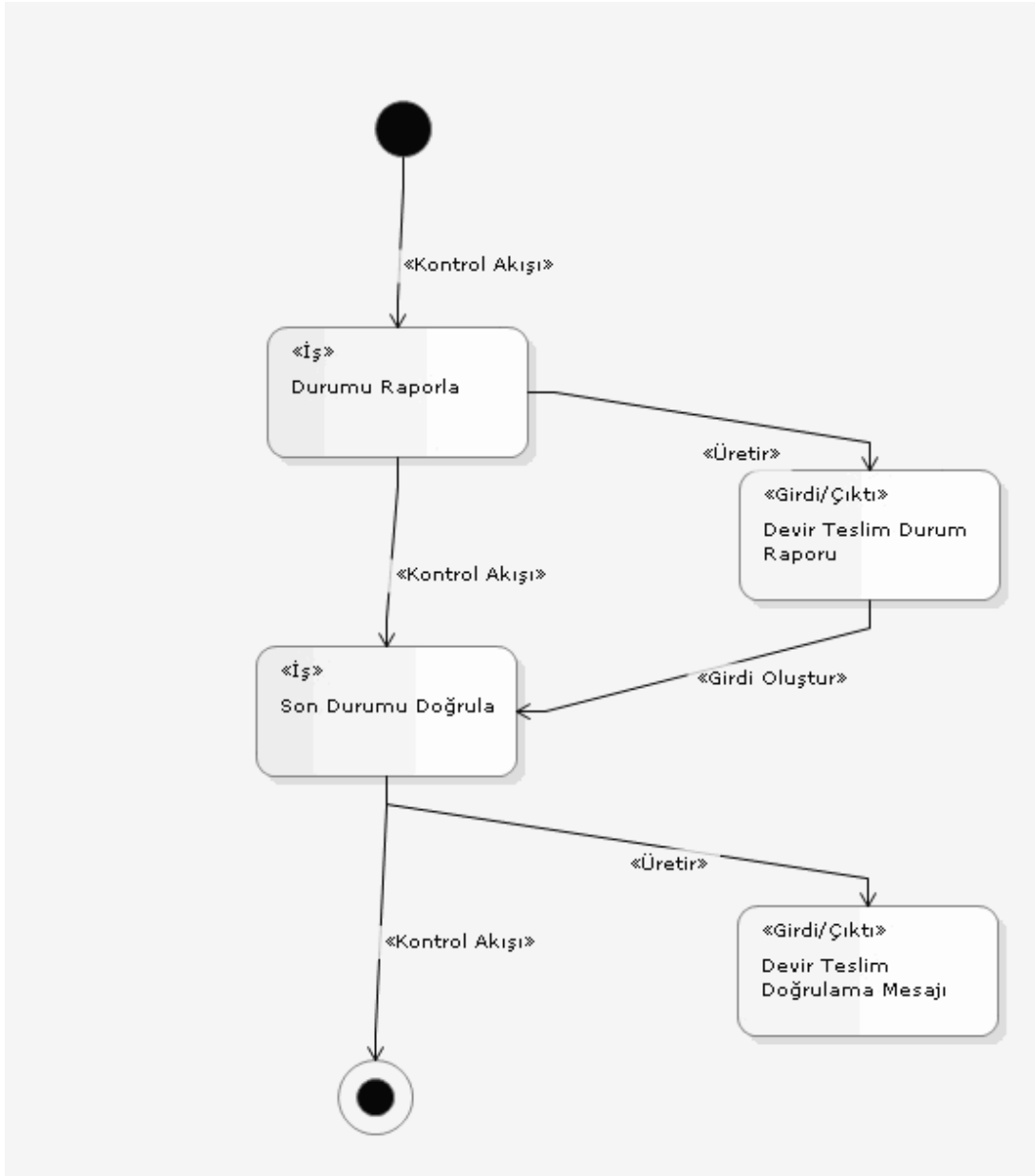
ID: 2.1.2

Name: Update the Database (Veri Tabanı Güncelle)



