Ministry of Higher Education and Scientific Research

Ferhat Abbas University – Setif 1
UFAS1 – Algeria

THESIS
entitled

Transcription des spécifications M-UML vers le système Maude

Presented at
Faculty of Sciences
Department of
Informatics
by
Mr Mourad KEZAI
For
Doctor of Sciences Degree

The 06/02/2023.

Committee:

Pr. Abderraouf Bouziane	University of Bourdj-bouariridj	Jury president
Pr. Abdallah Khababa	University of Setif 1	Supervisor
Dr. Ammar Boucherit	University of Eloued	Reviewer
Dr. Abdelaziz Lakhfif	University of Setif 1	Reviewer
Dr. Lyazid Toumi	University of Setif 1	Reviewer
Dr. Mohamed Amine Beghoura	University of Bourdj-bouariridj	Reviewer

2023

Acknowledgements

I would like,first of all, to express my deep gratitude to my advisor, *Professor Abdallah Khababa* to give me the opportunity to discover the field of formal methods and model transformation, both for his valuable scientific advice as well as his valuable human side through his patience, support and encouragement and his willingness to listen and give advice while still giving me enough independence to follow my own ideas, for all of this and more - thank you.

I would like to thank all the professors with whom I worked at the University of Setif over the past two decades and my colleagues now at the University of Batna.

I would also address a special greeting and word of thanks to *Professor Santiago Escobar* from the University of Valencia in Spain, *Professor Detlef Plump* from York University in the UK, and *Professor Alberto Lluch Llafuente* from the Technical University of Denmark. They generously gave their time to offer me valuable comments toward improving my work.

I would like to extend my thanks and appreciation to the Chairman and all members of the thesis committee to accept reviewing the thesis.

Finally, I would like to express my gratitude to my parents for their unlimited patience, Support and encouragement. I would also like to thank my brother, my sister, my sister-in-law and all the family members for their help and support during my PhD. I would like to thank all my friends and all the people who helped me in preparing this thesis, whether from near or far.

Abstract

First, multi-agent systems form an interesting type of company modeling and, as such, have very broad fields of application. Therefore, the technology field of "mobile agents" represents a new and important concept in artificial intelligence and software engineering. Despite the interest in this technology, most approaches at present do not allow a sufficient and complete design of mobile agents.

On the other hand, the Maude system is a specification, object-oriented programming and proof environment based on rewriting logic which is a unifying semantic framework of several concurrency models. In addition, the Maude system also has a set of tools allowing the simulation and the analysis of accessibility by the LTL (Linear Temporal Logic) Model-Checker enabling the verification and the prototyping of the properties of the specifications.

Finally, the UML graphical method and its extensions make it possible to represent systems synthetically and intuitively, multi-agent systems. However, they are not equipped with tools for formal verification. In this thesis, we address an approach for the transcription of M-UML diagrams in to the Maude system. We applied our proposed three-stage approach to the Statechart diagram, one of the most important diagrams in M-UML. The technique involves systematically deriving a formal Maude description from this type of diagram analysis, and we obtained very satisfactory results. The proposed transcription makes it possible to obtain an algebraic specification expressed with the rewriting logic. The latter will be used for the verification of properties of mobile agent-based systems.

Table of Contents

Acknowledgements i
Abstract ii
Table of Contents iii
List of Figures v
List of Tables vii
List of Abbreviations viii
General Introduction 1

Chapter 1: Distributed Systems and Mobile Agents 8

1.1 Introduction 8
1.2 Distributed systems and mobile agents 8
1.3 Mobiles agent 13
1.4 Mobile Agent Systems 15
1.5 Communication between mobile agents 19
1.6 Field of application of mobile agents 20
1.7 Presentation of some mobile agent platforms 21
1.8 Benefits of the Mobile Agent Paradigm 22
1.9 Limits and Disadvantages of the Mobile Agent Paradigm 24
1.10 Conclusion 26

Chapter 2: UML & Mobile UML 28

2.1 Introduction 28
2.2 Modeling 28
2.3 Unified Modeling Language (UML) 32
2.4 Extension of UML 40
2.5 Mobile UML 42
2.6 Conclusion 43

Chapter 3: Comparing Modeling Approaches for Mobile Agent-Based Systems . 46

3.1 Introduction 46
3.2 Related Works and Background 46
3.3 classification of software system modeling approaches based on mobile agents 51
3.4 The comparison framework 58
3.5 Conclusion 65

Chapter 4: Model Transformation 68

4.1 Introduction 68
4.2 An approach for the MDA architecture 68

4.3 Graph transformation 76
4.4 Conclusion 80
Chapter 5: Rewriting Logic and The Maude language 82
5.1 Introduction 82
5.2 Rewriting Logic 82
5.3 Rewrite theory 83
5.4 Deduction rules 84
5.5 Maude System 86
5.6 Maude language syntax 88
5.7 Maude modules 91
5.8 Predefined Modules 94
5.9 Execution and formal analysis under Maude 94
5.10 Formal analysis and verification of properties 95
5.11 Execution of Maude 98
5.12 Conclusion 100
Chapter 6: An integrated Mobile-UML/Maude system approach 102
6.1 Introduction 102
6.2 Related works 102
6.3 UML Mobile 104
6.4 The Maude short overview 107
6.5 The proposed graph grammar 108
6.6 The proposed approach 116
6.7 M-UML and Inconsistencies 120
6.8 discussion of the proposed approach 121
6.9 Potential limits 121
6.10 Conclusion 122
Chapter 7: Case study 125
7.1 Introduction 125
7.2 Presentation of the example 125
7.3 Application of the transcription process of M-UML diagrams 126
General conclusion and perspectives 143
Bibliography 145

List of Figures

Figure 1.1: Client/server organization ... 9
Figure 1.2: Remote assessment ... 11
Figure 1.3: Code on demand ... 12
Figure 1.4: Agent mobile ... 12
Figure 1.5: Place and Region ... 18
Figure 1.6: The migration of the Mobile Agent ... 19
Figure 1.7: Communication between mobile agents ... 20
Figure 2.1: Four-level meta-modeling architecture ... 33
Figure 2.2: Classification of UML types ... 36
Figure 2.3: Two simple states in a state-transition ... 38
Figure 2.4: Transition in Statechart diagram ... 40
Figure 2.5: Example of stereotype ... 41
Figure 2.6: Example of constraints ... 41
Figure 3.1: Classification of UML approaches mobile agent extension ... 48
Figure 3.2: Development life cycle of mobile agent-based software systems ... 49
Figure 3.3: Categories of Security Issues in Mobile Agent Systems ... 50
Figure 3.4: Classification of existing modeling approaches according to ... 52
Figure 3.5: The UML diagram types supported by the UML-Extended languages mobile agent based ... 60
Figure 3.6: The viewpoint types supported by the UML-Extended languages based mobile based ... 62
Figure 4.1: Variants of MDE ... 70
Figure 4.2: The four levels of abstraction for MDA ... 71
Figure 4.3: Model transformation process driven by meta models ... 73
Figure 4.4: Undirected graph ... 76
Figure 4.5: Simple directed graph ... 77
Figure 4.6: Labeled directed graph ... 77
Figure 4.7: G' is a subgraph of G ... 78
Figure 5.1: Execution of Maude ... 99
Figure 5.2: Running the reduce command ... 100

Figure 6.1: Mobile transition state graph concept 105
Figure 6.2 : Mobile Statechart diagram modeling 107
Figure 6.3 : Execution of Maude 108
Figure 6.4: Simple state transformation 109
Figure 6.5: Mobile state transformation 110
Figure 6.6: Simple transition transformation between two simple states 110
Figure 6.7: Mobile transition transformation between a simple state and a Mobile state 111
Figure 6.8: Mobile transition transformation between two mobile states 111
Figure 6.9: Mobile transition transformation between a mobile state and a simple state 112
Figure 6.10 : Remote transition transformation between two mobile states 112
Figure 6.11: Simple transition transformation between two mobiles states 113
Figure 6.12 : Transformation of an “agent return” simple transition between a mobile state and a simple state 113
Figure 6.13: Remote transition transformation between two simple states 114
Figure 6.14: Transformation of an “agent return” mobile transition between a mobile state and a simple state 114
Figure 6.15: Translation Process Overview 117
Figure 7.1: Mobile Statechart diagram of the Vote Collector (VC.) 127
Figure 7.2: Modeling vote statechart 1 in Maude 129
Figure 7.3: The model with mobility attributes on states in Maude 131
Figure 7.4: Final Maude specification for M-UML statechart diagram. 131
Figure 7.5: Maude's vote statechart code execution. 133
Figure 7.6: "search command" execution for Maude vote statechart 134
Figure 7.7: " Show graph instruction " execution for Maude vote statechart 135
Figure 7.8: " Search specific path " execution for Maude vote statechart 136
Figure 7.9: " search for all mobile states " execution for Maude vote 2 statechart .. 137
Figure 7.10: "search for all non-mobile states " execution for Maude vote 2 statechart138
Figure 7.11: " search for all non-mobile states with condition " execution for Maude vote 3 statechart 139
Figure 7.12: " Check for mobile states not reached via mobile transition " execution140

List of Tables

Table 2.1: Classification and Use of Languages or Methods 29
Table 2.2: Shorts description of structure UML diagrams. 34
Table 2.3: Shorts description of dynamic UML diagrams 35
Table 2.4: The structural elements of MSD 43
Table 3.1: Progress review of classification 52
Table 3.2 : Coverage of UML Extensions Diagrams 58
Table 3.3: Percentage of Coverage of UML extensions for initial diagrams 59
Table 3.4: Viewpoint Support for the UML- languages based mobile agent 61
Table 3.5: Percentage of Viewpoint Support for the UML- languages based mobile agent 62
Table 6.1: Maude commands example 108
Table 6.2: Representation of control structures in Maude. 115
Table 6.3: The three steps of the translation Process 118
Table 6.4: Comparative of our approach with the closest similar methods 118
Table 7.1: State abbreviations. 127
Table 7.2: Transition abbreviations. 128

List of Abbreviations

AADL: Architecture Analysis and Design Language

Atom3: A Tool for Multi-Formalism and Meta-Modeling

CASE: Computer Aided Software Engineering

CIM: Computational Independent Model

CPN: Colored Petri Nets

CSP: Communicating Sequential Processes

CWM: Common Warehouse Metamodel

DBMS: Database Management System

DPF: Destination Platform

FIPA: Foundation for Intelligent Physical Agents

LDAP: Lightweight Directory Access Protocol

LHS: Left Hand Side

LOTOS: Language of Temporal Ordering Specifications

LTL: Linear Temporal Logic

MAS: Multi-Agent System

MASIF: Mobile Agent System Interoperability Facility

MDA: Model-Driven Architecture

MDE: Model-Driven Engineering

MOF: Meta Object Facility

MOM: Message Oriented Middleware

MSD: Mobile Statechart Diagram

M-UML: Mobile Unified Modeling Language

OCL: Object Constraint Language

OMG: Object Management Group

OMT: Object-Modeling Technique

PCA: Personal Communicator Agent

PDA: Personal Digital Assistant

PIM: Platform Independent Model

PSM: Plateform Specific Model

RHS: Right Hand Side

RPC: Remote Procedure Calling

SPF: Source Platform

SSH: Secure Shell Protocol

SSL Secure Sockets Layer

UML: Unified Modeling Language

VSM: Virtual Shared Memory

XMI: Xml Metadata Interchange

XML: Extensible Markup Language

General Introduction

The last decade has seen considerable evolution in computing, especially with the emergence of smartphones and the expansion of networks, cloud computing and finally, the Internet of Things. This substantial development has led to the emergence of many new challenges and increasingly complex situations. This has pushed researchers to try to find other models than those that already exist (especially the Object-Oriented paradigm) to deal with these new difficulties encountered, especially in distributed systems.

The mobile agent paradigm has been considered one of the most successful models proposed to represent these systems, which provides excellent flexibility over previous techniques and will allow for significant expansion in the development of many important systems in various fields. This is mainly due to the various characteristics that characterize mobile agents, such as mobility, autonomy, adaptability, etc.

Mobile agent-based software systems are a special type of software system that takes advantage of the benefits of mobile agents. In order to give a simplified view of the working mechanism of this type of agent, we can say that it consists of several connected agents working in several distributed places, and they can move between them to achieve their objectives. This leads to a new paradigm that can solve many complex problems in several fields, such as network management, e-commerce, e-learning etc.

Modeling and analyzing these systems in the design phases can avoid many problems and thus significantly improve and the development of these systems. However, providing an approach of the formal development and verifying such systems is not an easy task. In fact, two main aspects must be taken into account, namely mobility and communication.

In return for the significant advantages that mobile agents give us, we face a serious problem: the absence of a formal specification for this kind of software system, therefore, this work attempts to look for an approach for specifying software systems based on mobile agents using the Maude system. Our choice of the Maude system is motivated, on the one hand, by the power of expression of the latter due to their rich syntactic notations based on an algebraic formalism, making it possible to provide highly compact models. On the other hand, they have healthy semantics defined in terms of the rewriting logic allowing them to express remarkably and intuitively the dynamic behavior and the parallelism which characterizes the distributed

systems. This system also allows for formal verification through specifications expressed in the Maude language, a rewriting logic language that can be executed on two compiler versions: Full Maude and Core Maude.

PROBLEM STATEMENT

Research work on software systems based on mobile agents aims to develop models, languages, and tools for the design of such software systems based on mobile agents. The latter is not only built in a modular way with a large number of components. Still, it must also consider the characteristics of this type of systems, such as mobility and communication. This implies that these components are modeled in different modeling languages or formalisms [92]. Verification of the properties of these systems is recognized as a difficult problem and faces several issues. The first obstacle appears at the level of the choice of modeling techniques or modeling languages to be used to model the various components of the system. If the modeling language is too expressive, then we cannot mathematically analyze it automatically. A second obstacle is the verification of the system's global behavior. The models representing such a system are modeled in different languages or formalisms, making any global reasoning about the system complex.

The main objective in this field is to study formal models, which allow us to describe these systems and their constraints (design, specification), build them (programming, simulation, synthesis, execution), and analyze them (validation, verification). Verification is a major concern for many software systems based on mobile agents. Hence the importance of validating these systems, i.e., testing, verification, and certification. There is a real and pressing need to develop effective methods and tools for verifying mobile agent-based systems to keep pace with their ever-increasing evolution. In addition, the slightest flaw will be exploited by potential malicious users.

The verification and validation phases make it possible to check that the system satisfies the expected properties. As a result, any problem related to qualitative behavior (security, fairness, absence of blocking, etc.), as well as quantitative behavior (loss of messages, average transmission speed, etc.), is detected and corrected very early in the cycle of development. One of the promising avenues for reducing these verification costs is the use of formal methods. These methods are based on mathematical foundations and make it possible to carry out verification tasks with high added value during development [64]. To be able to be carried out,

the verification requires a formal description of the system and a formal specification of its properties.

Model-driven engineering (MDE) methods are increasingly used in the industry to handle this level of complexity at the design level [43].

Model-Driven Architecture (MDA) is a development approach proposed by the Object Management Group (OMG) [20]. It makes it possible to separate a system's functional specifications from its implementation specifications.On a given platform the MDA approach makes it possible to carry out the same model on several platforms thanks to standardized projections.

The implementation of MDA is entirely based on models and their transformations. This approach consists of manipulating different application models to be produced, from a very abstract description to a representation corresponding to the effective implementation of the system [128].

The MDA approach can be instantiated using different languages. For this, many languages that can describe this have come into existence and now offer a number of functions.

In the literature, several suggestions exist to provide a complete expressive language for these systems. M-UML (mobile UML) [41] is one of the most powerful of these attempts; It is an extension of the famous UML (Unified Modeling Language) to model software systems based on mobile agents by covering the aspects of mobility which is the new concept inherent in these systems. M-UML provides a basic version of mobile statechart diagrams to describe agent behavior in terms of state change.

Since M-UML is an extension of UML, and therefore inherits its properties, it also has semi-formal semantics as well, which makes difficult any task of automatic analysis of its diagrams.

On the other hand, formal methods offer an interesting solution to the above problem. The formal specifications will have the effect of eliminating ambiguities in the interpretation of M-UML models. A appropriate combination of the mobile agent concept and formal methods can make software development more rigorous. However, a difficulty arises due to the variety of formal models used to represent the non-functional characteristics of the components and the requirements for the complete system. Many specification formalisms dedicated to several computer systems have been proposed in the literature, such as communicating automata, transition systems, CCS process algebras, CSP, LOTOS, Estelle, Promela , etc.

The Maude system [127] can fulfil this role satisfactorily. It is a very powerful algebraic language for describing the behavior of concurrent systems with mobile communications. We chose it because of its successful experiences in this field by providing the appropriate mechanisms to describe the behavior and movement of agents in M-UML diagrams.

Applying model-checking techniques in the design cycle of agent-based systems provides proof of properties, such as the absence of run-time errors or the satisfiability of properties.

Model checking consists of building a finished model of the analyzed system and verifying the desired properties of this model. Verification requires a complete or partial exploration of the model. The main advantages of this technique are the possibility of making the exploration of the model automatic and facilitating the production of counter-examples when the property is violated.

CONTRIBUTION

This thesis has two major contributions due to the treatment of the aforementioned problems: the first one is validated and published in a journal, and the other is a comparative study of the existing approaches and it is still in the scientific arbitration phase.

The first contribution of this thesis is to provide a comprehensive comparative study of the different proposals that exist in the literature as well as a classification of approaches to modeling software systems based on mobile agents; we focus on the coverage factor of the diagrams, which have not been addressed in previous comparative studies. Based on the results of this study, we have adopted M-UML and extended it because of its advantages over others.

The second and significant contribution of this work consists in the proposal of an integrated approach UML-Mobile/Maude, for the modeling and the verification of the applications of the mobile agents more precisely through the proposal of a diagram of states-transitions mobile MSD (Mobile Statechart Diagram) as an extension of UML statechart diagrams, followed by an integrated MSD/Maude approach for the modeling and verification of software systems based on mobile agents.

In our approach, the behavior of a software system based on mobile agents is modeled by a mobile state-transition diagram; the basic idea proposed consists of giving this diagram a corresponding Maude system which allows us, through our proposed graph grammar, to automate the transfer of a source formalism (meta-model for the mobile transition state diagram)

to a target formalism (meta-model for Maude's formalism), a reconfiguration of this system is made each time an agent moves from one state to another or from one place to another.

This means that the new specifications obtained are a detailed image of the converted system in real-time. The generated Maude specifications are then used to analyze and verify these systems using Maude analysis tools such as Core Maude. This verification consists, for example, in the early detection of errors (deadlocks, livelocks, etc.), checking if certain properties are satisfied, and (or) checking the equivalence between different diagrams; a literature case study illustrates the proposed approach.

DISSERTATION PLAN

The manuscript is organized into seven chapters, plus an introduction and a conclusion. The research work's problem and contributions are discussed in the general introduction.

The first chapter is dedicated to the mobile agent paradigm. The goal is to take a look at the advantages of this technology.

The second chapter is devoted to the two languages around which we have built our approach, namely the UML language and the M-UML. The first is a semi-formal language considered the standard for object-oriented modeling systems; the second is its extended version to support mobility.

The third chapter presents a comparative study of different approaches proposed in the literature for. The study consists of the proposal of a comparison framework, classification, and evaluation of several proposals.

In the fourth chapter, we introduce the concept of model transformation, starting with presenting the concept of model transformation in the general framework. After that comes the enumeration of some types of transformations. A classification of the different approaches will be followed by a presentation of a specific model transformation framework based on graph transformation.

The fifth chapter is devoted to the rewriting logic after the definition of the formal methods. We present an overview of rewriting logic and the Maude language. The chapter ends with the concepts of model-checking and the description of Maude’s LTL model-checker

In the sixth chapter, we propose a Mobile-UML/Maude system integrated approach for transforming the M-UML statechart diagram. we have introduced new elements in MSD statechart to support mobility more easily. Then, we presented our approach, which consists of

the formalization of a semi-formal diagram MSD that we exploited using Maude formal language.

In the last chapter (the 7th), we illustrated our approach by a case study which consists of modeling and verifying an electronic voting system.

Finally, the conclusion summarizes the essential points of this work and presents the research perspectives that were recommended.

Chapter 1

Distributed Systems and Mobile Agents

Chapter 1: Distributed Systems and Mobile Agents

1.1 INTRODUCTION

The last years witnessed impressive development in all areas of computing, including the field of networks, which in turn expanded to lead to the emergence of many distributed applications. These applications require strong interactions between different entities distributed in the network that share the same resources and aim to achieve the same goals. These distributed implementations use different distributed implementation models suggested in the literature.

We begin this chapter by briefly presenting the characteristics of some distributed execution models used to implement a distributed application. We will discuss the technology of mobile agents and recall the fundamental concepts of software systems using a mobile agent base. We will continue with the properties, domain of application, communication between agents, platforms,

Finally, we will detail the advantages and disadvantages of mobile agents.

1.2 DISTRIBUTED SYSTEMS AND MOBILE AGENTS

In the development of distributed applications, programming by mobile agents has found its place as a new paradigm. However, for the construction and management of distributed systems, other organizational structures can be classified into two categories. One is for organisational structures without mobility, and the other is for structures with mobility.

1.2.1 Non-mobile organizational structures for distributed systems

In the design of a classic distributed application, which does not require mobility, several organizational structures are involved, the most common of which are the following:

1.2.1.1 Client-server

In the client/server model, a server represents an object managed at a site and services to clients. The functionalities of an application are then encapsulated in services offered/executed within servers. Servers are thus seen as service providers. There are either: (i) data servers, in which clients access data located on each server (BDMS, LDAP, etc.), (ii) or calculation servers, in which clients use the resources of each server to perform a specific task.

In the client/server model, only the code initially installed on the server can be executed on the latter, and the latter's role is to respond to the requests sent by the client processes. The interaction with a server is established by the client requesting a service. We can consider that executing a task on a server requires interaction with other servers (such as the consultation of a database). In this case, the client and server can do the same process. In the client/server model, only the client represents an application in the proper sense of the term. The function of the server is to respond to client requests. The answers depend on the requests formulated and not on the client applications which interrogate them.

The client sends a request to the server. This request describes the operation and parameters to be executed (the requested service). The client is then said to invoke an operation on the server. The server performs the service requested by the client and returns the response. It is a synchronous invocation, indeed in this type of communication, the client which sends a request to the server blocks while waiting for its response (see **Figure 1.1**)

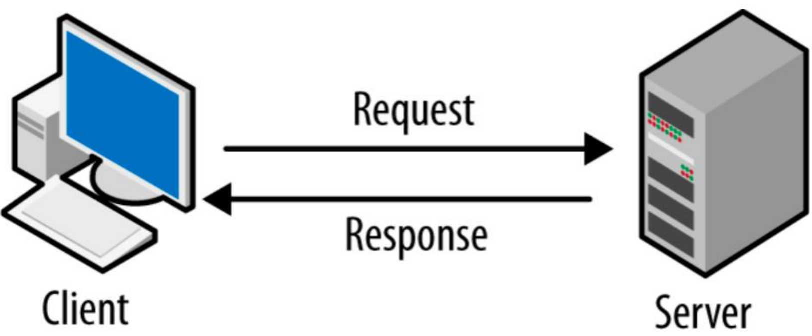


Figure 1.1: Client/server organization

1.2.1.2 asynchronous event

This model allows communication between several processes, and this is indirect. The different components of the application cooperate by sending and receiving notifications. The processes communicate via a base representing a subscription and event distribution manager**.**

This database is responsible for directing an event to its subscribers. In this model, there are two types of process: the senders who send or publish events in the database and the consumers who subscribe to specific categories of events. When the database receives an event notification, it distributes it to all the components that have declared their interest in receiving this message. Thus, this intermediary decouples the sources and the consumers of events. The

sender of communication is not obliged to specify the destination of his messages, nor does the recipient necessarily know the message's origin. This type of infrastructure [1] makes it possible to communicate using messages and is designated by MOM (Message Oriente Middleware).

1.2.1.3 Virtual shared memory (VSM)

The exchanges take place via a large shared memory which serves as a communication space between processes on computers which do not physically share their memory. Its interest is to allow the use of a programming model with advantages over models based on the exchange of messages.

Processes access shared memory by reading and updating what appears to be ordinary memory inside their address space. The system, running in the background, transparently ensures that processes running on different computers observe updates made by other methods.

The great advantage of the VSM [2] is to spare the programmer the management of the exchange of messages when he writes an application needed in the VSM. One cannot, however, completely avoid exchanging messages in a distributed system; it is, of course, necessary that the underlying system of the VSM sends the updates to the various processors, each with a local copy of the shared data [3]. Moreover, these data must be updated regularly for a question of performance and the validity of the data used.

1.2.2 Organizational structures with mobility for distributed systems

Applications using networks as infrastructure, particularly the Internet, have rapidly grown. Non-mobile organizational structures, such as the client/server approach, face significant challenges. Given the centralized nature of these approaches, the emergence of new trends for sharing knowledge and resources available on the planetary network is slowing down enormously. Therefore, the concept of mobility is an excellent alternative to non-mobile approaches.

In what follows, we present some definitions and basic concepts of mobility; we will continue with some organizational structures based on mobility for distributed systems.

1.2.2.1 Mobility

Mobility, in general, is physical or software movement in telecommunication networks. As a result, three basic concepts are widespread: physical mobility (Mobile computing), mobile code and mobile agents.

• Physical mobility is defined as a paradigm in which users of the shared network connect via portable machines [4]. The diffusion of wireless networks, and cellular telecommunication networks, explain the motivation of this paradigm. These networks require large computing and information processing capacities.

• The dynamic capacity of exchange between the code fragments and the locations on which this code will be executed defines the mobile code. The migration of the code (process, object, or procedure) ensures a load balancing between the processors connected to the network. This also improves performance in communication (gathering objects that communicate intensively on the same nodes).

• Mobile agents are software entities that can move from one site in the network to another to achieve their objectives autonomously.

1.2.2.2 Organizational structures

Among the organizational structures of mobility, we present the remote evaluation, the code on demand and the Mobile agent-based model.

1.2.2.2.1 Remote assessment

In a remote assessment interaction (see **Figure 1.2**), a client sends a code to a remote site. The receiving site uses its resources to execute the program sent.

Optionally, an additional interaction then delivers the results to the customer. In this scheme, only the code is transmitted to the server, and the code's execution takes place only on the latter. For example, the code for an SQL query sent to a database server is an example of remote evaluation.

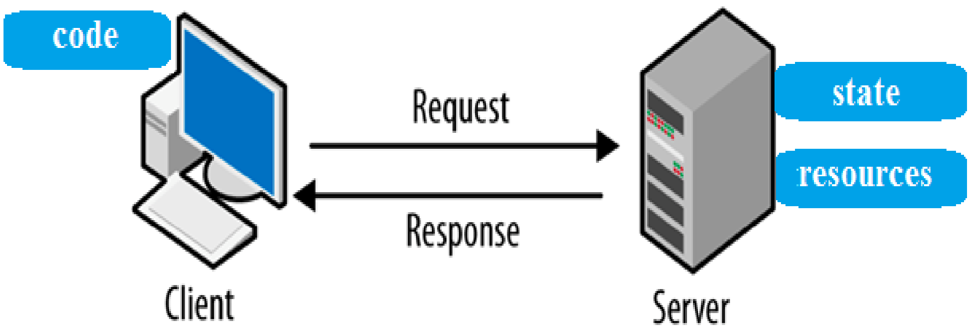


Figure 1.2: Remote assessment

1.2.2.2.2 Code on demand

In this diagram, the client process interacts with a remote site to retrieve know-how that will be executed on the client machine. Thus, the customer downloads the code necessary to perform a service. The role of the remote site is to provide the service code that will be executed on the client site (see **Figure 1.3**). Java Applets are based on this mobile code technology; it is a program loaded from a web page to be executed on the client's machine

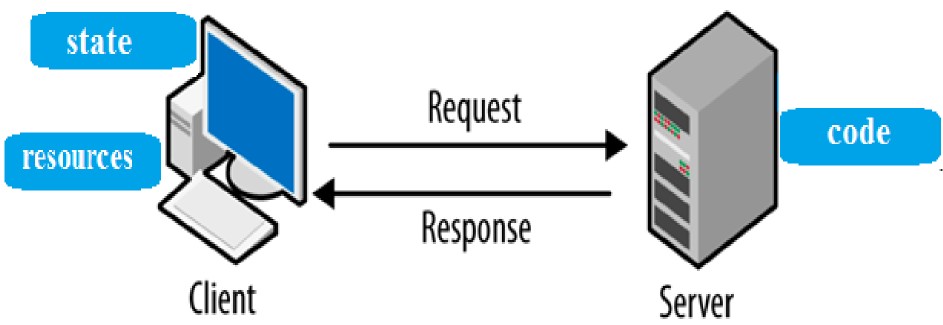


Figure 1.3: Code on demand

1.2.2.2.3 Mobile agents

Compared to the two previous diagrams, the execution of the process begins at the customer site. Since the client needs to interact with the server, that same process (code, execution state, and data) travels across the network to continue its execution and to interact locally with server resources (see **Figure 1.4**). After execution, the mobile agent eventually returns to its client to provide it with the results of its execution. In this scheme, the know-how belongs to the client, and the execution of the code is initiated on the client side and continues on the different machines visited.

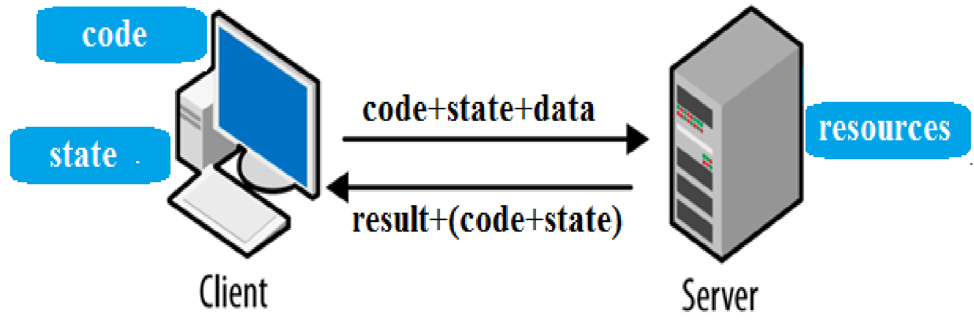


Figure 1.4: Agent mobile

In order not to get lost in the details of MAS, we directly try to focus on a particular type of multi-agent system that interests us in this work; it is mobile agents.

1.3 MOBILES AGENT

The interactions associated with the mobile agent paradigm are distinct from those associated with other mobile code paradigms due to the fact that they involve the mobility of an existing computational component. In other words, whereas the focus of the remote assessment and code-on-demand paradigms is on the transfer of code between components, the mobile agent paradigm moves an entire computational component to a remote site, along with its state, the code it needs, and some resources needed to perform the task. This is in contrast to the remote assessment and code-on-demand paradigms, which focus on the transfer of code between components.

1.3.1 Agent concept

1.3.1.1 Definitions

An agent is an entity that perceives and acts in an environment. Its agent function specifies the action it performs in response to a given sequence of percepts [5]. In general, an agent is substituted for any entity that can be considered to perceive its environment through sensors and acts on that environment through effectors. Here we present some alternative definitions according to the approach of artificial intelligence:

• According to [6], an agent is a physical or virtual entity that can act in an environment and communicate directly with other agents. This entity is driven by a set of trends (in the form of individual objectives, a function of satisfaction, or even survival, that it seeks to optimize). To achieve its goals, the agent takes into account the resources and skills at its disposal, as well as its perception and representations of its environment.

• According to [7], an agent is an autonomous active entity acting by the delegation on behalf of a client.

• In [8], we learn that an agent is a computer system located in an environment capable of acting autonomously to achieve its design objectives.

1.3.1.2 Motivation of mobile agents

A mobile agent can transport in the network where they move, code data, and the execution units associated with it. An agent migrates autonomously to the sites it deems interesting. This

new approach has made it possible to increase the robustness of applications for distributed systems, in particular for:

• **Reducing network load:** It is generally more advantageous in terms of performance to send code rather than data.

• **Moving from Code to Data**: Servers containing data provide a fixed set of operations. An agent can extend this set for the particular needs of treatment.

• **Reliability**: The life of a classic program is linked to the machine where it runs. A mobile agent can move to avoid a hardware or software error or simply a machine shutdown.

• **Dynamic system adaptation to problems:** Agents can distribute themselves across machines on the network to better consider the system's state when they need to perform their tasks.

• **The management of robustness and fault tolerance:** If a machine stops, the agent there can change the device to complete his current task.

• **Protocol encapsulation**: In distributed systems, protocols define how messages and data are exchanged. To modify a protocol, you must change the code on all the machines in the system. Agents encapsulate protocols. Changing protocol is equivalent to interacting with a new agent.

• **Component autonomy and synchronous/asynchronous execution**: Mobile terminals (PDAs, laptops, telephones, etc.) are not always connected to the network. Systems that require permanent connections are not suitable in this case. With agents, mobile devices can log in to check for messages. An agent can be dispatched to the mobile terminal at login time and work on the mobile terminal after disconnecting the connection. The same agent can wait for the next connection to send information on the network, meaning, for example, that a task has been completed

1.3.1.3 Properties

Agents generally have a large number of properties and abilities, among which we list here a few that are of interest to us:

• **Autonomy**: An autonomous agent is an entity that acts in its environment independently of external commands. An agent's autonomy concerns both its behavior and internal resources [9]. The "daemon" is an example of an autonomous agent.

• **Mobility**: A mobile agent can migrate from one machine to another to achieve its objectives.

• **Communication**: Agents can communicate with each other and with others.

By exchanging messages or according to a mode of communication established by the architecture in which the agents evolve, such as the MAS (Multi-Agent System).

• **Learning**: An agent can increase capabilities by learning using knowledge.

• **Reactivity and Pro-Activity**: a reactive agent is characterized by its reaction to external events to execute new tasks. On the other hand, a ProActive agent is characterized by the ability to anticipate changes rather than react to these changes.

1.3.1.4 Agent Classes

The term agent takes on different meanings depending on the fields of application and the expected perspectives. The classification of the different agent architectures is based on relevant criteria and a naturalistic mode (taxonomy). There are domain-based or viewpoint-based classifications [10]. In this section, we present classification as an overview.

• Cognitive agent: Classical artificial intelligence focused very early on the expression on logical bases of the deliberative behavior of a rational agent according to its beliefs and goals. This later gave rise to the first modern architectures of so-called cognitive agents.

• **Robotic agent:** We can consider the control software architecture of a robot as an agent. This architecture can also be tested and trained in simulation before being implemented. However, the physical reality of a robot brings specific problems (imprecision of perception and action, real-time aspects, evolution of the world), which make the design of such agents, particularly difficult, but also particularly rich.

• **Software agent:** this designation is justified in opposition to the term physical agent, such as a robot. The first Software Agents are Unix daemons (autonomous computer processes capable of waking up at certain times or depending on certain conditions). Computer viruses are already more sophisticated versions (notably endowed with the ability to reproduce) and harmful.

• **Mobile agent**: Born in the mid-90s. Téléscript is the first platform for mobile agents. It subsequently gave rise to many other platforms.

1.4 MOBILE AGENT SYSTEMS

1.4.1 Basic concepts

An agent that runs exclusively in the system where it was created is said to be a Static Agent. If the agent needs information unavailable in the system running, it can interact with other

agents in another system. In this case, the agent uses a communication mechanism such as RPC (Remote Procedure Calling). On the other hand, a mobile agent is not linked to the system in which it begins its execution. This Agent can move from one host to another in the network. It can transport its state and code from one environment to another in the network where it continues its execution.

A mobile agent is, therefore, able to move in its environment. It can be physical (real or simulated) or structural (execution levels, for example). Therefore, a mobile agent has devices ensuring mobility [10].

An agent system (also called an agent server) is the living space of agents. It is associated with an authority identifying the organization or party for which the agents act. The diversity of the systems developed in this way makes bridges between such systems often impossible. To allow interoperability between mobile agent systems, the OMG (Object Management Group) has shown its interest in this class of application by defining the specifications of MASIF (Mobile Agent System Interoperability) [11]. In addition, the FIPA (Foundation For Intelligent Physical Agents) [12] has launched another effort to specify the architecture of mobile agent systems.

1.4.1.1 Agent Mobility

The mobility of an agent refers to its ability to move from one computing environment to another, typically across different machines or networks, without losing its functionality or state. During its migration, the agent transports its code, its execution state as well as its resources according to the following legend:

• The agent and communication channels will be suspended on the originating machine.

• The agent state will be extracted and transferred over the network, as data, to the destination machine where it will be rendered.

• The agent resumes execution on the destination machine after being reactivated. Its communication channels will then be updated.

The details in the legend quoted above determine the mobility class of the agent. Generally, there are two major classes for mobility: high mobility and low mobility.

a-High mobility: In this class, the agent moves entirely with all its components to the destination machine, where it will resume its execution from the last interrupted instruction. This movement can be proactive or reactive. In this class, there are two types of mobility:

Cloning: In this type of mobility, an exact copy of the mobile agent is transferred over the network, and the other copy remains at the level of the original machine. The agent's communication strategies must be specified for this type of mobility.

Migration: This type of mobility defines migration itself. The mobile agent will be transferred to the destination machine without leaving a copy on the originating machine.

b- Low mobility: In this class, the mobile agent is deprived of its execution context during migration. It must restart its execution on the destination site from the beginning. Several recovery strategies can be integrated into applications using this mobility class. In the same way as for strong mobility, in this class, the movement of mobile agents can also be proactive or reactive.

1.4.1.2 The place

In the host environment of the migrating agent, the place can be seen as a fixed agent that can receive the mobile agent by offering services (**Figure 1.5**). For example, when a mobile agent moves to an agent server, it enters the space offered by the requested service. An agent server can be considered as a place in a multi-agent system. The departure and destination places of the migrating agent can be located within the same agent system or on different systems. Places are uniquely identified in the system under consideration.

1.4.1.3 The region

A region models a set of mobile agent systems that are not necessarily of the same type (see **Figure 1.5**). It represents the set of all the places and all the agents that belong to the same authority (organization).

1.4.1.4 The name of an agent

An agent's identity must be unique in its region to be able to identify it. The name assigned to an agent can be combined using their identity and other useful information.

1.4.1.5 The localisation

The location of a mobile agent is defined by the combination of its place of execution and the network address of the agent system where the place resides.

1.4.1.6 Agent and Place Authority

The authority of an agent makes it possible to identify his affiliation organization to authorize him access to requested resources. Similarly, a place has the authority of the region of its creation.

