MASD: Multi-Agent Systems Development Methodology

T. Abdelaziz¹, M. Elammari², R. Unland³, C. Branki⁴

¹,²IT Faculty, University of Garyounis, Benghazi, Libya
³ICB, University of Duisburg-Essen, Essen, Germany
⁴School of ICT, University of the West of Scotland, Scotland, UK

1 tawfig@informatik.uni-essen.d,

2 elammari@garyounis.edu,

3 unlandr@informatik.uni-essen.de

4 cherif.branki@uws.ac.uk

Abstract. In recent years, multi-agent systems gained growing acceptance as a required technology to develop complex distributed systems. As result, there is an increased need for practical methodology for developing such systems. This paper presents a new Multi-Agent System Development (MASD) methodology developed over several years through analyzing and studying most of the existing agent oriented methodologies. This new methodology is constructed based on the strengths and weaknesses of existing methodologies. MASD aims to provide designers of agent-based systems with a set of methods and guidelines that allow them to control the construction process of complex systems, enabling software engineers to specify agent-based systems that would be implemented within an execution environment, for example the Jadex platform. MASD differs from

existing methodologies in that it is a detailed and complete methodology for developing multi-agent systems. This paper describes the methodology's process and illustrates it using a running example—namely, a car rental system.

**Keywords:** agent-oriented methodologies, agent-oriented software engineering, multi-agent systems, software agents.
1. Introduction

There has been a wide and an increasing acceptance of multi-agent system to implement complex and distributed systems. In addition, with industry demanding increasingly sophisticated applications, agent-oriented technology is being supplemented. Building multi-agent applications for complex and distributed systems is not an easy task; indeed, the development of industrial-strength applications requires the availability of strong software engineering methodologies. These methodologies typically consist of a set of methods, models, and techniques that facilitate a systematic software development process.

Many agent-oriented methodologies and modeling languages have been proposed such as: Gaia [48,52], MaSE [22], MESSAGE [16], Tropos [12], HLIM [25], Prometheus [36], and AUML [5] etc. These methodologies have been established to develop multi-agent systems and to support tools that allow developers to create complex agent applications.

Several evaluation frameworks and comparisons of agent-oriented methodologies have been suggested such as [1,6,43,21,40,41,17]. Most of them agree that despite the fact that most of these methodologies are developed on strong foundations, they suffer from number of limitations: none of the existing agent-oriented methodologies has itself established as a standard nor have they been commonly accepted [34]. The lack of standard agent architectures and agent programming languages is actually the main problem in defining agent models and putting them into operation, or providing a useful “standard” code generation. Since there is no standard agent architecture, agent design needs to be customized for each agent architecture. Nevertheless, the analysis models are independent of the agent architectures; they describe what the agent-based system has to do, but not how this is done [30]. Moreover, there is no agreement on how existing methodologies identify and characterize some of the important agent aspects, such as the beliefs, goals, plans, roles that the agents play in the system, or interactions [21,23]. The existence of such an agreement would contribute to agent standardization. Until now, no agent-oriented standards have been established, and most previous research that has examined and compared agent-oriented methodologies suggested that none was completely suitable for industrial development of multi-agent systems [34,44]. Even though existing methodologies are based on a strong agent-oriented basis, they need to support essential software engineering issues such as accessibility and expressiveness, which has an adverse effect on industry acceptability and adoption of agent technology [42]. None of these methodologies is in fact complete (in the sense of covering all of the necessary activities involved in software development life cycle (SDLC)) . Furthermore, most of the existing agent-oriented methodologies suffer from a gap between the design models and the existing implementation languages [43]. One of the steps towards fulfilling this demand is to unify the work of different existing methodologies; work similar to development of the Unified Modeling Language in the area of object-oriented analysis and design. Furthermore, up to this moment, there is no well-established development process with which to construct agent-oriented applications. Therefore, the consequences expected by the agents’ paradigm cannot yet be fully achieved.
The main contribution of this paper is to build a new methodology for the development of multi-agent systems, called MASD (Multi-agent Systems Development). This methodology is expected to be a solid and reliable guide in building and developing such systems, as it is based upon other research, including previously existing methodologies. It aims to develop a complete life-cycle methodology for designing and developing MASs. In addition, MASD attempts to solve some of the problems mentioned in the above paragraph.