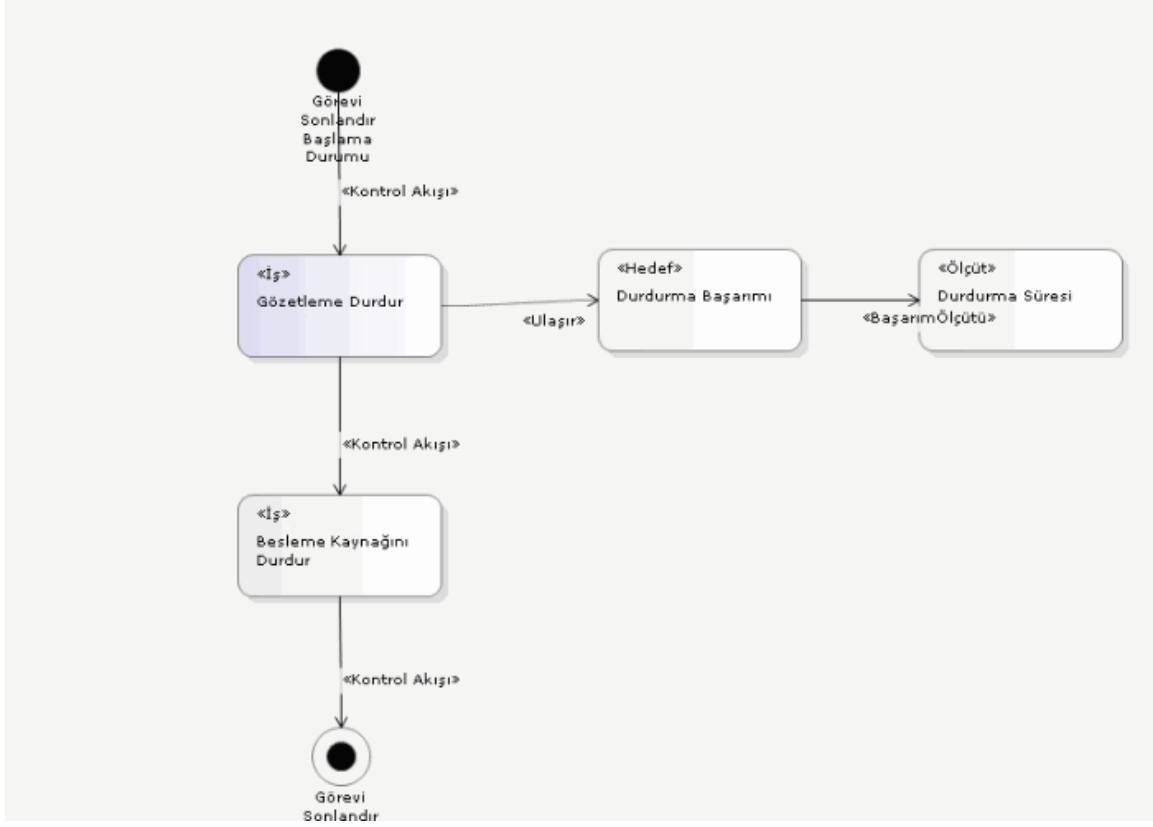
ID: 2.7

Name: Handover the Mission (Görev Devir Teslimini Gerçekleştir)



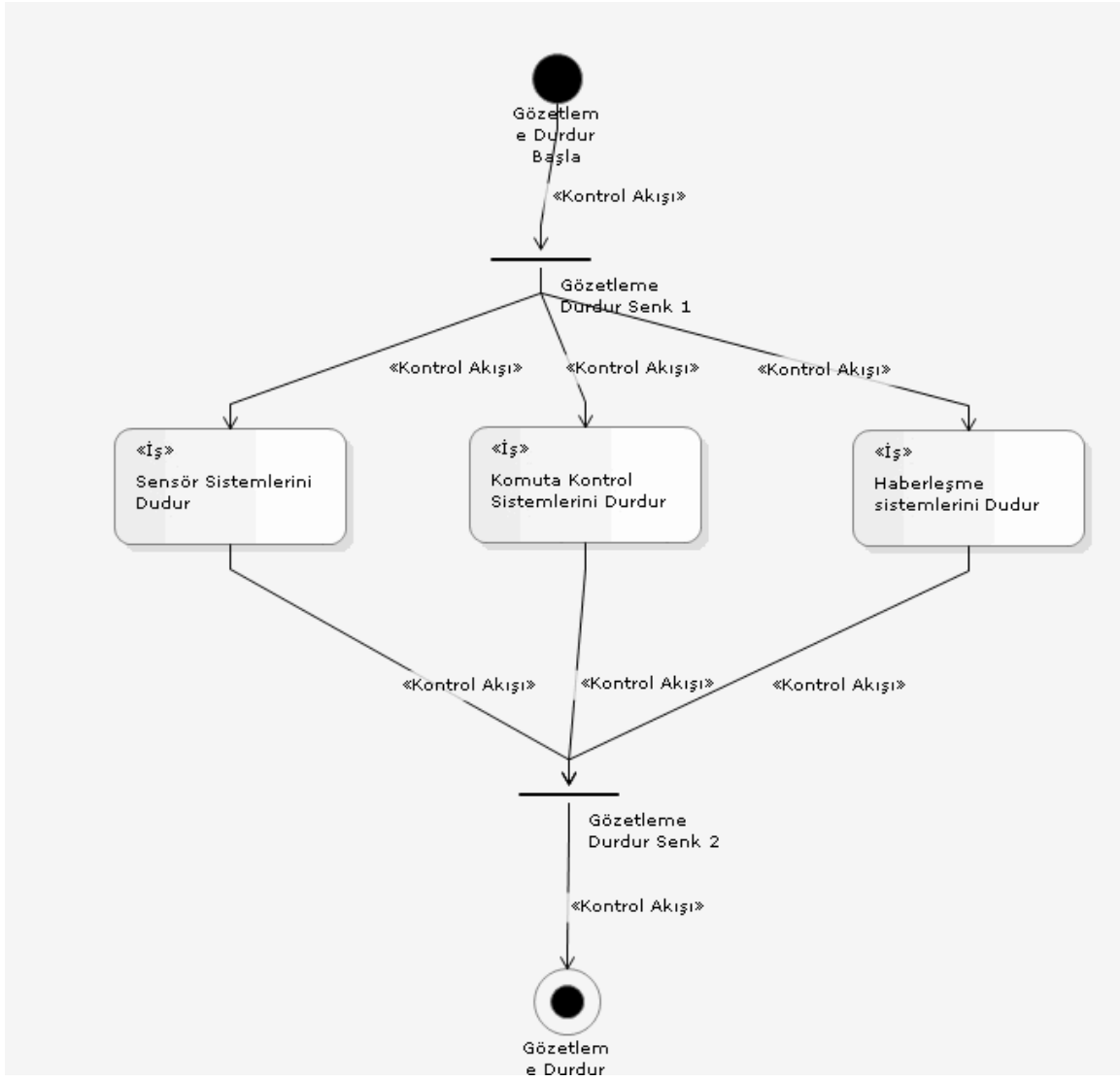
Id: 2.8

Name: End the Mission (Görevi Sonlandır)