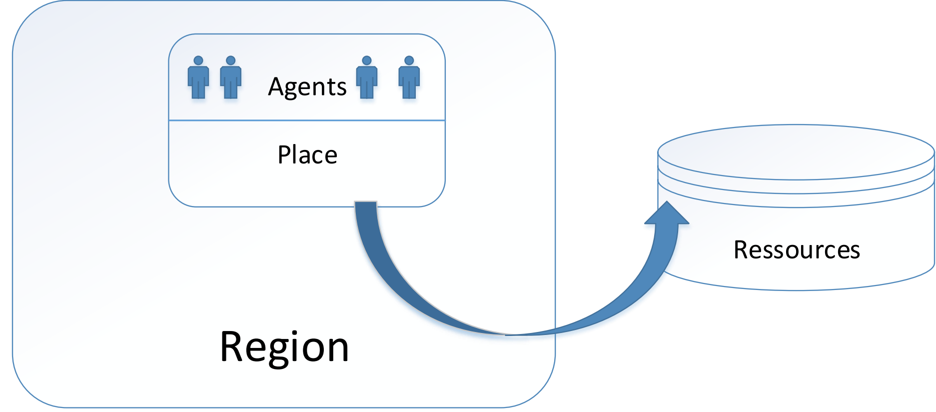


Figure 1.5: Place and Region

1.4.2 System services

We describe in this section the valuable elements of a mobile agent system for building applications.

The creation of an agent is generally done within a specified place. The creator (the system or others) must authenticate in the place and establish the authorizations and the references that the agent will possess. The creation of the agent goes through the following steps:

• **Instantiation and naming of the agent**: At the beginning, the system loads and executes the code of the agent class. The agent object is then instantiated. Ultimately, a unique identifier is assigned for the agent thus created.

• **Agent initialization**: The agent creator provides the necessary arguments to the agent to allow its initialization. After its initialization, the agent is considered completely installed in its place.

• **Migration of a mobile agent:** A mobile agent system offers all the necessary mechanisms for each agent to ensure its mobility. The migration process can be initiated by the agent itself or by others. The agent migrates according to its objectives from its current place to the specified destination. (See **Figure 1.6**)

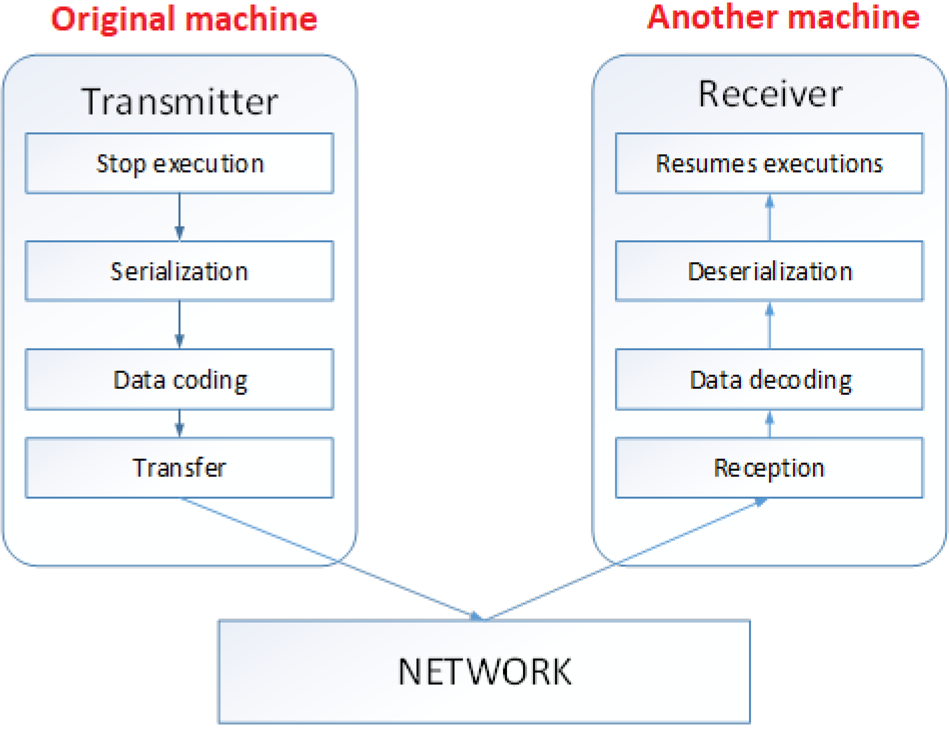


Figure 1.6: The migration of the Mobile Agent

• **Security**: Due to the distributed and heterogeneous nature of distributed systems, security plays a significant role in developing such systems. To implement a targeted security policy for developing systems based on mobile agents, designers generally use authentication mechanisms (password, certificate, etc.), and cryptography (SSH, SSL, etc.). ..) And access control (user rights, firewall). In the classic scheme of distributed applications, the critical elements are grouped on secure machines, and the focus is on external communication channels using authentication and cryptography mechanisms. For implementing code mobility, security policies are generally based on a close relationship between the site storing the program and the one executing it. Implicitly, the owner of the code is the same as that of the environment that will execute it. As for agents' mobiles, the security problem is not entirely solved. This is also the basic argument explaining the low use of this paradigm. Indeed, mobile agents represent a new field of investigation for the field of security research, on the one hand in the protection of sites against malicious agents and on the other hand in the protection of agents against malicious sites

1.5 COMMUNICATION BETWEEN MOBILE AGENTS

Agents can communicate with resident agents in the same place or with residents in other places. An agent can invoke a method of another agent as it can send messages to it if authorized. Inter-agent communication can follow three different patterns. (See **Figure 1.7**)

A. Now-type messaging: This is the most used type of messaging. It is a synchronous type. It blocks the execution of the sender of the message until the receiver has wholly downloaded the message and sent its response. (See **Figure 1.7**)

B. Future-type messaging: This is a non-blocking asynchronous messaging type. The sender retains a variable, which can be used to obtain the result. This type of messaging is beneficial when several agents communicate together. (See **Figure 1.7**)

C. One-way-type messaging: This is an asynchronous type that does not block current execution. The sender will not retain a variable for this message and the receiver will never respond. This type is proper when two agents initiate a conversation where the sending agent does not need a response from the receiving agent. This type of messaging is called fire-and-forget. (See **Figure** 1.7)

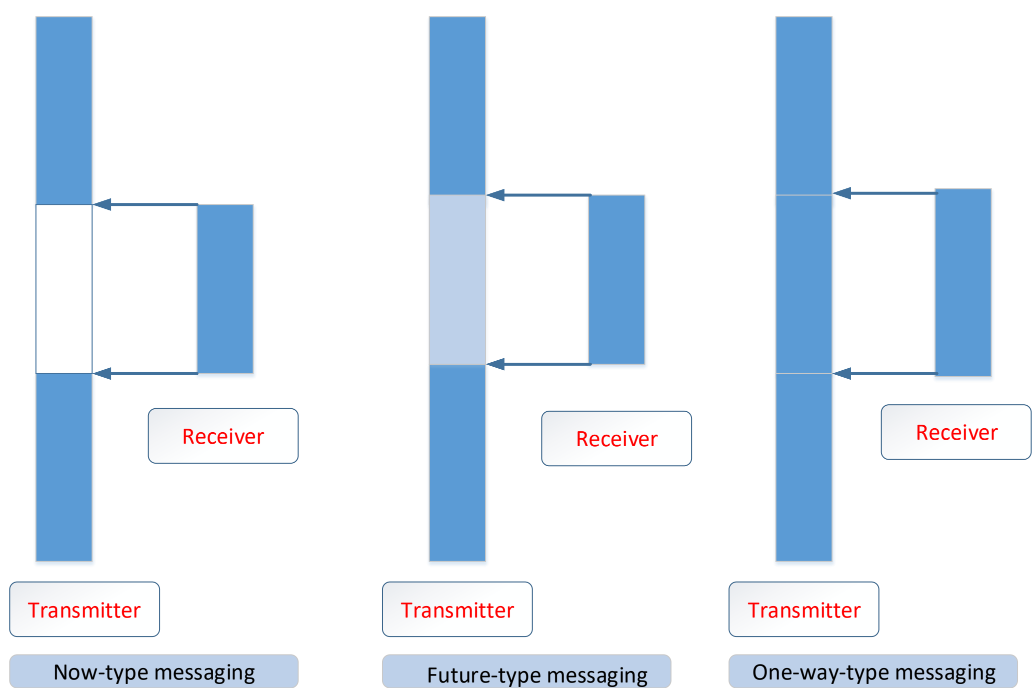


Figure 1.7: Communication between mobile agents

1.6 FIELD OF APPLICATION OF MOBILE AGENTS

Although mobile agents have been introduced recently, several application areas involving mobile agents as new technology have emerged, among which we cite:

• **Electronic Commerce:** The principles of autonomy and delegation are widely used in electronic commerce. Agents are, therefore, naturally suited to this type of application. The agent acts on behalf of a person and applies different strategies to fulfil the expected services. In

addition, the agent's mobility allows him to move from site to site to dialogue and negotiate with the services present to achieve the best possible transaction.

• **Telecommunication services**: Mobility functionality is gaining momentum in telecommunications. Mobile agents can be used to cover the layers of communication protocols, from network maintenance to mobile applications following the user in his movements. A “Personal Communicator Agent” (PCA) is a mobile agent responsible for delivering a message to the recipient, regardless of their device (telephone, computer, laptop or cordless phone). A user's PCA must be able to receive messages and route them uninterrupted across heterogeneous networks to the user. For example, if the only way to deliver an urgent message to a user is via a wireless phone, the personal agent must convert the text message to a voice message. Mobility allows these agents to follow the user even when he changes the machine.

• **Robotics**: Agents can also help improve the performance of physical robots. Using sensors, the robots base their behavior on agents who translate the information obtained to define the actions to follow (turn, move forward, etc.). Example: Asterix is an example. It enables high-level screening using opinion-based reactionary behavior.

• **Distributed computing**: Mobile agents can be used in distributed computing systems to move computation to where data resides, reducing network traffic and improving performance.

• **Intrusion detection**: Mobile agents can be used in intrusion detection systems to scan for vulnerabilities, detect attacks, and respond to security breaches.

• **Healthcare**: Mobile agents can be used in healthcare applications to monitor patients, collect data, and assist in the diagnosis and treatment of medical conditions.

• **Entertainment**: Mobile agents can be used in entertainment applications such as online gaming to manage game play, match players, and provide feedback to users.

• **Environmental monitoring**: Mobile agents can be used in environmental monitoring systems to collect data on weather, air quality, and other environmental factors.

1.7 PRESENTATION OF SOME MOBILE AGENT PLATFORMS

This part tries to give an idea of the existing platforms [13] allowing the development and manipulation of mobile agents. Unfortunately, this task is quite tricky for many reasons:

- This technology is booming, although the notion of the agent is relatively old
- The rapid evolution of the field does not always provide up-to-date information.

- There are several models to describe the behavior of agents. Each is the subject of research and documents that do not allow for a global and homogeneous view.
- The technological mastery of agents is a critical economic and strategic issue, which explains the difficulty of finding an acceptable standardization.

There are currently many different implementations that are incompatible with each other. Some are written in pure Java, thus benefiting from cross-platform porting. We quote:

- **Odyssey** [14] (from General Magic), One of the first agent platforms. It is independent of the transport protocol used and can therefore be adapted to RMI (from Sun), CORBA (from OMG) or DCOM (from Microsoft).
- **Aglets** [15] from IBM, use the notion of remote reference (AgletProxy), message passing (multicast, broadcast) and the ATP protocol developed by IBM. It also offers a certain notion of security by limiting the resources allocated to the aglets.
- **Dima** [16] from the LIP6 laboratory is a platform for multi-agent systems allowing the development of different types of agents (reactive, cognitive and hybrid).

This platform does not support mobility. This must be programmed

- **Jade** [19] from the CSELT of the University of Parma, is a software that simplifies set up multi-agent systems through a personalized interface that responds to the specifications of the Foundation for Intelligent Physical Agents -FIPA- and thanks to a set of tools that support bug fixing. Jade also offers security services.
- **Grasshopper** [66]: The Grasshopper platform developed by IKV++ (Innovation Knowhow Vision++) in 1997 is an implementation and execution environment for mobile agents. This product developed in Java has been tested on SUN Solaris 2.5 and 2.6 and on Windows (NT and 9x). For its operation, Grasshopper requires at least Java version 1.1.4Visibroker 3.1. Grasshopper has the distinction of being the first platform compatible with the standard protocol MASIF (Mobile Agent System Interoperability Facility) of the OMG (Object Management Group) and the standards of FIPA (Foundation for Intelligent Physical Agents).

1.8 BENEFITS OF THE MOBILE AGENT PARADIGM

Despite their recent appearance, mobile agents have quickly established themselves, particularly in research related to distributed applications. Therefore, this paradigm was immediately evaluated to verify what it could bring advantages compared to traditional

programming methods. In the rest of this section, we present some aspects of the evaluation of mobile agents:

- **Performance:** The first improvements made by mobile agents relate to the gain in performance due to better use of the physical resources employed.
- **Network traffic reduction**: Moving mobile agents can significantly reduce remote communications between network clients and servers. This reduction itself brings the following advantages:
    - The decrease in bandwidth consumption. Indeed, studies show that implementing mobile agents makes it possible to obtain a significant reduction in the network load in terms of the total number of data transferred by comparing with the remote sending of requests (procedures and methods). This decrease is observed in different applications requiring intense information exchanges between the client and the server. Examples include collecting information from distributed databases and crawling the Internet.
    - Reducing latency times. In the context of large-scale networks, the implementation of distributed applications, requiring frequent interactions between client and server, comes up against the latency times specific to network communications. The waiting time for a response to a request is often longer than the processing time required to perform the service. By bringing the client and server closer together in the same subnet, even on the same site, they are placed in an environment where the response time of interactions will be limited, reducing latency times.
    - By reducing remote communications as much as possible to transfers of mobile agents, the periods of connection between two sites are considerably reduced. This reduction in the window of use of network communications makes it possible to minimize the breakdowns of physical links, which can frequently occur in wireless environments
- **Computational independence**: In the remote evaluation model, the client and server must remain connected as long as the service runs. In addition, some services requiring long processing phases do not always support connection breaks with customers. On the other hand, maintaining communications links in large-scale or wireless networks is very difficult. Mobile agents allow customers to delegate service interactions without maintaining an end-to-end connection.

- **Physical fault tolerance**: Mobile agents can easily adapt to system errors by moving with their code and data. These errors can be purely physical, such as the disappearance of a node, or functional, such as the stopping of a service. If a service fails in a site, for example, the user agent of this service can then choose to move to another site containing the desired functionality. This allows for better fault tolerance.
- **The design**: From the design point of view, the mobile agents represent a method that allows characterising certain applications better. Still, they also bring an increased complexity compared to the traditional client/server, which is much better mastered. To begin with, designers generally prefer a way to describe real behavior easily. With the classic method, it is very restrictive to describe (network) exploration algorithms or even to characterize the movements of mobile users. With mobile agents, designers have a way to describe this kind of behavior naturally. Thus, one can easily set up application deployment and/or maintenance on a network or even follow users in their movements.

In addition, agents have a specific processing capacity to adapt to their environment. Generally, in the client/server model, the service is characterized beforehand by a strict interface and/or a well-defined usage protocol. Thus, if the customer is not aware of the description of the service, he will not be able to use it. On the other hand, the agents adapt to the characteristics of the services to express their requests. For example, if the service requests secure communication, the agent can retrieve a security module, update its communication stack and dialogue with the service.

On the other hand, the reasoning capacity of the agents will make it possible to design autonomous agents, adapting their movements according to the environment and modulating their functionalities during execution. However, these properties come with a much more challenging design than the classic client/server. Indeed, in the majority of distributed applications, we try as much as possible to hide the distribution by relying on a set of system services that allow us to design an application as if all its elements were local.

1.9 LIMITS AND DISADVANTAGES OF THE MOBILE AGENT PARADIGM

To know the limits and disadvantages of any field is very important to get familiar with and get a profound idea about it, a thorough understanding of these problems is the first step to solving them and thus developing them and obtaining robust technology with fewer drawbacks.

Mobile agent technology is no exception. However, while mobile agents have undeniable advantages, they also have shortcomings that must be addressed to make mobile agent-based

applications work reliably. Again, we can cite that development and insufficient standardization are this area's major limitations.

Development and insufficient standardization: The field of mobile agents is still relatively young; it comes up against solid constraints during the development phases. The first of these is that there are currently too many middlewares for mobile agents, each of which has its own shortcomings and qualities [17]. With this plethoric offer, it is difficult to speak of standardization. We can also mention some of the sub-reasons that may lead to the weakness of mobile agent technology [18].

- **Mobile agents do not perform well:** In general, mobile agents give worse performance than other mechanisms, such as remote evaluation. In addition, they are also costly
- **Mobile agents are challenging to test and debug**: distribution and mobility add complexity to the process of testing and debugging software. In addition, mobile agents can move from one node to another in an unpredefined order, and understanding and testing their implementation from such complex runtime history is very hard.
- **Mobile agents are difficult to authenticate and control**: they must be authenticated when they enter an environment. The problem is that they are associated with many identities, and it is unclear which ones should be authenticated and how access control mechanisms should take this information into account.
- **Mobile agents can be indoctrinated**: they are vulnerable to attacks from malicious execution environments when they travel across different hosts to perform their tasks. For example, a malicious host can modify an agent's code or memory image to change how the agent behaves, and as a result, it is possible to create a malicious agent.
- **Mobile agents can't keep secrets**: Mobile agents have been advocated as a means to implement critical applications. However, to perform sensitive transactions (for example, signing a contract), conducting operations that require secrecy, such as a private key is often necessary. Unfortunately, a secret cannot be effectively hidden if it is employed by an agent on a remote host and the agent cannot interact with the originating host.
- **Mobile agents are eerily similar to worms**: The mechanism of mobile agents bears some striking resemblance to how malicious worms spread through networks. In addition, worms are difficult to remove. A mobile agent infrastructure would support the execution of both benign and malicious agents, and, as a result, it would be prone to be leveraged to launch worm-like attacks.

In summary, we can say that until now, there is no standard language shared by all the development platforms for mobile agent applications. This represents a serious handicap because agents need to express the characteristics of the services they are looking for and obtain precise answers on their location and how to use them. Moreover, with all the existing languages, developers are, once again, faced with too broad a range of possibilities. To find a solution to this dilemma, it will be necessary to turn to the multi-agent systems field, which seeks to set up precise languages based on a set of ontologies and protocols characterizing inter-agent interactions.

1.10 CONCLUSION

We have presented in this chapter a general introduction to the paradigm of mobile agents. In the beginning, we introduced the concept of agents, in general, to make a brief presentation of the paradigm of mobile agents.

We then moved on to some examples of implementation platforms for mobile agent applications and, finally, a look at the criticisms of the paradigm.

To conclude, Mobile Agent Applications are proven to look good because of the following motivations: The first motivation is usually the minimization of remote communications (it is generally cheaper to migrate the processing code than the data to be processed, which can be much bulkier).

Another motivation is the case of nomadic computing (the mobile agent can continue its processing on servers while disconnecting from the client machine).

Nevertheless, the development of mobile agent applications comes up against problems inherent in security and interoperability in the various heterogeneous host platforms. Moreover, the insufficiencies of the efforts for the standardization of these platforms pushed the researchers to explore other horizons for better support and implementation of the applications based on mobile agents.

The research on mobile agents is relatively recent; the research in this field is numerous and has recently appeared to identify important concepts.

The following chapter will introduce the approaches and methods for modeling mobile agent applications, particularly the notion of mobile UML.

Chapter 2

UML & Mobile UML

Chapter 2: UML & Mobile UML

2.1 INTRODUCTION

The advent of computer network technology has prompted researchers to develop new communication paradigms whose processes can move over the network. Software development based on these paradigms requires appropriate extensions of traditional methods and concepts.

UML [20] **(Unified Modeling Language***)* provides extension mechanisms. These make it possible to adapt UML specifications to the requirements of a specific domain. Thus, the mobile UML extension has been proposed to satisfy mobility modeling needs.

In this chapter, we will recall the concept of modeling and its different forms; then, we will see in detail the UML language, its statechart diagram and its extended version, which covers mobility, making it possible to model the mobile code paradigm.

2.2 MODELING

2.2.1 What is a model?

Modeling is the process of producing a model [21]. According to Marvin L. Minsky [22], for an observer B, an object A* is a model of object A insofar as B can use A* to answer the questions that interest him concerning A.

A model is an abstraction of a system built for a specific purpose. The model is then said to represent the system. [23] A model is an abstraction that contains a restricted set of information about a system. It is built for a specific purpose, and the information it has is chosen to be relevant to the use that will be made of the model.

2.2.2 Why model?

Modeling a system before it is built provides a better understanding of how it works. It is also an excellent way to control complexity and ensure consistency [24]. A model is a common, precise language that all the members know of the team, and it is, therefore, a privileged vector for communication. This communication is essential to arrive at a standard and precise understanding of a given problem for the various stakeholders (particularly between project management and IT project management).

In software engineering, the model makes it possible to distribute tasks better and automate some of them. It is also a factor in reducing costs and lead times. For example, modeling platforms now know how to use models to generate code (at least at the skeleton level) or even go back and forth between the code and the model without losing information. Finally, the model is essential to ensure a good level of quality and efficient maintenance. Indeed, once put into production, the application will have to be maintained, probably by another team and, what's more, not necessarily from the same company as the one that created the application.

The choice of model, therefore, has a significant influence on the solutions obtained. Non-trivial systems are best modeled by a set of independent models. Depending on the models used, the modeling approach is not the same

2.2.3 Different Forms of Modeling

Modeling can be classified as formal, semi-formal or informal. **Table 2.1**[25] presents a definition of the categories of languages as well as examples of languages or methods that use them

Table 2.1:Classification and Use of Languages or Methods [25]

Language categories			
Informal language		Semi-formal language	Formal language
Simple	Standardized		
The language that does not have a complete set of rules to restrict a construct	Language with a structure, a format and rules for the composition of a construction	The language that has a defined syntax to specify the conditions on which constructs are allowed	A language that has rigorously defined syntax and semantics. There is a theoretical model that can be used to validate a construction
Examples of languages or methods			
Natural language	Structured text in natural languages	Entity Relationship Diagram,Object diagram	Petri net , finite state machine,Maude ,VDM,Z

2.2.3.1 Informal Modeling

The process of informal modeling according to an informal language can have its use justified for several reasons [25]:

- By its ease of understanding, it allows consensus between people who specify and those who order software;
- It represents a familiar way of communication between people. On the other hand, the use of informal language makes the modeling imprecise and sometimes ambiguous. Moreover, as human reasoning is the main factor for the analysis and verification of the specification, it can lead to errors of understanding, interpretation and verification. It is possible to use Standardized Informal Modeling to limit these problems; that is, modeling which uses a natural language, while introducing rules of use of this language in the construction of the modeling. Such modeling retains the advantages of informal modeling by making it less imprecise and less ambiguous.

2.2.3.2 Semi-Formal Modeling

The semi-formal modeling process is based on a textual or graphical language for which a precise syntax is defined as well as semantics; this semantics is relatively weak but allows a certain amount of control and the automation of some tasks [25].

Most semi-formal proposals rely heavily on a graphical language. This can be justified by the expressiveness that a well-developed graphic model can have; textual language is normally used as support for graphical models. Furthermore, semi-formal modeling using a graphical language can produce models that are easy to understand and can be very punctual; this largely justifies its use. However, the lack of complete semantics is a substantial handicap for this kind of modeling; the existing problem for the informal languages, of lack of precision compared to the comprehension of the modeling as well as the problem of ambiguity persists for the semi-formal languages. The use of constraints in graphical models has been introduced to try to solve this problem of lack of precision and ambiguity. An example of using textual constraints on a graphical model relates to the work of JJ Odell [26].

This work applies structural constraints to modeling based on object-oriented models. Another proposal is that of S. Cook and J. Daniels [27]. In this work, on a graphical model based on the OMT method [28] and State Diagrams, the formal language Z [29]is used to represent constraints.

2.2.3.3 Formal Modeling

A formal method is a rigorous development process based on formal notations with precise semantics and formal verifications. The main advantage of formal specifications is their ability to express precise meaning, thereby allowing checks on the consistency and completeness of a system. We can briefly summarize [25] the analyzes of formal methods carried out by A. Hall [30] and by JP Bowen and MC Hinchey [31], which more particularly concern the myths inherent in formal languages.

A. Hall [30] points out that formal methods help to find errors and reduce their incidence through a complete specification that can be applied to any system, software or hardware; however, they do not replace existing methods and must be used in conjunction with them. If solid mathematical knowledge is necessary to carry out proofs, it is not imperative to specify. Using formal methods can lower costs and does not cause project delays.

JP Bowen and MC Hinchey [31] show that with an appropriate translation, formal methods can help understand a system by a user, including in the case of large systems where they are also applicable.

In addition to the already existing proof tools, the studies for the creation of other tools which cover the whole specification show the community's interest in these methods. The use of formal methods, more than desirable, is beginning to be imposed by companies in certain cases, even if they are not used in all types of modeling

2.2.4 Object-oriented modeling

The fundamental concept in object-oriented modeling and design is the object, which combines both data structure and behavior; this object is organized around real-world concepts.

Object-oriented models are used to understand problems, communicate with application domain experts, model a company's business, repair documentation, and design programs and databases. [136]

2.2.4.1 what is object-oriented

Object-Oriented (OO) means that software is organized as a collection of independent objects that incorporate data structure and behavior [136]. data structure and behavior are loosely associated in this approach, like other programming approaches.

In general, the characteristics required by an object-oriented approach are identity, classification, inheritance and polymorphism**.**

2.2.4.2 Why the object approach

The stability of the modeling compared to real-world entities, the iterative construction facilitated by the weak coupling between components and the possibility of reusing elements from one development to another [138], as well as the simplicity of the model, which calls upon five founding concepts (objects, messages, classes, generalization and polymorphism) are advantages of the object approach. The object-oriented approach is widely adopted simply because it has demonstrated its effectiveness when building systems in a variety of business domains, and it encompasses all dimensions and all degrees of complexity.

2.3 UNIFIED MODELING LANGUAGE (UML)

The Unified Modeling Language (UML) is a visual modeling language used to specify, visualize, build and document the artifacts of a software system [32]. UML concerns any system built using the object-oriented (OO) approach.

It is used to understand, design, navigate, configure, maintain and control information about modeled systems [33].

UML is a digest of three important notations *(**Booch**, OMT* 5, *OOSE*6). It has had an undeniable impact on the way we view systems development [34].

The first version of UML, published by the OMG 7 was *(UML 1.0)* in 1997; several versions appeared until the latest version, UML 2.5, was released in 2015.

2.3.1 UML and Meta-Modeling

UML is based on the four-level meta-modeling architecture. Each successive level is marked from *M3* to *M0* and is usually called meta-meta-model. Meta-model**,** class diagram and object diagram, respectively [20].

A level diagram, *Mi,* is an instance of a level diagram *Mi +1.* The *M3* level diagram is used to define the structure of a meta-model. The Meta Object Facility (MOF*)* belongs to this level. The UML meta-model belongs to level *M2* [20]. UML is formally defined using a meta-model expressed in UML [33]. **Figure 2.1** illustrates this definition.

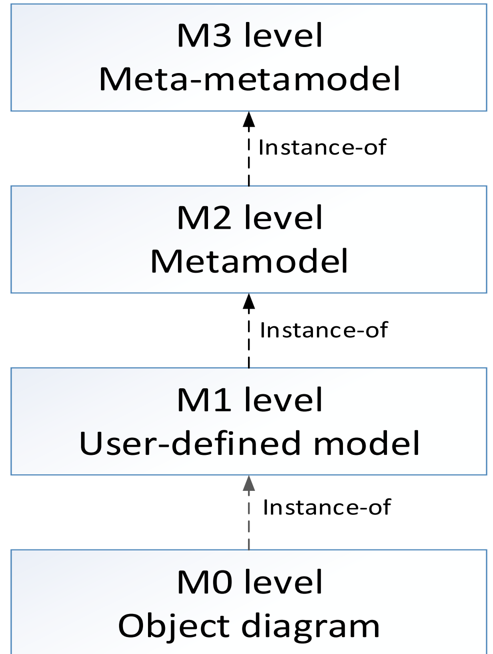


Figure 2.1: Four-level meta-modeling architecture

2.3.2 UML diagrams

The UML notation is described in the form of a set of diagrams. Modeling in UML specifies that it is defined in 13 diagrams grouped into two main classes, which are [20]:

1. Static diagrams
2. Behavioral diagrams

2.3.2.1 Static diagrams

Also called structural diagrams, they used to illustrate the static characteristics of a model.Static diagrams have the same structural characteristics [34] and are summarized in **Table 2.2.**

Table 2.2: Shorts description of structure UML diagrams.

Diagram name	**Description**
Class diagram	It represents the static description of the system by integrating into each class the part dedicated to data and processing. It is the pivot diagram of the whole modeling of a system
Object diagram	the representation of instances of classes is the objective of the object diagram
Component Diagram	it represents the different software components of a system
Deployments Diagram	it describes the technical architecture of a system
Package diagram	shows the packages and possibly the relationships between them
Diagram of composite structures	shows the internal structure of a classifier and/or the use of Collaboration in a collaboration occurrence

2.3.2.2 Dynamic diagrams

Also called behavioral diagrams, they describe how resources modeled in structural diagrams interact. These dynamic diagrams include [34] with a simple description of each in **Table 2.3** below:

Table 2.3: Shorts description of dynamic UML diagrams

Diagram name	**Description**
Use case diagram	It is intended to represent the needs of users in relation to the system
Activity Diagram	It gives a version of the sequences of activities specific to an operation or a use case
State-transition diagram	It shows the different states of objects in reaction to events.
Sequence diagram	It allows to describe the scenarios of each use case by emphasizing the chronology of the operations in interaction with the objects
Communication diagram	Graph whose nodes are objects and the arcs (numbered according to the chronology) the exchanges between objects
Overall interaction diagram	Shows aspects of an interaction.
Time diagram (timing)	Is an interaction diagram that describes both the behavior of individual classifiers and classifier interactions, focusing on the times of occurrence of events that cause state changes.

In our project, we are much more interested in the statechart diagram; for this the next section will be devoted to the statechart diagram,We will try to understand it in depth and familiarize ourselves with all its elements to facilitate the work in the last section

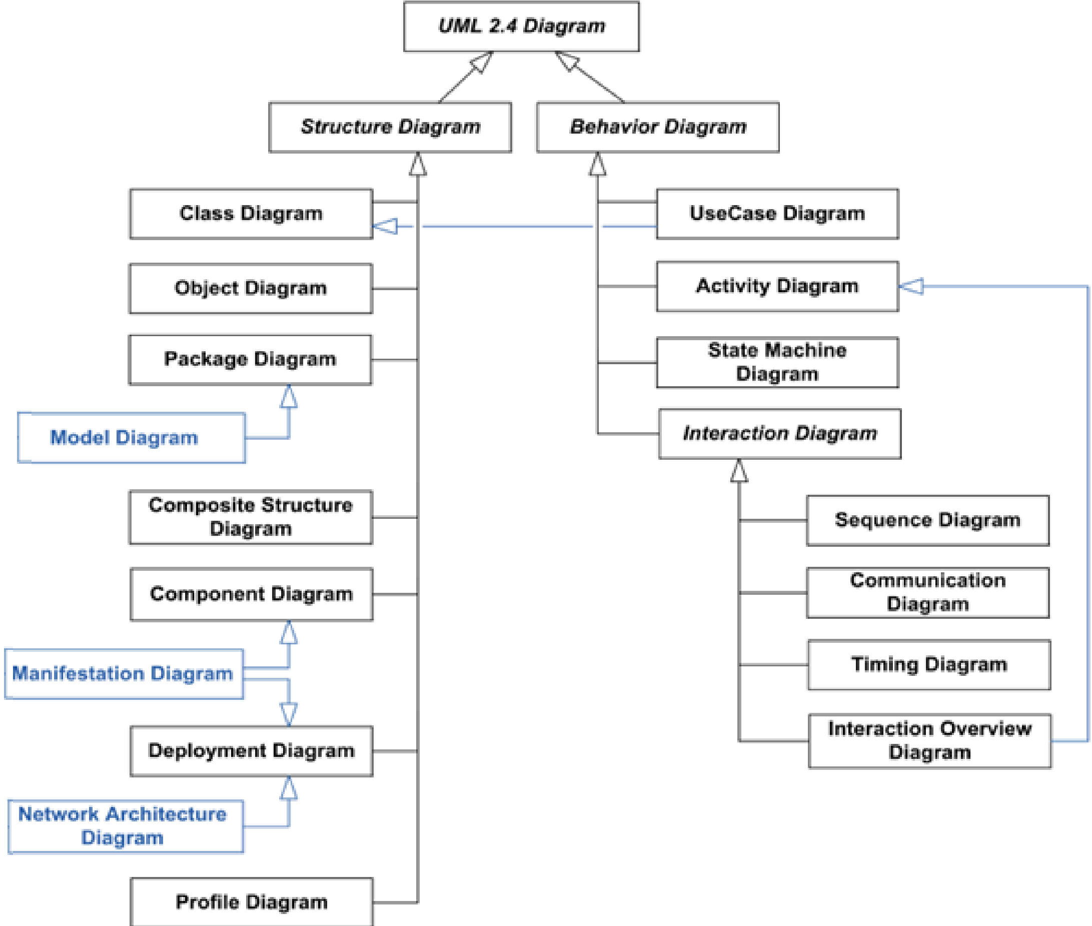


Figure 2.2: Classification of UML types [35]

2.3.3 Statechart diagrams

As we have already seen, the unified modeling language has become an essential standard in the software industry for several years. Nevertheless, several criticisms are addressed to this language; among the most important, UML is not a formal language. This is why several efforts have been founded to formalize this language.

The statechart diagram was the diagram most targeted by formalization work due to its specificity and its wide use in modeling the internal behavior of objects. The approach proposed by ourselves is also based on this diagram in its extended version covering mobility, so we saw that it is necessary to explain this diagram. We mention its most essential points and introduce the syntax and semantics of this diagram.

2.3.3.1 Definition

The UML statechart diagram describes the internal behavior of an object using a finite state machine. It specifies the possible sequences of states and actions that an object can process

during its lifetime in reactions to events (signals, invocation of methods). The state-transition diagram is the only diagram in UML that offers a complete and unambiguous view of all the components of the element to which it is attached [24].

2.3.3.2 States

The state of an object is a condition or situation during an object's lifetime in which it satisfies specific requirements, performs an activity, or awaits the occurrence of an event [37]. An object remains in a state for a limited time; for example, a heater could be in one of the following four states: inactive (waiting for a command to start), starting (gas is flowing through the heater, but the last one is waiting for the reached a certain temperature), active (heating in action) and closed state (once the heating is closed).

A state with several parts, namely:

- **Name:** a textual string which distinguishes the state from other states; a state can be anonymous.
- **Entry/exit** actions: these are the actions to be executed when entering and leaving the state.
- **Internal** transitions: the transitions to be made without causing a change of state of the object.
- **Substates:** the nested structure of the state. The state can be simple or composite.
- **Events postponed (deferred):** a list of events which are not processed in this state but which are saved in a queue so that they can be processed by the object in other states.

Figure 2.3 represents some simple states in a statechart diagram. A state is graphically represented by a rectangle with rounded corners.

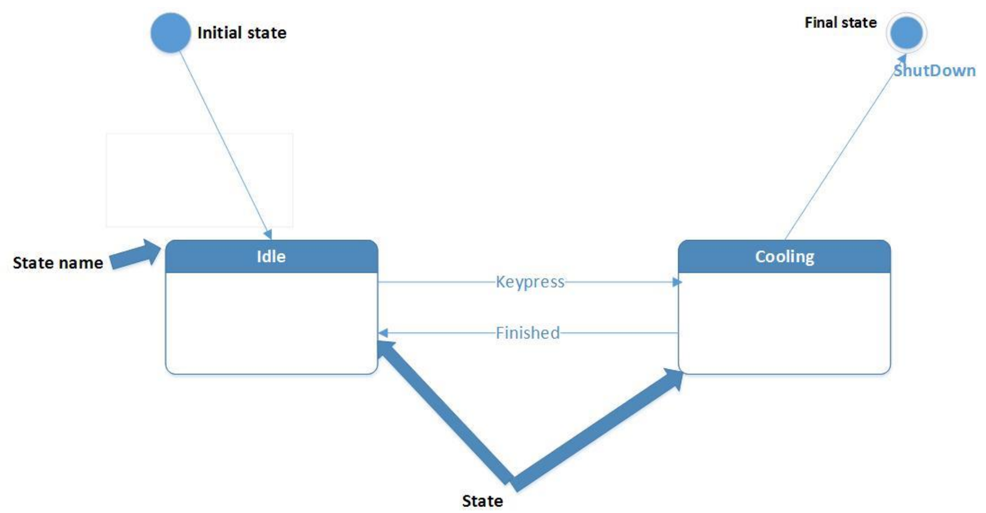


Figure 2.3: Two simple states in a state-transition [37]

As the **Figure 2.3** shows, two special states are defined in a state-transition diagram; first, an initial state that indicates the default starting place of the diagram or a composite state. Second, a final state that indicates that the flowchart execution or composite state is complete.

2.3.3.3 Events

An event is something that occurs during the execution of a system and is worth modeling [24]. State-transition diagrams make it possible to specify the reactions of a part of the system to discrete events. An event occurs at a specific time. When an event is received, a transition can be triggered, and a state change takes place to switch the object from one state to a new state.

There are several types of events, namely [24]:

- **signal event:** A signal is a type of event intended explicitly to convey a one-way asynchronous communication between two objects. The sender object does not wait for the receiver to process the signal to continue its flow. The same object can be both sender and receiver.
- **Call event:** A call event represents the receipt of a method invocation by an object. The operation parameters are those of the call event. Call events are usually declared at the class diagram level.
- **Change event:** A change event is generated by satisfying a Boolean expression on attribute values. This is a declarative way of waiting for a condition to be satisfied. A change event is continuously evaluated until it becomes valid when the transition fires.

- **Temporal event:** Temporal events are generated by the passage of time. They are specified either absolutely (precise date) or relatively. By default, time begins to elapse upon entering the current state. Therefore, the syntax for a relative specified time event is: **After (<duration>).** An absolutely specified time event is defined using a change event: **when (date= <date>).**

2.3.3.4 Transitions

A transition is a relationship between two states. The transition indicates that the object which is in the first state will perform certain actions and enters the second state once a specific event occurs and a specific condition is verified [37]. A transition consists of five parts, namely;

- **Source state:** This is the object's state before crossing the transition.
- **Event trigger:** this is the event recognized by the source state and with which the transition can be crossed once the guard of the transition is verified. An event can be a signal, a method call, a passage of time or a change of state [37].
- **Guard condition:** a transition can have a guard condition (specified by '['<guard> ']' in the syntax). This is a logical expression of the attributes of the object [24]. The transition only fires once the guard condition evaluates to true.
- **Effect:** once a transition is triggered (we also speak of firing (crossing) of a transition), its effect (specified by '/' <activity> in the syntax) runs. These are usually; primitive operations such as assignments, sending a signal to the object itself or to another object, calling methods, or a list of activities, etc. [24].
- **Target state:** is the object's active state (the new state) after crossing the transition. **Figure 2.4** shows a simple state-transition diagram; the figure focuses on the presentation of the transitions and its parts (guard, effects, events, etc.). The transition in the diagram is an arrow pointing from the source state to the target state.