The MASD methodology was established based on three fundamental aspects considered to be the foundation for building a solid methodology: concepts, models, and process. Concepts are the necessary MAS properties considered in building the models of the new methodology in a correct manner. Models include modeling techniques, modeling languages, a diagramming notation, and tools that can be used to analyze and design the agent system. Process is a set of steps or phases describing in detail how the new methodology works.

In addition, the MASD methodology bridges the gap between design models and existing implementation languages. It provides refined design models that can be directly implemented in an available programming language, or it can use a dedicated agent-oriented programming platform that provides constructs to implement the high-level design concepts such as Jadex [11], JADE [30], JACK [15], etc. In addition, MASD helps developers to map the developed complex design models into implementation constructs.

Furthermore, the MASD methodology proposes an important concept called triggers and relies heavily on agent roles. The role concept is considered one of the most important aspects representing agent behavior; therefore, MASD assumes each agent can play one or more roles in the system. The trigger concept is also considered as an important aspect as it represents the agent reactivity. Furthermore, MASD considers the social agent aspects by utilizing well-known techniques, such as use-case maps, which enable developers to identify social aspects from the problem specification. Therefore, MASD assumes that agent society architecture is to be derived from the problem specification.

Moreover, MASD methodology is developed based on essential software engineering issues such as precision, accessibility, expressiveness, domain applicability, modularity, refinement, model derivation, traceability, and clear definitions.

Finally, the MASD methodology provides a plain and understandable development process throughout the methodology phases. It captures a holistic view of the system components, and commutative aspects, which should be recognized before designing the methodology models. This is achieved by using well-known techniques such as UCMs and UML-UCDs.
2. MASD Methodology

MASD methodology is developed as a reliable systematic approach that proves a milestone for Software Development Life Cycle (SDLC). Fig. 1 illustrates the process of MASD methodology. The proposed methodology covers the most important characteristics of multi-agent systems and deals with the agent concept as a high-level abstraction capable of modeling a complex system.

MASD includes well-known techniques for requirement gathering and customer communication and links them to domain analysis and design models such as UCMs [13], UML use-case diagrams [47], activity diagrams [46], FIPA-ACL [27], etc. Furthermore, it supports simplicity and ease of use as well as traceability.

The MASD methodology is composed of four main phases: system requirements phase, analysis phase, design phase, and implementation phase. The next sections discuss further each of the four phases. A car rental system [26] is used to demonstrate the process of MASD methodology; the reservation scenario is described as an example.
3. System Requirements Phase

The system requirements phase describes the details of system scenarios in a high-level through System Scenario model. The system scenario model utilizes well-known techniques such as use-cases diagrams (UCDs) [47] and use-case maps (UCMs) [13] to describe the whole system scenario. Such techniques assist in discovering the system components such as agents, objects, roles, resources, etc. and their high-level behavior. The system requirements phase produces a model called the system scenario model.

3.1 System Scenario Model

This model is a starting point for generating more detailed visual descriptions. It describes the whole system scenario in terms of what a system does, but it does not specify how a system does it. The model captures the system's components and the responsibilities that have to be performed by each component within the system. Then, it illustrates how these components collaborate with each other and with the external environment. In addition, it captures the behavior of the system as it appears from the point of view of the external users. To construct this model, some specific, well-known techniques have been used such as Use-Case Diagrams (UCDs) and Use Case Maps (UCMs). These techniques are assembled together in order to understand and obtain a complete system requirement as far as possible. They are chosen to describe system requirements because these techniques are comparatively more appropriate than others for agent-based systems [2,3,4].

3.2 Reservation Scenario Example

The car rental reservation scenario is used as an example. Most rentals are done by advance reservation. The rental period and the car group are specified at the time of reservation. Reservation can be achieved online, by filling web application, by sending an e-mail, or by a phone call.

3.2.1 UCD of Reservation Scenario

UCDs for reservation scenario of the EU-Rent a car Rental Company are constructed. Each use case in the reservation scenario will be described with a diagram as well as describing and clarifying its components. Initially, use case diagrams of the dialogues for the reservation scenario are created.