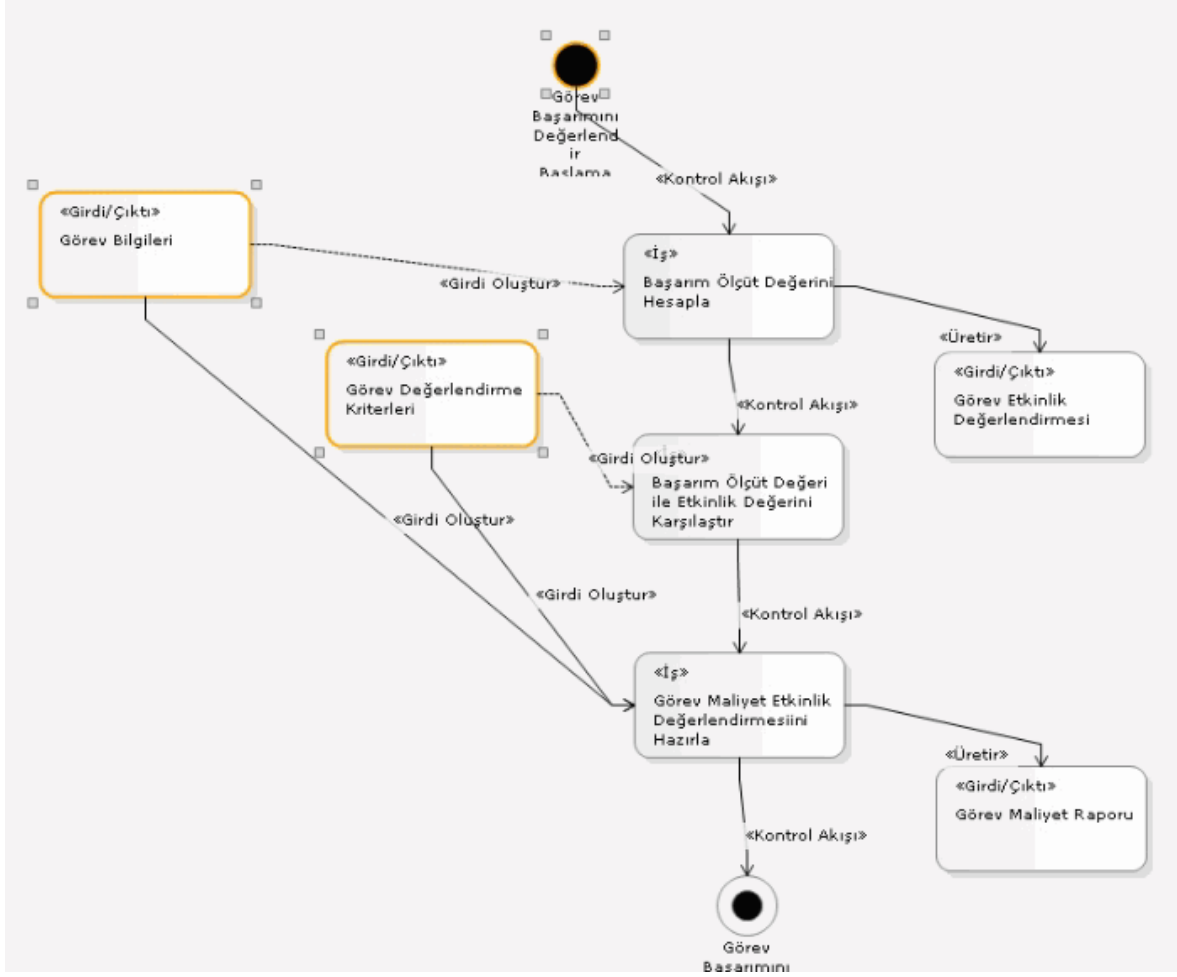
ID: 2.8.1

Name: End the Surveillance (Gözetleme Durdur)



Id: 2.4

Name: Evaluate Mission Performance (Görev Başarımını Değerlendir)



3. Unreferenced Items in Referent Model

Original Name (In Turkish)	SGKS Document	Notes
Grup Görev Emirlerinin Dağıtımı	3.4.1.3	Related to “Görev emri dağıtılması” 3.4.1.2
Görev Yerlerine İntikal Sonrası En Uygun Konuş Noktasının Bulunması	3.4.1.5	After the task 3.4.1.4 “Görev Yerlerine İntikal“
Görev Bölgesi Değistirme	3.4.1.6	No reference in the Document

4. Preconditions, Postconditions and Assumptions Defined for Tasks

Perform Coverage Analysis and Planning

Preconditions

All inputs should be ready

Mission (task) order distribution should be defined

Mission plan should be prepared

Postconditions

The confirmation should be received from all the units which make the mission order transmission

The confirmation should be received from any unit in a period previously determined

Move to Mission Location

Preconditions

Movement confirmation should be received from all the units (inconsistency)

Communication should be established

Communication devices should be determined

The specified communication medium (channels) should be available, appropriate and usable

Communication devices of the unit(s) shall be appropriate

Prepare for the Surveillance Mission

Preconditions

If one or more of the units performing the surveillance can not perform the mission, this work ends

Mission plan should be ready.

Devolution to the surveillance region should be performed

Communication and sensor systems should have at least one entity

Transfer Detection / Identification / Recognition Information

Preconditions

Detection / Identification / Recognition information to be transferred should be ready

Transmission used in communication environment should be available, appropriate and work

Update Database

Preconditions

Command-control computer should be switched on

New data should be gathered

Transmit

Assumptions

Only communication devices that are mentioned in the document will be utilized.

Preconditions

The information to be transmitted should be defined.

The specified communication medium (channels) should be available, appropriate and usable

Select the Communication Medium

Assumptions

In selection of the communication medium, no communication will occur except the ones that are supported by the entities communication devices and their communication channels.

Preconditions

Existence of necessary conditions for communication (requirement of notice of threat after Detection / Identification /Recognition, alarm situations etc.)

Handover Mission

Assumptions

Work termination conditions are not provided and it is performed for permanent prescriptive works

Preconditions

Mission shift plan should be prepared

Actors should be ready to make handover

Postconditions

New actors should review and verify the last reported case

Report Status**Preconditions**

A request should be made

Verify Latest Status**Preconditions**

The report should be received.

5. Screenshots of the Error Reports Displayed by EMF

Two screen shots show the errors found in the model when KAMA constraints are checked and KAMA-DV constraints are checked.

KAMA Rules Checking Screenshot

The screenshot shows the Eclipse IDE interface with the 'Problems' view open. The 'Problems' view displays 27 errors, 1 warning, and 0 others. The errors are listed in a table with columns for Description, Resource, Path, Location, and Type. The errors are categorized as EMF Problems.