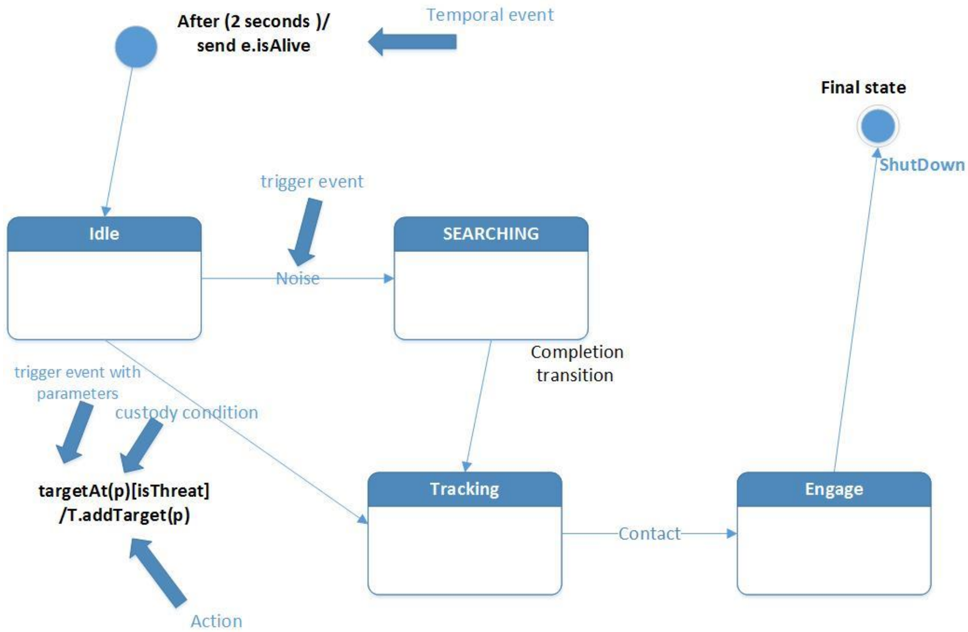


Figure 2.4: Transition in Statechart diagram [37]

A transition can have several sources (in this case, it represents a junction between several concurrent states). It can also have several target states.

2.4 EXTENSION OF UML

UML can be extended or adapted to a specific method, organization, or specific user. There are three extension mechanisms in *UML* which are [38]:

1. Stereotypes
2. Tagged values
3. Constraints

UML extension mechanisms have been very successful, leading to a proliferation of new model elements and extensions. Profiles provide a mechanism to manage these extensions; in the next chapter, we will try to make a comparative study of UML extensions which covers the aspect of mobility to reach the optimal choice of available extensions and complete our work on it.

2.4.1 Stereotypes

The stereotypes extension mechanism defines a new type of model element based on an existing model element. Thus, a stereotype is like the existing element, with additional semantics and properties that are not present in the existing element. Graphically, a stereotype

is rendered by a name surrounded by quotation marks (‹‹›› or by <<>> if quotation marks are not available) and placed above the name of another element. A stereotype can also be indicated by a specific icon [38]. **Figure 2.5** shows examples of stereotypes.

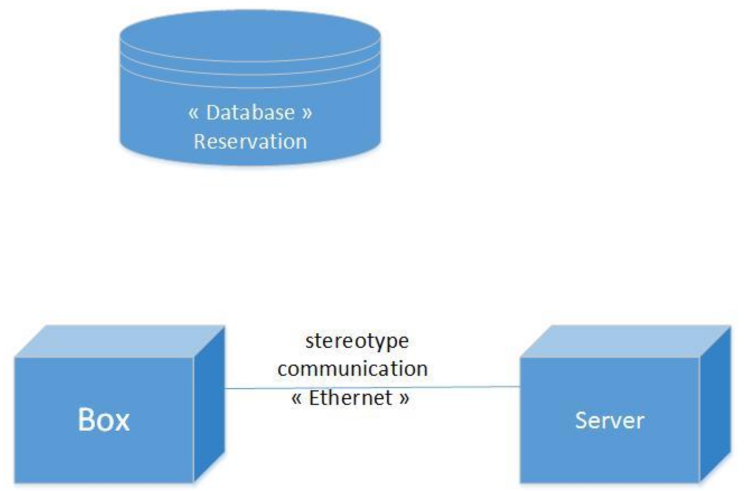


Figure 2.5: Example of stereotype [33]

2.4.2 Tagged Values

The purpose of a tagged value is to assign (add) **a** property to a model element, in addition to those properties already defined in the meta-model [34]. Tagged values are expressed as *"name=value",* eg author ="Amine", project_phase = 5, or last_update ="1-09-22" [34].

2.4.3 Constraints

Constraints define the invariants, making it possible to preserve the system's integrity. A constraint defines a condition that must be true for the duration of the context in which it is defined. It is represented by a text enclosed in braces "{}" [34]. **Figure 2.6** shows examples of constraints.

Figure 2.6: Example of constraints

2.5 MOBILE UML

The shortcomings of UML in modeling mobile agents have motivated the authors to propose extensions to adapt UML to be able to model this type of system. These works are very rare and limited in general. Among these works, we cite the extension of Hubert Baumeister [39], Cornel Klein [40], Kassem Saleh [41] and Haralambos Mouratidis [42].

In the following, we will see the Mobile statechart diagram taken from [41], which we will use in our project.

2.5.1 Mobile statechart diagram

We have already seen in detail in this chapter the statechart diagram, so in this section which presents the mobile statechart diagram, we try to show just the new concepts brought with this new concept of mobility.

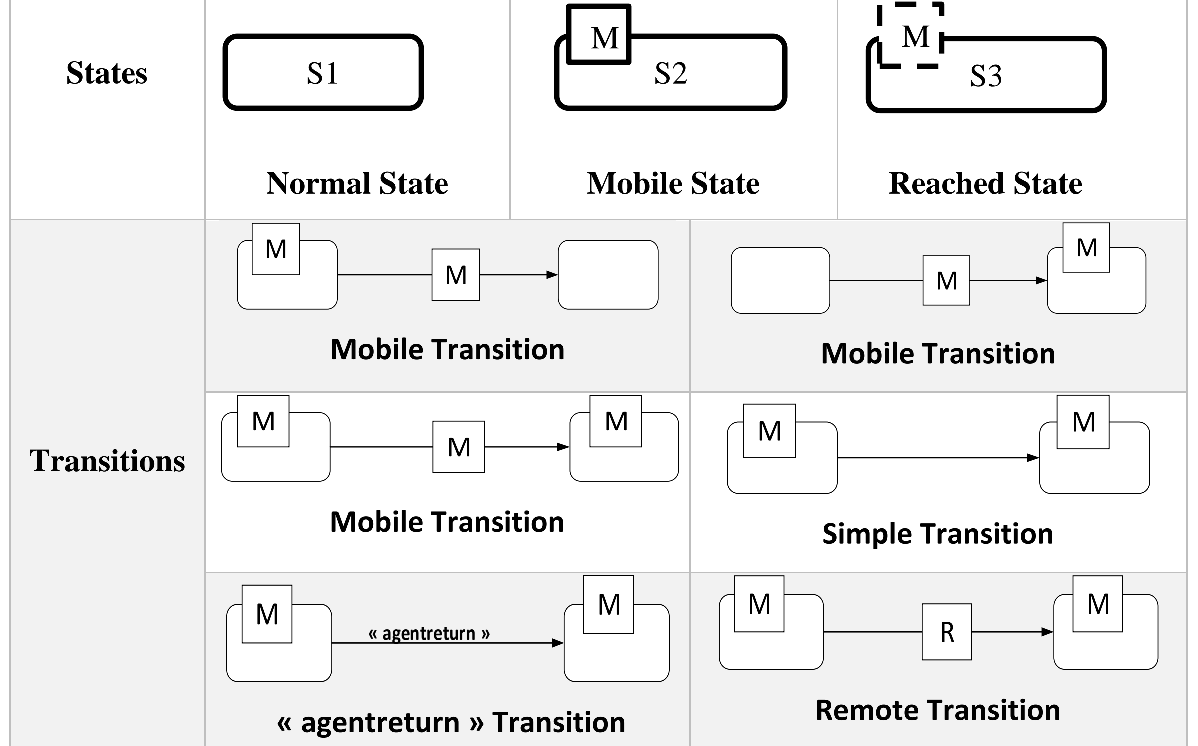
Kassem Saleh [41] propose a mobile statechart diagram (MSD, *Mobile Statechart Diagram)* as an extension of UML state-transition diagrams. The latter was introduced to model the behavior of mobile agents by exposing their different states. They can be used elegantly to represent the behavior of stationary and mobile agents. We present below (cf. TAB. 2.4) the structural elements of MSD, and we focus on the new elements introduced to deal with mobility in these diagrams.

A place is a logical node on which the agents of an application can be executed (one or more states), designated implicitly. The initial pseudo-state represents the agent's execution start point. At any given time, the agent may be in a distinct state at a particular place. A normal state represents the state of the agent in its basic place. A mobile state is represented with a box (M); it is a state accessible by the agent outside of its base place. A transition is a one-way link that goes from one state (the source state) to another state (the target state); it can carry a label according to the syntax *Event [Condition] / Action,* which means if an event occurs, action will be executed if the condition evaluates to true. A transition is simple if the two states at their two ends are located in the same place. A mobile transition is shown with a box (M); it is a transition between two states accessible by the agent and located in two different places; it has several forms as illustrated in (cf. **Table 2.4**)

The current state reached by an agent, either in its base place or outside it, is represented by a dotted box (M). A remote transition carries a box (R); it occurs if a mobile agent reaches a state while interacting with another remote agent. A transition with an << agentreturn >>

stereotype represents the return of the mobile agent to its original place by reaching a state on it. If the agent completes its execution, it will reach the pseudo-final state.

Table 2.4: The structural elements of MSD



2.6 CONCLUSION

In this chapter, we have tried to present UML and Mobile UML, starting by presenting the modeling in a general framework. Then, we introduce the UML modeling language and focus on his statechart diagram, and finally, a brief presentation of the extensions of UML for mobility.

As a result, UML has many facets. It is a standard, an object modeling language, communication support, and a methodological framework. UML is all of these. However, verifying such models is quite a tricky task despite the convenience of modeling and understanding UML models due to the semi-formal semantics. Moreover, mobile agents are primarily characterized by their power to move between nodes in a network to complete tasks assigned to them.

The specificities of mobile agents have made using traditional modeling methods and means, such as UML, unsatisfactory. For this reason, we have presented a mobile UML diagram extension that allows us to model them.

In the next chapter, we will see a comparative study of different approaches proposed to specify software systems based on mobile agents by focusing on those based on UML more precisely on the coverage factor of the diagrams.

Chapter 3

Comparing Modeling Approaches for Mobile Agent-Based Systems

Chapter 3: Comparing Modeling Approaches for Mobile Agent-Based Systems

3.1 INTRODUCTION

In the first chapter, we saw the importance and evolution of the mobile agent concept and its system. In the second chapter, we learned that obtaining a solid and coherent system begins with modeling and correct and efficient design. This is why the modeling and design of software systems based on mobile agents has been a topic of research in computer science and artificial intelligence for several decades..

Building a modeling language from scratch is a complicated task; researchers have attempted to shorten the path by importing the essentials of success from the object-oriented paradigm and introducing the features appropriate to support the mobile agent paradigm.

In this chapter, we have tried to make an effective extension of the work proposed by [44], which has made a comprehensive presentation, offers a classification and a distinguished critics of the available works of modeling software systems based on mobile agents, which have been developed to deal with the analysis and design phases of these systems.

We found that the diagram coverage factor is absent from the previous study [44], which focused on UML-based methods. That is why in this section we have tried to add a coverage factor of the diagrams to get a broader comparison between these languages. An in-depth discussion will then be conducted better to understand the pros and cons of different approaches.

3.2 RELATED WORKS AND BACKGROUND

Conducting a comprehensive survey of current methodologies for modeling software systems using mobile agents is a critical task. This survey will allow us to understand the available approaches and evaluate them, providing a comprehensive perspective that can help us choose the optimal approach to benefit from this research study. Additionally, it can serve as a valuable reference for future researchers to benefit from.

The first interesting work in the literature that examines techniques and tools for modeling mobile agent-based software systems is that of [45], which provides an overview of existing agent-oriented methodologies. We can see in this work that the researchers did not propose new methodologies, but rather extended existing methodologies to include aspects appropriate to

agents. Furthermore, it discusses the approaches taken to assist in all phases of an agent-based application's life cycle.

Belloni and Marcos [46] examined several UML extensions for modeling mobile agents to define a single extension that gathers and integrates the characteristics of the others UML extension.

Bauer and Muller [47] evaluated existing techniques and methodologies for agent-based systems, however modeling mobile agents was not the primary emphasis of this work; instead; it was mentioned secondarily.

In their article [48], Melomey et al. evaluate the strategies and approaches that have already been implemented to establish the mobile agent paradigm. To overcome the shortcomings of the suggested approaches in terms of agent mobility modeling, they offered several essential notions to model mobility; nonetheless, the authors combined wrongly modeling languages and methodology.

Melomey et al. [49] extend their last work in [48], to provide a comparative study of some agent-based systems modeling approaches. Cervenka and Trencansky [50] provided an overview of agent-oriented modeling techniques; however, they focused on multi-agent systems. Hachicha et al. [51] discussed several approaches to modeling mobile agent-based applications using UML.

But the most comprehensive work that exists to date is the work of [44], who tried to follow a logical path, started by first defining the evaluation criteria, then selecting the most critical available versions and by projecting onto them the criteria used to achieve a comprehensive and high-quality assessment.

Other approaches found in the literature could have been mentioned, but we noted a methodological problem in these contributions: the mixture of the concept of modeling languages and methodologies. Thus, in our study, as in [48], we made an effort to narrow our focus to modeling languages because, given that mobile agents are introduced in various domains, and the approaches cannot be general but instead adapted to particular circumstances, we believe that they are the key to a paradigm's success. As a result, it is outside the purview of this study to examine the methodology. Additionally, only mobile computation applications based on mobile agents with static locations are considered. Mobile computing applications that use dynamic locations fall outside the purview of our work.

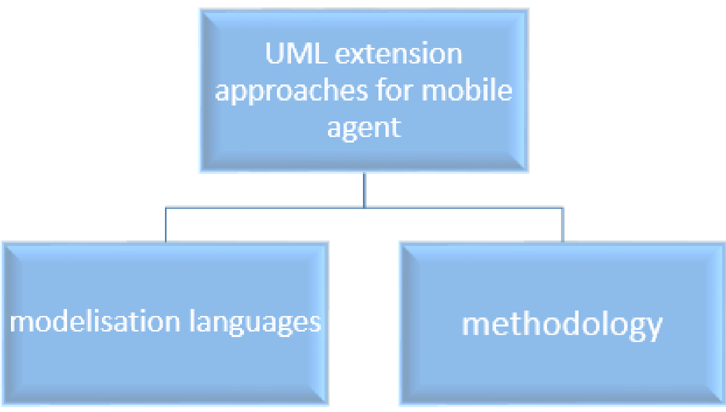


Figure 3.1: classification of UML approaches mobile agent extension

3.2.1 Modeling challenges and gaps

Mobility is an important issue that needs to be considered when modeling and examining mobility since it causes a complex dynamic reconfiguration of the system under consideration while it is being used.

Consequently, providing an approach with the capability to design it is not simple. Indeed, the approach must respond to questions raised by the use of agents [42], such as why a mobile agent moves from one host to another, when it moves, where it moves, and how it reaches the target host.

We know that UML [20] allows us to model every step of creating an object-oriented system; as a result, we can consider it crucial and decisive to the success of object-oriented systems, from the feasibility study to the deployment stage. Therefore, researchers have attempted to define and develop a modeling language that can play the same role in the mobile agent paradigm as UML has already played in the object-oriented paradigm by exploiting appropriate methods from the preceding paradigm for modeling mobile agents.

The development life cycle of mobile agent-based software systems consists of roughly the same stages as the simple software system development life cycle in three basic phases: analysis, design, and implementation. Note that the implementation phase is characterized by the abundance and presence of many different platforms that support and implement the mobile agent model, which explains the desire of developers to provide direct and rapid solutions to exploit by researchers and manufacturers.

As for the design and analysis phases, the approaches proposed for software systems based on mobile agents are individual tests and proposals, which are not widespread. Hence, not exhaustive, have a limited scope and are incomplete in themselves.

Figure 3.2 illustrates the current state of the development life cycle of mobile agent-based software systems

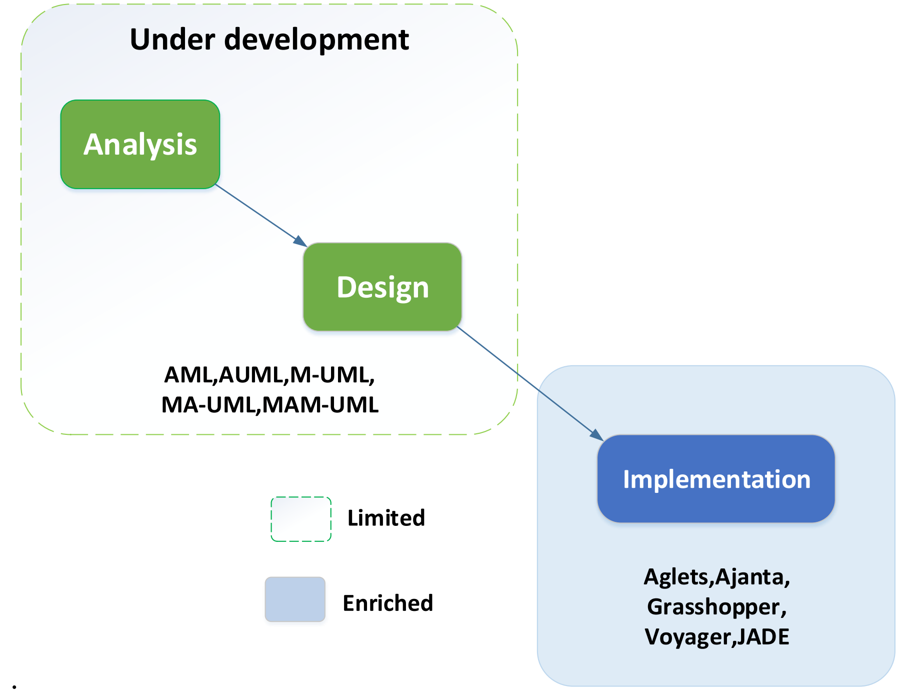


Figure 3.2: Development life cycle of mobile agent-based software systems [44]

3.2.2 Security modeling problem of mobile agent systems

Receiving a mobile agent by a host process is an action that carries maximum risk if security is not taken into account, as shown in **Figure 3.3** Therefore, many research works related to mobile agent security were realized. This research has proposed several mechanisms that follow different ways of approaching security problems and are divided into two axes [53]. The first concerns the protection of the host against malicious mobile agents, while the second focuses on the protection of the mobile agent against the maliciousness of the host on which it is running.

Mobile agent systems have several features to make them secure for network programming. In addition, designers may specify additional measures to improve application security. Various modeling techniques provide a security perspective that describes the many defences employed to keep applications based on mobile agents from being vulnerable.

These processes consist of, for instance: Authorization and access control to resources, agents and mobile agent platform authentication. However, given that mobile agents are used across a wide range of disciplines with vastly varied settings, it is genuinely unclear how one can create standard notions for security modeling (each domain has its peculiarities).

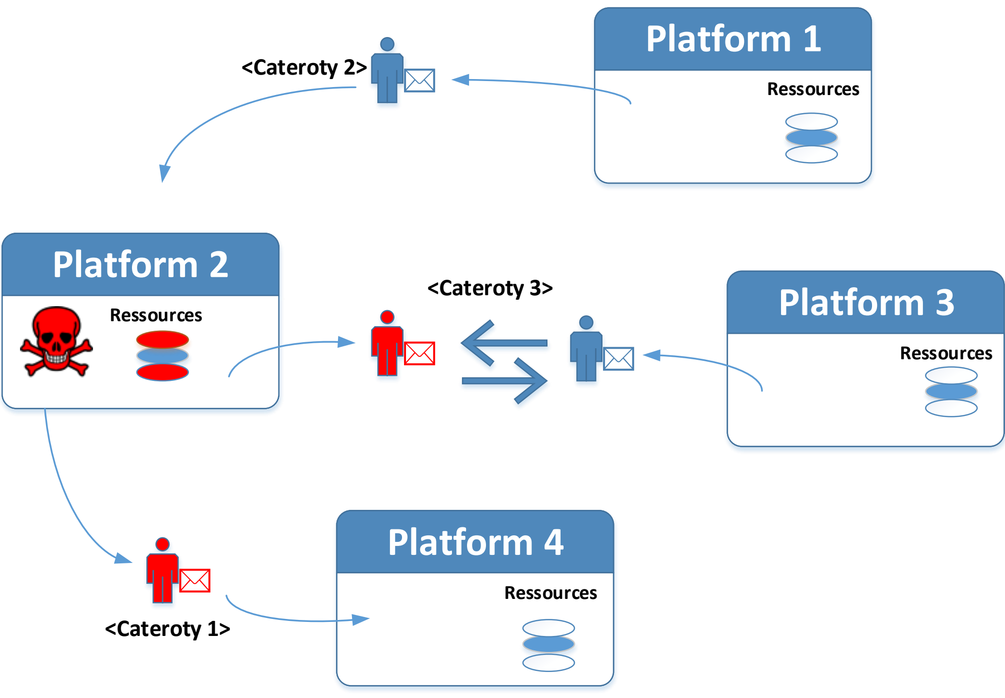


Figure 3.3: Categories of Security Issues in Mobile Agent Systems [36]

According to [54], there are three sorts of vulnerabilities that may be taken into account when considering an agent's mobility, and they are as follows:

- Agent against Platform: As a result of having unrestricted access to the runtime environment, a mobile agent could compromise its confidentiality, integrity, and availability by intercepting or altering data, fully utilizing its resources, or endlessly replicating or migrating.
- Platform against Agent: when an agent migrates to a new location, such as a new platform, it must reveal its code, status, and data in plain sight, making it vulnerable to threats to its confidentiality and integrity from the hosting platform, which uses its data to control its behaviors and outcomes.
- Agent against Agent: An agent travelling throughout the network may experience a number of attacks from other agents they come across. In this sense, there are two distinct types of attacks: passive attacks, which merely observe and analyze an agent's behaviors and results without changing the agent's code or data, and active attacks, which have the potential to change an agent's code or data by altering variable values or introducing a virus.

3.3 CLASSIFICATION OF SOFTWARE SYSTEM MODELING APPROACHES BASED ON MOBILE AGENTS

Several modeling approaches have been discussed in this section, and only a few others are cited. The aim is to cover several points of view on dealing with these systems. Thus, we have classified the existing proposals for modeling software systems based on mobile agents into three main categories (see **Figure 3.2**):

We found that two major works try to make a classification of software system modeling approaches based on mobile agents;

The first work is by [55], who defined three categories of approaches to model software applications agents; the pattern approach, the formal approach, and the semi-formal approach. We note that this work is concentrated only on semi-formal approaches and particularly on formalisms which proposed extensions to UML notations or\and AUML, which eventually reached the work classification of all of MAM-UML [46], Klein et al. [57], Gervais and Muscutariu [58], and Mouratidis et al. [42] to end up proposing his MA-UML approach [55] classified with this type.

The second considerable work of [44] is an extension of the previous work in which he tried to reformulate the concept, and the name of the classes, then extend and detailed the classification of existing approaches in the other class; we can cite the essential extensions that he made as follows:

- Modified the concept of the semi-formal approach, which was divided before into an agent-oriented methodology extension class, and the second class is the class of UML language extensions; it was reduced towards approaches based on UML because this class has the advantage of being more comprehensive. After all, it is aimed at a large community.
- he replaced the class of pattern approaches with hybrid approaches in which he proposed to combine the hybrid integration of his two previously proposed classes (approaches based on UML, formal approaches), and we can summarize the differences between his two works in the **Table 3.1** below:

Table 3.1:progress review of classification

Loukil classification [55]	**Belghiat classification** [44]
Semi-formal approaches [OOext/UML]	UML-based approaches
The formal approach	The formal approach
The pattern approach	Hybrid approaches

And we can explain the extended classification of [44] as follows:

UML-based approaches: these methods tackled the issue of modeling mobile agent-based systems using UML [20], the standard language of the object-oriented paradigm.

Formal approaches: these approaches use formal methods for modeling applications based on mobile agents.

Hybrid approaches: These approaches are the outcome of a hybrid combination of the traits of the two previous categories as well as other inspired strategies from different perspectives.

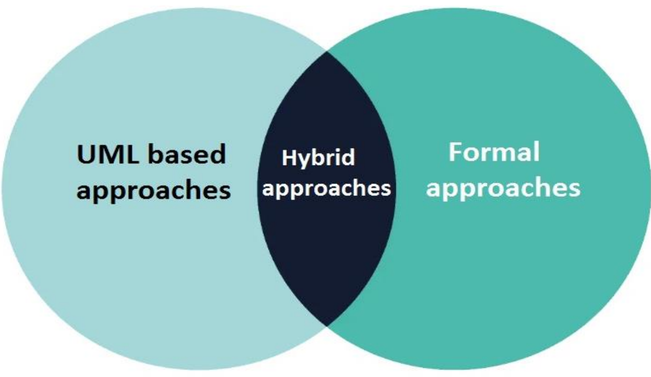


Figure 3.4: Classification of existing modeling approaches according to [44]

3.3.1 UML-based approaches

By expanding the object-oriented model and extrapolating its de facto standard to meet mobile agents' characteristics, academics have attempted to model mobile agents in this category. In fact, UML [20] offers ways for it to grow. As a result, researchers chose UML and created the necessary objects to support the new paradigm based on its expressiveness and extensibility. Due to the fact that these proposals take into account various system viewpoints, we have classified them as complete languages.

AUML (UML Agent): According to Bauer et al. [60], Agent UML (AUML) is the most well-known UML extension for agents. The ability to describe both an agent's internal and

external behavior is provided by AUML. The agent class diagram and interaction protocol diagram, which respectively expand the UML class diagram and sequence diagram, are the two main components of AUML. It was initially designed to exclusively deal with multi-agent systems, but later on, its deployment and activity diagrams were expanded [42] [61] to allow the modeling of the agent's mobility. The AUML deployment diagram extension's new annotations make it easier to describe agent migration between various nodes and the locations that are affected by the movement. The AUML activity diagram can also be used to express the mobility moment question, or the time when a mobile agent must move. The language has been developed with a variety of concepts and notations [62], but the key elements of applications based on mobile agents are not taken into consideration. Mobility, for example, is described statically; in addition, security is entirely omitted.

AML (Agent Modeling Language): AML is a UML extension that combines various multi-agent system principles and functionalities, according to Cervenka et al. [50] [63]. The complete agent-oriented modeling language at the moment is AML. It includes modeling entities' static, dynamic, and behavioral aspects inside a multi-agent system. AML also tries to model support for mobility by identifying and recognizing agent environment, host, and behavior features (migration and clone). However, when a multi-agent system is used, the language only considers mobility at the implementation level. Because of this, in our opinion, modeling actual mobile agent apps is not appropriate.

MAM-UML (Mobile-Agent Modeling with UML): To try to define a single UML extension that collects and combines the properties of a significant portion of mobile-based techniques, Belloni and Marcos [46] [56] suggest MAM-UML. UML extension. A UML profile called MAM-UML represents the authors' stated set of views, including organizational, life cycle, interaction, and mobility views. The itinerary of a mobile agent, the various types of movement, and the work plan are just a few of the intriguing features of a mobile agent-based application that are described in detail in this extension.

M-UML (Mobile UML): To deal with the modeling of software-based systems, Saleh and El-Morr [41] suggest a comprehensive modification of the UML 1.4 standard M-UML, in which all the diagrams have been expanded. Of mobile agents. The authors demonstrate and describe the modifications made in each UML diagram to depict mobility characteristics through an exemplary case. The authors leave out the security measures and their modeling. The authors only emphasize modeling of mobility. The strategy is well explained, but it needs to be tried to see how good it is.

MA-UML (Mobile Agent UML) : Seven diagrams were proposed by Hachicha and colleagues [51] based on the AUML sequence diagram, the UML state/transition diagram, and the activity diagram extension. The writers went into detail about a few issues the other extensions did not address. Additionally, to facilitate the use of the MA-UML profile and the automatic production of Java code from its diagrams, they created the CASE tool known as MAMT (Mobile Agent Modeling Tool) [65]. The modeling and execution of mobile agent applications are substantially facilitated by this. We were surprised to know that MA-UML is the most recent language to be established in the literature (which deals with different views of applications based on mobile agents). This demonstrates that despite this field's value, the scientific community has partially given up on it.

In their study, Klein et al. [57] offer certain UML extensions for mobile agents. New ideas have been proposed to model strong mobility, weak mobility, and cloning action. Additionally, they have added new stereotypes to model the elements of surroundings for mobile agents (mobile-agent, mobile-agent systems, places, and regions). They have also proposed extensions to the sequence diagram in order to model agents' movement from one location to another. The limitation of this approach is that it is restricted to a particular mobile agent platform (Grasshopper platform [66]) and a specific class of mobile agent applications. In addition, they do not provide a mobility description for all views and aspects of systems.

An Architecture Description Language (ADL) specific to the architecture of mobile agent systems has been defined by Gervais and Muscutariu [58]. This ADL has proposed some conceptual and operational interfaces required for interoperability among diverse mobile-agent systems. It has also suggested modifications to component and deployment diagrams and offered fresh archetypes to represent mobile agents and mobile agent systems. The stereotyped flow relationship "becomes," an operation "move," and operations "beforeMove" and "afterMove," which get the agent ready for the migration, all assist modeling agent mobility.

To address various issues raised by the use of mobile agents, Mouratidis et al. [42] added additions to the UML deployment and activity diagrams. The AUML formalism now includes these extensions. First, developers can record the cause (what motivates) and position (where) of an agent's movement using the AUML deployment diagram. Developers can record the time when a mobile agent leaves one node to migrate to another using the AUML activity diagram.

There are also some intriguing proposals outside of these languages. These are, however, only partial solutions because they don't cover all angles of a mobile agent-based software system.

The activity diagram of UML 2.0 is suggested by Miao Kang et al. [67] as a suitable model for mobile agent applications. Furthermore, it is demonstrated that using this model with stereotypes provides satisfactory answers to the questions on the particularities of mobile agent applications. Subsequently, others have extended his work [68] by proposing a UML sequence diagram extension, which aims to model the migration path of a mobile agent and the three fundamental components of mobility (the process of creating a mobile agent, current location of a mobile agent and finally the path that follows the mobile agent).

Bahri et al. [69] suggest an expansion of various UML 2.0 diagrams in their work in order to explain mobile agent-based software systems and cope with novel notions offered by these systems, such as migration, cloning, and locations.

Even though these approaches have tried to help model software systems with mobile agents, almost all of them have been taken from the field of multi-agent systems, where it is more important to show how agents relate to each other. Moreover, from the perspective of distributed systems, the description of mobility (and its consequences like security) is not sufficiently described to offer effective and reliable UML-based modeling approaches for agent-based mobile apps, as is the case with object-oriented applications modeled with UML.

3.3.2 Formal approaches

Formal methods are a specific category of mathematics-based approaches used in many fields, including software engineering, to specify , develop, and verify software systems [70]. Due to the strength of correct mathematical analysis, it is extremely appealing to utilize formal approaches to specify and verify the functioning of mobile agent-based software systems in the early phases of the development life cycle. This should improve the design's reliability and resilience.

The two effective formalisms, process algebras and Petri nets, have spawned many formalisms that have been utilized for mobile agents. Some of them are recalled in this section.

With regard to the application of process algebras,

The first work we can cite in this context is Milner's [22] proposal of the π-calculus language for modeling mobile agents. This language provides the opportunity for the movement of links within a virtual space of linked processes to describe mobility, and the majority of the proposed methods are based on this formalism.

Using the π-calculus formalism also, Jezic and Lovrek [72] presented a method for formally specifying and verifying the movement and communication of mobile agents in a network

The distributed Join-calculus, which is as expressive as the asynchronous π-calculus, is used by Fournet et al. [73] to encode their computation for mobile agents. This computation has a precise definition of migration, failure, and failure detection. The asynchronous π-calculus supports explicit localities and primitives for mobility, which makes it possible to express the movement of mobile agents between nodes clearly.

The π-ADL is a formal, theoretically sound language based on higher-order typed π-calculus and can be used to define static, dynamic, and mobile architectures. It was proposed by Oquendo et al. [74] and designed in the European ArchWare1 project to address the specification of dynamic and mobile architectures. As for the description of mobile agents, formalities have been proposed in other works.

A π-calculus extension was presented by Schmitt and Stefani [75] to express the mobility of agents in complex scenarios.

A communication architecture between mobile agents is formalized using a π-calculus extension in the work of Sewell et al. [76].

According to the development of their localities, the features of agents in systems based on mobile agents are dynamically expressed by Bettini et al [70].

The "Mobile Ambients," a computation to characterize the motion of both processes and gadgets, was proposed by Cardelli and Gordon [77].

More formal methods for capturing mobility are introduced in [78] and [79].

In the context of Petri nets, several proposals have also been developed. They benefit t from this formalism which is a powerful formal model in terms of tools, mathematical description, and graphical depiction.

Xu and Deng [80] attempted to model mobile agents using high-level Petri nets.

Predicate/transition (PrT) networks are used by Xu et al. [81] to suggest a technique for the formal modeling of logical agent mobility. They describe a mobile agent system as a collection of agent spaces, and the agents can move between these spaces.

Xu and Schatz [82] tried to design mobile agents using high-level Petri nets G-net.

3.3.3 Hybrid approaches

In recent years, the combination of formal approaches and visual modeling tools for mobile agent-based software systems has attracted a lot of attention.

The objective is to try to provide powerful modeling languages allowing the specification and verification of these systems while taking into account the shortcomings of formal models in terms of graphic notation and the weaknesses of visual models in terms of formal semantics because of their semi-formal nature. This has encouraged many researchers to try to mix the two in order to benefit from the advantages of each.

In his thesis, Bahri [84] suggests a hybrid approach that is based on a UML extension and the nested Petri nets formalism to represent software systems based on mobile agents. The method takes advantage of UML's benefits while overcoming its semantics deficiencies by utilizing nested Petri nets to enable the formal analysis of these models.

Knapp et al. [85] proposed a semi-formal/formal modeling approach for moving objects using a UML state/transition diagram extension and the MTLA formalism [32], aiming to study the concepts of refinement.

Latella et al. [86] proposed a UML state/transition diagram extension accompanied by formal semantics for specifying the behavior of moving objects.

Belghiat et al. [87] proposed an interesting combined approach based on an extension of UML and the π-calculus for the modeling and verifying mobile agent-based applications.

Kezai et al. [88] also proposed a combined approach, based on an extension of UML and the formal language Maude to extract a formal specification with integrity constraints to ensure verification of these rules-based mobile agents.

Jian-Min [83] presented a method for modeling and evaluating a moving system utilizing an event-based formalism with a comprehensive four-step framework. Their primary goal was to establish a theoretical framework for modeling and analyzing intelligent transportation systems from the viewpoint of system behavior (to establish a theoretical framework for modeling and analyzing intelligent transportation systems from a system behavior perspective.)

Other approaches exist, some of which propose utilizing design patterns to construct mobile agent-based software systems, such as the ones described in [89], [90], and [91]. The research on mobile agents has uncovered a great deal of information regarding design patterns, including agent patterns, interaction patterns, and many more. While some are described using UML, others do not use this notation.

3.4 THE COMPARISON FRAMEWORK

3.4.1 Comparison criteria

As we have seen above, many studies have tried to compare and rank the research work related to adding the concept of mobility to UML to cover mobility. Still, all these studies only looked at features and characteristics from the conceptual level. Here for the first time, we have compared the proposed approaches by comparing the degree of coverage of these extensions of UML for its initial diagrams. As a first and quick note, we found that most proposals only concern some diagrams, and only M-UML, which suggests complete modeling, covers most UML modeling diagrams (see **Table 3.2**), we summarize our opinion and compare all the methods.

The second part of this comparative study relates to the point of view model for a greater understanding of the different points of view available and, thus, the ease and effectiveness of choosing the modeling language in proportion to the need to use it according to the system.

We proposed to treat a factor which has not been treated in the previous works. We cannot say that this coverage factor is the most important of the factors discussed in the literature. Still, we can say that it is a complementary factor that helps us figure out how important and successful each proposed mobile modeling approach is for agent-based applications.

3.4.2 Comparison results

Table 3.2 shows the UML extension coverage for its initial diagrams

Table 3.2 : Coverage of UML Extensions Diagrams

	UCD	STD	SQD	COD	ACD	CLD	OBD	CPD	DPD	PD	DGI
M-UML	X	X	X	X	X	X	X	X	X		
AUML		X	X		X	X				X	X
MAM-UML		X		X	X	X					
Mouratidis					X						
M.kusek			X								
MA-UML		X			X	X					
Kang					X						
Klein			X								
Muscutariu									X	X	

Notes:

AML[50] does not appear in the previous table because it does not offer extensions of diagrams directly based on mobile agents but based on the direct proposal of a new architecture, dedicated multi-agents despite that can be considered as a quasi-optimal approach by its power and its completeness like M-UML, the problem is that it is not designed to support mobility [50] [63].