In order to develop UCDs for reservation scenario, actors involved in the reservation scenario are captured and the use cases associated with those actors. For each
use case the following attributes are identified: description of use case, actors, goal, preconditions and postconditions, triggering events, and extensions and alternatives to the use case.

Actors can be a human or an automated system. A use case is made up of a set of scenarios. Each scenario is a sequence of steps that encompasses an interaction between a user and a system. Fig. 2 describes the UCDs for the reservation scenario. The figure shows two actors: the customer actor and the car rental clerk actor. Each actor represents a role that a user system plays within the system. The customer actor requests the car rental clerk actor to perform an action, such as reserve a car or cancel a reservation. The clerk actor performs actions such as replying to customer requests and so on.

![Use Case Diagrams for reservation scenario.](image)

The semi colon symbol “;” in preconditions, postconditions and triggering events represents the “or” operator. The comma symbol “,” is used to represent the “and” operator.

**Use case Request reservation**

**Use case name:** Request reservation

**Description:** The customer requests a reservation from the car rental clerk
**Actors:** Customer

**Goal:** To make request

**Precondition:** Customer wants reservation

**Postcondition:** Request rejected; Request accepted

**Triggering event:** A customer requests a reservation

**Extensions:**

**Alternatives:**

---

**Use case Request reservation cancellation**

**Use case name:** Request reservation cancellation.

**Description:** The customer requests canceling reservation from the car rental clerk

**Actors:** Customer

**Goal:** To make a request to cancel reservation

**Precondition:** Customer has reservation and requested cancellation

**Postcondition:** Request rejected; Request accepted

**Triggering event:** A customer requests to cancel reservation

**Extensions:**

**Alternatives:**

---

### 3.2.2 UCMs of Reservation Scenario

In this model, use case maps for reservation scenarios of the car rental system are described. This section describes how UCMs can be used to represent and describe the reservation scenario of the car rental system. UCMs are applied to capture the behavior of the system as high-level description and explain how UCMs describe the reservation scenario in visual views. The following scenarios represent interactions between some components in the system. By tracing application scenarios, the high-level view of the system is derived. These scenarios describe functional behavior as UCM paths within the system. This discovers system roles, responsibilities, and plug-ins along the way. To capture UCMs, the following steps are performed:

- **Identify scenarios and major components involved in the system.**
• Identify roles for each component.

• Identify preconditions and postconditions for each scenario.

• Identify responsibilities and constraints for each component in a scenario.

• Identify sub scenarios and replace them with stubs.

• Identify components collaborations for the major tasks.

Fig. 3 demonstrates the important symbols in the UCM notations.

The reservation scenario will be performed between two components of the system called: customer and car rental clerk. Fig. 4 shows the use case map for the reservation scenario.
Fig. 4 Use Case Map for the Reservation Scenario

The customer component represents the customer in the application environment, and the car rental clerk represents the employee of the car rental company. The preconditions for the reservation scenario are:

- A customer wants to rent a car, or
- A reservation exists and the customer wants to cancel the reservation.

When the first precondition is satisfied the scenario starts with the request reservation stub, which hides the detailed information of the request reservation process. The request reservation can be achieved in several ways. For example, it can be done by a phone call, or by filling a web form, or by an Email. Therefore, the request reservation stub is represented as a dynamic stub. Fig. 5 illustrates the plug-ins for the request reservation dynamic stub. After all responsibilities for the request reservation process are performed, the path leads to the car rental clerk component. In this component there is a responsibility called request information, which requests the customer to provide personal information such as address, phone, personal ID, driving license etc. The path leads to the customer component where there is a responsibility called provide info, which provides a confirmation that the customer has filled in the application form for the rental transaction.
After the previous responsibilities are performed, the path leads to the static stub **verify car rentals regulations** which hides the detailed information of the **verify car rentals regulations** process.

Fig. 6 illustrates the plug-in for the **verify car rentals regulations**. In this plug-in the car rental clerk checks whether the customer meets the rental rules.