Description	Resource	Path	Location	Type
The 'ExistsOutgoingGuardCondition' constraint is violated on 'Decision Point Detection End Conditions Satisfied?'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'ExistsOutgoingGuardCondition' constraint is violated on 'Decision Point Select the Activity'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'HangingTask' constraint is violated on 'Task Detect the locations of Friend Forces'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'HangingTask' constraint is violated on 'Task Update deployment points and routes of the platforms'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Data Visualization on Map'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Delete Data'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Detect using Day vision Camera'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Detect using Embedded Sensor'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Detect using Personnel'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Detect using Radar'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Detect using Thermal Camera'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Examine the deployment points and routes of the platforms'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Handover the mission'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Input New Data'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Query Data'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Recognize and Identify Targets using Day Vision Camera'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Recognize and Identify Targets using Personnel'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Recognize and Identify Targets using Thermal Camera'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Recognize using Radar'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'NoFlowDeadEnd' constraint is violated on 'Task Sequence Data'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'TaskflowTargetType' constraint is violated on 'Task Flow Perform Coverage Analysis and Planning TF1'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem
The 'TaskflowTargetType' constraint is violated on 'Task Flow Surveillance TF4'	Task.xmi	/KAMAv2/...	Unknown	EMF Problem

The 'HangingTask' constraint is violated on 'Task Update deployment points and routes of the platforms'

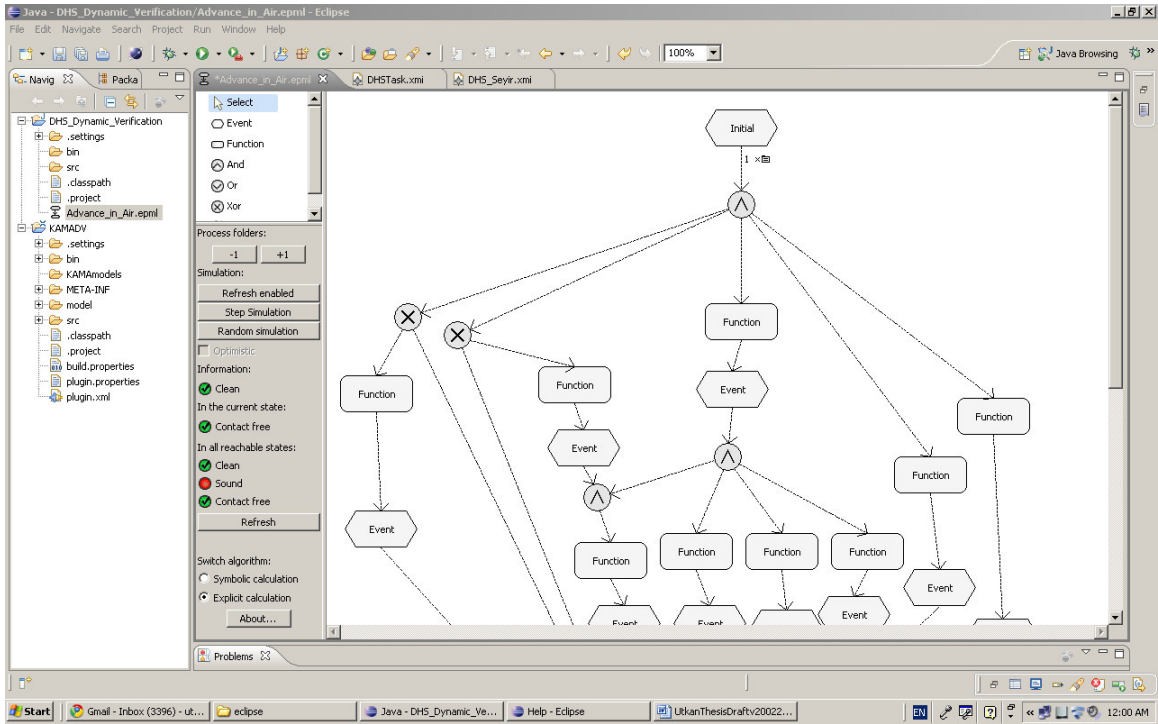
KAMA-DV Rules Checking Screenshot

The screenshot shows the Eclipse IDE interface with the following components:

- Navigator:** Shows the project structure for KAMAv2, including folders like .settings, bin, KAMAModels, META-INF, model, and src.
- Editor:** Displays the Task.xml file with a tree view of tasks: Task Surveillance, Task Perform Coverage Analysis and Planning, Task Perform Pre-Analysis of Coverage, and Task Perform Mission Planning for Platforms.
- Problems View:** Lists 56 errors. The selected error is: "The 'NoFlowDeadEnd' constraint is violated on 'Task Examine the deployment points and routes of the platforms'".

Description	Resource	Path
Errors (56 items)		
The 'ExistsOutgoingGuardCondition' constraint is violated on 'Decision Point Detection End Conditions Satisfied?'	Task.xml	/KAMAv2/...
The 'ExistsOutgoingGuardCondition' constraint is violated on 'Decision Point Select the Activity'	Task.xml	/KAMAv2/...
The 'HangingTask' constraint is violated on 'Task Detect the locations of Friend Forces'	Task.xml	/KAMAv2/...
The 'HangingTask' constraint is violated on 'Task Update deployment points and routes of the platforms'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Data Visualization on Map'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Delete Data'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Detect using Day vision Camera'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Detect using Embedded Sensor'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Detect using Personnel'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Detect using Radar'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Detect using Thermal Camera'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Examine the deployment points and routes of the platforms'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Handover the mission'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Input New Data'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Query Data'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Recognize and Identify Targets using Day Vision Camera'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Recognize and Identify Targets using Personnel'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Recognize and Identify Targets using Thermal Camera'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Recognize using Radar'	Task.xml	/KAMAv2/...
The 'NoFlowDeadEnd' constraint is violated on 'Task Sequence Data'	Task.xml	/KAMAv2/...
The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Detection End Conditions Satisfied?'	Task.xml	/KAMAv2/...
The 'SequentialIntermediateNode' constraint is violated on 'Decision Point End conditions Satisfied?'	Task.xml	/KAMAv2/...
The 'SequentialIntermediateNode' constraint is violated on 'Decision Point End Mission Order Received?'	Task.xml	/KAMAv2/...
The 'SequentialIntermediateNode' constraint is violated on 'Decision Point End Mission Order Received?'	Task.xml	/KAMAv2/...
The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Insufficient Resources?'	Task.xml	/KAMAv2/...
The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Insufficient Resources?'	Task.xml	/KAMAv2/...
The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Mission Duration Exceeded?'	Task.xml	/KAMAv2/...