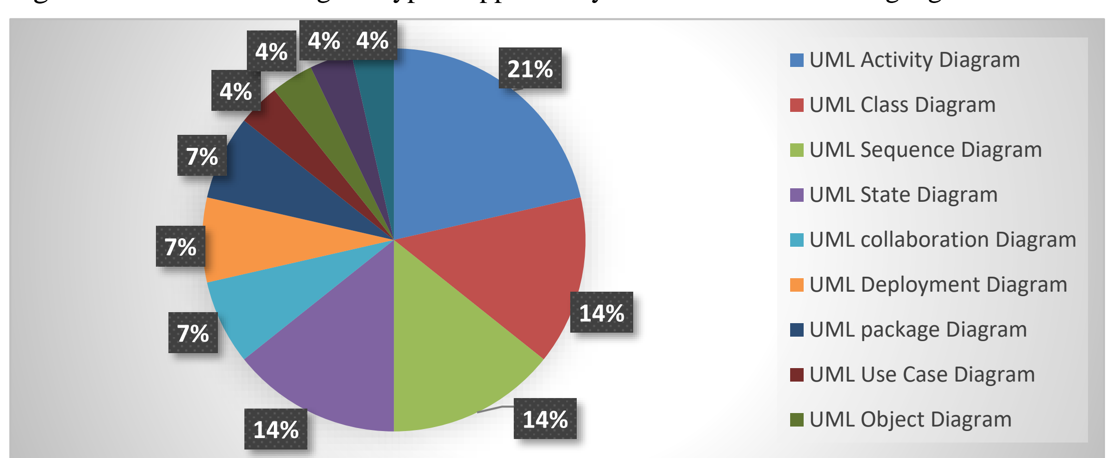
In MA-UML [51], it was said before that this approach offers seven diagrams. However, we will mention in the tables just three diagrams because the Navigation Diagram is an extension of the statechart diagram, as well as the mobile agent lifecycle diagram is an extension of the statechart diagram also by adding rental concepts and the other diagrams are either new concept diagrams or compound diagrams which do not really exist in the UML language.

Table 3.3: percentage of Coverage of UML extensions for initial diagrams

Mechanism diagram abbreviation	Mechanism diagram name	Pourcentage
ACD	Activity Diagram	21,42%
CLD	Class Diagram	14,28%
SQD	Sequence Diagram	14 ,28%
STD	Statechart Diagram	14 ,28%
COD	Collaboration Diagram	7 ,14 %
DPD	Deployment Diagram	7 ,14 %
PD	Package Diagram	7,14 %
UCD	Use case Diagram	3,57%
OBD	Object Diagram	3,57%
CPD	Component Diagram	3,57%
GID	Global interaction Diagram	3,57%

Modeling languages extend UML in two ways, i.e., via the UML profile diagram as the AML language, which is not shown in the table or extending the UML diagrams of interest as represented by the languages offered in the previous table. Indeed, extend particular UML diagrams to adapt them to the needs of their domain.**Figure 3.5.** It gives a percentage of mobility-supporting languages based on UML that extend different types of UML diagrams. Thus, of the proposed languages that we believe are the most important and influential in the literature, the UML activity diagram is the most selected extension mechanism used by 21% of languages. In addition, 14% of languages extend UML's class diagram, and 14% of languages extend UML's sequence and state diagrams.

Figure 3.5: The UML diagram types supported by the UML-Extended languages mobile



3.4.2.1 Modeling point of view

The concept of software viewpoints was proposed to promote modular and manageable software systems specifications.

The software viewpoints each address the specific concerns of a software system to be modeled and support its separate specification without addressing other issues. Kruchten & al. [95] proposed four key software perspectives: logic, process, physics, and development. Later, the IEEE [96] community standardized the concept of software perspectives. Rozanski & al. [97] recently proposed an exhaustive set of software perspectives: logical, behavioral, concurrent, physical, deployment, development, and operational. Thus, in this most recent study, many viewpoint proposals by Rozanski & al. were considered, and UML-based languages for each viewpoint were analyzed.

They proposed a logical viewpoint concerning the decomposition of software systems into logical components and their relationships; the behavioral viewpoint concerns the evolution of the state of system units according to certain events. (e.g., method call), The concurrency view is concerned with simultaneous interactions of system units; the physical and

deployment views are concerned with the physical structure of systems and their mapping to logical structures, respectively; and finally, the operational point of view concerns questions of organization, administration, and support.

Table 3.4 gives the extended modeling languages and their support for different software viewpoints. So apparently, M-UML is the extended language of mobile agent-based UML that supports the largest number of different viewpoints (i.e. five viewpoints), which further strengthens our choice in this study; then comes AML [50], and Muscutariu [58] also support three different viewpoints. And finally, the rest of the proposed languages with one or two maximum viewpoints. In **Figure 3.6**, the logical and behavioral viewpoints are the most popular, used by 29-43% of languages. On the other hand, the rest of the viewpoints are rarely used by languages.

Table 3.4: Viewpoint Support for the UML- languages based mobile agent

UML-based Languages	Multiple-Viewpoints Support
M-UML [41]	Logical and behavior, deployment, physical, concurrency
AUML [60]	Logical and behavior
AML [50]	Logical and behavior, operational
MAM-UML [46]	Logical and behavior,
Mouratidis [42]	behavior
Mario Kusek[68]	behavior
MA-UML [51]	Logical and behavior,
Kang [67]	behavior
Klein[57]	behavior
Muscutariu [58]	Logical, deployment, physical

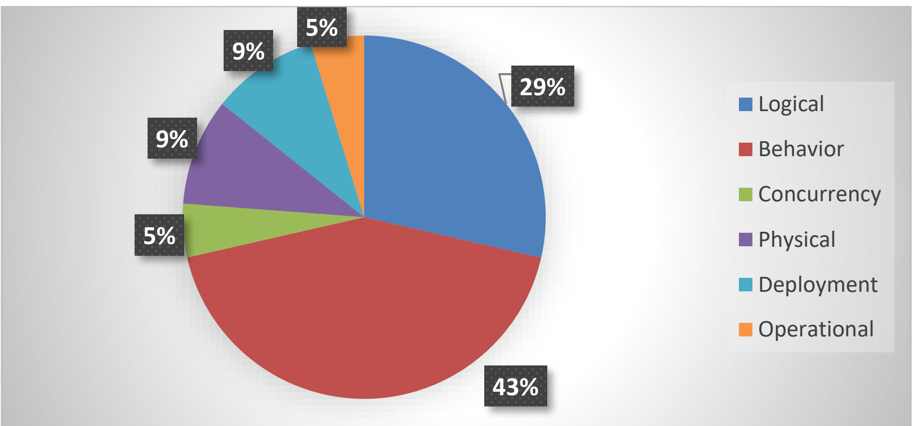


Figure 3.6: The viewpoint types supported by the UML-Extended languages based mobile based

Table 3.5: Percentage of Viewpoint Support for the UML- languages based mobile agent

Viewpoints Support	Pourcentage
Logical	28,57%
Behavior	42,85%
Concurrency	4,76%
Physical	9,52%
Deployment	9,52%
Operational	4,76%

3.4.3 Comparison discussion

The table above (**Table 3.2**) summarizes the coverage of UML extensions for the initial UML diagrams on both sides, structural and behavioral

From (**Table 3.2**) and the results obtained in [44] we can enumerate the results of the comparison obtained as follows:

- Regarding the coverage of diagrams in the extensible versions of UML for mobility, we note that the coverage of M-UML was the best and most complete of all existing UML extensions [41], where the coverage included the two classes with five diagrams for the behavioral aspect and four diagrams for the structural aspect, Then, AUML is in second place because it proposed a good number of diagrams, with a focus on both structure and behavior, with three diagrams for behavior and two for structure. On the

other hand, MAM-UML is strong because it covers many diagrams, even though it came in third place on our list.

- Although it is a simplified representation for state and activity diagrams on the behavioral side and class diagrams on the structural side, MA-UML is considered one of the powerful extensions in certain criteria in the following lines. We also note that most of the UML extensions presented have chosen the CLD class diagram to represent it in the structural part, this explains why class diagrams are also among the most used and most important types of UML diagrams [98] and are vital importance in software development. Furthermore, class diagrams are the best way to illustrate the structure of a system in detail, showing its attributes, operations as well as interrelationships, which also prompted us to choose the extension of this M-UML diagram class diagram as the second diagram in order to continue our research, to specify it and to verify it.

From all the above, we can summarize the following points:

- M-UML is arguably the most comprehensive language in terms of diagram coverage and the simplest and most suitable for mobile agent-based software systems that broadly support the modeling of the mobility paradigm, we believe that an extension of M-UML which takes into account the previous issues, could present a complete and sufficient modeling tool, this reinforces our good choice in this search for the language mentioned M-UML
- The MA-UML language comes in the second position, which is also interesting for these systems, but the major obstacle that will hinder its use is the lack of a mechanism for modeling the architecture of the execution environment. Also, note that there is some redundancy in the representation of information. More precisely, MA-UML provides two diagrams “Navigation diagram” and “Itinerary Diagram” to describe the agent's travel plan with almost the same description that we find redundant.
- We can say that the MAM-UML is very well organized from its different points of view. But its 'disadvantage' is that it has not been tested enough and does not provide illustrative examples because it was suggested as a preliminary research idea, which makes it very difficult for users to use.
- AUML Although it was one of the first and most popular UML extensions for mobile agents because of its effectiveness in generating test cases that exercise agent interactions and scenarios, it has suffered a terrible decline and underutilization because Agents are represented as objects in AUML, which limits the ability to express and test proactive behavior [99] [100].

- AML was initially developed to model multi-agent systems, but it proved unsuitable for modeling software systems because it is not well suited for modeling software systems based on mobile agents. We can also add that AUML lacks modeling tools dedicated to it and lacks a clearly defined metamodel and precisely specified semantics to explain the different modeling elements, which can result in the poor interpretation of AUML models [50].

3.4.4 Approaches comparison summary

From our analysis and evaluation of the mobile agent paradigm and its modeling approaches, we can draw the following conclusions:

- We found that all the existing proposals for modeling mobile agent-based systems have been proposed individually by researchers, either by suggesting a direct extension to UML or by developing an earlier version and that there is still no version developed by well-known international organizations as has been done in UML provided by OMT as a powerful and comprehensive modeling language.
- The more research deals with and develops tools for modeling systems based on mobile agents, the wider our angles of vision as researchers in this field and the greater our ability to deal with and manage the difficult unsolved problems of this paradigm. Such as security, which is a significant concern for these systems.
- The mobile agent (MA) paradigm has garnered considerable interest. However, although MAS have generated considerable excitement in the research community, they have not led to a substantial number of practical applications. One of the primary reasons for this is the lack of quantitative research comparing the effectiveness of mobile agents to that of conventional approaches. This can support recommendations for initiating projects to evaluate the implementation of practical and industrial applications utilizing mobile agents in the real world.
- There is not yet a standardized and effective way to verify the integrity of the proposed methods because their authors, while demonstrating the applicability of the approach, relied on simple illustrative examples that do not allow us to discover the weaknesses of the approaches unless we apply them to the most complex examples. It is, therefore, necessary to work more to try to extract strong points from each language and standardize them by first creating a standard language, then by extending them to more complex examples until their final application in the real world, and then verifying its completeness or making the necessary corrections.

- The combination and integration of visual modeling tools and formal methods provide powerful languages to specify and verify these systems, taking into account the absence of semantics in visual models on the one hand (such as OCL for UML) and graphical notations for formal models on the other side to avoid any form of interference and repetition.
- To advance and organize this field, researchers must define a generic meta-model with its associated notations to build a single reference extension that integrates all the features of the proposed approaches.
- Most designers do not have a sufficiently deep and complete view of mobility modeling to support the full life cycle of the mobile agent model and to avoid its issues, such as security, control and agent management and dynamic resource discovery, and most of the mobility modeling suggestions were primitive and simple solutions lacking in maturity and completeness. Therefore, a design that integrates security into the basic agent infrastructure would be preferable)
- Distributed applications are based on many different paradigms. Mobile agents are the most recent paradigm, but based on what we know about their most important features, we can say that mobile agents are not the best paradigm for all of these types of applications. Instead, they are a new option with many benefits in some fields but less useful in others.

In this study, we have attempted to address modeling approaches of the most important mobile agent-based software systems and their impact because it is impossible to list them all.

Finally, it is necessary to challenge languages based on mobile agents to industrial and real applications to verify their completeness. This is a very difficult task, as finding suitable industrial applications in the real world is difficult. Nevertheless, the research community encourages companies to participate in its efforts to develop a robust and useful agent-based modeling language.

3.5 CONCLUSION

In this chapter, we have presented a comparative framework for mobile agent-based software system modeling approaches that enhances the understanding of previous work on this topic. First, we presented and identified the challenges and shortcomings that need to be overcome to model mobile agent-based applications. Afterwards, we proposed classifying the approaches into three broad categories according to certain factors: UML-based, formal, and hybrid approaches. As a result, we can say that the combination of visual approaches, called

informal (which attract the engineer's attention) and formal approaches (which are analyzable) represents a promising axis in the near future.

More importantly, we can also conclude that M-UML is one of the most powerful languages for its simplicity and completeness and its coverage of almost all diagrams in both structural and functional aspects and among the most suitable also for mobile agent-based software systems which broadly supports the modeling of new features mainly the mobility property of the paradigm this puts us on the right track to complete the work of specification and verification of M-UML diagrams

The analysis results of different mobile agent-based software system modeling languages are expected to be very useful for the community of software researchers and engineers who can now realize the importance of mobile agent-based languages for practitioners and their benefits. And the disadvantages from a practical point of view. Thus, further research may be conducted in the future with the aim of further improving mobile agent-based UML languages and tools based on them. Additionally, language developers can also use the results in their language design to better target gaps in the domain. Finally, although we cannot say with certainty that the comparative framework is definitely perfect, but the results of the analysis can be used by practitioners to choose the mobile agent-based language(s) that best meets their needs in terms of targeted requirements.

To benefit from the advantages of UML in the modeling of systems without omitting the task of verifying the models resulting from such modeling, several works have focused on the principle of model transformation to obtain models whose verification is affordable.

The next chapter will introduce the notion of model transformation and, more precisely, the transformation of graphs.

Chapter 4

Model Transformation

Chapter 4: Model Transformation

4.1 INTRODUCTION

In the broad sense, modeling effectively uses a simplified representation of an aspect of reality for a given purpose. Far from being reduced to the expression of a solution at a higher level of abstraction than the code [101], computer modeling can be seen as separating the different functional needs and extra-functional concerns (such as safety, reliability, efficiency, performance, flexibility, etc.).

The Model Driven Engineering (MDE) approach offers the mechanization of the process experienced engineers follow by hand [101]. Interest in MDE was strongly felt at the end of the 20th century when the standardization organization OMG (Object Modeling Group) made its MDA (Model Driven Architecture) public. This can be seen as a restriction of the MDE to manage the particular aspect of software dependence on an execution platform.

In the context of MDE, the notion of model transformation plays a fundamental role. The design process is reduced to a set of partially ordered model transformations. Each transformation takes models as input and produces others as output, until executable artifacts are obtained.

In this thesis, we are particularly interested in the concepts of model transformation. For the UML diagrams (source model), we try to verify them by a formal language Maude (target model). The transformation of the source model (M-UML) to the target model (Maude), will be carried out by a graph transformation using graph grammar which we will also propose.

We start with a presentation of the basic concepts for model transformation (in the general framework). After that, we will expose some types of these transformations and their classification. Finally, we will focus by refinement on graph transformation as a specific type of model transformation, considering that the statechart diagram the subject of our study here is fundamentally a graph [33,149], and that we can treat it and apply to it the same processes applied to graphs.

4.2 AN APPROACH FOR THE MDA ARCHITECTURE

The spirit of the MDA architecture stems from a model-driven approach to application and system development. MDA aims to model applications and systems

independently of the target implementation (hardware or software level), allowing the developed models to be reused.

The models thus created (PIM: Platform Independent Model) will be transformed to obtain application models specific to the target platform (PSM: Platform Specific Model). Automatic code generation tools then allow programs to be generated directly from the models.

This approach makes it easy to develop applications and their architectures based on models. Therefore, the MDA invests in a large workshop whose challenge is technically the manipulation of models. In this specific context, model transformation based on metamodeling techniques and model engineering opens up a promising field of investigation for research.

4.2.1 Principles of implementation

The MDA is seen as a specific MDE-type process that is based on the following principles:

- **Mastery of complexity**: the MDA is generally illustrated according to two main distinct concerns, one for developing business models independent of the PIM implementation platforms and the other for developing specific models of the PSM implementation platform [101]. This separation reduces the complexity of developing applications.
- **Abstraction and capitalization of models**: models must be independent of the technologies of implementation to adapt the business logic to different contexts and allow the evolution of applications towards new technologies, thus allowing better reuse of these models.
- **Modeling:** It is approached according to a productive vision (to generate the final code of the software developed for a given implementation technology) as opposed to the traditional contemplative idea (goal of documentation, specification and communication).
- **Metamodeling**: this process makes it possible to define the entities necessary for modeling specific systems [101]. A metamodel makes it possible to determine the structure of the models by specifying their syntax and semantics.
- **Model transformation:** metamodeling techniques make it possible to perform model transformations, considering these transformations as models.
- **The OMG standards**: the contribution of abstraction due to the introduction of metamodeling makes it possible to detect certain inconsistencies at the level of the

dialects presented by the metamodels. In this context, the OMG adopts MOF (Meta Object Facility), which allows the definition of the essential elements, the syntax and the structure of the metamodels.

In addition, the various MDA tools use XMI (XML Metadata Interchange) as the standard Metadata exchange format. This OMG standard makes it possible to describe, for example, an instance of the MOF in textual form using XML (Extensible Markup Language).

On the other hand, UML (Unified Modeling Language), with its ability to model systems independently of any platform, has established itself as a tool for modeling PIMs and some PSMs in MDA.

To name a few, other OMG standards find their roles in MDA, such as OCL (Object Constraint Language), which brings more precision to source models and language definitions, as well as CWM (Common Warehouse Metamodel), the modeling language dedicated to modeling data warehouse applications (**Figure 4.1**).

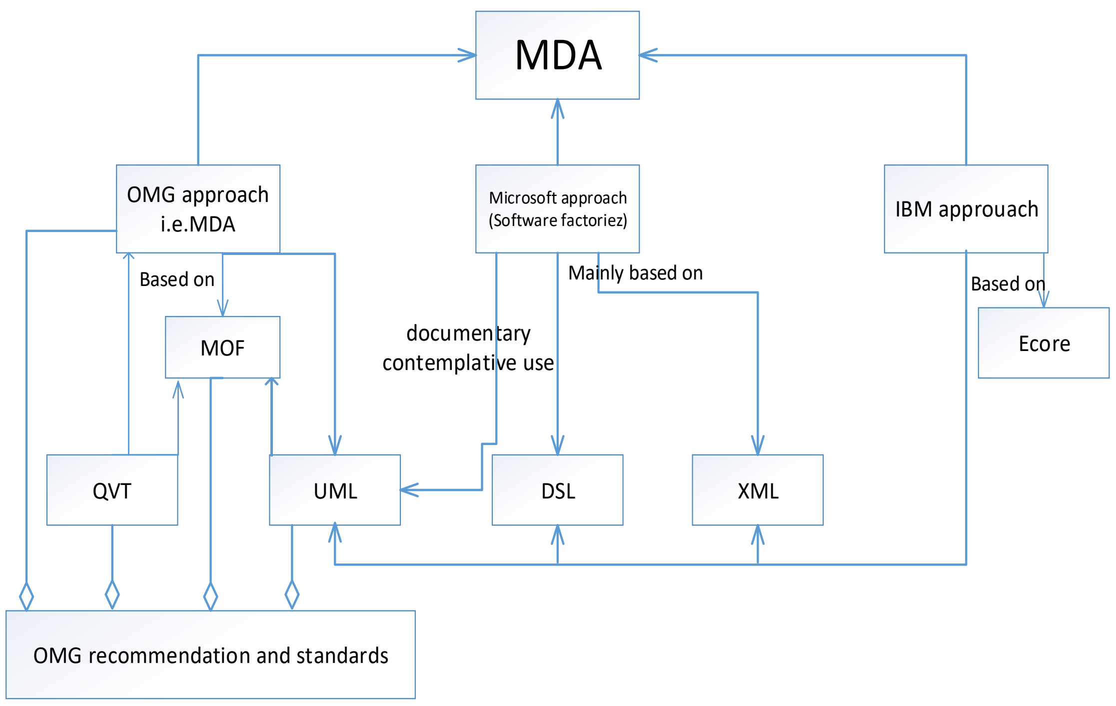


Figure 4.1: Variants of MDE [102]

4.2.2 Four-tier MDA architecture model

The OMG has defined architecture with four levels of abstraction as a general framework for integrating metamodels based on the MOF, as shown in **Figure 4.2**. In this architecture, the models of two adjacent levels are linked by an instantiation relation:

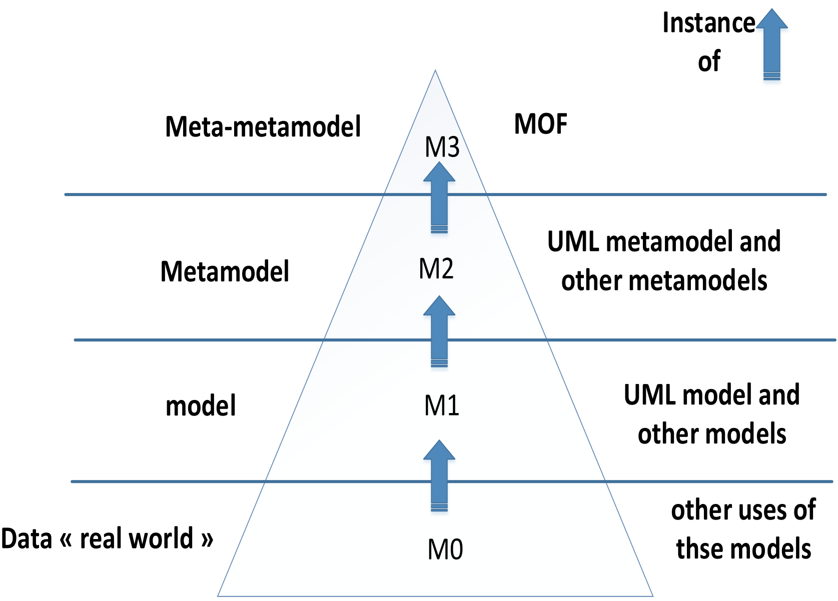


Figure 4.2: The four levels of abstraction for MDA

- **Level M0**: Level of model instances. It defines information for modeling real-world objects.
- **The M1 level**: This level represents all the instances of a metamodel. M1-level models must be expressed in a language defined at the M2-level.

UML is an example of M1 level model

- **The M2 level**: This level represents all the instances of a meta-metamodel. It is composed of information model specification languages. The UML metamodel, which is described in the UML standard and defines the internal structure of UML models, belongs to level M2
- **Level M3:** This level defines a unique language for the specification of metamodels. The MOF reflexive element of the M3-level defines the structure of all the metamodels of the M2 level

4.2.3 Models in the MDA architecture

The continuous evolution of technologies and the high cost of adapting software applications to these technologies prompted the OMG to propose the MDA [20] towards the end of the year 2000. The main goal of this approach is to separate the business parts of their technical implementation and guarantee the interoperability of functional models for different implementation choices.

Conceptually, the MDA proposes three points of view associated with their respective models: The CIM, the PIM and the PSM.

- **The CIM model**: the CIM (Computational Independent Model) corresponds to the needs model at the business level. It makes it possible to identify the different needs of the customers independently of any implementation
- **The PIM model**: the PIM (Platform Independent Model) corresponds to the specification model of the business part of an application. This specification must conform to an IT analysis seeking to meet business needs regardless of the implementation technology.
- **The PSM**: the PSM (Platform Specific Model) corresponds to the design and implementation model of an application about a specific platform

4.2.4 Model transformation in the MDA approach

4.2.4.1 Principle of transformation

The transformation of models is essentially based on the relationships between the models. Changing a source model into a target model requires the perfect knowledge of the relationships between the two models. This transformation must implement operations allowing the creation of a target model from the information provided by the source model.

In MDA, model transformation is driven by metamodels and includes two successive steps [101], which consist of:

- The specification of transformation rules expresses the correspondence between the concepts of the metamodel describing the source model and the concepts of the target model.
- Application of the defined transformation rules to the source model to generate the target model.

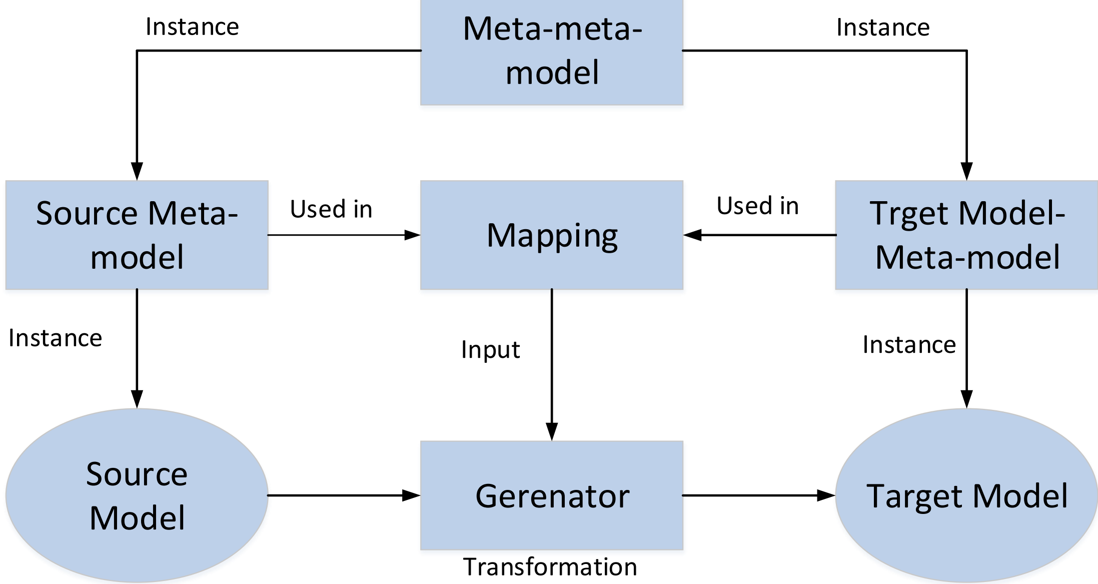


Figure 4.3: Model transformation process driven by meta models

4.2.4.2 Types of transformation

The MDA paradigm is based on model transformation. According to the literature [101], two types of basic transformation are distinguished:

- **Horizontal transformation:** in this type of transformation, the level of abstraction in modeling is preserved (PIM to PIM, or PSM to PSM). This transformation makes it possible to make updates at the level of the transformed models. The example of the passage from analysis to design illustrates the transformation PIM to PIM.
- **Vertical transformation:** in this type of transformation, the level of abstraction of the two models, source and target is different (PIM to PSM, or PSM to PIM).

The PIM to PSM refinement makes it possible, for example, to integrate elements specific to a technical platform, such as J2EE, Corba, .NET, etc. On the other hand, the PSM abstraction in the PIM is handy for reconstructing migrating systems.

The development of systems in MDA most often requires a series of model transformations. This can lead to a mixture of horizontal and vertical transformations, possibly integrating another hybrid transformation, using the two types defined above.

4.2.4.3 Classification of model transformation approaches

The classification of model transformation approaches is based, according to NPrakash [103], on several points of view, each of which allows a particular classification. According to CZarnecki [104] The classification of model transformation approaches is based on the

transformation techniques used in its approaches and the facets that characterize them, such as: the technique of patterns and that of programming languages. On the other hand, NPrakash [103] proposes a classification according to a multidimensional point of view; it is particularly interested in the field of application of the models and their generosity (ability to transform any input model to any output model)

In this section, we will present a classification of model transformation approaches, drawing inspiration from the work of CZarnneki [104] where we find a decomposition of model transformation into two categories: model-to-code type transformations and model-like transformations to model.

a- Transformation from model type to code

This transformation offers two transformational approaches. The first approach is based on the principle of the visitor (Visitor-based approach), and the second is based on the principle of patterns (Template-based approach).

- Visitor-based approaches transform the model as input to code written in a programming language as output. The visitors added to the input model reduce the semantic difference between the model and the target language. The target code is obtained by traversing the visitor-enriched model to create an output text stream.
- Pattern-based approaches rely on target code meta-code fragments to access source model information. These approaches are currently widely used in MDA tools, such as: AndroMDA (a code generator that uses open Velocity technology for writing patterns).

b- Transformation from model-to-model type

- **Modeling of the transformation:** Model-to-model transformation continues to develop in the significant MDA shipyard. The differences in semantics and syntax between the two models, source and target, such as PMIs and PSMs, impose a rigorous approach for such a transformation. Model transformation modeling then appears as a solution to this problem. Modeling a transformation falls under metamodeling, which is currently emerging as a promising technique in model transformation.
- **Transform structure :** A model transformation is a model in itself. It is defined by a set of elements: transformation rules, their organization and scheduling, traceability and orientation. The combination of its elements makes it possible to

describe the transformation. However, these transformation rules must first be specified in order to be able to express the correspondences between the concepts of the source and target Metamodels.

- **Transformation rules:** a transformation rule describes how one or more constructs in the source model language can be transformed into one or more constructs in the target model language [101]. In addition, any transformation rule has declarative or imperative form logic to express constraints or calculations on the source and target models of the transformation. Technically, a transformation rule consists of two parts: a left part called LHS (Left Hand Side), which accesses the source model, and a right part called RHS (Right Hand Side), which accesses the target model. Executing a left-to-right oriented transformation rule replaces source model constructs conforming to the LHS of the rule with RHS of the same rule for the target model construct. This technique requires the organization and scheduling of the rules as well as their orientation. The traceability mechanism also makes it possible to strengthen the technique by archiving the correlations that exist between the elements of the transformation models.

  the transformation rules express the correspondence between the concepts of the source metamodel and the target metamodel. The specification's role is to describe such relations regardless of any execution. MDA plans to automate this phase entirely. Currently, there is no standard for expressing transformation rules. However, the current work of the OMG on the MOF/QVT (Query Views Transformation) standard seems promising.

- **Approaches to the definition of the transformation**: the Model-to-Model transformation is based on a varied structure. Therefore, several approaches attempt to define this type of transformation. In the literature, one generally distinguishes five approaches:

  - Direct manipulation approaches,

  - Relational approaches.

  - Approaches based on graph transformation,

  - Structure-based approaches,

  - Hybrid approaches.

In our work, we are particularly interested in approaches based on the transformation of graphs because our objective is to transform UML diagrams which are graphs, towards its formal specification in Maude languages which are also in the form of graphs.

4.3 GRAPH TRANSFORMATION

The modeling of systems comes up against the difficulty of describing their complex structures. Models and metamodels (for example, UML) most often have a representation in the form of a graph. Furthermore, the graphs offer pleasant and effective support that allows the modeling of systems. Therefore, graph transformation and rewriting techniques can be applied to model transformation

Before discussing graph transformation techniques and the tools making such techniques applicable, it is necessary to recall some basic concepts relating to graph theory.

4.3.1 Graph concept

A graph is generally defined as a set of vertices connected by edges. Mainly, there are two categories of graphs: directed and undirected (see **Figure 4.4** And **Figure 4.5)**

In the rest of this section, we will present some definitions relating to the properties and characteristics of graphs:

- **The Graph:** we call graph G the pair (S, A) made up of a non-empty set S of vertices and a non-empty set A of edges. Each element of A connects two elements of S.
- **Directed graph**: a graph is said to be directed if its edges have an orientation (starting vertex is the arrival vertex). Otherwise, the graph is said to be undirected

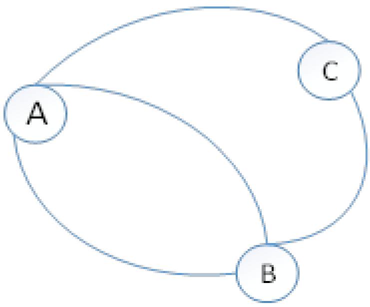


Figure 4.4: Undirected graph

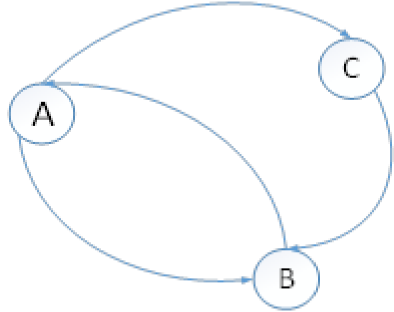


Figure 4.5:Simple directed graph

- **Labeled graph:** a graph used for the representation of a model must be able to support the different relationships between the elements of the model represented. Directed graphs are not enough to express the relationships between model elements. Labelling arcs in graphs makes it possible to specialize the relations between different vertices in a graph.

A labeled graph is, therefore, a directed graph whose arcs have labels

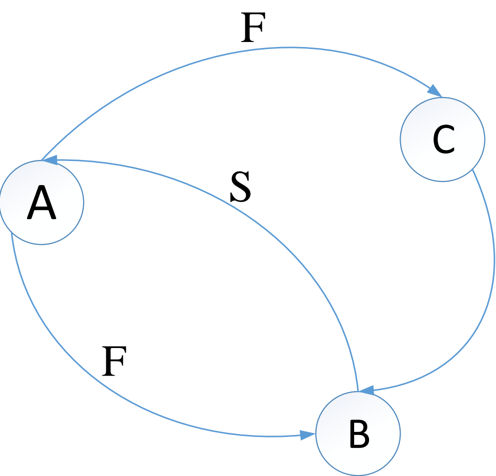


Figure 4.6: Labeled directed graph

- **The Degree of a graph:** the degree of a vertex in a graph is the number of edges in this vertex. If the graph is oriented, the degree of a vertex is defined in degree entering and degree exiting, relative respectively to the number of arcs entering and exiting in this vertex.The degree of a vertex in **Figure 4.4** is 3, and the degrees in and out of vertex A in **Figure 4.5** are 1 and 2, respectively.
- **The order of a graph:** the order of graph G is defined by the number of vertices present in this graph. The order of the graph in **Figure 4.4** is 3

- **The Path in a graph:** the path in a graph is defined by the succession of arcs traversed in a direction defined in the graph, it has the following properties and characteristics:
- The length of the path is equal to the number of arcs covered;
- The chain is said to be a chain if the direction of the arcs is not taken into account;
- The path is said to be a circuit if it returns to its starting point.
- The distance between two vertices in a graph is defined as the length of the shortest path between these two vertices; if this length is 1 the vertices are said to be adjacent.
- The diameter of a graph is the greatest distance separating two vertices in this graph
- **The subgraph:** a subgraph G'(S', A') of a graph G(S, A) is a graph composed of a subset of vertices S'∈ S and a subset d 'edges A'∈A' such that A' represents the edges connecting the vertices S' in the graph G (cf. **Figure 4.7**).

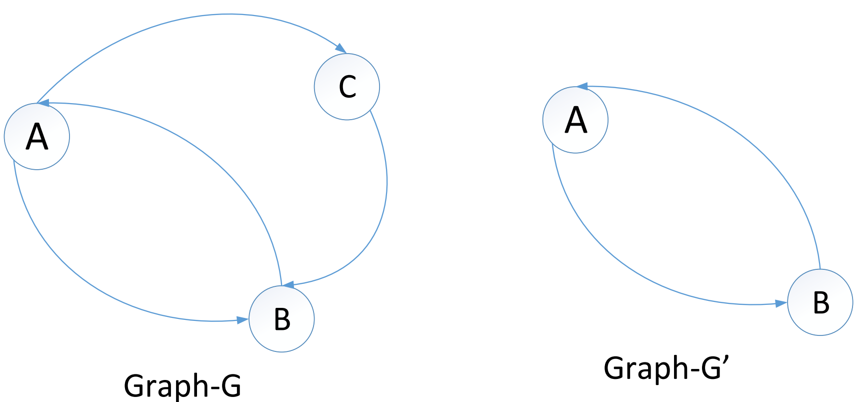


Figure 4.7:G' is a subgraph of G

4.3.2 The basic of graph transformation

Graph transformation is a system for transforming models [101]; it is carried out thanks to the transformation rules [105]. The rules apply to a source graph, based on the principle of pattern matching, to produce a target graph. The basic principle is to apply transformation rules that replace subgraphs in the source graph with subgraphs in the target graph, using the production defined in the transformation rules. These fragments of subgraphs can be expressed in respective concrete or abstract syntaxes of the source and target models.

ATOM3[148], UMLX and VIATRA are examples of tools that support model transformation based on graph transformation.

By analogy to Chomsky's grammar which applies to texts, the transformation of graphs is carried out thanks to the rules of transformation organized in the grammar of graphs. The basic idea is to be able to derive target models from source models.

4.3.2.1 Graph grammar

From a certain point of view, graph grammars are a generalization of Chomsky's grammar applied to graphs. The graph transformation is specified as a model of graph grammar. These grammars are composed of rules, each of which consists of two parts: a left part called LHS (Left Hand Side), which corresponds to a graph, and a right part called RHS (Right Hand Side), which also corresponds to a graph.

4.3.2.2 Principle of transformation

The graph transformation process distinguishes two types of graphs: non-terminal graphs, which are the intermediate results on which the rules of the grammar are applied, and terminal graphs, which are in the language generated by the grammar and on which one does not can no longer apply rules.

- **Composition of a graph transformation rule:** A graph transformation rule consists of a set of attributes allowing him to substitute the graphs of the source model with the graphs of the model target [155]:
  L: graph on the left side on which the rule will apply.
  R: graph on the right side that the rule will produce.
  K: a subgraph of L.
  F: a set of functions allowing the rule to perform the substitution.
- **Application of a graph transformation rule:** The transformation of a source graph 'G' towards a target graph 'G' 'consists of the application of one or more times of the rules of the transformation grammar to graph G. The application of a rule of transformation is generally carried out according to the following principle:
  - Choose an occurrence 'l' of the graph on the left side L, according to the rule to be applied.
  - Check the conditions of application of the rule.
  - Realize the substitution of the subgraph’s 'l' of L by subgraphs 'l' ' in G'

Applying a transformation rule to a graph G to produce a graph G', is called a direct derivation from G to G' through the transformation rule. The order in which the rules of grammar are triggered remains arbitrary. However, it is possible to express an order of execution of the rules by assigning them priorities, for example.

4.4 CONCLUSION

In this chapter, we introduced model transformation, one of the promising techniques in the MDA approach. This approach automates modeling processes, including code generation, from development to testing.

In this context, we have presented some modeling approaches from the literature, starting with an introduction to the MDA approach. We then gave an enumeration of models' transformation types, followed by a classification of some transformation approaches. In the second phase, we focused on a specific model transformation framework based on graph transformation. Finally, we briefly presented certain envisaged transformation tools used in our contribution, which consists of the proposal of a graph grammar, allowing the transformation of UML-Mobile models towards a Maude specification.

Chapter 5

Rewriting Logic and The Maude language

Chapter 5: Rewriting Logic and The Maude language

5.1 INTRODUCTION

The rewriting logic is a replacement logic where each rewrite rule is a general form of an atomic action that can occur in competition with other actions in a concurrent system. This logic is a consequence of the work of José Meseguer [106]. On general logic to describe concurrent systems. The rewriting logic is a unifying logical and semantic framework in which other logic, concurrency models and languages can be represented (CCS, LOTOS, π-calculus, networks of Petri, etc.). In rewriting logic, a rewrite of a term consists of replacing it with an equivalent term, per the laws of term algebra[107].