The **verify car rentals regulations** stub has two outgoing ports. If the customer passed the car rentals regulations, port “b” will be followed, which means that the customer is allowed to reserve a car. Otherwise, port “c” is followed, which means that the customer reservation request is rejected. The path that comes from port “b” leads to the **check customer demands** stub, which hides the detailed information of the **check customer demands** process. This stub checks whether the customer demands are available or not.

At the end of this phase, the general behavior of the system as a whole is described in a high-level view using UCMs scenarios. The interactions that take place between the customer and the car rental system are described and the system requirements are understood using UCDs and UCMs. UCMs and UCDs describe the system without referring to any details regarding the implementation of the system. They provide a clear idea about how the entire system works.
4. Analysis Phase

The objective of the analysis phase is to transform the system requirements into a representation of the system that can be forwarded to the design phase. This phase starts with analyzing the system requirements phase; it utilizes the system scenario model that is constructed by UML use-cases and use-case maps. This system scenario model is considered as a base to produce the models of the analysis phase. The analysis phase is concerned with the description of the agent architecture as well as the MAS architecture. It is divided into two stages; the first stage describes agent architecture, while the second stage describes the MAS architecture.

4.1 Agent Architecture Stage

The agent architecture stage describes the following models:

- Roles model: discovers the roles that agents play or perform in the system, determines responsibilities for each role and specifies activities for each responsibility.
- Agent model: identifies agents in the system and assigns roles to them. Refines the roles to fit agent capabilities.
- Beliefs model: identifies agent beliefs.
- Goals model: identifies agent’s goals.
- Plans model: specifies plans for each goal.
- Triggers model: identifies the triggers of which each agent should be aware. Triggers could be events or change in beliefs.

The MASD methodology requires the development of all models of the agent architecture stage. They are always developed even if the proposed agent system is just as a single agent.

4.1.1 Roles Model

The agent role represents an agent behavior that is recognized, providing a means of identifying and placing an agent in a system. Role modeling is appropriate for agent systems [32] because of the following reasons:

- Roles and role models present a new abstraction that can unify a range of aspects of a system. Software agents, objects, processes, organizations, and people can play roles, and
this is especially important in applications that include all these types of entities, such as information and process management.

- Role models are prototypes that should be documented and shared.
- Role model collectively integrates roles and may be significant for agent design.
- Role model can be used to model adaptive behavior, context switching, mobility and other aspects of agent systems.

Furthermore, the roles model presents the agent system as an organization by considering it as a set of roles working together. Each role has its own responsibilities, and these roles improve and systematize the agent functionality and emphasize social or interactive behavior. The agent can perform more than one role in the system, and more than one agent can perform a role. Roles are encapsulated units that can be transferred easily from one agent to another when the need exists.

### 4.1.1.1 Discovering Roles

This step is responsible for identifying the main roles that are found in the system. In order to be able to capture those roles, UCMs and UML use cases scenarios are to be exploited. In the system scenario model, the UCM components that are involved in the system are identified. These components could be agents, objects, or actors. Roles are discovered by analyzing path segments that cross UCM components in the system scenario model.

This process is performed by passing through all responsibilities and all stubs in all UCM scenarios for each component in the system separately. Roles are also discovered by tracing use cases in UCDs. It is possible then to define the responsibilities and tasks that identify the role or roles which are played by every component of the system (at this moment, only roles are considered and agents are identified later on).

Fig. 7 shows some examples to illustrate how the roles are assigned to UCM components. Components are listed in one row and the roles are listed in a second row. Each role can be associated with one or more components. Each component can be associated with one or more roles.
4.1.1.1.1 Determining Responsibilities of the Roles

Once roles have been identified then the next step is to determine the duties and responsibilities of each role separately. This process starts by tracing scenarios of use case diagrams that have been developed during the system requirements phase. Identifying each actor individually and determine all its use-cases, then transferring them directly to responsibilities in the role that it plays in the system. The scenario paths of the UCMs are then traversed and all the responsibilities and stubs are individually defined and transferred directly to responsibilities and functions that are carried out by the role. This process is an attempt to fully, clearly and accurately capture most of responsibilities and functions of each role.