EPC Soundness Checking Screenshot



6. List of Errors and Issues for the First Case Study

No.	Description
1	The 'UniqueOutgoingforJoinType' constraint is violated on 'Synchronization Point Seyir SP1'
2	The 'UniqueOutgoingforJoinType' constraint is violated on 'Synchronization Point Recognize and Identify SP1'
3	The 'UniqueOutgoingforJoinType' constraint is violated on 'Synchronization Point Detect the Targets SP1'
4	The 'UniqueOutgoingFlowGuardCondition' constraint is violated on 'Decision Point Select the Activity'
5	The 'UniqueOutgoingFlowGuardCondition' constraint is violated on 'Decision Point Select Communication Medium for Voice'
6	The 'UniqueOutgoingFlowGuardCondition' constraint is violated on 'Decision Point Select Communication Medium for Image'
7	The 'UniqueIncomingforForkType' constraint is violated on 'Synchronization Point Seyir SP1'
8	The 'UniqueIncomingforForkType' constraint is violated on 'Synchronization Point Recognize and Identify SP1'
9	The 'UniqueIncomingforForkType' constraint is violated on 'Synchronization Point Detect the Targets SP1'
10	The 'SequentialIntermediateNode' constraint is violated on 'Synchronization Point SeyirSP2'
11	The 'SequentialIntermediateNode' constraint is violated on 'Synchronization Point Recognize and Identify SP1'
12	The 'SequentialIntermediateNode' constraint is violated on 'Synchronization Point Proceed to Mission Location SP2'
13	The 'SequentialIntermediateNode' constraint is violated on 'Synchronization Point Havada İlerleme Faaliyetleri SP4'
14	The 'SequentialIntermediateNode' constraint is violated on 'Synchronization Point Havada İlerleme Faaliyetleri SP3'
15	The 'SequentialIntermediateNode' constraint is violated on 'Synchronization Point Havada İlerleme Faaliyetleri SP1'
16	The 'SequentialIntermediateNode' constraint is violated on 'Synchronization Point Detect the Targets SP1'
17	The 'SequentialIntermediateNode' constraint is violated on 'Initial Task Update the Database IT'
18	The 'SequentialIntermediateNode' constraint is violated on 'Initial Task Transmission of Image IT'
19	The 'SequentialIntermediateNode' constraint is violated on 'Initial Task Surveillance of the Area IT'
20	The 'SequentialIntermediateNode' constraint is violated on 'Initial Task Scan the Area IT'
21	The 'SequentialIntermediateNode' constraint is violated on 'Initial Task Recognize and Identify IT'
22	The 'SequentialIntermediateNode' constraint is violated on 'Initial Task Prepare Platforms for Transportation IT'
23	The 'SequentialIntermediateNode' constraint is violated on 'Initial Task Perform Transmission of Voice IT'
24	The 'SequentialIntermediateNode' constraint is violated on 'Initial Task Perform Transmission IT'
25	The 'SequentialIntermediateNode' constraint is violated on 'Initial Task Perform Transmission IT'
26	The 'SequentialIntermediateNode' constraint is violated on 'Initial Task Intercept IT1'

27	The 'SequentialIntermediateNode' constraint is violated on 'Initial Task Havada İlerleme Faaliyetleri IT'
28	The 'SequentialIntermediateNode' constraint is violated on 'Initial Task End the Surveillance IT'
29	The 'SequentialIntermediateNode' constraint is violated on 'Initial Task Detect the Targets IT'
30	The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Yakıt Miktarı'
31	The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Seyir DP1'
32	The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Select the Activity'
33	The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Proceed to Mission Location DP1'
34	The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Need for Reorganization?'
35	The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Mission End Conditions Satisfied?'
36	The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Mission Duration Expired?'
37	The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Mission Duration Expired?'
38	The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Insufficient Resources?'
39	The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Insufficient Resources?'
40	The 'SequentialIntermediateNode' constraint is violated on 'Decision Point End Mission Order Received?'
41	The 'SequentialIntermediateNode' constraint is violated on 'Decision Point End Mission Order Received?'
42	The 'SequentialIntermediateNode' constraint is violated on 'Decision Point End conditions Satisfied?'
43	The 'SequentialIntermediateNode' constraint is violated on 'Decision Point Detection End Conditions Satisfied?'
44	The 'IncomingFlowMultiplicityFT' constraint is violated on 'Final Task Transmission of Image FT'
45	The 'IncomingFlowMultiplicityFT' constraint is violated on 'Final Task Surveillance of the Area FT'
46	The 'IncomingFlowMultiplicityFT' constraint is violated on 'Final Task Scan the Area FT'
47	The 'IncomingFlowMultiplicityFT' constraint is violated on 'Final Task Prepare for the Mission FT'
48	The 'IncomingFlowMultiplicityFT' constraint is violated on 'Final Task Perform Transmission of Voice FT'
49	The 'IncomingFlowMultiplicityFT' constraint is violated on 'Final Task End the Surveillance FT'
50	The 'IncomingFlowMultiplicity' constraint is violated on 'Task Sağlam Personeli Tedavi Gücüne Göre Rassal Olarak Hastalandýr/Yaralandýr'
51	The 'IncomingFlowMultiplicity' constraint is violated on 'Task Prepare Path Plan '
52	The 'IncomingFlowMultiplicity' constraint is violated on 'Task Perform Surveillance of the area'
53	The 'IncomingFlowMultiplicity' constraint is violated on 'Task Perform Pre-Analysis of Coverage'
54	The 'HangingTask' constraint is violated on 'Task Detect the locations of Friend Forces'