Rewriting for this logic allows computing a rewrite ability relation between algebraic terms. The essential idea of rewriting logic is that the semantics of rewriting can be rigorously changed in a very fruitful way [106]. Furthermore, it has been widely demonstrated that it makes it possible to reason perfectly about the behavior of concurrent systems. So, rewriting logic is a computational model and a semantic framework that expresses communication, interaction, parallelism, concurrency, and parallelism.

In this context, Maude represents a formal language widely used to verify concurrent systems; Maude is a formal language for algebraic specification and declarative programming, simple, expressive and efficient. This chapter will introduce rewriting logic and its semantics and present its basic concepts. Then, Maude rewriting logic language and its different levels of specification.

5.2 REWRITING LOGIC

Rewriting logic is concurrent change logic that can process the state and computation of concurrent systems. This logic has been widely used to specify and analyze systems and languages in different application domains.

Consequently, rewriting logic offers a formal foundation for describing and researching the behavior of concurrent systems. In fact, it enables it to consider potentially complicated alterations that would correspond to atomic acts axiomatized by the rewriting rules. This logic's main argument is that computation in a concurrent system corresponds to logical deduction, which is intrinsically concurrent. [106].

5.3 REWRITE THEORY

A concurrent system is defined by a rewrite theory R = (Σ, E, L, R) in rewriting logic, where (Σ, E) is the signature (an equational theory) describing the specific algebraic structure of the system's states (multi-set, binary tree), which are distributed according to this structure. The dynamic structure of the system labeled R is described by the rewrite rules (L is a set of labels of these rules). The elementary and local transitions that are possible in the concurrent system's current state are also specified by the rewriting rules.

Each rule (denoted [t] → [t']) corresponds to an action that can occur in competition with other actions. So, rewriting logic is logic that captures the concurrent change in a system.

5.3.1 Labeled rewrite theory

A rewriting theory R is a 4-tuple (Σ, E, L, R) such that:

1. Σ is a set of function symbols and sorts.

2. E a set of Σ-equations (the set of equations between the Σ-terms).

3. Read a set of labels.

4. R is a set of rewrite rules, defined as $R \subseteq L \times (T_{\Sigma,E}(X))^2$

Each rule consists of two components: a label and a pair of equivalence classes of terms

$T_{\Sigma, E}(X)$ on the signature (Σ, E), modulo the equations E, with X = {x*1,...,xn,…*} an infinite and countable set of variables [108].

5.3.2 Conditional rewrite systems

For a rewrite rule of the form r ([t], [t'], A1, …, Ak), the following notation is used, r : [t] → [t'] if A1 ∧ … ∧ Ak, where a rule r expresses that the equivalence class containing the term t can be rewritten as the equivalence class containing the term t' if the condition of rule A1 ∧ … ∧ Ak is verified. The latter is called the condition of the rule and can be abbreviated by the letter C, and the rewrite rule is called conditional.

The conditional part of a rule can be empty; in this case, the rules are called unconditional rewrite rules and are denoted by r: [t] → [t']

A rewrite rule can be parameterized by a set of variables {x1, … , xn } which appear either in t, t' or A , and we write: *r : [t(x1, ..., xn)] → [t' (x1, ..., xn)] if A(x1, ..., xn)* [106].

5.4 DEDUCTION RULES

In a concurrent system, the sequence of transitions executed from a given initial state constitutes the computation corresponding to a proof or a deduction in the rewriting logic. Hence, the rules formalizing the operation of rewriting terms are called deduction rules. That is, for Given a rewrite theory R, = (Σ, E, L, R), we say that the sequence $[t] \rightarrow [t']$ is provable in R and we write $R \vdash [t] \rightarrow [t']$ if and only if $[t] \rightarrow [t']$ is obtained by a finite application of the rules following deductions [139].

- **Reflexivity**

For each term $[t] \in T_{\Sigma,E}(X)$, $\overline{[t]} \rightarrow \overline{[t]}$ where $T_{\Sigma,E}(X)$ is the set of Σ terms with variables constructed on the signature Σ and the equations E

- **Congruence**

For each function $f \in \sum n$, $n \in \mathbb{N}$

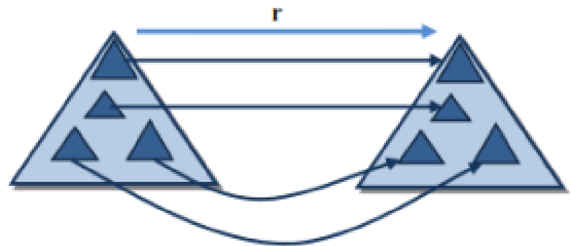
$$\frac{[t1] \rightarrow [t1'] \dots \dots [tn] \rightarrow [tn']}{[f(t1 \dots \dots tn)] \rightarrow [f(t1' \dots \dots tn')]}$$

- **Replacing**

For each rule r : $[t(x1\ ...\ Xn)] \rightarrow [t'(x1\ ...\ Xn)]$ in R

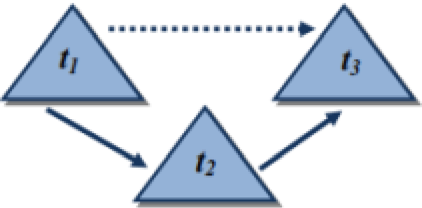
$$\frac{[w1] \rightarrow [w1'] \ \dots \dots \ [wn] \rightarrow [wn']}{[t(\bar{w} / \bar{X})] \rightarrow [t'(\bar{\bar{w}}' / \bar{X})]}$$

Knowing that $t(\overline{w}/\overline{X})$ denotes the simultaneous substitution of xi by wi in t with $\overline{x}$ representing $x1\ \dots \dots \dots .\ xn$.



- **Transitivity**

$$\frac{[t1] \rightarrow [t2]\ [t2] \rightarrow [t3]}{[t1] \rightarrow [t3]}$$



Intuitively, we should consider the inference rules above as different ways to construct all the concurrent computations of the concurrent system specified by R.

The rule of reflexivity says that for any state t there is an inactive transition in which nothing changes.

The congruence rule is a general form of "lateral parallelism", so each operator f can be seen as a constructor of parallel states, allowing its unfixed arguments to evolve in parallel.

The replacement rule supports a different form of parallelism, which might be called "parallelism underfoot" because, in addition to rewriting a left-side instance of a rule over the

corresponding right-side instance, the fragments of The state in the rule variable substitution can also be rewritten, provided that the variables involved are not frozen.

The transitivity rule allows us to build longer simultaneous computations by componentizing them sequentially [108].

5.5 MAUDE SYSTEM

Based on a mathematical theory of rewriting logic, the Maude language emerged as declarative programming and a formal specification. Jose Meseguar and his group in the computer lab at SRI International took an interest in rewriting logic and developed the Maude language.

Maude is a powerful and expressive simple language regarded as one of the finest for modeling concurrent systems and algebraic specification. [109].

Maude specifies theories of rewriting logic, data types are defined algebraically by equations, and the dynamic behavior of the system is defined by rewrite rules.

Maude also supports object-oriented programming with the inclusion of multiple inheritance and asynchronous communication through message passing.

Maude's group also focused their efforts on performance; the current version of the interpreter can reach millions of rewrites per second. Of this fact, Maude competes with high-level languages in terms of efficiency.

Maude also supports using sockets for network programming, which allows not only modeling, simulating and analysing competing systems but also programming them.

To specify a concurrent system, Maude offers three modules: consequently, we differentiate between three types of modules: functional modules to implement equational theories. System modules implement rewrite theories and define the dynamic behavior of a system. Finally, object-oriented modules implement object-oriented rewriting theories (they can be reduced to system modules) [110]. These modules must be declared respectively, respecting the following syntaxes:

```
fmod NAME is … endfm
```

```
mod NAME is … endm
```

```
omod NAME is … endom
```

Dots represent declarations and expressions that may appear in the module. It is also important to note that there are two levels in the current version of Maude (Maude 2.6): Core Maude and Full Maude:

- Core Maude is the base level of Maude, programmed directly in C++. It implements all the basic features of the language, the functional modules and the system modules.
- Full Maude is the top level. Programmed in Core Maude, Full Maude is used with the object-oriented programming paradigm and modules object-oriented Maude. All commands and modules in Full Maude must be declared in parentheses.

5.5.1 Characteristics of Maude

We can enumerate the most important characteristics of Maude language as follows:

- **Simple**: It is simple to program with Maude because it is a declarative language with few syntactic constructions that are very simple and easy to understand. Maude is seen as a meta-language that allows the definition of its own ratings.
- **Expressive**: With Maude, it is possible to express deterministic calculations using equations in functional modules easily. Thus, concurrent and non-deterministic calculations with rules for rewriting system modules.
- **Powerful semantics**: a Maude program has very clear semantics; it is based on sound logic, called rewriting logic. A Maude program should be seen as a statement of a rewriting theory. Thus, the execution of this program must be interpreted as a logical deduction from the axioms defined in the program.
- **Wide spectrum**: It supports the formal specification, prototyping, and concurrent programming.
- **Multi-paradigms**: it combines functional programming paradigms, concurrent and object-oriented.
- **Executable**: a Maude specification is directly executable.

- **Equipped with formal verification tools**: Maude offers its users a set of tools to facilitate the task of verification significantly; among these tools, we can cite:
  - The Reachability Analysis tool: is a key feature that enables users to explore and analyze term rewriting systems by determining whether a given state can be reached from an initial state which can help users identify critical properties and detect potential errors in modeled systems and this is the tool we will use in our verification phase.
  - Model Checker: for property verification linear temporal models specified in Maude.
  - Inductive Theorem Prover: this tool is used to verify the properties of equational specifications.

5.6 MAUDE LANGUAGE SYNTAX

In this section, we present the most interesting syntactic notions of the Maude language (sorts, subsorts, operations, etc.).

5.6.1 Sorts

Sorts (data types) are the first things to define in a specification. To declare a new type in Maude, use the keyword "sort" followed by the name of the sort as follows:

sort *(Sort name).*

In the case where we have several sorts to declare, we can use the keyword “sorts” in the following way:

Sorts *(first Sort name) (second Sort name) (last Sort name)* .

The example below shows the announcement of two sorts Nat (the natural integers) and NzNat (the non-zero natural integers):

Sort Nat .
Sort NzNat .
Or
Sorts Nat NzNat .

5.6.2 The sub-sorts

The subsort relation is equivalent to the subset relation. i.e. Let S1 be a sub-sort of S2: all the elements of S1 belong to S2, while the converse is not necessarily true. Subsorts are declared using the "subsort" keyword as follows:

subsort *(subsort name) < (Sort name).*

The statement below means that 'NzNat' is a sub-sort of 'Nat':

subsort NzNat < Nat .

5.6.3 The operations

To declare an operation, you must use the “op” keyword as follows:

op *(operation name) : (Sort1) (Sort2) (SortK)* **->** *(Sort)*

(Sort1) (Sort2) (SortK) is the list of sorts forming the arguments (the domain) of the operation . While (Sort) represents the sort of the result (the co-domain). Maude offers the possibility of declaring several operations thanks to the keyword "ops", provided that all these operations have the same domain and co-domain. In the following example, we present some declarations of operations in Maude.

op *successor_* : Nat -> Nat .
ops _+_ _*_ : Nat Nat -> Nat

5.6.4 The overload of operations

In Maude, operators can be overloaded. i.e. we can have multiple declarations for the same operator with different domains and co-domains. For example, the _+_ operation can be overloaded by giving two declarations to this same operator for two different sorts (natural numbers and character strings):

op _+_ : Nat Nat -> Nat .
ops _+_ : String String -> String .

5.6.5 The variables

A variable represents an undefined value of some kind. It is declared in Maude with the keyword "var" (or "vars" for several variables of the same kind). An example of variable declarations (S, X, Y and Z) is as follows:

var S : String .

var X : Nat .

vars Y Z : Nat .

5.6.6 Equations

We recall that a term is a constant, a variable or the result of the application of an operation on a set of terms. Maude supports two types of equations:

Unconditional equations: an unconditional equation is declared using the keyword "eq" as follows:

eq *(Terme) = (Terme).*

Conditional equations: to declare a conditional equation, you must use the keyword "**ceq**". The form of a conditional equation is:

ceq *(Term₁) = (Term₂) if (Condition₁) / \ (Condition₂) / \ / \ (Conditionₙ).*

The "/ \" operator (condition-concatenation operator) links the different conditions of a conditional equation.

5.6.7 The rewrite rules

Like equations, Maude offers two types of rewrite rules:

Unconditional rewrite rules: an unconditional rewrite rule has the following syntax:

rl [*label*]: (*Term1*) => (*Term2*) .

Conditional rewrite rules: the form of a conditional rewrite rule is as follows:

crl [label]: (Term1) => (Term2) if (Condition1) / \ / \ (Conditionn).

5.6.8 Module importation

As in most programming languages, Maude offers the possibility to import other modules. This allows developers to benefit from all declarations and definitions of imported modules.

This mechanism minimizes the redundancy in specifications. In addition, it allows for offering better modularity.

Maude supports three import modes:

- "Protecting": in this mode, the developer cannot modify the declarations of the imported module. i.e. all defined operations and sorts are used strictly as they are in the imported module.
- "Including": in this mode, the developer can freely change the meaning of the elements declared in the imported module.
- "Extending": in this mode, the developer can add new terms to a sort but he cannot modify the meaning of the operations.

The syntax for using these three modes is as follows:

- protecting (module name).
- including (module name).
- extending (module name).

5.7 MAUDE MODULES

In Maude, a module will provide a collection of sorts and a collection of operations about these sorts, as well as the information needed to reduce and rewrite expressions entered by the user in the Maude environment. There are three types of modules defined in Maude.

- Functional modules are used to define data types.
- System modules are used to define the dynamic behavior of a system.
- object-oriented modules are used to define the object-oriented paradigm.

5.7.1 Functional modules

These modules define the data types and operations used by the equations. For example, in Maude, an equational specification is a functional module which is represented by the following syntax:

```
fmod MODULE-NAME is ***(the name of the module)
 BODY ***(announcements of sorts, operations, variables,
 equations, membership axioms and comments)
 Endfm.
```

MODULE-NAME is the introduced module name, and BODY is the set of announcements of sorts, operations, variables, equations, membership axioms and comments.

Comments start with '***' or '---' and end with the end of the current line, or they begin with *** (or --- (and end with the occurrence of)."

The module's body specifies a theory *(Σ, EUA, Φ)* in the membership equational logic. The signature *Σ* includes sorts (indicated by the keyword sort), subsorts (specified by the keyword subsort) and operators (introduced with the keyword op). The operator syntax is defined by users by indicating the position of the arguments by the symbol (_). Some of these arguments can be specified as frozen using the frozen (Position Argument) keyword.

The set *E* denotes the equations and membership tests (which can be conditional). *A* is a set of equational axioms introduced as attributes of Certain operators in the signature *Σ*such as the axioms of associativity (specified by the word assoc keyword), commutativity (specified by the comm keyword) or identity (specified by the id keyword). The latter is defined in such a way that the equational deductions are made modulo the axioms of A.

Equations are specified by the keyword eq or the keyword ceq (for conditional equations), and membership or membership tests are introduced with the keywords mb or cmb (for conditional membership tests). A condition linked to an equation or membership test can be formed by conjunction with equations and unconditional membership tests. Variables can be declared in modules with the var or vars keywords, or introduced directly into equations and adhesion tests as a var:sort expression [111].

An example of a functional natural number module is:

```
fmod NAT is ***(the module name is: NAT)
sort Nat ****(natural sort announcement)
sort PositiveNat .
subsort PositiveNat < Nat .
op 0: -> Nat [ ctor] . *** (announcement the existence of an operation<mathematically: function>
which returns 0)
op s_ : Nat -> Nat [ ctor ] .***( announcement the existence of an operation)
op _+_: Nat Nat -> Nat .
vars NM: Nat . ** *(announcement the existence of natural variables N and M)
var P: PositiveNat.
cmb P: PositiveNat if P =/= 0 .**( membership axiom indicating that P is of type PositiveNat)
*** if P is different from 0
eq N + 0 = N . *** N + 0 and N denote the same term (same equivalence class) which is [N]
eq (S N) + (S M) = S S (N + M) .
endfm
```

5.7.2 System modules

System modules define the dynamic behavior of concurrent systems by enriching the functional modules with a set of rewriting rules. The introduction of rewriting rules allows for the expression of concurrency in systems.

A system module describes a rewrite theory that includes sorts, operations, variables, equations, membership axioms (conditional and unconditional) and conditional and unconditional rewrite rules.

A rewrite rule executes when its left part matches a portion in the overall status of the system and with the satisfaction of the condition in the case of a conditional rule [109].

In Maude, the following syntax represents the system module:

```
mod MODULE-NAME is
 BODY
endm.
```

R rewrite rules are introduced with *rl* keywords or *crl* . They are specified in Maude with the syntax:

crl [l]: t => t' if cond . If the rule is unconditional, the
crl keyword is replaced by *rl,* and the *"if cond"* clause is omitted [111].

An example that demonstrates the use of these modules is as follows:

```
mod CHOICE-INT is ***(CHOICE-INT is the module name)
including INT ***(import the INT module (the module for integers)
to use its operators)
op _?_ : Int Int -> Int ***( an operator are announced)
vars YZ: Int . ***(Two integers Y and Z are announced)
rl [ choose_first ]: Y ? Z =>Y .
rl [ choose_second ]: Y? Z =>Z .
endm[110]
```

5.7.3 Object-oriented Modules

Object-oriented modules are supported by the Full-Maude system [109]. Note that Full Maude is an extension of Maude (Core Maude), whose code is written in Maude, which enriches the Maude language with a very powerful and extensible algebraic module. These object-oriented modules are introduced by the keywords:

(omod MODULNAME is

…..

Endom)

The module body is a rewrite theory $\Re = (\Sigma, EUA, \Phi, R)$. They support the specification and manipulation of objects, messages, classes and inheritance. A concurrent object-oriented system, in this case, is modeled by a multiset of juxtaposed objects and messages, where concurrent interactions between objects are governed by rewrite rules.

An object is represented by the term < O : C | a1: v1, …, an: vn >, where O is the name of the instance object of class C, ai, ∈ 1..n, the names of the attributes of the object, and vi, their respective values.

The declaration of classes follows the syntax: class C | a1: s1, …, an: sn. where C is the class name and si is the sort of the ai attribute. It is also possible to declare subclasses and thus benefit from the notion of inheritance. Messages are declared using the msg keyword. The general form of a rewrite rule in Maude's object-oriented syntax is :

crl [r]: M1 ... Mn < O1: F1 | a1> ... < Om: Fm | am > =>< Oi1 : F'i1 | a'i1 > ... < Oik : F'ik | a'ik > M1' ... Mp ' if Cond.

Where *r* is the rule label, *Ms, s ∈ 1..n, and M'u, u ∈1..p* are messages, *Oi , i ∈1..m, and Oil , l ∈1..k* , are objects, and *Cond* is the rule condition. If the rule is unconditional, we replace the *crl* keyword with r.l. and remove the *if Cond* clause.
We will not expand on the explanation here further because an object-oriented unit is not of interest to us in our study as it can be defined using system units.

5.8 PREDEFINED MODULES

Maude's predefined modules are stored in a specific library and can be imported by other user-defined modules. They are introduced in the source files of Maude *prelude.maude* and model-checker.maude, such as *BOOL*, *STRING* and *NAT*. These modules declare sorts and operations to handle, respectively, Boolean values, character strings and natural numbers. The *model-checker.maude file* contains the predefined modules interpreting the tools necessary for the use of Maude's LTL Model Checker [112]

5.9 EXECUTION AND FORMAL ANALYSIS UNDER MAUDE

In a module written in Maude, the rewriting rules constitute the elementary units of execution. They interpret the local actions of the modeled system and can be executed in a

constant time and concurrently (at any time). Maude offers the possibility of simulating the execution of such rewrites (via rewrite rules) or equational rewrites (via equations) in an M module by implementing the two commands: *reduce* and *rewrite.*

The reduce command (abbreviated as red) allows an initial term to be reduced by applying the membership equations and axioms in a given module.

It has the following syntax: Reduce {in module:} term.

The rewrite command rewrite (abbreviated as *rew*) and the fair rewrite command *frewrite* (abbreviated as *frew)* perform a single rewrite sequence (among several possible sequences) from an initially given term following the syntax:
rewrite {in module : } term .

frewrite {in module:} term.

These commands make it possible to rewrite an initial term using the rules, equations and membership axioms in the specified module [113].

5.10 FORMAL ANALYSIS AND VERIFICATION OF PROPERTIES

The formal verification of models on the systems specified in Maude is carried out using tools available around the Maude system. Among these tools, we distinguish an accessibility analysis tool (the *search command)* and a verification module by *model-checking* properties expressed in linear temporal logic (LTL) [114]. To analyze all the sequences of possible rewrites from an initial state (term) *t 0, the search* command is used. This searches whether states corresponding to given patterns and satisfying certain conditions can be accessed from *t 0.* In addition, this command performs a deep traversal of the computation tree (reachability tree) generated during this search to detect invariant violations in infinite-state systems[115].

The formal analysis of systems by Model Checking is a set of techniques to automatically verify temporal properties relating to the behavior of systems [116]. Model Checking makes it possible to verify the satisfiability of a given property in a state or a set of states by conducting an exhaustive exploration of the set of states accessible from an initial state. It receives as input an abstraction of the behavior of the system (a system of transitions), represented by a Kripke structure *K,* and a property ϕ of this system, formulated in some temporal logic, and answers whether or not the abstraction satisfies the formula ϕthat is, whether $K \models \phi$. The advantage of Model Checking is that it returns a trace of the execution of the system violating the property when the latter is invalid.

5.10.1 Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) is an extension of classical logic with temporal operators, such as *G* and *F* (representing "globally", "finally or fatally", respectively), which allow for expression properties relating to the execution of a sequence of states. It is called temporal because it describes the sequence of events observed in a system. This logic makes it possible to specify interesting properties for concurrent systems, particularly the properties of safety, accessibility, liveness and fairness:

- **Safety:** Nothing bad ever happens.
- **Accessibility:** a certain situation can be reached.
- **Liveliness:** something good is always possible.
- **Fairness:** something will repeat itself infinitely often.

LTL formulas are built from propositional variables called atomic propositions, Boolean operators ($\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$), and temporal operators, *F* (Future), *G* (Global), *U* (Until),*X* (neXt).

Atomic properties make it possible to describe the states of the system: a state *t* is said to be labeled by an atomic proposition ϕif $\phi$is true in *t.* Moreover, temporal operators make it possible to link system states within an execution sequence.

- The formula *Xf* indicates that the LTL formula *f* should be checked the immediate next time along the execution (neXt).
- The formula *f U g* indicates that *f* is always true until a state where *g* is true (Until).
- The formula *Gf* means *f* is always verified (Generally) throughout the execution.

Ff formula *indicates* that *f* should be checked later, at least in some execution state (Fatally) [112].

5.10.2 The Maude LTL Model Checker

Maude's model-checking tool uses LTL logic. It requires the system to be verified, described by a rewrite theory $\mathfrak{R} = (\Sigma, E, \Phi, R)$, or finite state, i.e., from an initial state [*t0]*, the set of accessible states defined by $\{[u] \in T\ \Sigma, E\ /\ \mathfrak{R} \vdash [t\ 0] \rightarrow [u]\}$ is finite. So, Maude's LTL model checker takes as input a rewrite theory $\mathfrak{R}$ from which it generates the Kripke structure $K\ (\mathfrak{R}, State)$ as well as an LTL temporal formula $\phi$and checks if this structure $K\ (\mathfrak{R}, State)$ satisfies the formula ϕ. If this formula is not valid, the LTL model checker returns a counter example showing a sequence of states leading to the violation of this formula. This tool is implemented as a module, specified in terms of rewriting logic, in the Maude language. The latter imports other predefined modules (also specified in the Maude language):

- The ***LTL-SIMPLIFIER module*** implements formal procedures to simplify the LTL formula expressing a property.
- The ***SATISFACTION*** module specifies the syntax and semantics of the satisfaction operator (|=), indicating whether a given formula is true or false in a certain state.
- The ***SAT-SOLVER module*** makes it possible to check the satisfiability and the topology of a formula specified in LTL logic [114].

In the following, we find some of the LTL operators in Maude's syntax:

```
fmod LTL is
…
*** LTL op defined operators
_->_: Formula Formula -> Formula . *** implication
op _<->_: Formula Formula -> Formula . *** equivalence
op <>_: Formula -> Formula . *** possibly
op[]_: Formula -> Formula . *** always
op _W_: Formula Formula -> Formula . *** unless
op _|->_ : Formula Formula -> Formula . *** leads- to -
op _=>_: Formula Formula -> Formula . *** strong implication
op _<=>_: Formula Formula -> Formula . *** strong equivalence
…
endfm
```

LTL operators are represented in Maude using a syntactic form similar to their original form.

The State state is generic. After specifying the behavior of his system in a Maude system module, the user can specify several predicates expressing certain properties related to the system. These predicates are described in a new module that imports two modules: the one that describes the dynamic aspect of the system and the ***SATISFACTION module.***

Let, for example, ***M-PREDS*** be the name of the module describing the predicates on the states of the system:

```
mod M-PREDS is
protecting M .
including SATISFACTION.
subset Configuration < State.
…
endm
```

M is the name of the module describing the behavior of the system. The user must specify that the state is chosen (configuration chosen in this example) for his own system under a type of type State. In the end, we find the ***MODEL-CHECKER module,*** which offers the ***model-Check function*** [113].

The user can call this function by specifying a given initial state and a formula. Then, Maude's Model Checker verifies whether this formula is valid (according to the nature of the formula and the Model Checker procedure adopted by the Maude system) in this state or all of the states accessible from the initial state. If the formula is not valid, a counterexample is displayed. The counterexample concerns the state in which the formula is not valid:

```
fmod MODEL-CHECKER is
including SATISFACTION.
…
op counterexample: TransitionListTransitionList -> ModelCheckResult [ ctor ] .
op modelCheck : State Formula ~> ModelCheckResult .
…
endfm
```

5.11 EXECUTION OF MAUDE

We can start a session with Maude under Windows by clicking on the core Maude 2.6 file. Maude's main window opens, as shown in **Figure 5.1 .**

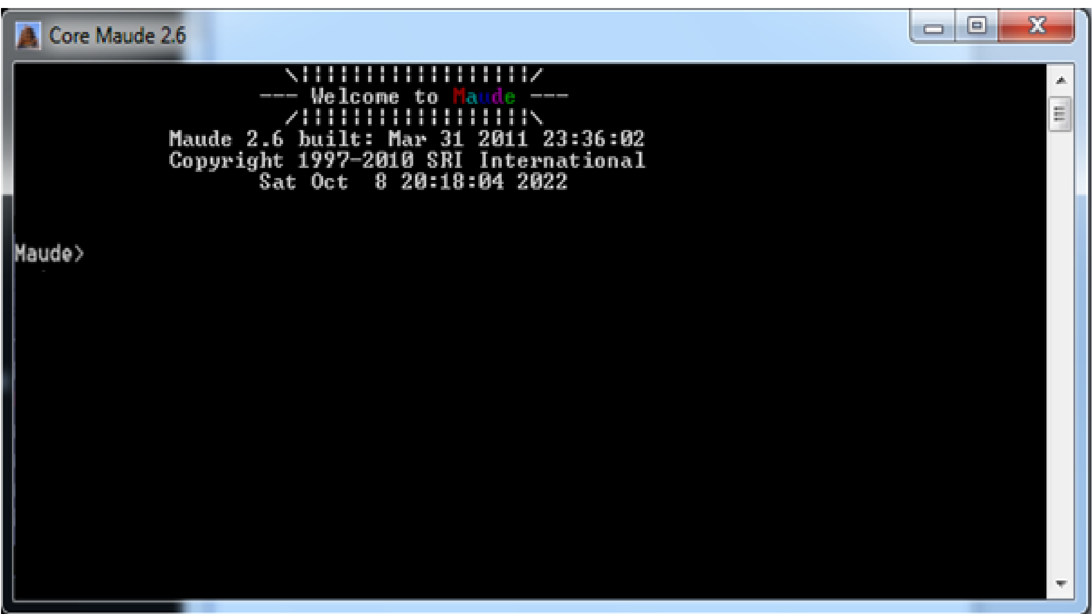


Figure 5.1:Execution of Maude

The Maude system is now ready to accept commands or modules. The user interacts with the system during a session by entering his request at the 'Maude prompt'. For example, if we want to leave Maude

Maude > quit

' ***q*** ' can be an abbreviation for the command ' ***quit*** '. You can also enter modules and use other commands. It's not really convenient to enter a module in the 'prompt', rather, the user can write one or more modules in a text file and have the file enter the system by calling the ' ***in*** ' command or the ' ***load*** ' command [113]. Assuming **my-*nat.maude file*** containing the NAT module described as follows:

```
fmod SIMPLE-NAT is
sort Nat .
opzero : -> Nat .
op s_: Nat -> Nat .
op _+_ : Nat Nat -> Nat .
vars NM: Nat .
eq zero + N = N .
eq s N + M = s (N + M ) .
endfm
```

We can introduce it to the system by doing the following command:

Maude> in my-nat.maude

After entering the NAT module, we can for example, reduce the term s szero + s sszero (which corresponds to 2 + 3 in the Peano notation) in such a module. Running and committing the ' reduce ' command returns a result like this:

Maude > reduce in NAT: s s zero + s sszero.

reduce in NAT: s s zero + s sszero .

rewrites: 3 in 0ms cpu (0ms real) (~ rewrites/second)

result Nat: s ssss zero

It is not necessary to give the name of the module explicitly in which we reduce the term. All the commands that module needs refer to the current module by default, otherwise you must explicitly give the name of the module. The current module is usually the last introduced or used, or we can use the **reduce command**

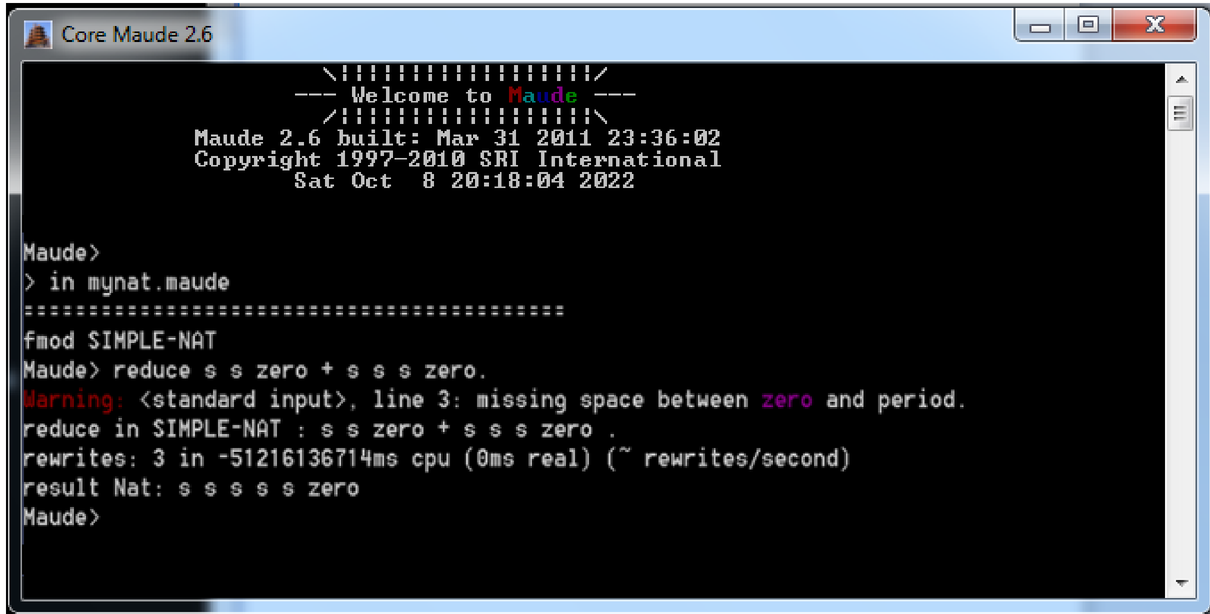


Figure 5.2:Running the reduce command

5.12 CONCLUSION

In this chapter, we have presented the basic concepts of rewriting logic and the Maude system. Furthermore, we emphasized verification with the Maude system and temporal logic used in our work. In the following chapter, we tried to present our integrated mobile-UML/Maude system approach with propose a graph grammar to automatically transform a mobile UML digraph into a formal specification in Maude language for verification reasons.

Chapter 6

An integrated Mobile-UML/Maude system approach (Contribution)

Chapter 6: An integrated Mobile-UML/Maude system approach

6.1 INTRODUCTION

In the third chapter of this thesis, we exposed a comparative study on the available extensions of UML diagrams that support Mobility. On the other hand, the transformation of all the extended diagrams proposed in this study requires a very large investment in time and effort to ensure its completeness.

In this chapter, we propose a Mobile-UML/Maude system integrated approach [88]for the transformation of the M-UML statechart diagram[41],[117].In order to be able to implement, in this thesis, the process of transformation of models, we are interested in the diagram of mobile statechart because it makes it possible to describe the behavior of an object of the modeled system. Therefore, this diagram will capture the mobility of an agent. Despite the convenience of modeling and understanding UML models, verifying such models is quite a difficult task because of the semi-formal semantics of UML. In this context, several research works have focused on the verification and validation of UML models [118], [105], [119], etc.

Our contribution, based on graph transformation principles presented in chapter four, proposes a graph grammar to transform the diagrams of mobile statechart towards a formal specification in Maude. We started by using this graph grammar, transforming the source models to the target models, and obtaining an initial Maude code. Then by exploiting a simple method by using Boolean predicates to redefine our system and make known the states and mobile transition among the states and the simple or non-mobile transition, We add mobility attributes to the states and transitions with a Boolean predicate which finally allows us to check if a state is a state mobile or not and if a transition also a mobile transition or not, thus, Global knowledge of the image of our system

6.2 RELATED WORKS

In the literature, much research work has been done concerning the integration of different UML diagrams and formal methods, such as Petri nets [120], [121], [122], Colored Petri nets (CPN) [123], Object-Z [124], the B method [125], communicating sequential processes (CSPs) [126], and Maude [127].

Many works in the literature attempt to formalize the semantics of UML state diagrams, which is very difficult to review. That is why we consider only those most related to our work. The authors in [93] categorised and compared 26 approaches to state machine semantics formalization. In [129], a temporal logic-based formalisation gives formal semantics of statechart diagrams. In [130] and [131], CSP is used to define a formal semantics of statechart diagrams. Other formalizations of statechart diagrams use PROMELA [132], Esterel [133], and Petri nets [134]. Finally, a bottom-up approach based on the π-calculus formalism has been proposed in [135].

Regarding the formalization of M-UML statechart diagrams with a formal method, we found three major works that seem related to our contribution. First, the authors in [85] proposed a formalization of mobile state machines using the TLA logic to define the notion of refinements. In [137], an approach for transforming mobile UML statechart diagrams into nested nets using meta-modeling and graph grammar is described for modeling and analysis purposes. Finally, in [87], the authors proposed an approach for formalising M-UML statechart diagrams using π-calculus for modeling and analysis.

Although there is already a formal semantics of M-UML statechart diagrams in [137], the target semantic domain chosen is very limited in semantics and tools. In [135], the authors propose a formalization of UML statechart diagrams without taking mobility into account, apart from the fictitious strategy used in the formalization, which often considers that fictitious (non-identified) processes that perform certain tasks at the source. By examining this approach, we quickly detect that it is impossible to analyze and verify a π-calculus specification generated by this formalization; the informal definition of the semantic mapping in general of all the previous methods, especially in [135, 137], makes them insufficient to define the translation fully. The work presented in [85] proposes a formalization of M-UML state machines using MTLA, which extends Lamport's temporal logic of actions (TLA) with spatial modalities. The approach can be classified as those that use statechart diagrams for refinement purposes in addition to lessening the poverty of the target model (MTLA formalism) in theory and with tools, from another point of view, about the use of the same tool that we used of the rewriting logic to generate formal specifications based on software concepts as MAS, we can mention the work of [151] that propose an algorithm allowing the automatic generation of Maude specifications from Petri nets models to help designers to effectively obtain the rewriting logic based specification of their multi-agent systems and then facilitate their analysis.

We note that all the previous contributions did not consider the fact that the user has not mastered these formal languages and does not specialize in them in most cases; most of them do not even provide automation, which hampers their use substantially. Thus, on the contrary, we propose a simple method for directly mapping mobile statechart diagrams into a Maude specification. The Maude tool is chosen because it offers us the capabilities needed to achieve our purposes.

6.3 UML MOBILE

6.3.1 Description of the mobile statechart diagram

The UML statechart diagram consists of states and the relationships that connect their states. This diagram describes how objects work in terms of changing states, knowing that an object must be in a state at a given time.

The ***mobile state*** is a state in which an actor or an object is in a platform different from that of its origin. Graphically, the ***mobile state is represented*** by adding a square with the symbol ' **M** ' in the top left end of the standard UML state.

A ***transition*** is a unidirectional line connecting two states (the source state and the target state). A ***mobile transition is*** a transition that connects two agents or two objects that are on different platforms. A ***non-mobile transition*** can connect two ***mobile states,*** and a ***mobile transition*** connects two ***mobile states*** or a ***mobile state*** to a ***non-mobile state*** and vice versa.

Graphically, a ***mobile transition*** is represented as a standard transition by adding a small square with the symbol ' **M** ' in the upper left end of this transition. Additionally, if an agent arrives in a state, a small dotted square with the symbol ' **M** ' is added to the upper left end of it to indicate the current state. In the same way, if an agent arrives in a state and performs a remote interaction with another agent, we add a small square bearing the symbol ' **R** ' in the upper left end of this transition. Finally, a mobile transition stereotyped ***"agentreturn»,*** corresponds to the return of an agent to its base platform, with a change of state (cf. **Figure 6.2**).