4.1.1.1.2 Specifying Activities of Each Responsibility

Once the responsibilities and functions of each role are identified, then the following step is to identify all the activities undertaken by each responsibility. This will in fact, represent the functions of the proposed role to be implemented in the system.

The important attributes of the roles model are: role name, role description, responsibilities, permissions, perceptions, obligations and constraints. The role name states the name of the role. The role description is a textual explanation of the function of the role. Responsibilities are the activities that the role is responsible to perform. Obligations are requirements that should be available to enable the role to start its functionality and carry out its responsibilities and activities. Permissions are the authorities related to numbers and types of resources that will be exploited by agents in
the system. **Constraints** are restrictions and boundaries that the role must not violate through executing its tasks. Table 1 shows the attributes of the role model. Obligations, permissions, and constraints can be derived from UCM scenarios.

Developers systematically apply phrase heuristics to classify the statements as permissions, obligations, or constraints. Heuristics include modality (can, may, must), condition key words (if, unless, except) and English conjunctions (and, or, not). Developers must document their interpretation (e.g., “may” indicates a permission) and assign logical meanings to each conjunction. Due to logical disjunctions, each sentence may have multiple obligations, permissions, and constraints.

Once all responsibilities and stubs (request reservation, cancel reservation request, etc.) that the component customer performs have been recognized, then it is quite possible to define and specify the role played by the customer component. Here, it is obvious that it plays a renter role. Table 1 illustrates the renter role that the customer component will play in the reservation scenario. In the same situation, the car rental clerk component plays the rentier role. For simplicity, only the renter role in the reservation scenario is described.

<table>
<thead>
<tr>
<th>Role name:</th>
<th>Renter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role description:</td>
<td>Renter who pays rent to use a car that is owned by the car rental company.</td>
</tr>
<tr>
<td>Responsibilities &amp; Activities:</td>
<td>Res.1 Request reservation</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td></td>
<td>Act.1 Reserve car by phone</td>
</tr>
<tr>
<td></td>
<td>Act.2 Reserve car by E-mail</td>
</tr>
<tr>
<td></td>
<td>Act.3 Reserve car online</td>
</tr>
<tr>
<td></td>
<td>Res.2 Cancel reservation request</td>
</tr>
<tr>
<td></td>
<td>Act.1 Cancel reservation by phone</td>
</tr>
<tr>
<td></td>
<td>Act.2 Cancel reservation by Email</td>
</tr>
<tr>
<td></td>
<td>Act.3 Cancel reservation online</td>
</tr>
<tr>
<td></td>
<td>Res.3 Notify real customer.</td>
</tr>
<tr>
<td></td>
<td>Act.1 Notify customer of canceled reservations</td>
</tr>
<tr>
<td></td>
<td>Act.2 Notify customer of rejected reservations</td>
</tr>
<tr>
<td></td>
<td>Act.3 Notify customer of confirmed reservations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obligations:</th>
<th>The renter should pass rental regulations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permissions:</td>
<td>Null</td>
</tr>
<tr>
<td>Constraints:</td>
<td>The renter should not have more than one reservation at the same time</td>
</tr>
</tbody>
</table>

Table 1  **Renter role for customer component**

### 4.1.1.2 Agent Model

The agent model describes the internal structure of agents within the system and how these agents employ its internal structure to perform its tasks. In the MASD methodology, the building process of the agent model is based on the BDI agent architectures [28]. The BDI architecture is based on agent goals, plans, and beliefs. Agents also have triggers which assist agents to determine the appropriate goal or plan to be selected.
4.1.1.2.1 Identifying Agents

The agent identification step is performed to extract those agents that are assumed to exist within the system. The agents are identified using use case maps that have been developed during the system requirements phase. Agents are identified by analyzing UCM components. A component can be identified as an agent or several components can be combined to constitute one agent. Hence, several roles are combined into one agent.

Fig. 8 shows how roles are assigned to agents. Each agent should be able to fully and logically carry out the roles assigned to it. The developer must select the most appropriate agent for the role. In the car rental system, the customer and car rental clerk components are assigned respectively to a customer agent and a car rental clerk agent. The car rental manager agent represents the branch manager of the car rental company. This agent can play two roles in the system. The first and main role is a director role. The second role is rentier role. It can play the role of rentier when it wants to serve customers as a rentier.