55	The 'FlowDeadEnd' constraint is violated on 'Task Sequence Data '
56	The 'FlowDeadEnd' constraint is violated on 'Task Recognize using Radar'
57	The 'FlowDeadEnd' constraint is violated on 'Task Recognize and Identify Targets using Thermal Camera'
58	The 'FlowDeadEnd' constraint is violated on 'Task Recognize and Identify Targets using Personnel'
59	The 'FlowDeadEnd' constraint is violated on 'Task Recognize and Identify Targets using Day Vision Camera'
60	The 'FlowDeadEnd' constraint is violated on 'Task Query Data'
61	The 'FlowDeadEnd' constraint is violated on 'Task Perform Transmission of Voice '
62	The 'FlowDeadEnd' constraint is violated on 'Task Perform Transmission of Voice '
63	The 'FlowDeadEnd' constraint is violated on 'Task Perform Transmission of Image'
64	The 'FlowDeadEnd' constraint is violated on 'Task Perform Transmission of Image'
65	The 'FlowDeadEnd' constraint is violated on 'Task Perform Transmission of Data'
66	The 'FlowDeadEnd' constraint is violated on 'Task Perform Transmission of Data'
67	The 'FlowDeadEnd' constraint is violated on 'Task Input New Data'
68	The 'FlowDeadEnd' constraint is violated on 'Task Handover the mission'
69	The 'FlowDeadEnd' constraint is violated on 'Task Detect using Thermal Camera '
70	The 'FlowDeadEnd' constraint is violated on 'Task Detect using Radar'
71	The 'FlowDeadEnd' constraint is violated on 'Task Detect using Personnel'
72	The 'FlowDeadEnd' constraint is violated on 'Task Detect using Embedded Sensor'
73	The 'FlowDeadEnd' constraint is violated on 'Task Detect using Day vision Camera'
74	The 'FlowDeadEnd' constraint is violated on 'Task Delete Data'
75	The 'FlowDeadEnd' constraint is violated on 'Task Data Visualization on Map'
76	The 'FinalTaskRequirement' constraint is violated on 'Task Perform Transmission'
77	The 'FinalTaskRequirement' constraint is violated on 'Task Perform Transmission'
78	The 'ExistsOutgoingGuardCondition' constraint is violated on 'Decision Point Select the Activity'
79	The 'ExistsOutgoingGuardCondition' constraint is violated on 'Decision Point Select Communication Medium for Voice'
80	The 'ExistsOutgoingGuardCondition' constraint is violated on 'Decision Point Select Communication Medium for Image'
81	The 'ExistsOutgoingGuardCondition' constraint is violated on 'Decision Point Detection End Conditions Satisfied?'

APPENDIX B: KAMA-DV METAMODEL DEFINITION IN EMF

In this Appendix, we include the KAMA-DV metamodel definition expressed in ecore format.

1. KAMADVMetamodel.ecore

```
import ecore : 'http://www.eclipse.org/emf/2002/Ecore#/' ;

package kamametamodel : kamametamodel = 'http://kamametamodel/1.0'
{
class Task
{
invariant FlowDeadEnd:
self.incomingFlow->size()= 1
implies self.outgoingFlow->size()>= 1;
invariant HangingTask:
self.outgoingFlow->size()= 1
implies self.incomingFlow->size()>= 1;
invariant InitialTaskRequirement:
self.consistsOfIN->size()> 0
implies self.consistsOfIN->exists(oclIsTypeOf(InitialTask));
invariant FinalTaskRequirement:
self.consistsOfIN->size()> 0
implies self.consistsOfIN->exists(oclIsTypeOf(FinalTask));
invariant SelfconsistOf:
self-> closure(consistsOf)->excludes(self);
invariant IncomingFlowMultiplicity:
self.incomingFlow->size()<= 1;
invariant OutgoingFlowMultiplicity:
self.outgoingFlow->size()<= 1;
property incomingFlow#targetTask : TaskFlow[*];
property outgoingFlow#sourceTask : TaskFlow[*];
attribute ModelID : String[?];
attribute Name : String[?];
property consistsOf : Task[*] { composes };
property RepresentedBy#Represents : TaskFlowDiagram[*]
{ composes };
property consistsOfIN : IntermediateNode[*] { composes };
property ConsistsOfTF : TaskFlow[*] { composes };
}

class TaskFlow
{
invariant SourceTargetDifference2:
```

```

    (self.targetTask->size() = 1 and self.sourceTask->size() = 1)
    implies not (self.targetTask = self.sourceTask);
invariant TaskflowTargetType:
    (self.targetTask.oclIsTypeOf(Task) or
     self.target.oclIsTypeOf(DecisionPoint) or
     self.target.oclIsTypeOf(SynchronizationPoint) or
     self.target.oclIsTypeOf(FinalTask));
invariant TaskflowSourceType:
    (self.sourceTask.oclIsTypeOf(Task) or
     self.source.oclIsTypeOf(DecisionPoint) or
     self.source.oclIsTypeOf(SynchronizationPoint) or
     self.source.oclIsTypeOf(InitialTask));
invariant SourceTargetDifference1:
    (self.target->size() = 1 and self.source->size() = 1)
    implies not(self.target = self.source);
attribute ModelID : String[?];
attribute GuardCondition : String[?];
property target#incomingFlow : IntermediateNode[?];
property source#outgoingFlow : IntermediateNode[?];
property sourceTask#outgoingFlow : Task[?];
property targetTask#incomingFlow : Task[?];
}
class IntermediateNode { abstract }
{
invariant SequentialIntermediateNode:
    self.outgoingFlow->forall (target=null);
property incomingFlow#target : TaskFlow[*];
property outgoingFlow#source : TaskFlow[*];
attribute ModelID : String[?];
}
class SynchronizationPoint extends IntermediateNode
{
invariant UniqueIncomingforForkType:
    self.outgoingFlow->size() >1
    implies self.incomingFlow->size() = 1;
invariant UniqueOutgoingforJoinType:
    self.incomingFlow->size() >1
    implies self.outgoingFlow->size() = 1;
invariant FlowType:
    (( self.incomingFlow->forall(oclIsTypeOf(TaskFlow))) and
     (self.outgoingFlow->forall(oclIsTypeOf(TaskFlow))));
attribute isJoin : Boolean[?];
attribute isJoinDerived : Boolean[?] { derived,volatile }
{
    derivation:
    if (incomingFlow->size>1) then isJoinDerived=true
    else isJoinDerived=false
    endif;
}
}
class DecisionPoint extends IntermediateNode
{
invariant UniqueOutgoingFlowGuardCondition:
    self.outgoingFlow->forall(c1, c2 | c1 <> c2 implies
c1.GuardCondition <> c2.GuardCondition);
invariant ExistsOutgoingGuardCondition:
    self.outgoingFlow->forall(GuardCondition <> null);
}