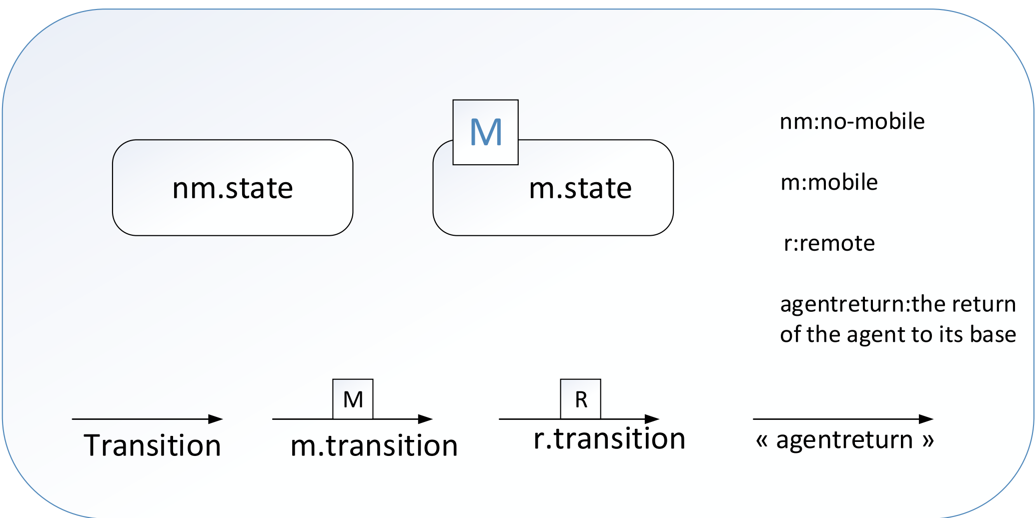


Figure 6.1:Mobile transition state graph concept

6.3.2 Meta-model of the mobile statechart diagram

Our meta-model is composed of 3 classes (one meta-class of state and two sub-class simple state and mobile state and we note that the meta-class state are omitted in transformation) and 9 transitions **.**

- **The State Class:** This class generally represents a state in the mobile transition state diagram. It has a "name**"** attribute of **String** type to designate the report's name. This class is considered the parent class of the two following classes: the ***Simple State class*** and the ***Mobile State class***
- **The Mobile State Class:** this class represents the mobile state. It inherits the attributes of the ***State class,*** and it is graphically represented by a white rectangle which bears at its upper left end a small square marked by ' **M** '. If this state is the current state, it will be represented by a white rectangle which carries a small dotted square marked by ' M 'at its upper left end, as shown in **Figure 6.1**. This class is connected to the ***Simple State class*** by the association. ***TransitionMobileMS*** in the case where a mobile transition exists between this state and a simple state; it is connected by the association ***TransitionAGR*** if there is a mobile transition which models the return of the mobile agent of the platform (**"agentreturn" transition**). These associations have,like all the associations of the metamodel, the attributes **SPF, DPF , Events and Activities.** They connect a single instance of the ***Mobile State class*** to a single instance of the ***Simple State class.*** They are graphically represented by a black arrow carrying all the attributes by adding a small square marked with " **M**" to indicate that these transitions are mobiles.

Additionally, we add an **"agentreturn" stereotype** for the ***TransitionAGR relation*** to model the return of the mobile agent to its bass platform.

- The relation ***Mobile State*** class is linked to itself by three relations: **-** The ***TransitionMM relation*** models the simple transition between two mobile states, and it is graphically represented by a black arrow carrying all the attributes. This relation links a single instance of the ***Mobile State class*** to a single instance of itself.
- The second relation ***TransitionMobile*** MM models the mobile transition between two mobile states. It is represented graphically by a black arrow carrying all the attributes, adding a small square marked by " **M**" to indicate that the transition is mobile. This association links a single instance of the ***Mobile State class*** to a single instance of itself.
- The last relation ***TransitionADistanceMM*** models the distance transition between two mobils states. It is graphically represented by a black arrow that carries all the attributes, adding a small square marked with **"R"** to indicate the distance transition. This relation links a single instance of the ***Mobile State class*** to a single instance of itself.

- **Simple State class:** This class represents the simple state (the state where the mobile agent is in its base platform) of the mobile agent. It inherits attributes and relationships from the ***State class.*** It is represented graphically by a white rectangle (cf. **Figure 6.1**), moreover, a small dotted square marked by ' **M** ' is added to the top left end of the rectangle to designate the current state (the platform which owns the mobile agent at the current time). This class is connected to the ***Mobile State class*** by the association ***TransitionMobileSM*** which also has SPF, DPF for represents Events and Activities attribut**es.** This association connects a single instance of the ***Simple State class*** to a single instance of the ***Mobile State class;*** it is represented graphically like all the mobile transitions in the metamodel. *Simple State* class is connected with itself by two relations:
  - The ***TransitionSS association*** models the simple transition between two simple states, it is represented graphically by a black arrow carrying all the attributes. This association links a single instance of the ***Simple State class*** to a single instance of itself.
  - The second ***TransitionADistanceSS association,*** models the distance transition between two simple states; it is graphically represented by a black arrow carrying all the attributes by adding a small square marked with **"R"** to show

that it is a transition from a distance. This association links a single instance of the ***Simple State class*** to a single instance of itself.

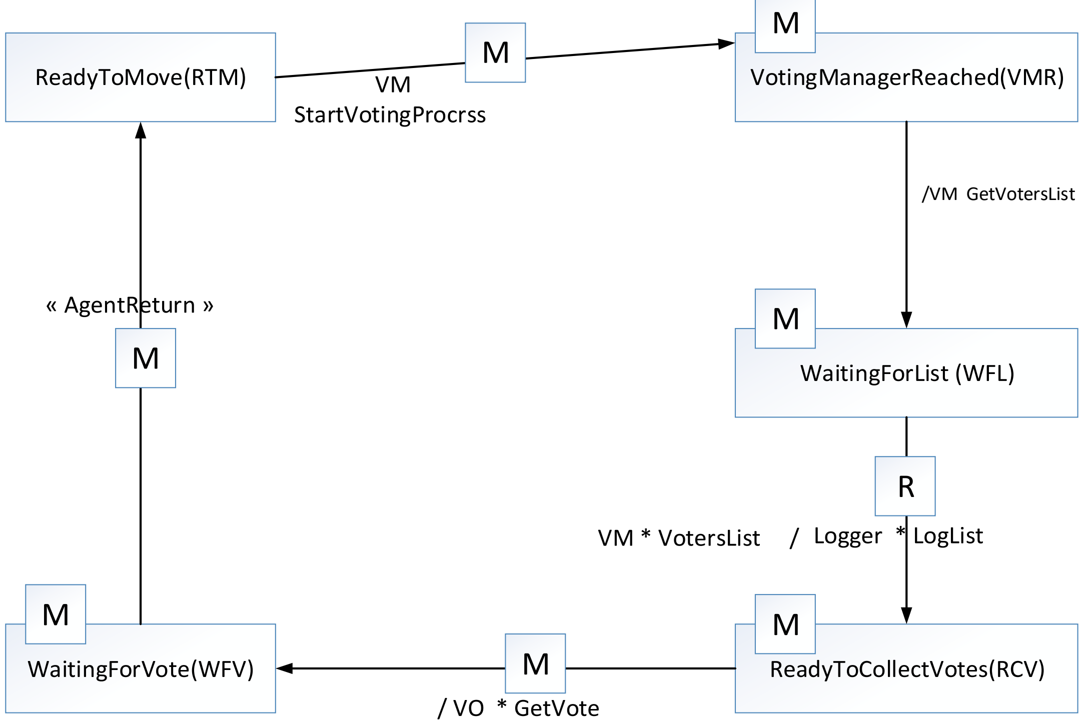


Figure 6.2 : Mobile Statechart diagram modeling

6.4 THE MAUDE SHORT OVERVIEW

We give an overview of running a Maude session on Windows to put the reader in context after we talked about it at length and its most important commands in the previous chapter.

The Maude FW interpreter (Maude for Windows 2.6) is free. It can be downloaded from the MOMENT PROJECT team website (www.moment.dsic.upv.es). Maude has a standard library of predefined modules, which, by default, are loaded by the system automatically at the start of each session. Furthermore, each of its predefined modules can be imported by another user-defined module.

These predefined modules are specified in a file named prelude.maude, this file is in the same directory as the executable file of Maude. Among these modules, we can find BOOL, STRING, NAT. These modules declare sorts and operations to handle Boolean values, character strings and natural numbers.

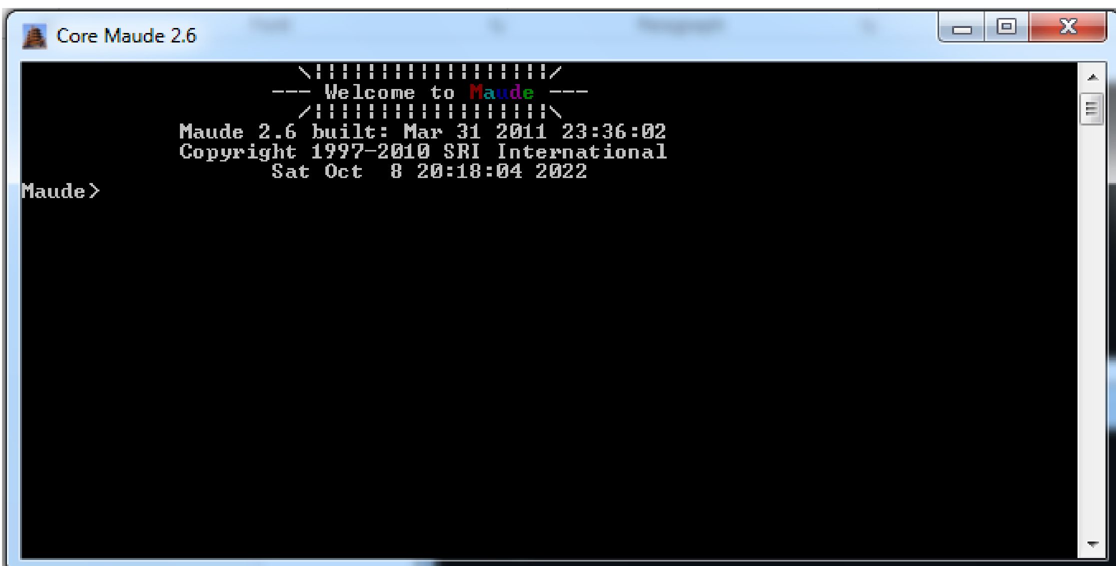


Figure 6.3 :Execution of Maude

Figure 6.4 represents an open session of Maude 2.6; during this Maude session, the user interacts with the environment by entering modules, and Maude commands directly in a prompt. But it is very convenient to use a graphical environment with an advanced text editor to create and run Maude programs. The following list (**Table 6.1**) presents some Maude commands. Indeed, Maude has several other commands, which are described in detail in [115].

Table 6.1:Maude commands example

The command	**Description**
red {in mod: } T .	The command asks to reduce (reduce) the term "T" using the equations in the "mod" module.
rew {in mod: } T .	This command rewrites the "T" term using the "mod" module rewrite equations and rules until no rule can be applied.
rew [n] {in mod: } T .	Rewrite the term "T" for a number n of rewrites. (The number of equational reductions does not influence n).
select {mod} .	select the "mod" module so that it is the current module.
show module {mod} .	view the specified module
show spells {mod} .	Visualize sorts and sub-sorts.
show ops {mod} .	view the list of operations.
show eqs {mod} .	displays the list of equations

6.5 THE PROPOSED GRAPH GRAMMAR

Our grammar (MobileStatechartVersMaudespcgrammar) comprises eleven rules divided into two categories. one for the states and the second for the transitions.

Compared to a classic UML diagram, we find that the mobile UML must include a duplication of state and transition types to be able to support the new mobility paradigm that covers it; this is why we find the mobile state next to the simple non-mobile state and the mobile transition next to the simple transition**,** in addition to the remote transition to cover a feature to the mobile agent, it is the remote intervention.

6.5.1 Rules for transforming all states into Maude rewriting rule

This category includes two rules that aim to transform the states existing in the source model.

Rule 1: Transformation of a simple state (non-mobile)

Name: EtatSToMaude

Role: This first rule allows us to convert a simple state of the statchart diagram to its corresponding Maude specification (cf. **Figure 6.4**). It is a simple abbreviation of a non-mobile state which, when used in the Maude system, represents a unit of grammar for this formal language.

M-UML Statechart Diagram Control Structure	Equivalent	Corresponding Maude Rewriting Rules
State	:=	nm.state

Figure 6.4: Simple state transformation

Rule 2:Transformation of a mobile state

Name: EtatMToMaude

Role: This second rule makes it possible to convert a mobile state from the statechart diagram to its corresponding Maude specification (cf. **Figure 6.5**). It is a simple abbreviation of a mobile state, the essential feature in the mobile statechart diagram. When used in the Maude system, it also represents a unit of grammar for this formal language.

M-UML Statechart Diagram Control Structure	Equivalent	Corresponding Maude Rewriting Rules
M State	:=	m.state

Figure 6.5: Mobile state transformation

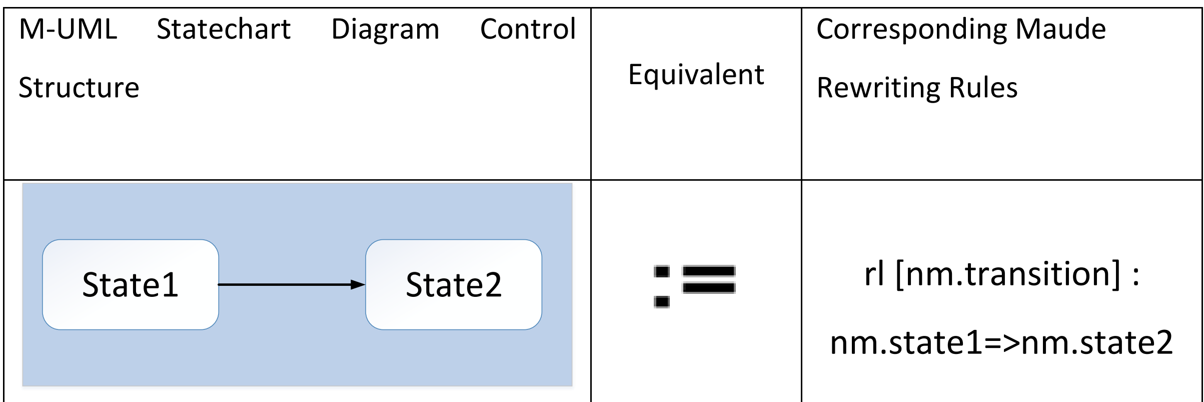
6.5.2 Rules for transforming all transitions into Maude rewriting rule

This category includes nine rules that aim to convert transitions from the source model to transitions in the destination model.

Rule 3: Transformation of a simple transition between two simple states

Name: EtatSToEtatSRMD

Role: This rule converts a simple transition between two simple states from the source diagram (Mobile statechart diagram) to a line code corresponding to its corresponding Maude specification (cf. **Figure 6.6**).



M-UML Statechart Diagram Control Structure	Equivalent	Corresponding Maude Rewriting Rules
State1 → State2	:=	rl [nm.transition] : nm.state1=>nm.state2

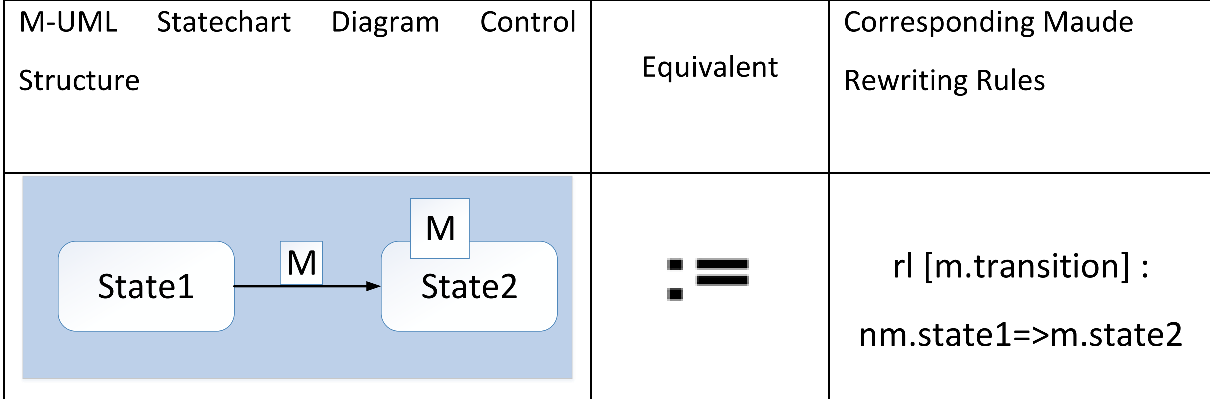
Figure 6.6: Simple transition transformation between two simple states

This transition borrows the name of the source diagram transition, and there is no mobility in this case because the agent does not move from one platform to another (the two states are on the same platform). Until now, the agent could only be in one state at a time.

Rule 4: Transformation of a mobile transition between a simple state and a mobile state

Name: EtatSToEtatMRMD

• **Role:** This rule makes it possible to convert a mobile transition between a simple state and a mobile state of the source diagram (Mobile statechart diagram) to its online correspondent Maude specification. (cf. **Figure 6.7**).



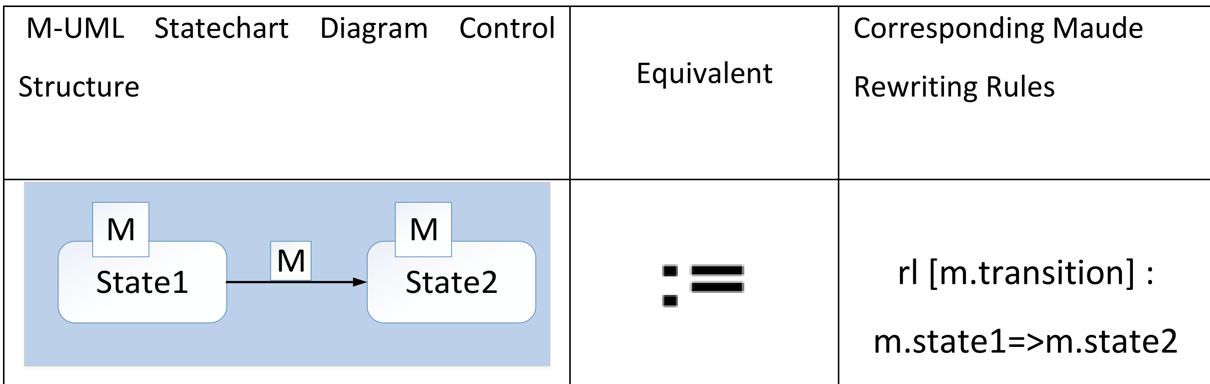
M-UML Statechart Diagram Control Structure	Equivalent	Corresponding Maude Rewriting Rules
State1 —M→ M State2	:=	rl [m.transition] : nm.state1=>m.state2

Figure 6.7: Mobile transition transformation between a simple state and a Mobile state

Rule 5: Transformation of a mobile transition between two mobile states

Name: EtatMToEtatM

Role: This rule makes it possible to convert a mobile transition between two mobile states of the source diagram (mobile statechart diagram) towards its corresponding in Maude specification. This transition takes as its name from the transition of the source diagram, the concatenation between the following attributes: SPF, DPF (cf. **Figure 6.8**).



M-UML Statechart Diagram Control Structure	Equivalent	Corresponding Maude Rewriting Rules
M State1 —M→ M State2	:=	rl [m.transition] : m.state1=>m.state2

Figure 6.8: Mobile transition transformation between two mobile states

Rule6: Transformation of a mobile transition between a mobile state and a simple state

Name: EtatMToEtatSRMD

Role: This rule makes it possible to convert a mobile transition between a mobile state and a simple state of the source diagram (Mobile statechart diagram) towards the rule corresponding to it in line Maude specification. This rule takes as its name, from the transition of the source diagram, the concatenation between the following attributes: SPF, DPF. (cf. **Figure 6.9**).

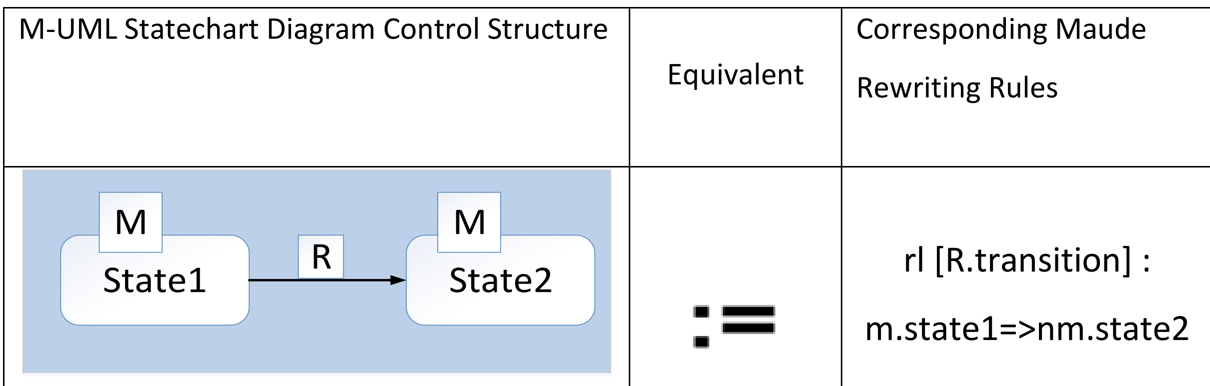
M-UML Statechart Diagram Control Structure	Equivalent	Corresponding Maude Rewriting Rules
M State1 M State2	:=	rl [m.transition] : m.state1=>nm.state2

Figure 6.9 Mobile transition transformation between a mobile state and a simple state

Rule 7: Transformation of a remote transition between two mobile states

Name: EtatMToEtatMRRMD

Role: This rule makes it possible to convert a remote transition between two mobile states of the source diagram (Mobile statechart diagram) towards its corresponding in Maude specification. This transition takes its name from the transition of the source diagram, the concatenation between the attributes: SPF, DPF. (cf. **Figure 6.10**).



M-UML Statechart Diagram Control Structure	Equivalent	Corresponding Maude Rewriting Rules
M State1 R M State2	:=	rl [R.transition] : m.state1=>nm.state2

Figure 6.10: Remote transition transformation between two mobile states

Rule 8: Transformation of a simple transition between two mobile states

Name: EtatMToEtatMRMD

Role: This rule makes it possible to convert a simple transition between two mobile states from the source diagram (Mobile statechart diagram) to its corresponding Maude specification (cf. **Figure 6.11**). This transition takes as appeared in its name, from the source diagram transition, in this case, the agent does not move from one platform to another (both states are in the same platform). As a result, the agent can only be in one state simultaneously.

M-UML Statechart Diagram Control Structure	Equivalent	Corresponding Maude Rewriting Rules
M State1 → M State2	:=	rl [nm.transition] : m.state1=>nm.state2

Figure 6.11: Simple transition transformation between two mobiles states

Rule 9: Transformation of an "agent return" simple transition between a mobile state and a simple state

Name: EtatMToEtatSAG

Role: This rule makes it possible to convert a stereotyped simple transition "agentReturn" between a mobile state and a simple from the source diagram (mobile statechart diagram) towards its corresponding Maude specification. This transition takes as its name from the transition of the source diagram, the concatenation between the following attributes: SPF, DPF. (cf. **Figure 6.12**).

M-UML Statechart Diagram Control Structure	Equivalent	Corresponding Maude Rewriting Rules
M State1 « agentreturn » → State2	:=	rl [agentreturn] : m.state1=>nm.state2

Figure 6.12: Transformation of an "agent return" simple transition between a mobile state and a simple state

Rule10:Transformation of a remote transition between two simple states

Name: EtatSToEtatSRR

Role: This rule makes it possible to convert a remote transition between two simple states from the source diagram (Mobile statechart diagram) to its equivalent in Maude specification. This transition takes as its name from the transition of the source diagram, the concatenation between the following attributes: SPF, DPF. (cf. **Figure 6.13**).

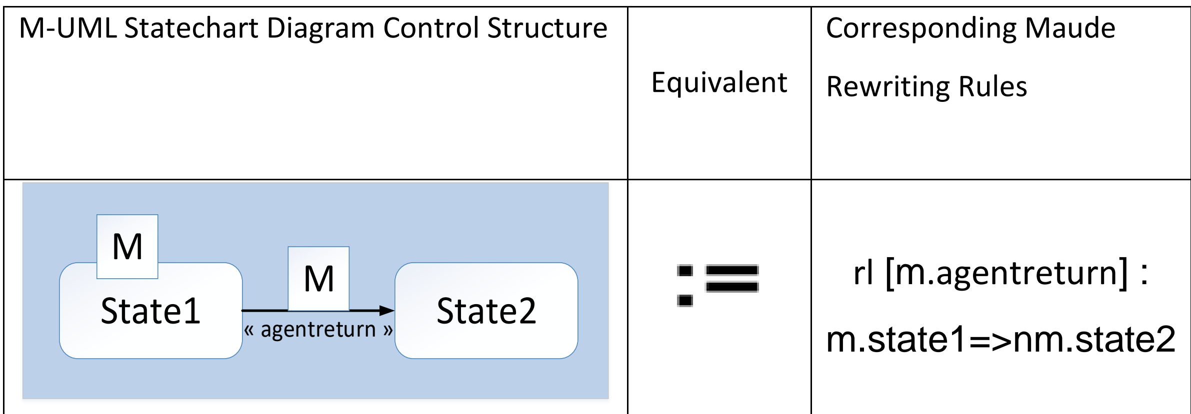
M-UML Statechart Diagram Control Structure	Equivalent	Corresponding Maude Rewriting Rules
State1 —R→ State2	:=	rl [R.agentreturn] : m.state1=>nm.state2

Figure 6.13: Remote transition transformation between two simple states

Rule 11: Transformation of an "agent return" mobile transition between a mobile state and a simple state

Name: EtatMToEtatSAG

Role: This rule makes it possible to convert a stereotyped mobile transition "agentReturn" between a mobile state and a from the source diagram (Mobile statechart diagram) to its corresponding Maude specification. This transition takes its name from the transition of the source diagram, the concatenation between the following attributes: SPF, DPF. (cf. **Figure 6.14**).



M-UML Statechart Diagram Control Structure	Equivalent	Corresponding Maude Rewriting Rules
M State1 —M « agentreturn »→ State2	:=	rl [m.agentreturn] : m.state1=>nm.state2

Figure 6.14: Transformation of an "agent return" mobile transition between a mobile state and a simple state

Table 6.2: Representation of control structures in Maude.

Rule number	M-UML Statechart Diagram Control Structures	Corresponding Maude Rewriting Rules	Explanation
01	State	nm.state	A transaction of simple state on Maude rewriting rules
02	M State	m.state	A transaction of mobile state on Maude rewriting rules
03	State1 → State2	rl[nm.transition]: nm.state1 => nm.state2	a non-mobile transition linking a non-mobile state to a non-mobile state
04	State1 M→ M State2	rl [m.transition]: nm.state1 => m.state2	a mobile transition linking a non-mobile state to a mobile state
05	M State1 M→ M State2	rl [m.transition]: m.state1 => m.state2	a mobile transition linking a mobile state to a non-mobile state
06	M State1 M→ State2	rl [m.transition]: m.state1 => nmstate2	a mobile transition linking a mobile state to a non-mobile state
07	M State1 R→ M State2	rl [R.transition]: m.state1 => m.state2	a remote transition linking two mobile states
08	M State1 → M State2	rl [nm.transition]: m.state1 => m.state2	a non-mobile transition linking two mobile states
09	M State1 « agentreturn »→ State2	rl [agentreturn]: m.state1 => nm.state2	In this case state2 = the base platform for this agent
10	State1 R→ State2	rl [R.transition]: nm.state1 => nm.state2	a remote transition linking two non-mobile states
11	M State1 M « agentreturn »→ State2	rl [m.agentreturn]: m.state1 => nm.state2	Agent return come back to the initial state with a mobile transition

Finally, we can summarize the graph grammar proposed in our M-UML Statechart to Maude specification transformation approach in the **Table 6.2** above.

We now have all the components needed to construct our Maude specification corresponding to the Mobile Statechart Diagram. As a result, we must collect as a process the entire system composed of a restricted composition of several interactive processes that evolve dynamically by-passing messages between them.

6.6 THE PROPOSED APPROACH

6.6.1 Formalization of MSD

The behavior of a software system based on mobile agents is given by a set of communicating mobile state-transition diagrams MSD 1 ... MSD k. The basic idea in our formalization is to reduce the work to a single MSD and generate its corresponding code**.** The latter consists of a well-defined program portion ready to check and execute it. This Maude code must inevitably include the aspect of mobility of states and transitions, so it must provide a reconfiguration of the Maude system each time an agent passes from one state to another. This allows for capturing in a clear way the mobility function, which is the objective of this work.

6.6.2 Translation of M-UML diagrams to a formal specification Maude

In the previous chapter, the concepts relevant to formal specifications were introduced obviously, the advantages of such a technique are numerous and offer interesting possibilities when they are used to develop a system. Also, in the same chapter, the Maude environment was introduced. With a solid mathematical basis, this language is a perfect candidate to specify a system formally. Besides the fact that it was designed with this objective, Maude offers the possibility of simulating the developed system, as will be discussed later in this section.

6.6.3 The approach proposed translation of M-UML diagrams to Maude

This section presents our approach to translating one of the most important M-UML diagrams into a formal Maude notation. The diagram used is the statechart diagram. The technique consists of systematically deriving a formal Maude description from this type of diagram analysis.

Our approach presented here, which was mainly inspired by the works presented in [85, 140] and [137], wich concerns only standard UML diagrams without regard to the mobility property; As a result, we present the Maude specification of M-UML statechart diagrams. Furthermore, we decompose the M-UML statechart into its essential elements to better define

the mapping to the corresponding Maude specification. More precisely, the proposed approach entails converting an M-UML statechart diagram into a Maude specification. The translation process is divided into three significant steps, summarized in the **Table 6.3** below. Additionally, **Figure 6.15** visually summarizes these same steps.

The first step of the translation process begins with automatically translating the M-UML diagram to the first Maude specification. Then, we use the control structures (graph grammar) mentioned in de Maude, and we get the initial Maude code; we can say that, in reality, this step is the usual analysis step of the development process of a software system. It consists of the description of a system using the devices of UML Mobile. Then, of course, we try to cover the different artifacts of M-UML [41], such as mobile states, mobile transitions, and remote transitions. The proposed approach first considers the statechart diagram and suggests extending it to other diagrams.

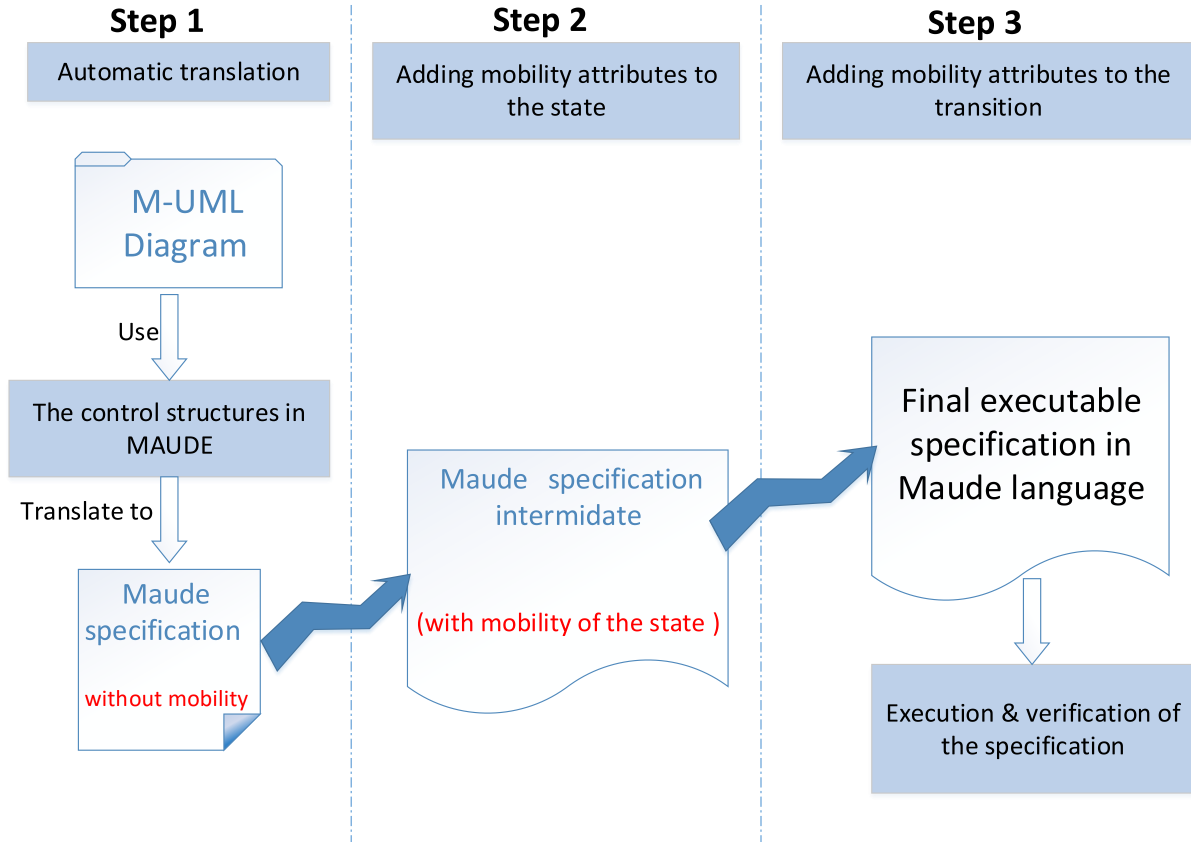


Figure 6.75: Translation Process Overview

The second step of the translation process comes after obtaining the initial code of the specification in Maude; we continue to enrich it to cover the aspect of mobility by adding mobility attributes to states with a Boolean predicate which enables us to perform some tests such as checking whether the condition is mobile or not. This step aims to distinguish between

the types of states used. An inter-diagram check is performed on all the structure of the diagram used to ensure the correctness of all the constituent elements of the statechart diagram.

The third step of the translation process is the last step which consists in adding mobility attributes also to the transitions (the second component, which represents mobility). Finally, we obtain a complete Maude specification defining our M-UML state diagram.

Table 6.3 The three steps of the translation Process

	Brief description
Step 1	It can be considered an automatic translation from the M-UML state diagram to the Maude specification since we directly use the control structures shown in Table 6.2
Step 2	This stage consists in adding mobility feature to states
Step 3	This stage consists in adding mobility feature to transitions, and we finally obtained a complete Maude specification representing our M-UML statechart diagram.

We focused on the main elements that affect the mobile behavior of the object/agent in these diagrams in this study. In the future, we intend to broaden our approach to include all notational aspects of M-UML statechart diagrams, we will also try to work towards finding formal specifications for the structural diagrams, such as the M-UML class diagram, and then search for consistency between the structural and behavioral perspectives to achieve maximum systems guarantee, and it is the point that we will address in details in the following item.

We can also highlight the importance of our proposed approach by comparing it in the **Table 6.4** below with two of the most close and similar approaches, trying to clear the strengths and weaknesses of each approach.

Table 6.4 Comparative of our approach with the closest similar methods

Autours	Specification tool use	Pros	Cons
Kezai et all [57]	Integred M-UML /Maude	• Affordable Complexity and better than the two others tools . • Equipped with tools for verification. • Support for simulation and reduction	• Meduim Complex formal language that requires significant learning time. • Requires high technical skills. • The complexity make verification process more difficult and more expensive
Bahri et all [57]	Integred M-UML /Petri net	• Graphical modelisation • Describe in formal and precise manner • Equipped with tools for verification	• Limits in expressiveness • Requires high technical skills
Belghiat et all [57]	Integred M-UML /π-calculus	• Describe in formal and precise manner . • High expressiveness • Mobile Process Modeling Support	• Complex formal language that requires significant learning time . • Requires high technical skills • The complexity make verification process more difficult and more expensive. • Not equipped with tools for verification

In summary, the 3 approaches allow to describe distributed systems in a formal and precise way with a high expressiveness, which allows to model in a detailed way these systems, including their behavior, their interaction and their structure. Maude system and π-calculus share that they are formal algebraic languages on the other hand petri net is a language with graphical modeling, while Maude is less complex and contains verification tools, π-calculus and more complex and does not contain any verification tools but it does it with external tools

6.7 M-UML AND INCONSISTENCIES

Using these different diagrams, M-UML, like UML, offers the possibility of describing several "views" of the same system. However, these different views may conflict as they may present inconsistencies [141], and here, for the mobility paradigm, it is even worse. Moreover, experience shows that many modeling faults are perceptible through detecting inconsistencies [142, 143, 144, 145]. Inconsistency is the violation of a property associated with the UML language, which must be respected by any UML model [146]. Indeed, the coherence can concern a single diagram (we speak of intra-diagram coherence) or several diagrams (we talk of inter-diagram coherence). Among the intra-diagram inconsistencies that must be taken care of and dealt with, we can mention the existence of a transition whose source state is a final state in a statechart diagram and for other types of diagrams, the presence of cycles in an inheritance graph for a class diagram, and the violation of the message sending order of a synchronization point in a communication diagram (for example, the message number blocked is less than the number of the blocking message). Using the various diagrams, the representation of complex systems can generate inter-diagram inconsistencies. For example, one can quote, among others, a call event of a statechart diagram which do not appear as a method in the corresponding class. Also, a message sent (procedure call) in a communication diagram is not listed as a method with proper visibility in the class of the recipient object. Furthermore, the final state of a finite system does not consist of the different final states of the other objects involved in the communication diagram.

Reviewing and discovering these contradictions will be crucial to obtaining safe systems free from faults and errors. For this, finding a formal specification of diagrams must be inevitably followed by an essential step, which consists of checking the consistency of these diagrams. Then, through their formal representation to return to make the necessary corrections in the semi-formal prototype, such as UML or M-UML.

6.8 DISCUSSION OF THE PROPOSED APPROACH

We tried in this approach to provide an understandable, readable, and simple formalization by immediately linking the elements of mobile state-transition diagrams to their corresponding Maude expressions.

We have focused here on the essential elements of mobile statechart diagrams to focus on mobility (and concurrency) which is our main concern, and for simplification purposes. We, therefore, did not consider other elements that we judged that do not affect the mobile behavior of these diagrams, such as, for example, the historical and deep historical pseudo-states, the junctions and the decisions. Still, our formalization can be extended to all elements of mobile state-transition diagrams by a structural induction/recursion paradigm thanks to the formal definition of the transformation.

The mobility provided by the Maude specification allows the trivial modeling of the agent's movements and the type of each state between simple or mobile state in real-time. In our approach, the scope extrusion mechanism in Maude is used to model the mobility depicted in the diagrams.

It is not useful to discuss how to prove that the semantics are preserved after the transformation of the mobile statechart diagrams into Maude since the former is a notation and the latter is a formal language. Mathematically speaking, there is no way to prove the correctness of our semantics. However, we can verify the soundness and completeness of the transformation by model checking using a verification tool of the same language full Maude [45]. It can be shown that the conversion is sound, i.e. If the system Maude model checking tool says that an error (the deadlocks, for example) exists in the input, then it should be the case that the error (the deadlocks, for example) exists in the specification MSD. We can also show that the translation is complete, i.e., if the specification has an error (deadlock, for example), then the model checking tool will be sure to find it. In fact, we experimented with our transformation with several real-world examples in both directions (solidity and completeness) and got good results.