![Fig. 8 Assigning roles to agents](image)

4.1.1.2.2 Refining Roles

The refining roles step is merely used to revise the roles that the agent plays within the system. The refinement process consists of two steps. The first step is to match the roles that are captured in the roles model with agents that play these roles according to the agent's capabilities. The role's responsibilities are classified based on who is responsible for performing them. The second step is to separate, or isolate those responsibilities that are to be carried out by real persons from those responsibilities that are to be carried out by agents on their behalf.

The refining roles process keeps the responsibilities that are to be carried out by the agent within the roles model. The responsibilities that are to be carried out by real users are stated as preconditions. These preconditions are translated into beliefs. Agents use these beliefs to keep track of whether the real person performs those responsibilities or
Agents should be able to sense the environment to check whether these beliefs are changed or not. In other words, an agent may wait for a signal (e.g. a message) that confirms that a task to be performed by the real user has been completed. This refinement process assists developers to build a clear design that is free from confusion and responsibility overlap.

**4.1.1.2.3 Agent Beliefs Model**

The agent beliefs are identified either by the preconditions or by postconditions of the agent’s plans and goals, or by the obligations, permissions, and constraints that were obtained in the roles model. Furthermore, the beliefs can be obtained by tracing the UCM scenarios. The stubs and responsibilities are considered as bases of beliefs that are used to trace whether these stubs and responsibilities are achieved by the agent or not. In addition, the beliefs store information about the internal state of agents. Agent beliefs are classified into two types: constant belief, these beliefs are set beliefs and not allowed to change, and variable beliefs, the values of these beliefs can change many times. Beliefs can be assigned initial values or their values are computed using some kind of expressions or deduced by inference rules. According to Parsons [37] it is reasonable to assume that the values of the beliefs are obtained in several ways:

1) Initial beliefs (basic facts which represent the agent's initial beliefs).
2) Beliefs deduced from previous beliefs by deductive inference rules.
3) Beliefs obtained as answers to questions put to the environment by the agent.
4) Beliefs perceived by a sensor (facts that the agent perceives in its environment).
5) Beliefs communicated by external agents (messages received from other agents).

Also MASD classified the purposes of the beliefs as the following: Storage belief, when the belief is stored and the agent can use it during its lifecycle; maintain belief, when the agent must keep the belief at a certain value e.g. when the agent must keep the temperature constantly at 20 degrees; and achieve belief, the agent stores a required value of the belief and during its lifecycle tries more than one time to check the value of the belief and run plans if the value is not the required value. The agent may not be able to change an achieve belief to a required value but it must keep the value of a maintained belief true at all times. These classifications and its purposes assist the developers to identify the mechanism of how the beliefs are stored and exploited. Accordingly, the agents will be able to reason about the beliefs to select the appropriate actions.

Table 2 shows the beliefs model for the customer agent. A notification must be made about the beliefs that are captured from obligations, permissions, and constraints.
of the role. These beliefs are considered as initial beliefs. Therefore, they do not belong to any goals or plans.

<table>
<thead>
<tr>
<th>Belief</th>
<th>Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent-Id</td>
<td>Constant</td>
<td>Storage</td>
</tr>
<tr>
<td>Customer wants to rent a car</td>
<td>Variable</td>
<td>Storage</td>
</tr>
<tr>
<td>Customer decides to reserve by phone</td>
<td>Variable</td>
<td>Storage</td>
</tr>
<tr>
<td>Customer decides to reserve by E-mail</td>
<td>Variable</td>
<td>Storage</td>
</tr>
<tr>
<td>Customer decides to reserve car online</td>
<td>Variable</td>
<td>Storage</td>
</tr>
<tr>
<td>Reservation confirmed</td>
<td>Variable</td>
<td>Storage</td>
</tr>
<tr>
<td>Reservation rejected</td>
<td>Variable</td>
<td>Storage</td>
</tr>
<tr>
<td>Customer wants to cancel a reservation</td>
<td>Variable</td>
<td>Storage</td>
</tr>
<tr>
<td>The renter should not have more than one reservation at the same time</td>
<td>Variable</td>
<td>Maintain</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 2  Beliefs Model for Customer Agent

4.1.1.2.4 Agent Goals Model

The goal represents a specific target state that the agent is trying to achieve. MASD methodology assumes that the concept of goals in relation to agents has a very strong relation with the BDI architecture. The goals represent the desires and intentions that the agent possesses. The definition of the relationship that links goals with the desires and intentions is formulated as being a similarity or matching relationship. Intentions are considered and defined as being goals that possess previously prepared plans to be executed. Desires are defined as goals with no plans for future execution.