```



```

invariant FlowType:
    ((self.incomingFlow->forall(oclIsTypeOf(TaskFlow))) and
     (self.outgoingFlow->forall(oclIsTypeOf(TaskFlow))));
attribute isMerge : Boolean[?];
attribute isMergeDerived : Boolean[?] { derived,volatile }
    {
        derivation:
        if (incomingFlow->size>1) then isMergeDerived = true
        else isMergeDerived=false
        endif;
    }
}
class InitialTask extends IntermediateNode
{
    invariant NoIncomingFlow:
        self.incomingFlow->size() = 0;
    invariant OutgoingFlowMultiplicityIT:
        self.outgoingFlow->size() <= 1;
}
class FinalTask extends IntermediateNode
{
    invariant NoOutgoingFlow:
        self.outgoingFlow->size() = 0;
    invariant IncomingFlowMultiplicityFT:
        self.incomingFlow->size() <= 1;
}
class TaskFlowDiagram
{
    invariant NameEquivalence:
        self.Name = self. Represents.Name;
    attribute ModelID : String[?];
    attribute Name : String[?];
    property Represents#RepresentedBy : Task[?];
}
}
}

```

2. OCL Constraints Related to KAMA Metamodel Rules

In this section, rules of original KAMA metamodel are listed as OCL constraints. ID field is based on the constraints listing in the original KAMA metamodel. If it is not traceable to KAMA the code is based on the section it occurs in this work. This enables checking of the rules using EMF environment.

Code	Rule	OCL Constraint
3.3.b.3A	$(\forall t_i, t_j \in T((t_i, t_j) \in TF \text{ implies } \neg \exists t_k \in T(k \neq j, (t_i, t_k) \in TF)))$	invariant OutgoingFlowMultiplicity: <code>self.outgoingFlow->size() <= 1;</code>
3.3.b.3B	$(\forall t_i, t_j \in T((t_i, t_j) \in TF \text{ implies } \neg \exists t_k \in T(k \neq i, (t_k, t_j) \in TF)))$	invariant IncomingFlowMultiplicity: <code>self.incomingFlow->size() <= 1;</code>
3.3.g.1	“The connections coming into and going out of a <i>decisionPoint</i> must be <i>taskFlows</i> .”	invariant FlowType: <code>((self.incomingFlow->forall(oclIsTypeOf(TaskFlow))) and (self.outgoingFlow->forall(oclIsTypeOf(TaskFlow))));</code>
3.3.g.2	$\forall d_i \in D(((d_i, t_j) \in TF, (d_i, t_k) \in TF, i \neq k) \text{ implies } (d_i, t_j).guard \neq (d_i, t_k).guard)$	invariant UniqueOutgoingFlowGuardCondition: <code>self.outgoingFlow->forall(c1, c2 c1 <> c2 implies c1.GuardCondition <> c2.GuardCondition);</code>
3.3.g.3	$\forall d_i \in D((d_i, t_j) \in TF \text{ implies } (d_i, t_j).guard \neq \emptyset)$	invariant ExistsOutgoingGuardCondition: <code>self.outgoingFlow->forall(GuardCondition <> null);</code>
3.3.h.1	$\forall s_i \in S(((t_j, s_i), (t_k, s_i) \in TF, j \neq k) \text{ implies } (s_i, t_l) \in TF \text{ for only one } l).$	invariant UniqueIncomingforForkType: <code>self.outgoingFlow->size() > 1 implies self.incomingFlow->size() = 1;</code>
3.3.h.2	$\forall s_i \in S(((s_i, t_j), (s_i, t_k) \in TF, j \neq k) \text{ implies } (t_l, s_i) \in TF \text{ for only one } l).$	invariant UniqueOutgoingforJoinType: <code>self.incomingFlow->size() > 1 implies self.outgoingFlow->size() = 1;</code>

3.3.h.3A	$\forall s_i \in S((t_j, s_i) \in AS$ <i>implies</i> $(t_j, s_i) \in TF)$ <i>and</i> $\forall s_i \in S((t_j, s_i) \in AS$ <i>implies</i> $(t_j, s_i) \notin (AS - TF))$	invariant SynchronizationFlowType: $((\mathbf{self.incomingFlow} \rightarrow \text{forall}(\text{oclIsTypeOf}(\text{TaskFlow}))$ <i>and</i> $(\mathbf{self.outgoingFlow} \rightarrow \text{forall}(\text{oclIsTypeOf}(\text{TaskFlow})))$);
3.3.h.3B	$\forall s_i \in S((s_i, t_j) \in AS$ <i>implies</i> $(s_i, t_j) \in TF)$ <i>and</i> $\forall s_i \in S((s_i, t_j) \in AS$ <i>implies</i> $(s_i, t_j) \notin (AS - TF))$	Same as 3.3.h.3A.
3.3.i.1	$\forall i_j \in I((x, i_j) \notin TF).$	invariant NoIncomingFlow: $\mathbf{self.incomingFlow} \rightarrow \text{size}() = 0$;
3.3.j.1	$\forall f_i \in F((f_i, x) \notin TF)$	invariant NoOutgoingFlow: $\mathbf{self.outgoingFlow} \rightarrow \text{size}() = 0$;
3.3.p.1.A	$\forall (x, y) \in TF$ $(x \in (T \cup D \cup S \cup I))$	invariant TaskflowSourceType: $(\mathbf{self.sourceTask.oclIsTypeOf}(\text{Task})$ or $\mathbf{self.source.oclIsTypeOf}(\text{DecisionPoint})$ or $\mathbf{self.source.oclIsTypeOf}(\text{SynchronizationPoint})$ or $\mathbf{self.source.oclIsTypeOf}(\text{InitialTask}))$;
3.3.p.1.B	$\forall (x, y) \in TF$ $(y \in (T \cup D \cup S \cup F))$	invariant TaskflowTargetType: $(\mathbf{self.targetTask.oclIsTypeOf}(\text{Task})$ or $\mathbf{self.target.oclIsTypeOf}(\text{DecisionPoint})$ or $\mathbf{self.target.oclIsTypeOf}(\text{SynchronizationPoint})$ or $\mathbf{self.target.oclIsTypeOf}(\text{FinalTask}))$;
3.3.p.2	“Only one <i>taskFlow</i> may exist between the same source and target”	invariant SourceTargetDifference2: $(\mathbf{self.targetTask} \rightarrow \text{size}() = 1$ <i>and</i> $\mathbf{self.sourceTask} \rightarrow \text{size}() = 1)$ implies not $(\mathbf{self.targetTask} = \mathbf{self.sourceTask})$;