6.9 POTENTIAL LIMITS

The method proposed in this thesis has some limitations. First, the process of adding mobility attributes to states and transitions (Step 2 and Step 3) is done manually at the moment. However, these two steps can certainly one day be automated.

In addition, the generation of the Maude source code when applying the M-UML diagram translation to a formal Maude notation is intended to be automatic. The contribution of this dissertation is that it is considered the first experience of using rewriting logic to develop an adequate notation to represent a mobile agent-based system formally. The M-UML diagram that we will start with in this project is the state-transition diagram. However, we see many other diagrams that are very interesting to continue working with, either in the behavioral aspect, like the activity and sequence diagram or in the structural aspect, like the class diagram. From a state-transition diagram chosen to sometimes very rich structures expressing a large number of characteristics of a system, it is also very rich in its notations of states and transitions, the input and output instructions, etc.

Finally, the proposed approach considers only one M-UML statechart diagram at a time. A large system will obviously include several of these diagrams. It will then suffice to apply the same process several times, one statechart diagram at a time, to validate an entire system. The static structure of the system will already be described at this time, and this piece of Maude source code will not have to be repeated each time insofar as no correction is to be made; here it is possible to move to a higher level of verification by ensuring consistency between diagrams of the system, but this idea goes beyond the scope of this thesis.

6.10 CONCLUSION

In this chapter, we have proposed an integrated semi-formal/formal approach to specifying and verifying mobile agent-based software systems using MSD and the Maude language. The approach consists in defining a set of formal rules which define the formal semantics of the behavior of the elements of the mobile state diagrams (MSD) using the Maude language.

We start by proposing a graph grammar for the automatic transformation of mobile state diagrams into a Maude specification in three simple and clear steps in order to be able to perform automatic verifications. We used the compiler associated with this language, Core Maude, for our verification tests.

We have focused on the main elements that model the mobile behavior of agents in these diagrams.

As an extension of the approach, we plan to extend it to take into account all elements of mobile statechart diagrams. Therefore, more complex systems will be covered by the approach.

We also plan to continue our future work by completing our project. In addition, we will transform all other mobile UML diagrams, such as mobile sequence and mobile activity diagrams, because they represent different aspects of a system and provide multi-view modeling.

Another second perspective that we consider very important is to check the consistency between different diagrams which express several points of view of a mobile UML after obtaining the formal specification for each one.

The next chapter will illustrate and test the approach with a complete case study.

Chapter 7

Case study

Chapter 7: Case study

7.1 INTRODUCTION

The previous chapter presented a formal framework for translating and verifying M-UML state-transition diagrams using the Maude environment. This process, at first glance, may seem abstract and difficult to understand. To better illustrate this, a case study is presented in this chapter:

A structure will be followed to illustrate the translation process as follows:

1. Presentation of the example and the M-UML diagram associated with it;

2. Application of the process of translating M-UML diagrams into a formal

Maude description and validation of the system description using simulations:

3. Application of the process of formal verification of UML diagrams with the help of properties of the system, of desirable and undesirable nature, stated using linear temporal logic (LTL).

We will start by giving a complete case study modeled by an MSD diagram. Next, we will show how to generate our Maude specification step by step. We will then proceed to the analysis and verification of the Maude specification generated using the Code Maude system compiler.

7.2 PRESENTATION OF THE EXAMPLE

We will illustrate our approach defined in the previous chapter by applying it to a specification of a software system based on mobile agents borrowed from [41]. It models a mobile voting system VS, which consists of three interacting agents, a stationary agent VA (voting authority), a mobile agent VC (vote collector), and a stationary agent VM (vote manager). The VC is mandated by a VM; its role is to collect the votes from the polling stations. During an election period, the VA begins the voting process by sending the VC to its VM manager. The VC obtains the list of voters from the VM, then by interacting remotely with the VA agent it obtains the log of the list.

After that, he visits the polling stations in their place to collect the polling results and sends them to the VM that mandated that VC.

7.2.1 Modeling in MSD

The behavior of the system is defined through a diagram of communicating mobile statechart, one for each agent, as illustrated in **Figure** 7.1 . That shows the behavior of the VC Mobile Agent. First, the VC is in the ReadyToMove (RTM) state at its base place (P1) when it receives the StartVotingProcess event, and then it moves through a mobile transition to the VotingManagerReached (VMR) state in another seat (P2). At VMR, the VC mobile agent goes through a normal transition to the WaitForList (WFL) state. The transition involves sending a GetVotersList signal to the VM through asynchronous communication to get the voters list. In WFL, when the mobile agent receives a VotersList event, the VC interacts remotely with VA by executing the remote action LogList, which consists of a synchronous call to get the list log, and then it moves to the state ReadyToCollectVotes (RCV). In RCV, the mobile agent VC moves through a mobile transition to the WaitingForVote (WFV) state in place of the first voting station (P3). The transition has a GetVote action to get the votes. Finally, the mobile agent VC performs an « agentreturn " transition to return to its base place (P1). The transition has an action which consists in sending a Votes message to VM by asynchronous communication to give it the results of the vote of the station.

7.3 APPLICATION OF THE TRANSCRIPTION PROCESS OF M-UML DIAGRAMS

Many works try to propose a specification for mapping a UML diagram to a formal method and, more precisely, to a Maude specification [127, 52, 59]. However, the difference between UML and M-UML lies in mobility, so the modeling of mobile agent applications takes into account the following concepts: route, location, migration, remote cloning, and security.

For a successful Maude specification for an M-UML, we must inevitably take into consideration at least one of the mobility properties (the route, location, migration, remote cloning, or security).

In the literature, there is a lack of M-UML examples. The best way to successfully provide reliable case studies is to use case studies by Kassem Saleh [41] related to a mobile voting system and try to find the Maude specification for our mobile statechart diagram.

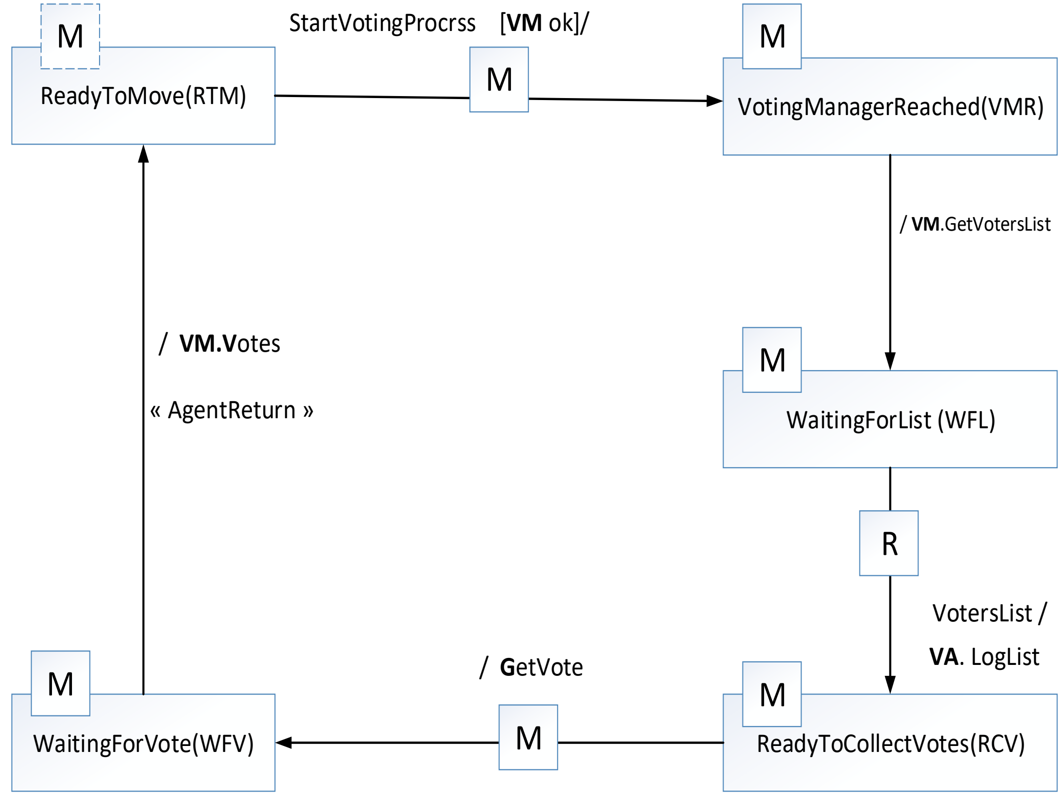


Figure 7.1:Mobile Statechart diagram of the Vote Collector (VC.) [41]

We start by finding the Maude specification of the statechart diagram for the case study: a mobile voting system, represented in **Figure 7.1.**

We use, in what follows, several examples to illustrate the defined process to support the translation of our statechart diagram into a Maude specification. To understand the initial code, see the explanation of state abbreviations in **Table 7.1** and the explanation of transition abbreviations in **Table 7.2.**

Table 7.1: State abbreviations.

Abbreviation	**Explanation**
RTM	Ready To Move
VMR	Voting Manager Reached
WFL	Waiting For List
RCV	Ready to Collect Votes
WFV	Waiting For Votes

Table 7.2:Transition abbreviations.

Abbreviation	Explanation
SVP	Start Voting Process
GVL	Get Voter List
VLL	Voter List Logger
GV	Get Vote
Agentreturn	The return of the agent to the initial platform

A rewrite theory, specified in Maude as a system module, provides an executable mathematical model of a system. We can use the Maude specification to simulate the system so specified. But we can do more. Under appropriate conditions, we can check that our mathematical model satisfies some important properties or obtain a useful counterexample showing that the property in question is violated.

To apply our approach to the proposed example, we will try to sequentially follow the steps mentioned in the previous section to obtain the desired result, which is a code similar to our initial M-UML statchart diagram.

This section demonstrates the possibility of a simpler, but beneficial model checking, namely, checking the model for constants, which can only be accomplished with the search command.

First step: Validation of the generated description of vote code (the initial code generated)

We will follow the path of building our Maude specification to reach our final specification in parallel with testing some properties to verify the integrity or malfunction of our specification of the system.

we start first with an automatic translation from the M-UML diagram to Maude specification. We use the control structures in Maude and we obtain the following Maude code shown in **Figure 7.2.**

- **Property 1:** The reachability problem is one of the most important issues in the verification of systems [150]; for this, we have taken it as the first property to be dealt with and checked: This command enables us to search for all reachable states beginning with RTM.: ***search in vote: RTM =>* s:State.***

```
mod vote is
sort State .
ops RTM  VMR  WFL  RCV WFV  : -> State [ctor] .
rl [SVP] : RTM  => VMR  .
rl [GVL] : VMR  => WFL  .
rl [VLL] : WFL  => RCV  .
rl [GV] : RCV  => WFV  .
rl [agentretturn] : WFV  => RTM  .
endm
```

Figure 7.2:Modeling vote statechart 1 in Maude

Knowledge of the reachable states allows us to build an accurate perception of the system's state and enable us to make changes to suit our needs or correct it if there are errors.

- **Property 2:** This command enables us to show the search graph (which should be like the diagram): ***show search graph.***

Displays the search graph generated by the last search [115].

- **Property 3:** This command enables us to search for states that reach RCV or WFV beginning from RTM:

search in vote: RTM =>* s:State such that (s:State == RCV or s:State == WFV) [115],.

The module vote is the main Maude code subject, and where the search takes place can be omitted;

RTM is the starting term;

s:State is the pattern that has to be reached;
– =>* means a proof consisting of none, one, or more steps,

(s:State == RCV or s:State == WFV) or the *Condition* states an optional property that has to be satisfied by the reached state; the syntactic form of the condition is the same as the one of conditions for conditional equations and memberships.

Second step: Validation of the generated description of vote2 code (the intermediate code generated)

After we applied some commands to the initial Maude code and verified its correctness, we go to the second step in our transcription process by adding mobility attributes to states with a Boolean predicate that checks if a state is a mobile state and we obtained the Maude code shown in **Figure 7.3.** .

Adding mobility attributes to states as all states with an M box in **Figure 7.1** are mobile states and it is encoded by: eq M(X) = true where X is a state, and the rest of the states are not mobile states which in turn encodes by: eq M(s) = false [owise]

In this second step, which we consider an intermediate step, we will work on adding the mobility feature for states that are characterized by this characteristic so that at the end of this process, we will be able to distinguish between mobile states and non-mobile states.

- **Property 4:** This command enables us to search for all mobile states beginning from RTM: ***search in vote2: RTM =>* s:State such that M(s:State).***
- **Property 5:** This command enables us to search for all non-mobile states beginning from RTM : ***Search in vote2: RTM =>* s:State such that not M(s:State).***

We note from the Maude code, that we used a specific property to give the mobility feature by using a boolean predicate that allows us to check if a state is a mobile state.

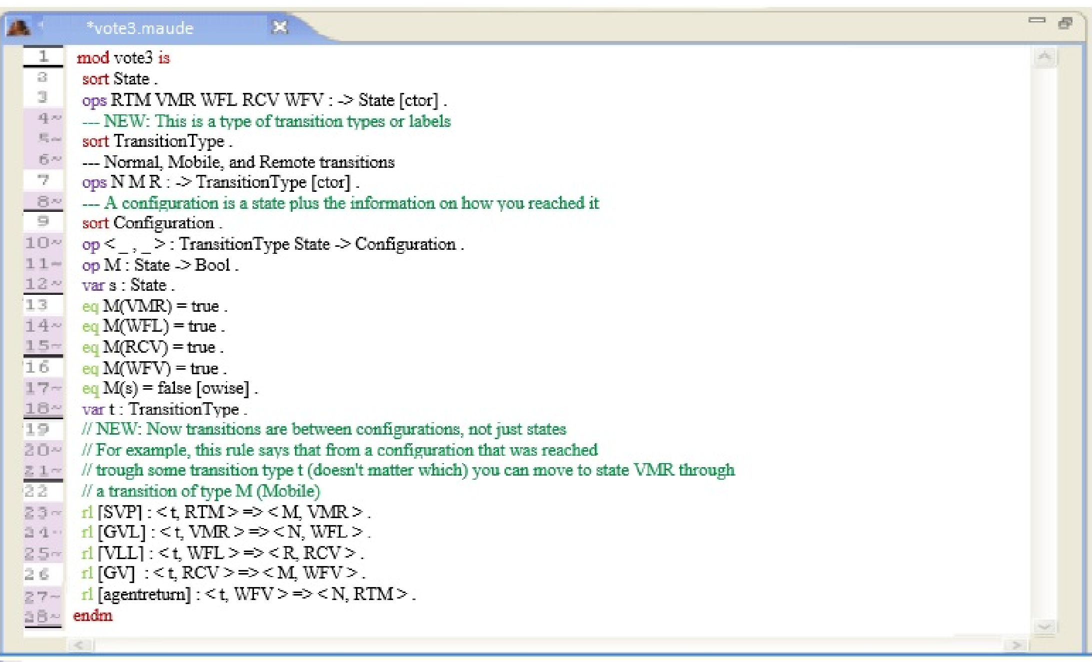
This mobile property to be checked is described by using a specific property specification logic, namely, linear temporal logic (LTL) (see [152, 153] and [115]), which allows the specification of specific properties, in our case, the mobility feature, In other, more general cases, we can touch upon the safety properties (ensuring that something bad never happens) and liveness properties (ensuring that something good eventually happens). Then, the reachability analysis tool can be used to check whether a given initial state, represented by a Maude term, fulfils a given property.

```
*vote2.maude
mod vote2 is
 sort State .
 ops RTM VMR WFL RCV WFV : -> State [ctor] .
 --- NEW: This is a boolean predicate that checks if a state is a mobile state
 op M : State -> Bool .
 var s : State .
 --- All states with an M box are mobile states
 eq M(VMR) = true .
 eq M(WFL) = true .
 eq M(RCV) = true .
 eq M(WFV) = true .
 --- The rest of the states are not mobile states
 eq M(s) = false [owise] .
 rl [SVP] : RTM => VMR .
 rl [GVL] : VMR => WFL .
 rl [VLL] : WFL => RCV .
 rl [GV] : RCV => WFV .
 rl [agentreturn] : WFV => RTM .
endm
```

Figure 7.3:The model with mobility attributes on states in Maude

Third step: Validation of the generated description of vote3 code (the final code generated)

After we made sure in the previous step that the generated code enables us to distinguish between mobile and non-motile states, we now move on to the third and final step in our transcription process by adding mobility attributes to transitions. We find that we obtain a complete Maude specification that represents our M-UML statechart diagram presented in the **Figure 7.4** .



```
*vote3.maude
mod vote3 is
 sort State .
 ops RTM VMR WFL RCV WFV : -> State [ctor] .
 --- NEW: This is a type of transition types or labels
 sort TransitionType .
 --- Normal, Mobile, and Remote transitions
 ops N M R : -> TransitionType [ctor] .
 --- A configuration is a state plus the information on how you reached it
 sort Configuration .
 op < _ , _ > : TransitionType State -> Configuration .
 op M : State -> Bool .
 var s : State .
 eq M(VMR) = true .
 eq M(WFL) = true .
 eq M(RCV) = true .
 eq M(WFV) = true .
 eq M(s) = false [owise] .
 var t : TransitionType .
 // NEW: Now transitions are between configurations, not just states
 // For example, this rule says that from a configuration that was reached
 // trough some transition type t (doesn't matter which) you can move to state VMR through
 // a transition of type M (Mobile)
 rl [SVP] : < t, RTM > => < M, VMR > .
 rl [GVL] : < t, VMR > => < N, WFL > .
 rl [VLL] : < t, WFL > => < R, RCV > .
 rl [GV]  : < t, RCV > => < M, WFV > .
 rl [agentreturn] : < t, WFV > => < N, RTM > .
endm
```

Figure 7.4: Final Maude specification for M-UML statechart diagram.

We have added mobility attributes to transitions by:

First, we declared that a new type of specification needs to be described as transition by the identifier sort as:

Sort TransitionType.

Secondly, we make the operator's declarations for this transition to three types: Normal, Mobile, and Remote transitions, and It can be expressed thus

ops N M R : -> TransitionType [ctor]

Thirdly, we declared a configuration, knowing that a configuration is a state plus the information on how you reached it, and it can be expressed thus

op< _ , _ > : TransitionType State -> Configuration .

Finally, it defines and adjusts transitions by making them between configs, not just states it, and it can be expressed thus

rl [Transition-name] : < specification-type , state-source> =>< transition-type, state-destination > .

For example, this rule says that from a configuration that was reached through some transition type t (doesn't matter which) you can move to state VMR through a transition of type M (Mobile) see **Figure 7.1.**

To check the last feature we added to our Maude code, which is a feature of transition mobility or not (Normal, Mobile, and Remote), we will suggest two queries in the following that allow us in the first query to search for all accessible configurations starting from a specific state with the customization of the nature of its transition in our example we suggested to start with the state RTM and the nature of the normal transition encoded by N to reach all possible configurations in our system.

- **Property 6:** This command enables us to search for all reachable configurations beginning from < N , RTM >: ***search in vote3: < N, RTM > =>* < t: Transition Type, s: State >***.

In our second query to ascertain the nature of the transition, we will continue to check the transition feature by checking if there are specific states that were not reached by a particular transition:

- **Property 7:** This command enables us to check if there are mobile states that are not reached through a mobile transition: ***search in vote3: < N, RTM > =>* <t: TransitionType,s: State> such that t:TransitionType =/= M and M(s:State)***.

1.1 IMPLEMENTATION OF OUR GENERATED CODE UNDER THE MAUDE SYSTEM

In this section, we will try to perform all the queries proposed in the previous section 7.3under the Maude system to get graphical results and then try to check that the Maude code obtained matches the base mobile state diagram in **Figure** 7.1 by partially checking each query whether is it correct or false.

We will try to shorten the method of launching a Core Maude 2.6, which is found in all its details in[115], Knowing that we used a Maude version on the Windows operating system directly, unlike the original on Unix .

When we try to execute this Maude module with Core Maude 2.6, we obtain the following interface result.

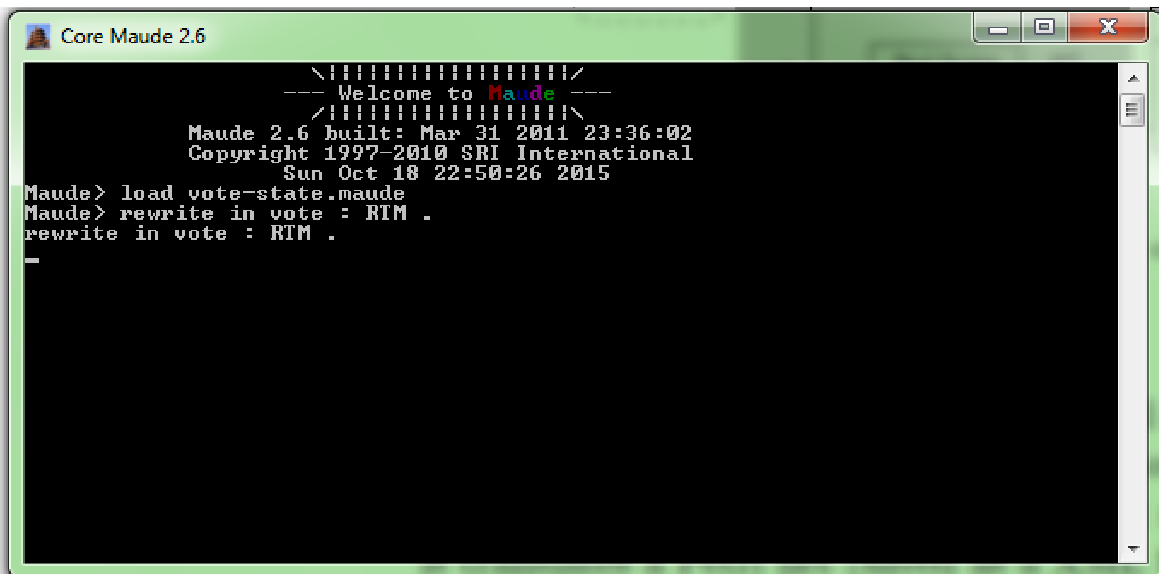


Figure 7.5: Maude's vote statechart code execution.

We note that we do not need to write our units in the prompt, so we can enter one or several units by saving them in a file and then entering the file with the in or load commands (the only difference between both commands is that the latter does not produce detailed output as modules are entered). To enter the file vote-state.maude above, we can write the following command to enter it:

Maude> load vote-state.maude

Check the property 1 :

We start by executing the first command that will allow us to search for all reachable states beginning from RTM:

Search in vote: RTM =>* s: State.

```
Core Maude 2.6
search in vote : RTM =>* s:State .

Solution 1 (state 0)
states: 1  rewrites: 0 in 4964127748ms cpu (0ms real) (0 rewrites/second)
s:State --> RTM

Solution 2 (state 1)
states: 2  rewrites: 1 in 4964127748ms cpu (1ms real) (0 rewrites/second)
s:State --> VMR

Solution 3 (state 2)
states: 3  rewrites: 2 in 4964127748ms cpu (3ms real) (0 rewrites/second)
s:State --> WFL

Solution 4 (state 3)
states: 4  rewrites: 3 in 4964127748ms cpu (6ms real) (0 rewrites/second)
s:State --> RCV

Solution 5 (state 4)
states: 5  rewrites: 4 in 4964127748ms cpu (8ms real) (0 rewrites/second)
s:State --> WFV

No more solutions.
states: 5  rewrites: 5 in 4964127748ms cpu (11ms real) (0 rewrites/second)
Maude>
```

Figure 7.6: "search command" execution for Maude vote statechart

If we look at **Figure 7.1**, which shows our mobile statechart diagram, we can confirm the path of our example (RTM => VMR => WFL => RCV => WFV); as it indicates our executable code shown in **Figure 7.6**, we confirmed that this property is true.

Check the property 2:

It is also possible to print out the current search graph generated by a search command using the following command show search graph. After the above search, we get (which must be exactly identical to the diagram):

Show search graph.

```
Core Maude 2.6
states: 4  rewrites: 3 in 4964127748ms cpu (6ms real) (0 rewrites/second)
s:State --> RCV

Solution 5 (state 4)
states: 5  rewrites: 4 in 4964127748ms cpu (8ms real) (0 rewrites/second)
s:State --> WFV

No more solutions.
states: 5  rewrites: 5 in 4964127748ms cpu (11ms real) (0 rewrites/second)
Maude> show search graph .
state 0, State: RTM
arc 0 ===> state 1 (rl RTM => VMR [label SVP] .)

state 1, State: VMR
arc 0 ===> state 2 (rl VMR => WFL [label GVL] .)

state 2, State: WFL
arc 0 ===> state 3 (rl WFL => RCV [label VLL] .)

state 3, State: RCV
arc 0 ===> state 4 (rl RCV => WFV [label GV] .)

state 4, State: WFV
arc 0 ===> state 0 (rl WFV => RTM [label agentreturn] .)
Maude> _
```

Figure 7.7: " Show graph instruction " execution for Maude vote statechart

If we look at **Figure** 7.1, which shows our mobile statechart diagram, we can confirm the path of our example (RTM => VMR => WFL => RCV => WFV => RTM);In addition, this show search graph command also allows us to show all the transitions between statesex: arc 0→state 1 (r1 RTM->VMR [label SVP].) Etc, thus creating a comprehensive picture of our system, as it indicates our executable code shown in **Figure 7.7** (this code matches our diagram), we confirmed that this property is true.

Check the property 3 :

We are now executing the command that will allow us to search for states that reach RCV or WFV beginning from RTM:

Search in vote: RTM =>* s:State such that (s:State == RCV or s:State == WFV).

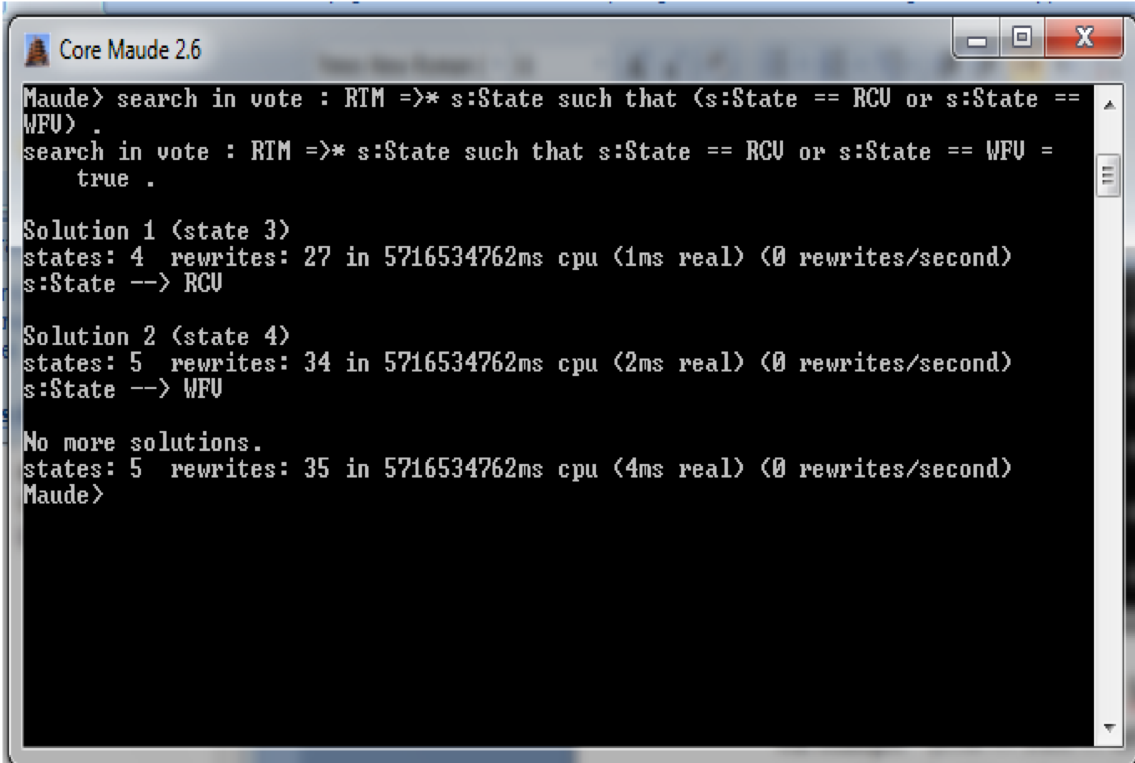


Figure 7.8: " Search specific path " execution for Maude vote statechart

Figure 7.8 shows that there is a path that allows us to get to the two states (RCV or WFV) from a source state (RTM). If we look at **Figure 7.1**, which shows our mobile statechart diagram, we can confirm this condition, so we confirmed that this property is true.

We note that the previous execution window in **Figure 7.8** gave two solutions; the first solution **RCV** related to the first case and gave us state No. 4, which is the same order of the state from **Figure 7.1**, the same thing for the second found solution **WFV** that pertains to the second case and gave us state No. 5 which is the same order of the case from **Figure 7.1.**

From here, we started working on the second intermediate code vote2, and inquired about the nature of the states starting from the system's first state (RTM).

Fortunately, we have only two types of states (mobile, non-mobile) here in vote2. Therefore, after adding the lines that give some states the mobility feature and distinguishing the rest by not mobility, it will not be difficult to use queries for this. Two queries will be enough to identify and verify the mobility feature for each state that constitutes the diagram; let's get started with the mobile states.

Check the property 4 :

We are now executing the command that will allow us to search for all mobile states beginning from RTM:

search in vote2: RTM =>* s:State such that M(s:State)

```
Core Maude 2.6
Maude> search  in vote2 : RTM =>* s:State such that M(s:State) .
search in vote2 : RTM =>* s such that M(s) = true .

Solution 1 (state 1)
states: 2  rewrites: 3 in 4964130762ms cpu (1ms real) (0 rewrites/second)
s --> VMR

Solution 2 (state 2)
states: 3  rewrites: 5 in 4964130762ms cpu (3ms real) (0 rewrites/second)
s --> WFL

Solution 3 (state 3)
states: 4  rewrites: 7 in 4964130762ms cpu (5ms real) (0 rewrites/second)
s --> RCV

Solution 4 (state 4)
states: 5  rewrites: 9 in 4964130762ms cpu (8ms real) (0 rewrites/second)
s --> WFV

No more solutions.
states: 5  rewrites: 10 in 4964130762ms cpu (11ms real) (0 rewrites/second)
Maude>
```

Figure 7.9: " search for all mobile states " execution for Maude vote 2 statechart

If we look at **Figure** 7.1, which shows our mobile statechart diagram, we can confirm that there are four mobile states (VMR, WFL, RCV, and WFV); as it indicates our executable code shown in **Figure 7.9**, we confirmed that this generate Maude specification is true and very expressive and accurate.

Check the property 5 :

We continue here with the second part of the nature of states, which is the non-mobile states. We are now executing the command that will allow us to search for all non-mobile states beginning from RTM:

Search in vote2: RTM =>* s:State such that not M(s:State).

If we look at **Figure 7.1**, which shows our mobile statechart diagram, we can confirm that there is only one non-mobile state (RTM); as it indicates our executable code shown in **Figure 7.10**, we confirmed that this generated Maude specification is true and very expressive and accurate.

We will start inquiries on the final code obtained ,as in the previous inquiries, we will try to clarify the importance of each query used and to show the method for verifying the proposed properties for our final code obtained and trying to find out whether it matches our initial mobile statechart diagram or contains errors.

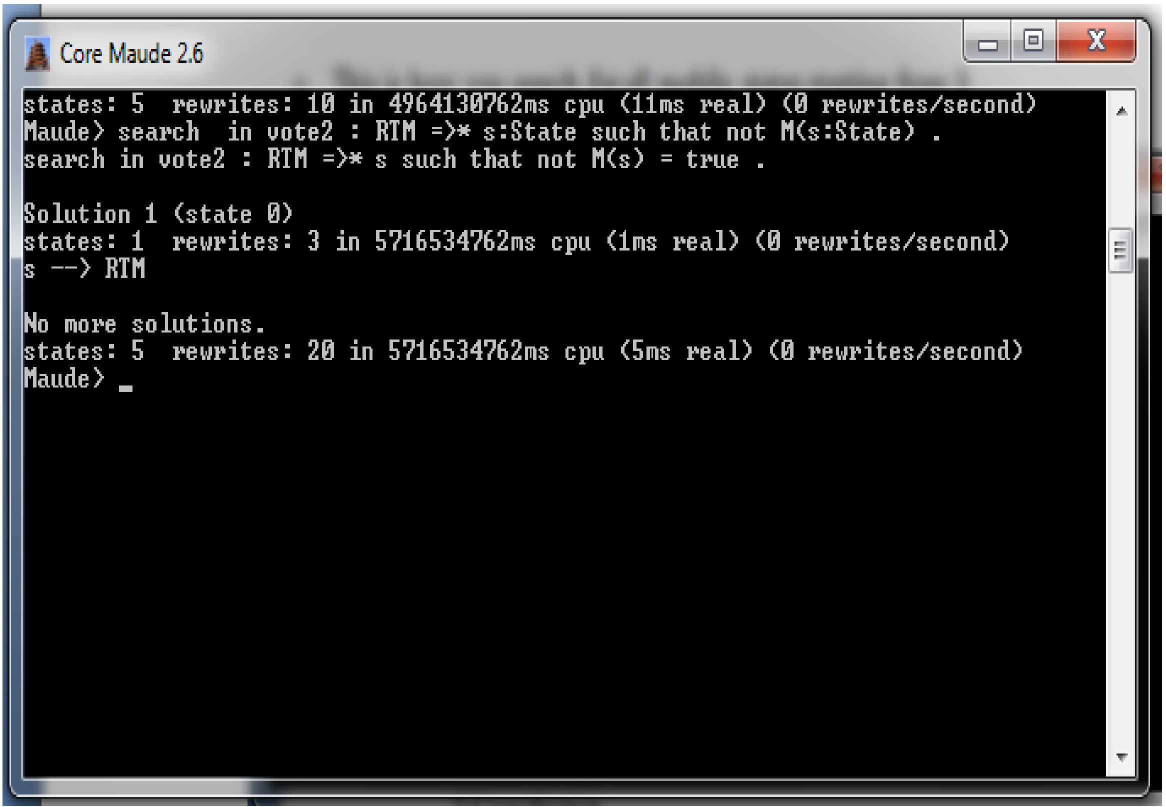


Figure 7.10: "search for all non-mobile states " execution for Maude vote 2 statechart

All reachable configurations starting from < N , RTM > are all possible configurations that make up our system, thus, obtaining a comprehensive and accurate visualization of the diagram representing our system and even reshaping and drawing the diagram from the result of executing this command without going back to the original diagram.

Check the property 6 :

We are now executing the command that will allow us to search for all reachable configurations beginning from < N , RTM >:

search in vote3: < N , RTM > =>* < t:TransitionType , s:State > .

If we compare **Figure 7.1**, which shows our mobile statechart diagram, and our executable previous code shown in **Figure 7.11**, we can confirm that this generated Maude specification is true and very expressive and accurate because the code shows us all states with

their transitions (< t=N ,s= RTM ><t= M ,s= VMR ><t=N , WFL >< t=R , s=RCV >< t=M , s=WFV >), knowing that N = Normal, M = Mobile, and R = Remote transitions.

```
Core Maude 2.6
Maude> search in vote3 : < N,RTM > =>* < t:TransitionType ,s:State > .
search in vote3 : < N,RTM > =>* < t,s > .

Solution 1 (state 0)
states: 1  rewrites: 0 in 5061174107ms cpu (0ms real) (0 rewrites/second)
t --> N
s --> RTM

Solution 2 (state 1)
states: 2  rewrites: 1 in 5061174107ms cpu (1ms real) (0 rewrites/second)
t --> M
s --> VMR

Solution 3 (state 2)
states: 3  rewrites: 2 in 5061174107ms cpu (3ms real) (0 rewrites/second)
t --> N
s --> WFL

Solution 4 (state 3)
states: 4  rewrites: 3 in 5061174107ms cpu (6ms real) (0 rewrites/second)
t --> R
s --> RCV

Solution 5 (state 4)
states: 5  rewrites: 4 in 5061174107ms cpu (9ms real) (0 rewrites/second)
t --> M
s --> WFV

No more solutions.
states: 5  rewrites: 5 in 5061174107ms cpu (12ms real) (0 rewrites/second)
Maude>
```

Figure 7.11: " search for all non-mobile states with condition " execution for Maude vote 3 statechart

Check the property 7 :

Knowing all the possible configurations that make up our system with the addition of special conditions for the allow or disallow test is the last command we will make about our generated Maude specification, this type of test allows us to get quick results even with very complex systems, It also allows us to inspect the initially proposed specifications for any system and thus verify the validity of these specifications or discover errors early in the design stage of the system, in our example, we wanted to know if we could access to a mobile state that is not reached through a mobile transition:

We are now executing the command that will allow us to check if there are mobile states that are not reached through a mobile transition:

search in vote3: < N, RTM > =>* < t:TransitionType , s:State > such that t:TransitionType =/= M and M(s:State).

```
Core Maude 2.6
Maude> search in vote3 : < N,RTM > =>* < t:TransitionType ,s:State > such that t
:TransitionType =/= M and M(s:State) .
search in vote3 : < N,RTM > =>* < t,s > such that M(s) and t =/= M = true .

Solution 1 (state 2)
states: 3  rewrites: 11 in 5061174107ms cpu (1ms real) (0 rewrites/second)
t --> N
s --> WFL

Solution 2 (state 3)
states: 4  rewrites: 15 in 5061174107ms cpu (5ms real) (0 rewrites/second)
t --> R
s --> RCV

No more solutions.
states: 5  rewrites: 20 in 5061174107ms cpu (8ms real) (0 rewrites/second)
Maude> _
```

Figure 7.12: " Check for mobile states not reached via mobile transition " execution

This rewriting rule is true, because from the **Figure 7.12** it shows us that there exist two mobile states (WFL and RCV) that are not reached through a mobile transition. When we check these properties through a comparison with our mobile statechart diagram of the voting system, we find that WFL is a mobile state and it was not reached through a mobile transition; however, it was reached through a normal transition (VM.GetVoterList). The second mobile state RCV was also not reached through a mobile transition, because it was reached through a remote transition (VM.VoterList/Logger.LogLIst).

A few implementations of verifications instructions were sufficient because we wanted to demonstrate the efficacy of our approach only through the chosen example. Still, we can extend it to more implementation of all the instructions provided by the Maude system tool or on other examples as well.

7.5 CONCLUSION

In this chapter, we have shown the application of our approach. We used a complete case study which consists of a specification of an electronic voting system. We have illustrated step-by-step how to apply the approach and how to explore its various advantages in modeling and analysis. We have also defined the main verification tasks that can be done with the derived

Maude code. Finally, we have tried to give an overview of what can be checked and obtained encouraging results.