In order to identify the goals of the agent, each responsibility of a given role is converted to a specific goal. Therefore, it can be stated that each responsibility within a specific role is considered a goal for the agent who plays the role. Moreover, each activity within a specific responsibility is the foundation for one plan of the goal.
The agent goals model consists of several fields as shown in Table 3. The first field is the goal title or the goal name. The second field is the priority, which specifies the goal precedence from the execution point of view in cases where there is a need to execute the agent’s goals based upon a priority. The priorities are classified as follows: High, above normal, normal, below normal and low. The third field is preconditions. The preconditions are the conditions that must be satisfied in order to consider this goal. The fourth field is the postconditions. Postconditions are the conditions that are to be satisfied when the goal is fully achieved. They are considered as an indication showing that the goal has been fully accomplished. Finally, the fifth field is the plans through which goals can be achieved. Table 3 illustrates the goals model of the customer agent.

<table>
<thead>
<tr>
<th>Goal</th>
<th>Priority</th>
<th>Preconditions</th>
<th>Postconditions</th>
<th>Plans</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request reservation</td>
<td>High</td>
<td>• Customer wants to rent a car</td>
<td>• Reservation confirmed.</td>
<td>• Reserve car by phone</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Reservation rejected.</td>
<td></td>
<td>• Reserve car by E-mail</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Reserve car online</td>
</tr>
<tr>
<td>Cancel reservation request</td>
<td>Normal</td>
<td>• Customer wants to cancel reservation</td>
<td>• Cancellation confirmed</td>
<td>• Cancel reservation by phone</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Cancel reservation by Email</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Cancel reservation online</td>
</tr>
<tr>
<td>Notify real customer</td>
<td>High</td>
<td>• Real customer must be notified</td>
<td>• Real customer notified</td>
<td>• Notify customer of canceled reservations</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Notify customer of rejected reservations</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Notify customer of confirmed reservations</td>
</tr>
</tbody>
</table>

Table 3 Goals Model for Customer Agent
4.1.1.2.5 Agent Plans Model

After the goals of the agents are identified by the previous step, it is time to describe the plans that should be performed by an agent in order to achieve its goal. Since each agent has a goal or set of goals that it wants or wishes to achieve, a plan or a set of plans for each individual goal must exist. Such a plan needs to be adhered to and followed in order for it to be achieved or performed. Each of those plans consists of a set of tasks to be executed.

4.1.1.2.5.1 Specifying Plans for each Goal

Plans are a deliberately prepared means through which agents achieve their goals. A plan is not just a sequence of basic actions, but it may also include sub-goals. Other plans are executed to achieve the sub-goals of a plan thereby forming a hierarchy of plans. The agent keeps track of the actions and sub-goals carried out by a plan to determine and handle plan failures.

Plans are specified by matching and transforming the activities that belong to the responsibilities within the roles. Each plan consists of a set of tasks. These tasks implement the plan and they complete the required work. Completion and implementation of these tasks is considered a success of the plan. A given goal is considered to be accomplished if at least one plan related to it was implemented. Plans may be executed in a sequential manner, according to the priority of each plan, or in parallel manner.

The plans model consists of six parts: a plan name, preconditions, postconditions, successful internal actions, failed internal actions and a plan body. Optional preconditions define the preconditions of the plan, i.e., what conditions must be true in order to execute the plan. Postconditions are conditions that would be true when the plan completes. Successful internal actions are the actions that are performed if the plan succeeds. Failed internal actions are the actions that are performed if the plan fails. Finally, the plan body defines a tree representing a kind of flow-graph of actions to perform. UML activity diagrams [46] are used to represent the plan body. Activity diagrams are used to model the workflow of the process of the internal operation of the agent system. Activity diagrams illustrate the dynamic nature of the agent system by modeling the flow of control from one activity to another.