3. OCL Constraints for Dynamic Properties defined in KAMA-DV

In this section the constraints of KAMA-DV metamodel that are used in the case studies are listed as OCL constraints. Id is the section number in the KAMA-DV metamodel. The OCL constraints are simplified based on the metamodel definition for the case study.

ID	Rule	OCL Constraint
6.2.2.6.1	Derived Attribute isJoin	<pre>attribute isJoinDerived : Boolean[?] { derived,volatile } derivation: if (incomingFlow->size>1) then isJoinDerived=true else isJoinDerived=false endif; }</pre>
6.2.2.7.1	Derived Attribute isMerge	<pre>attribute isMergeDerived : Boolean[?] { derived,volatile } derivation: if (incomingFlow->size>1) then isMergeDerived = true else isMergeDerived=false endif; }</pre>
6.2.2.1.2	Hanging Task	<pre>invariant HangingTask: self.outgoingFlow->size()= 1 implies self.incomingFlow->size()= 1;</pre>
6.2.2.1.2	Flow Dead End	<pre>invariant FlowDeadEnd: self.incomingFlow->size()= 1 implies self.outgoingFlow->size()= 1;</pre>
6.2.2.1.4	Existence of initialTasks	<pre>invariant InitialTaskRequirement: self.consistsOfIN->size()> 0 implies self.consistsOfIN->exists(oclIsTypeOf(InitialTask));</pre>
6.2.2.1.5	Existence of finalTasks	<pre>invariant FinalTaskRequirement: self.consistsOfIN->size()> 0 implies self.consistsOfIN->exists(oclIsTypeOf(FinalTask));</pre>
4.2.5 (optional rule)	Multiple sequential occurrence of intermediateNodes	<pre>invariant SequentialIntermediateNode: self.outgoingFlow->forall (target=null);</pre>
6.2.2.1.3	Task does not ConsistOf itself	<pre>Invariant ForbiddenSelfConsistOf: self-> closure(consistsOf)->excludes(self)</pre>

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Eryılmaz, Utkan
Nationality: Turkish (TC)
Date and Place of Birth: 1 January 1977 , Denizli
Marital Status: Single
email: utkaneryilmaz@gmail.com

EDUCATION

Degree	Institution	Year of Graduation
MS	METU Industrial Engineering	2006
MBA	METU Management	2001
BS	METU Electrical and Electronics Engineering	1999
High School	Denizli Anatolian High School, Denizli	1995

WORK EXPERIENCE

Year	Place	Enrollment
1999-2009	METU MODSIMMER	Research Assistant
1998	Erbakır A.Ş., Denizli	Intern Engineering Student
1997	Türk Telekom, Ankara	Intern Engineering Student

FOREIGN LANGUAGES

Advanced English

PUBLICATIONS

1. Eryılmaz U., Bilgen S., “Verification of Conceptual Models Using Metamodel Evolution”, submitted to “Simulation, Transactions of the Society for Modeling and Simulation International”.
2. Eryılmaz U., Bilgen S., “A Metamodel Based Approach for Conceptual Model Verification”, submitted to 26th International Symposium on Computer and Information Sciences.
3. Eryılmaz U., Karagoz A. “KAMA: A Tool for Developing Conceptual Models For C4ISR Simulations”, Proceedings of the European Simulation Interoperability Workshop (EUROSIW), (2009), İstanbul, Turkey.

4. Sancar H., Cagiltay K., Isler V., Tamer G., Ozmen N., Eryılmaz U. “Developing a Validation Methodology for Educational Driving Simulators and a Case Study”, Proceedings of the 13th International Conference on Human-Computer Interaction. Part IV: Interacting in Various Application Domains, (2009), p. 760-769.
5. Keskin U., Eryılmaz U., Parlak F., İşler V. “Kritik Altyapıların Korunması ve Kriz Yönetimi Alanında Modelleme ve Simülasyonun Kullanımı”, 4th National Defense Technologies Conference, Vol. 1, (2008), p. 311-320 (In Turkish)
6. Eryılmaz U., Bilgen, Molyer O. “Doğrulama ve Geçerleme Yaklaşımlarının Görev Uzaı Kavramsal Modellerine yönelik İncelenmesi”, 3rd National Defense Technologies Conference, Vol. 1", (2006), p. 63-72 (In Turkish)
7. Eryılmaz U. “Hybrid Ranking Methods Based on DEA and Outranking Methods”, MSc thesis, Industrial Engineering, Middle East Technical University.
8. Eryılmaz U., Karasakal E. “Veri Zarflama Analizi ve Baskınlık İlişkileri Metodlarını Temel Alan Hibrid Bir Sıralama Metodu”. 26th Operations Research and Industrial Engineering National Conference, (2006), p.498-501. (In Turkish)
9. Eryılmaz U., Karasakal E. “A Hybrid Ranking Method Based on DEA and Outranking Methods”. 18th International Conference on Multiple Criteria Decision Making, (2006), p.81.

HOBBIES

Trekking, hiking, playing football, reading on post-structuralism