In future work, we plan to extend our approach to account for all notational elements of UML mobile statechart diagrams. Consequently, more mobile agent-based software systems will be covered. We also plan to develop a fully integrated environment for the automatic generation and analysis of Maude specifications and formalize more M-UML diagrams using Maude specifications. We could focus on structural aspects, like mobile class diagrams, or behavior aspects, like mobile sequence diagrams and mobile activity diagrams, since they represent different aspects of a mobile system.

General conclusion and perspectives

General conclusion and perspectives

In this thesis, we are interested in the paradigm of mobile agents. We introduced in the first chapter the notion of agent in general, followed by the paradigm of mobile agents. The observation at this level has shown that the development of mobile agent applications comes up against problems inherent in security and interoperability in the various heterogeneous hosting platforms. The insufficiencies of the efforts for the standardization of these platforms pushed the researchers to explore other horizons for better support and implementation of the applications based on mobile agents.

On the other hand, very few research works focus on methods and tools for the analysis and design of mobile agent systems.

With this in mind, we have introduced in the third chapter, The first contribution of this thesis consists in completing previous comparative studies of the different proposals that exist in the literature of approaches to modeling software systems based on mobile agents, especially on the factor of coverage via the diagrams by a statistical study. Based on the results of this study, we have adopted M-UML and extended it because of the advantages it offers over others. The interest of such a contribution is to reinforce the robustness of applications in distributed systems by modeling them with standard tools such as UML. However, the flip side is that UML is a semi-formal modeling language; this aspect considerably reduces the possibility of verification and validation of the modeled systems.

The second part of our thesis introduces, in the fifth chapter, the Maude system as an alternative which makes it possible to compensate for the insufficiencies of the semi-formal approach of UML for the verification and the validation of complex systems and this, by drawing from the genesis of formal tools, or else, the richness and completeness of system tools Maude.

Our second and significant contribution, presented in the sixth chapter, proposes an automated approach based on model transformation technique, presented in the fourth chapter using the graph grammar. It is a question of transforming the diagram of transition states of M-UML as a graph towards a formal specification. The generated Maude specification will be considered as a formal semantics, which will be used after that for the analysis, such as early detection of errors (*deadlocks* , *livelocks* , etc ...), checking if certain properties are satisfied, and (or) checking equivalence between different diagrams, using Maude system analysis tools**,**

based on our contribution, was the subject of a publication in a specialized journal (JACIII JAN 2022).

Although we have achieved the objectives set for this thesis, in future work, we intend to transform the other extended UML diagrams remaining towards its Maude specifications corresponding especially to the structural aspect diagrams like the class diagram, for finalized to study the coherence between the two modes of structural and behavioral aspect.

In future work, we will plan to include other UML diagrams and add them to our integrated environment (such as mobile sequence and activity diagrams).

In this way, we will guarantee that all views of a mobile agent-based system will be covered. Another problem we are currently working on is hiding the verification process from the average user, who does not usually master formal techniques, by making an interface of abstraction of the verification process.

An interesting perspective of this work consists of the attempt to use the tools of artificial intelligence to detect design errors and work to optimize them via the exploitation of formal methods.

Where recent studies [154] indicate the main achievements and future potential of artificial intelligence to contribute effectively to the development of software through the automation of lengthy routine jobs in software development and testing using algorithms, e.g. for debugging and documentation, AI thus contributes to speeding up development processes and realising development cost reductions and efficiency gains. AI, to date depends on man-made structures and is mainly reproductive. Still, the automation of software engineering routines entails a major advantage: Human developers multiply their creative potential when using AI tools effectively.

Bibliography

[1] M. Leclercq, V. Quema, J.-B. Stefani, Dream, Proceedings of the 3rd Workshop on Adaptive and Reflective Middleware -. (2004). doi:10.1145/1028613.1028625.

[2] J.K. Bennett, J.B. Carter, W. Zwaenepoel, Munin: Distributed shared memory based on type-specific memory coherence, Proceedings of the Second ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming - PPOPP '90. (1990). doi:10.1145/99163.99182.

[3] H. Guyennet, J.-C. Lapayre, M. Trehel, A new consistency protocol implemented in the calif system, Proceedings Fourth International Conference on High-Performance Computing. (n.d.). doi:10.1109/hipc.1997.634474.

[4] G. Gardarin, Le client-serveur, Eyrolles, Paris, 1997.

[5] S. Russsell, P. Norvig, Intelligence artificielle, Pearson Education, France, 2006.

[6] J. Ferber, Les systèmes multi-agents Vers une intelligence collective, Masson, 1995.

[7] S.Perret, Agents mobiles pour l'accès nomade à l'information répartie dans les réseaux de grande envergure, thèse de doctorat, Université Joseph Fourier - Grenoble I, 1997.

[8] P. Ciancarini, M. Wooldridge, Agent-oriented software engineering (workshop session), Proceedings of the 22nd International Conference on Software Engineering - ICSE '00. (2000). doi:10.1145/337180.337833.

[9] B.Drieu, L'intelligence artificielle distribuée appliquée aux jeux d'équipe situés dans un milieu dynamique, l'exemple de RoboCup

[10] J.-P. Briot, Y. Demazeau, Principes et architecture des systèmes multi-agents, Hermes Science Publications, Paris, 2001.

[11] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, Masif: The OMG Mobile Agent System Interoperability Facility, Personal Technologies. 2 (1998) 117–129. doi:10.1007/bf01324942.

[12] J. Dale, Welcome to the Foundation for Intelligent Physical Agents. http://fipa.org/.

[13] D. B. Lange, M. Oshima, Programming And Deploying Java Mobile Agents with Aglets, 1998.

[14] ODYSSEY(GeneralMagic).URL:http://www.genmagic.com/technology/mobile_agent.html

[15] AGLETS (IBM Corporation). URL:http://www.trl.ibm.co.jp/aglets/.

[16] DIMA, OASIS, Laboratoire d‟Informatique de Paris 6 (LIP6) <URL :http://wwwwpoleia.lip6.fr/~guessoum/DIMA.html>.

[17] C.C.D. Cros, Padiou Gérard, Agents mobiles coopérants pour les environnements dynamiques, thesis, National Polytechnic Institute of Toulouse 2005.

[18] G. Vigna, Mobile agents: Ten reasons for Failure, IEEE International Conference on Mobile Data Management, 2004. Proceedings. 2004. (n.d.). doi:10.1109/mdm.2004.1263077.

[19] JADE: Java Agent Development Framework. <URL: http://www.jade.cselt.it/>.

[20] OMG, Object Management Group, (2011). Unified Modeling Language, Superstructure, version 2.4,http://www.omg.org/spec/UML/2.4.

[21] A. Maria, Introduction to modeling and Simulation, Proceedings of the 29th Conference on Winter Simulation - WSC '97. (1997). doi:10.1145/268437.268440.

[22] M. Minsky, Matter, Mind, and Models. In: Minsky, M., Ed., Semantic Information Processing, MIT Press, Cambridge, M. 425-432 , M. (1968).

[23] P.Alain Muller, From object modeling of software to meta-modeling of computer languages, HDR thesis presented to the University of Rennes .

[24] L. Audibert, UML 2.0 , University Institute of Technology of Villetaneuse, IT http://www-lipn.univ-paris13.fr/audibert/pages/enseignement/cours.htm, November 2007.

[25] J. Celso Freire Junior, J. Jean-Pierre Giraudin – Agnès Front Atelier MODSI: A Meta - Modeling and Multi-Modeling Tool. Network Systems Software Laboratory – IMAG BP 72 - 38402 – Saint Martin d' Heres Cedex – France.

[26] J.J. Odell – Specifying structural constraints. Journal of Object Oriented Programming , 1993, pp. 12–16.

[27] S.Cook ,J. Daniels , Object-Oriented Modeling with Syntropy . Designing Object Systems - , Prentice Hall, 1994.

[28] J. Rumbaugh ,M. Blaha ,W. Premerlani , F. Eddy ,W. Lorensen Oriented Modeling and Design, Object – Prentice Hall, 1991.

[29] J.M. Spivey , The Z Notation, A reference Manual. – Prentice Hall, 1989.

[30] A. Hall, Seven myths of formal methods, IEEE Software. 7 (1990) 11–19. doi:10.1109/52.57887.

[31] J.P. Bowen, M.G. Hinchey, Seven more myths of formal methods, IEEE Software. 12 (1995) 34–41. doi:10.1109/52.391826.

[32] R. Miles, K. Hamilton, Learning UML 2. 0, O'Reilly, Cambridge, 2006.

[33] J. Rumbaugh , I. Jacobson, and G. Booch , The Unified Modeling Language Reference Manual . Addison Wesley Longman, Inc. , 1998

[34] T. Penders, UML Bible, Wiley Pub., Indianapolis, IN, 2003.

[35] M .Fowler UML distilled: A brief guide to the standard object modeling language, Addison Wesley, 2004.

[36] H. Idrissi, A. Revel, E. Souidi , Security of Mobile Agent Platforms using RBAC based on Dynamic Role Assignment. , International Journal of Security and its Applications · April 2016
DOI: 10.14257/ijsia.2016.10.4.13

[37] G. Booch , J. Rumbaugh , I. Jacobson, the Unified Modeling Language User Guide second edition , Addison-Wesley Professional, USA, 2005

[38] H.-E. Eriksson, M. Penker , B. Lyons, and D. Fado , UML2 Toolkit . John Wiley & Sons, 2004

[39] H. Baumeister, N. Koch, P. Kosiuczenko, M. Wirsing, Extending activity diagrams to model mobile systems, Objects, Components, Architectures, Services, and Applications for a Networked World. (2003) 278–293. doi:10.1007/3-540-36557-5_21.

[40] C. Klein, A. Rausch, M. Sihling, Z. Wen, Extension of the unified modeling language for mobile agents, Unified Modeling Language. (2001) 117–129. doi:10.4018/978-1-930708-05-1.ch008.

[41] K. Saleh, C. El-Morr, M-UML: An extension to UML for the modeling of mobile agent-based Software Systems, Information and Software Technology. 46 (2004) 219–227. doi:10.1016/j.infsof.2003.07.004.

[42] H.Mouratidis .J. OdellJ. G Manson , Extending the Unified Modeling Language to Model Mobile Agents. Workshop on Agent-oriented methodologies. OOPSLA 2002, Seattle, USA, November 2002.

[43] M.Y._Chkouri , Modélisation des systèmes temps-réel embarqués en utilisant AADL pour la génération automatique d'applications formellement vérifiées, PhD thesis, Joseph Fourier University, France, April 7, 2010.

[44] A. Belghiat, Mobile agent-based Software Systems Modeling Approaches: A Comparative Study, Journal of Computing and Information Technology. 24 (2016) 149–163. doi:10.20532/cit.2016.1002695.

[45] C.A. Iglesias, M. Garijo, J.C. González, A survey of agent-oriented methodologies, Intelligent Agents V: Agents Theories, Architectures, and Languages. (1999) 317–330. doi:10.1007/3-540-49057-4_21.

[46] E. Belloni, C. Marcos, Modeling of mobile-agent applications with UML. In Proceedings of the Fourth Argentine Symposium on Software Engineering (ASSE 2003) (Vol. 32, pp. 1666-1141).

[47] B. Bauer, J. P Müller, Methodologies and modeling languages. Agent-based Software Development, (2004) (pp 77-131).

[48] D. Melomey, H., Mouratidis, C.Imafidon, An Evaluation of Current Approaches for Modelling Mobility of Agents. Advances in Computing and Technology, The School of Computing and Technology 2nd Annual Conference, (2007) ICGES Press.

[49] D. Melomey, C.Imafidon, G. Williams, A Comparative Study of Modeling Languages for Agent Systems. Systems and Information Science Notes (SISN) (2007) (Vol. 2, pp 207-212).

[50] R. Cervenka, I. Trencansky, The agent modeling Language - AML a comprehensive approach to modeling multi-agent systems, Birkhäuser Basel, Basel, 2007.

[51] H. Hachicha, A. Loukil, K. Ghedira, Ma-UML: A conceptual approach for mobile agents' modelling, International Journal of Agent-Oriented Software Engineering. 3 (2009) 277. doi:10.1504/ijaose.2009.023640.

[52] F. Mokhati, M. Badri, Generating Maude specifications from UML use case diagrams., The Journal of Object Technology. 8 (2009) 119. doi:10.5381/jot.2009.8.2.a2.

[53] P. Bellavista, A. Corradi, C. Federici, R. Montanari & D. Tibaldi. Security for mobile agents : Issues and challenges. In M. Ilyas & I. Mahgoub, éditeurs. Mobile Computing Handbook, chapitre 39, pages 941–960. CRC Press, 2005.

[54] P. Ahuja and V. Sharma, “A Review on Mobile Agent Security”. International Journal of Recent Technology and Engineering (IJRTE). 2, 1, 2277–3878. (2012).

[55] A, Loukil , H. Hachicha, K. Ghedira, A proposed Approach to Model and to Implement Mobile Agents, SOIE, Institut Supérieur de Gestion de Tunis, Université de Tunis.2003.

[56] E. Belloni, C. Marcos, Mam-UML, An UML profile for the modeling of Mobile-Agent Applications, XXIV International Conference of the Chilean Computer Science Society. (n.d.). doi:10.1109/qest.2004.14.

[57]C. Klein, A. Rausch, M. Sihling, Z. Wen, Extension of the unified modeling language for mobile agents, Unified Modeling Language. (2001) 117–129. doi:10.4018/978-1-930708-05-1.ch008.

[58] F. Muscutariu, M.-P. Gervais, On the modeling of mobile agent-based systems, Lecture Notes in Computer Science. (2001) 219–233.doi:10.1007/3-540-44651-6_21.

[59] E. Kerkouche, K. Khalfaoui, A. Chaoui, A. Aldahoud, UML activity diagrams and Maude Integrated Modeling and analysis approach using graph transformation, The 7th International Conference on Information Technology. (2015). doi:10.15849/icit.2015.0093.

[60] B. Bauer, *J.P. Müller, +J. Odell, Agent UML: A formalism for specifying Multiagent Software Systems, Agent-Oriented Software Engineering. (2001) 91–103. doi:10.1007/3-540-44564-1_6.

[61] A. Poggi, G. Rimassa, P. Turci, J. Odell, H. Mouratidis, G. Manson, Modeling deployment and mobility issues in multiagent systems using AUML, Agent-Oriented Software Engineering IV. (2004) 69–84. doi:10.1007/978-3-540-24620-6_5.

[62] B. Bauer, J. Odell, UML 2.0 and agents: How to build agent-based systems with the new UML standard, Engineering Applications of Artificial Intelligence. 18 (2005) 141–157. doi:10.1016/j.engappai.2004.11.016.

[63] R. Červenka, I. Trenčanský, M. Calisti, D. Greenwood, AML: Agent modeling language toward industry-grade agent-based modeling, Agent-Oriented Software Engineering V. (2005) 31–46. doi:10.1007/978-3-540-30578-1_3.

[64] A.F. Pires, V. Wiels, T. Polacsek, Amélioration des Processus de vérification de programmes par combinaison des méthodes formelles avec l'ingénierie Dirigée par les modèles, thesis, Thèse de doctorat Sureté de logiciel et calcul de haute performance, Université de Toulouse, 26 juin 2014.

[65] H. Hachicha, A. Loukil, K. Ghedira, MAMT: an environment for modeling and implementing mobile agents. In Sixth International Workshop from Agent Theory to Agent Implementation (AT2AI-6) (2008, May) (pp. 75-82).

[66] Grasshopper mobile agent system (2003). IKV++ GmbH, documentation and software. Available at: http://www.grasshopper.de/.

[67] Miao Kang, Modelling mobile agent applications in UML2.0 activity diagrams, "Third International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'04)" W16L Workshop - 26th International Conference on Software Engineering. (2004). doi:10.1049/ic:20040366.

[68] M. Kusek, G. Jezic, Extending UML sequence diagrams to model agent mobility. In International Workshop on Agent-Oriented Software Engineering (2006, May) (pp. 51-63). Springer Berlin Heidelberg.

[69] M. R., Bahri, R. Mokhtari, A. Chaoui, Towards an extension of UML2. 0 to model mobile agent based systems. International Journal of Computer Science and Network Security (2009),, 9(10), 124-13.

[70] L. Bettini, R. De Nicola, M. Loreti, Formalizing properties of Mobile Agent Systems, Lecture Notes in Computer Science. (2002) 72–87. doi:10.1007/3-540-46000-4_9.

[71] A. Jeffrey, *communicating and mobile systems: The π-calculus* by Robin Milner, Cambridge University Press, 1999, Journal of Functional Programming. 11 (2001) 433–436. doi:10.1017/s0956796801224113.

[72] G. Jezic , I. Lovrek Using Pi-Calculus for specification of mobile agent communication. In IASTED Conf. on Software Engineering and Applications, (2004) (pp. 356-361).

[73] C. Fournet, G. Gonthier, J.-J. Levy, L. Maranget, D. Rémy, A calculus of Mobile Agents, CONCUR '96: Concurrency Theory. (1996) 406–421. doi:10.1007/3-540-61604-7_67.

[74] F. Oquendo, π-Adl: an Architecture Description Language based on the higher-order typed π- calculus for specifying dynamic and mobile software architectures, ACM SIGSOFT Software Engineering Notes. 29 (2004) 1–14. doi:10.1145/986710.986728.

[75] A. Schmitt, J.-B. Stefani, The kell calculus: A family of higher-order distributed process calculi, Global Computing. (2005) 146–178. doi:10.1007/978-3-540-31794-4_9.

[76] P. Sewell, P.T. Wojciechowski, B.C. Pierce, Location-independent communication for mobile agents: A two-level architecture, Internet Programming Languages. (1999) 1–31. doi:10.1007/3-540-47959-7_1.

[77] L. Cardelli, A.D. Gordon, Mobile ambients, Theoretical Computer Science. 240 (2000) 177–213. doi:10.1016/s0304-3975(99)00231-5.

[78] R. De Nicola, G.L. Ferrari, R. Pugliese, Klaim: A kernel language for agents interaction and Mobility, IEEE Transactions on Software Engineering. 24 (1998) 315–330. doi:10.1109/32.685256.

[79] T.A. Kuhn, D. von Oheimb, Interacting state machines for Mobility, FME 2003: Formal Methods. (2003) 698–718. doi:10.1007/978-3-540-45236-2_38.

[80] Dianxiang Xu, Yi Deng, Modeling Mobile Agent Systems with high level petri nets, SMC 2000 Conference Proceedings. 2000 IEEE International Conference on Systems, Man and Cybernetics. 'Cybernetics Evolving to Systems, Humans, Organizations, and Their Complex Interactions' (Cat. No.00CH37166). doi:10.1109/icsmc.2000.886486.

[81] Dianxiang Xu, Jianwen Yin, Yi Deng, Junhua Ding, A formal architectural model for Logical Agent Mobility, IEEE Transactions on Software Engineering. 29 (2003) 31–45. doi:10.1109/tse.2003.1166587.

[82] H. Xu, S. M. Shatz, A Design Model for Intelligent Mobile Agent Software Systems. Computer Science Department, The University of Illinois at Chicago (2002).

[83] J.-M. Jiang, H. Zhu, Q. Li, Y. Zhao, L. Zhao, S. Zhang, & al., Event-based Mobility Modeling and analysis, ACM Transactions on Cyber-Physical Systems. 1 (2017) 1–32. doi:10.1145/2823353.

[84] M.R. Bahri, Une approche intégrée Mobile-UML/Réseaux de Pétri pour l'analyse des systèmes distribués à base d'agents mobiles (Doctoral dissertation, (2010) , Doctoral thesis, University of Constantine, Algeria.

[85] A. Knapp, S. Merz, M. Wirsing, Refining Mobile UML State Machines, Algebraic Methodology and Software Technology. (2004) 274–288. doi:10.1007/978-3-540-27815-3_23.

[86] D. Latella, M. Massink, H. Baumeister, M. Wirsing, Mobile UML Statecharts with localities, Global Computing. (2005) 34–58. doi:10.1007/978-3-540-31794-4_3.

[87] A. Belghiat, A. Chaoui, M. Maouche, M. Beldjehem, Formalization of mobile UML Statechart diagrams using the π-calculus: An approach for modeling and analysis, Communications in Computer and Information Science. (2014) 236–247. doi:10.1007/978-3-319-11958-8_19.

[88] M. Kezai, A. Khababa, Generating Maude specifications from M-UML Statechart diagrams, Journal of Advanced Computational Intelligence and Intelligent Informatics. 26 (2022) 8–16. doi:10.20965/jaciii.2022.p0008.

[89] Y. Aridor, D.B. Lange, Agent design patterns, Proceedings of the Second International Conference on Autonomous Agents - AGENTS '98. (1998). doi:10.1145/280765.280784.

[90] R. Tolksdorf, Coordination Patterns of Mobile Information Agents, Cooperative Information Agents II Learning, Mobility and Electronic Commerce for Information Discovery on the Internet. (1998) 246–261. doi:10.1007/bfb0053689.

[91] E.F. Lima, P.D. Machado, F.R. Sampaio, J.C. Figueiredo, An approach to modelling and applying Mobile Agent Design Patterns, ACM SIGSOFT Software Engineering Notes. 29 (2004) 1–8. doi:10.1145/986710.986726.

[92] E. Kerkouche, Modélisation Multi-Paradigme: Une Approche Basée sur la Transformation de Graphes, doctoral thesis , University of Mentouri Constantine, 2011.

[93] M. L. Crane, J. Dingel, On the semantics of UML state machines: Categorization andcomparison. In In Technical Report 2005-501, School of Computing, Queen's (2005).

[94] E.V . Berard, A comparison of object-oriented methodologies. Technical report. Object Agency Inc (1995)..

[95] P.B. Kruchten, The 4+1 view model of Architecture, IEEE Software. 12 (1995) 42–50. doi:10.1109/52.469759.

[96] M.W. Maier, D. Emery, R. Hilliard, Software architecture: Introducing IEEE Standard 1471, Computer. 34 (2001) 107–109. doi:10.1109/2.917550.

[97] N. Rozanski, E.Woods ,Software systems architecture: Working with stakeholders using viewpoints and perspectives, Upper Saddle River, NJ: Addison-Wesley Professional; 2005. doi:10.5860/choice.49-5717.

[98] M.Girschick,T. Darmstadt, Difference detection and visualization in UML class diagrams. Technical university of darmstadt technical report,2006, TUD-CS-2006-5, 1-15.

[99] E.J. Gonçalves, M.I. Cortés, G.A. Campos, Y.S. Lopes, E.S.S. Freire, V.T. da Silva, et al., MAS-ML 2.0: Supporting the modelling of multi-agent systems with different Agent Architectures, Journal of Systems and Software. 108 (2015) 77–109. doi:10.1016/j.jss.2015.06.008.

[100] A.G. Clark, N. Walkinshaw, R.M. Hierons, Test case generation for agent-based models: A Systematic Literature Review, Information and Software Technology. 135 (2021) 106567. doi:10.1016/j.infsof.2021.106567.

[101] H. Kadima, MDA,une conception orientée objet guide par les modèles. DUNOD 2005.

[102] J. Favre, J. Estublier, M.Blay-Fornarino, L'ingénierie dirigée par les modèles au-delà du MDA Hermes Science publications, Lavoisier.

[103] N. Prakash, S. Srivastava, S. Sabharwal, The classification framework for Model Transformation, Journal of Computer Science. 2 (2006) 166–170. doi:10.3844/jcssp.2006.166.170.

[104] K. Czarnecki, S. Helsen, Classification of Model Transformation Approaches, University of Waterloo, Canada.2003, shelsen@computer.org.

[105] E. Guerra, J. de Lara, A Framework for the Verification of UML Models. Examples using Petri Nets. Ecole Politechnique Superieur, Ingenieríe de l' Informátique, Université Autónoma de Madrid.Spain 2003.

[106] J. Meseguer, Conditional rewriting logic as a unified model of concurrency, Theoretical Computer Science. 96 (1992) 73–155. doi:10.1016/0304-3975(92)90182-f.

[107] J. Meseguer, Software Specification and Verification in Rewriting logic, Germany, August 2002.

[108] J. Meseguer, G. Roşu, Rewriting logic semantics: From language specifications to formal analysis tools, Automated Reasoning. (2004) 1–44. doi:10.1007/978-3-540-25984-8_1.

[109] M.Theodore, Maude 2.0 Premier Version 1.0, August 2003.

[110] M. Clavel, F. Durán, S. Eker, P. Lincoln, Martí-Oliet N., J. Meseguer,J.F. Quesada, Maude: Specification and Programming in Rewriting Logic, SRI International Lab,http://maude.csl.sri.com, 1999.

[111] M. Clavel, F. Durán, S. Eker, P. Lincoln, Martí-Oliet N., J. Meseguer, et al., Maude: Specification and programming in rewriting logic, Theoretical Computer Science. 285 (2002) 187–243. doi:10.1016/s0304-3975(01)00359-0.

[112] M. Benammar, An Architecture-Based Approach for the Formal Specification of Embedded Systems, Doctoral Thesis in Sciences, Mentouri University of Constantine, 2011.

[113] M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln,N. MartiOliet , M. José, C. Talcott, Maude Manual (Version 2.7.1), July 2016.

[114] S. Eker, J. Meseguer, A. Sridharanarayanan, The Maude LTL model Checker and its implementation, Model Checking Software. (2003) 230–234. doi:10.1007/3-540-44829-2_16.

[115] M. Clavel, All about Maude: A high-performance logical framework: How to specify, program and verify systems in rewriting logic, Springer, New York, 2007.

[116] M. Clarke Edmund,O. Grumberg,D. Peled, Model Checking. MIT Press ,Cambridge ,Massachusetts United States, ISBN: 978-0-262-03270-4, 2001

[117] K. Saleh, C. El Morr, A. Mourtada, Y. Morad, A mobile-agent platform and a game application specifications using M-UML, The Electronic Library. 22 (2004) 32–42. doi:10.1108/02640470410520096.

[118] G. Csertan, G. Huszerl, I. Majzik, Z. Pap, A. Pataricza, D. Varro, Viatra - visual automated transformations for formal verification and validation of UML models, Proceedings 17th IEEE International Conference on Automated Software Engineering. doi:10.1109/ase.2002.1115027.

[119] J. de Lara, H. Vangheluwe, Computer aided multi-paradigm modelling to process petri-nets and Statecharts, Graph Transformation. (2002) 239–253. doi:10.1007/3-540-45832-8_19.

[120] J. A. Saldhana, M. Shatz, and Z. Hu, Formalization of object behavior and interactions from UML models, International Journal of Software Engineering and Knowledge Engineering. 11 (2001) 643–673. doi:10.1142/s021819400100075x.

[121] X. Hei, L. Chang, W. Ma, J. Gao, G. Xie, Automatic transformation from UML Statechart to petri nets for safety analysis and verification, 2011 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering. (2011). doi:10.1109/icqr2mse.2011.5976760.

[122] M. Wang, L. Lu, A transformation method from UML Statechart to Petri Nets, 2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE). (2012). doi:10.1109/csae.2012.6272734.

[123] E. Kerkouche, A.A. Chaoui, E.B. Bourennane, O. Labbani, A UML and Colored Petri nets integrated modeling and analysis approach using graph transformation., The Journal of Object Technology. 9 (2010) 25. doi:10.5381/jot.2010.9.4.a2.

[124] J. Araujo and A. Moreira, pecifying the Behavior of UML Collaborations Using Object-Z,Departamento de Infomatica, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal, 2000.

[125] H. Ledang and J. Souquières, Formalizing UML Behavioral Diagrams with B, Tenth OOPSLA Workshop on Behavioral Semantics: Back to Basics, Tampa Bay, Florida, USA, 2001.

[126] C. A. R. Hoare, Communicating sequential processes. Prentice-Hall International, London, 1985, VIII+256 pages., Science of Computer Programming. 9 (1987) 101–105. doi:10.1016/0167-6423(87)90028-1.

[127] P. Gagnon, F. Mokhati, M. Badri, Applying model checking to concurrent UML models., The Journal of Object Technology. 7 (2008) 59. doi:10.5381/jot.2008.7.1.a1.

[128] D. Poole John, Model-driven architecture: Vision, standards and emerging technologies, In ECOOP 2001, Workshop on Metamodeling and Adaptive Object Models, 2001.

[129] C. Rossi, M. Enciso, I.P. de Guzmán, Formalization of UML state machines using temporal logic, Software & Systems Modeling. 3 (2004) 31–54. doi:10.1007/s10270-003-0029-7.

[130] Muan Yong Ng, M. Butler, Towards formalizing UML state diagrams in CSP, First International Conference OnSoftware Engineering and Formal Methods, 2003.Proceedings. (2003). doi:10.1109/sefm.2003.1236215.

[131] W.L. Yeung, K.R.P.H. Leung, Ji Wang, Wei Dong, Improvements towards formalizing UML state diagrams in CSP, 12th Asia-Pacific Software Engineering Conference (APSEC'05). (2005). doi:10.1109/apsec.2005.70.

[132] J. Lilius, I.P. Paltor, Formalising ,The Semantics of UML State Machines, TUCS Technical Report No. 273, (1999) 430–444. doi:10.1007/3-540-46852-8_31.

[133] S.A. Seshia, R.K. Shyamasundar, A.K. Bhattacharjee, S.D. Dhodapkar, A translation of statecharts to esterel, FM’99 — Formal Methods. (1999) 983–1007. doi:10.1007/3-540-48118-4_3.

[134] W.L. Yeung, K.R.P.H. Leung, Ji Wang, Wei Dong, Improvements towards formalizing UML state diagrams in CSP, 12th Asia-Pacific Software Engineering Conference (APSEC'05). (2005). doi:10.1109/apsec.2005.70.

[135] V.S.W. Lam, J. Padget, Formalization of UML Statechart diagrams in the π-calculus, Proceedings 2001 Australian Software Engineering Conference. (n.d.). doi:10.1109/aswec.2001.948515.

[136] M. Balaha, J. Rumbaugh : Modélisation et conception orientées objet avec UML 2, deuxième édition, Pearson Education, France,2005

[137] M.R. Bahri, A. Hettab, A. Chaoui, E. Kerkouche, Transforming mobile UML Statecharts models to nested nets models using graph grammars: An approach for modeling and analysis of Mobile Agent-based Software Systems, 2009 Fourth South-East European Workshop on Formal Methods. (2009). doi:10.1109/seefm.2009.21.

[138] P. Muller, N. Gaertner : Modélisation objet avec UML, Deuxième édition, Eyrolles, 2000

[139] J. Meseguer. A logical theory of concurrent objects. ACM SIG-PLAN Notices, 25(10), pp101- 115, 1990. Proc. OOP-SLA ECOOP’ 90

[140] S. Eker, J. Meseguer, A. Sridharanarayanan, The Maude LTL model Checker, Electronic Notes in Theoretical Computer Science. 71 (2004) 162–187. doi:10.1016/s1571-0661(05)82534-4.

[141] J. Derrick, D. Akehurst, A framework for UML consistency January 2002

[142] C. Lange, An empirical assessment of completeness in UML designs, "8th Internation Conference on Empirical Assessment in Software Engineering (EASE 2004)" Workshop - 26th International Conference on Software Engineering. (2004). doi:10.1049/ic:20040404.

[143] G. Reggio. R.J. Wieringa Thirty one Problems in the Semantics of UML 1.3 Dynamics.

[144] A. Moreira . Defining Precise Semantics for UML Object-Oriented. Technology. ECOOP 2000 Workshop Reader. ECOOP 2000 Workshops.

[145] E. Astesiano , G. Reggio, UML as a Heterogeneous Multiview Notation Strategies for a Formal Foundation (1998).

[146] H. Malgouyres,S. Jean-Pierre, G. Motet, Identification de Règles de Cohérence d'UML 2.0. Journée SEE "Systèmes Informatiques de Confiance" sur le thème "Vérification de la cohérence de modèles UML", Paris, France, Mars 2005.

[147] F. Durán and J. Meseguer. The Maude specification of Full Maude. Manuscript, May 1999. http://maude.cs.uiuc.edu/papers/.

[148] AToM3 home page, (online) available at: http://atom3.cs.mcgill.ca/.

[149] D. Harel, A. Naamad, The statemate semantics of statecharts, ACM Transactions on Software Engineering and Methodology. 5 (1996) 293–333. doi:10.1145/235321.235322.

[150] T. Gan, M. Chen, Y. Li, B. Xia, N. Zhan, Reachability analysis for solvable dynamical systems, IEEE Transactions on Automatic Control. 63 (2018) 2003–2018. doi:10.1109/tac.2017.2763785.

[151] A. Boucherit, A. Khababa, L.M. Castro, Automatic Generating Algorithm of Rewriting Logic Specification for multi-agent system models based on Petri Nets, Multiagent and Grid Systems. 14 (2019) 403–418. doi:10.3233/mgs-180298.

[152] Z. Manna, A. Pnueli, The temporal logic of reactive and concurrent systems, (1992). doi:10.1007/978-1-4612-0931-7.

[153] E.M., Clarke, O. Grumberg, D.A. Peled, Model Checking. MIT Press, Cambridge (1999)

[154] M. Barenkamp, J. Rebstadt, O. Thomas, Applications of AI in Classical Software Engineering, AI Perspectives. 2 (2020). doi:10.1186/s42467-020-00005-4.

[155] G. Rozenberg, Handbook of Graph Grammars and computing by graph transformation, World Scientific, Singapore, 1997.

.

Contributions

International Publications

<table>
<tr><td rowspan="6">International Publications</td><td>Title:</td><td colspan="2">Generating Maude Specifications from M-UML Statechart Diagrams</td></tr>
<tr><td rowspan="2">Autors:</td><td>Last Name : Kezai</td><td>First Name :Mourad</td></tr>
<tr><td>Last Name :Khababa</td><td>First Name :Abdallah</td></tr>
<tr><td>Year :</td><td colspan="2">2022</td></tr>
<tr><td>Journal</td><td colspan="2">Journal of Advanced Computational Intelligence and Intelligent Informatics (Vol.26 No.1, 2022)</td></tr>
<tr><td>URL</td><td colspan="2">https://doi.org/10.20965/jaciii.2022.p0008</td></tr>
</table>

ملخص

تشكل الأنظمة متعددة الوكلاء نوعًا مثيرًا للاهتمام من نماذج الشركات، وعلى هذا النحو، لها مجالات تطبيق واسعة جدًا. لذلك فإن مجال تكنولوجيا "الوكلاء المتنقلون" يمثل مفهوماً جديداً ومهماً في الذكاء الاصطناعي وهندسة البرمجيات. على الرغم من الاهتمام بهذه التكنولوجيا، فإن معظم الأساليب في الوقت الحالي لا تسمح بتصميم كاف وكامل للوكلاء المتنقلين.

من ناحية أخرى، فإن نظام Maude عبارة عن مواصفات وبرمجة موجهة للكائنات وبيئة إثبات تعتمد على إعادة كتابة المنطق الذي يعد إطارًا دلاليًا موحدًا للعديد من نماذج التزامن. بالإضافة إلى ذلك، يحتوي نظام Maude أيضًا على مجموعة من الأدوات التي تسمح بمحاكاة وتحليل إمكانية الوصول بواسطة مدقق النماذج LTL (المنطق الزمني الخطي) الذي يتيح التحقق من خصائص المواصفات ووضع نماذج أولية لها.

أخيرًا، فإن الطريقة الرسومية لـ UML وامتداداتها تجعل من الممكن تمثيل الأنظمة بشكل بديهي وتركيبي، أنظمة متعددة العوامل. ومع ذلك، فهي غير مجهزة بأدوات للتحقق الرسمي. في هذه الأطروحة، نتناول نهجًا لنسخ مخططات M-UML في نظام Maude. قمنا بتطبيق نهجنا المقترح المكون من ثلاث مراحل على أحد أهم الرسوم البيانية فيM-UML ، وهو مخطط الحالة، وتتكون التقنية من اشتقاق وصف رسمي منهجي من هذا النوع من تحليل الرسم البياني وحصلنا على نتائج مرضية للغاية. يجعل النسخ المقترح من الممكن الحصول على مواصفة جبرية معبراً عنها بمنطق إعادة الكتابة. سيتم استخدام هذا الأخير للتحقق من خصائص الأنظمة القائمة على الوكيل المحمول.

Résume :

Les systèmes multi-agents forment un type intéressant de modélisation de sociétés, car ils ont des domaines d'application très larges. Par conséquent, le domaine de la technologie des "agents mobiles" représente un concept nouveau et important en intelligence artificielle et de l'ingénieries de logiciels. Malgré l'intérêt porté à cette technologie, la plupart des approches à l'heure ne permettent pas une conception suffisante et complète des agents mobiles.

D'autre part, Maude est un environnement de spécification, de programmation orientée objet et de preuve basé sur une logique de réécriture qui est un cadre sémantique unificateur de plusieurs modèles de concurrence. En plus, le système Maude dispose également d'une batterie d'outils permettent la simulation, l'analyse d'accessibilité par le LTL (Linear Time Logic) Model-checker permettant la vérification et le prototypage des propriétés de spécification.

Enfin, la méthode graphique UML généralement et ses extensions permettent de représenter de manière synthétique et intuitive les systèmes multi-gent. Cependant, elles ne sont pas dotées permettant de faire la vérification. Dans cette thèse, nous adoptons une approche de la spécification formelle des diagrammes M-UML dans le système Maude. Nous avons appliqué notre approche en trois étapes proposées au diagramme d'états-transitions, l'un des diagrammes les plus importants de M-UML. La technique consiste à dériver systématiquement une description formelle de Maude à partir de ce type d'analyse de diagramme, et nous avons obtenu des résultats très satisfaisants. Cette transcription proposée permet d'obtenir une spécification algébrique exprimée en logique de réécriture. Ce dernier sera utilisé pour vérifier les spécifications des systèmes à base d'agents mobiles.

Abstract

First, multi-agent systems form an interesting type of company modeling and, as such, have very broad fields of application. Therefore, the technology field of "mobile agents" represents a new and important concept in artificial intelligence and software engineering. Despite the interest in this technology, most approaches at present do not allow a sufficient and complete design of mobile agents.

On the other hand, the Maude system is a specification, object-oriented programming and proof environment based on rewriting logic which is a unifying semantic framework of several concurrency models. In addition, the Maude system also has a battery of tools allowing the simulation and the analysis of accessibility by the LTL (Linear Temporal Logic) Model-Checker enabling the verification and the prototyping of the properties of the specifications.

Finally, the UML graphical method and its extensions make it possible to represent systems synthetically and intuitively, multi-agent systems. However, they are not equipped with tools for verification. In this thesis, we address an approach for the transcription of M-UML diagrams in the Maude system. We applied our proposed three-stage approach to the statechart diagram, one of the most important diagrams in M-UML. The technique involves systematically deriving a formal Maude description from this type of diagram analysis, and we obtained very satisfactory results. The proposed transcription makes it possible to obtain an algebraic specification expressed with the rewriting logic. The latter will be used for the verification of properties of mobile agent-based systems.