The plans for the request reservation goal are constructed. Activity diagrams are used to represent such plans. Table 4 illustrates reserve car online plan for the customer agent and the tasks
that should be performed in this plan. Each activity of the activity diagram represents a task in the plan.

<table>
<thead>
<tr>
<th>Plan-name</th>
<th>Reserve cars online</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>Customer decides to reserve a car online</td>
</tr>
<tr>
<td>Postconditions</td>
<td>Reservation confirmed</td>
</tr>
<tr>
<td>Successful internal actions</td>
<td>Inform the real customer to pick up the car</td>
</tr>
<tr>
<td>Failed internal actions</td>
<td>Try with another car rental company</td>
</tr>
</tbody>
</table>

**Plan body**

<table>
<thead>
<tr>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get car Rental website</td>
</tr>
<tr>
<td>Read car Rental rules</td>
</tr>
<tr>
<td>Verify rules</td>
</tr>
<tr>
<td>Fill in reservation application form</td>
</tr>
<tr>
<td>Approve application</td>
</tr>
<tr>
<td>Close car Rental website</td>
</tr>
</tbody>
</table>

Table 4 Reserve Car Online Plan for the Customer Agent

### 4.1.1.2.6 Agent Triggers Model

This model identifies and captures triggers that occur during system runtime. The idea of the trigger concept is somewhat similar to the ECA rule (event, condition and action) [24]. Triggers are the events and the changes in the beliefs. All events and the change of beliefs that are expected to occur within the system are identified. This model helps developers to identify these events and select the appropriate reaction for such triggers. Triggers can be caused by information coming from the environment, which has an effect on the behavior of agents. According to that information, the agent performs certain actions as a reaction.
Triggers are obtained by capturing and analyzing the beliefs of each agent that can change during runtime. Triggers are also obtained by capturing and identifying expected events that occur in the system during runtime. The selection of triggers that prompts a goal or a specific plan is then followed by transferring them into triggers that motivate the agent to perform some given reactions.

The triggers model consists of four attributes: trigger name, trigger type, trigger activator and the actions. Each trigger is identified by a unique trigger name. The trigger type can be either an event or a change of belief. The trigger activator represents the entity that is responsible for causing such trigger. This entity can be an agent, an object, a human or a resource within the system. Actions are either goals to be achieved or plans to be executed. Therefore, actions are labeled accordingly. For each agent, a trigger model is developed which identifies the triggers that are of interest to it.

Table 5 describes the triggers model for the customer agent. It shows a list of the triggers that might occur in the reservation scenario during system runtime. By looking at the beliefs model of the customer agent and the reservation scenario in UCMs that was built during the system requirements phase, it is found that the scenario begins with belief (precondition) such as “a customer wants to rent a car”. The belief can be true or false. When this belief changes from false to true, it motivates the agent to perform a certain action (to start to execute a specific plan or to start to achieve a specific goal such as “request reservation” goal) as a reaction.

<table>
<thead>
<tr>
<th>Trigger name</th>
<th>Trigger type</th>
<th>Trigger activator</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer wants to rent a car</td>
<td>Change of belief</td>
<td>Real customer</td>
<td>• Request reservation (Goal).</td>
</tr>
<tr>
<td>Customer decides to reserve by phone</td>
<td>Change of belief</td>
<td>Real customer</td>
<td>• Reserve by phone (plan).</td>
</tr>
<tr>
<td>Customer decides to reserve by E-mail</td>
<td>Change of belief</td>
<td>Real customer</td>
<td>• Reserve by E-mail (plan).</td>
</tr>
<tr>
<td>Customer decides to reserve online</td>
<td>Change of belief</td>
<td>Real customer</td>
<td>• Reserve Online (plan).</td>
</tr>
<tr>
<td>Reservation confirmed</td>
<td>Event</td>
<td>Car rental clerk agent</td>
<td>• Notify real customer to pickup the car (plan).</td>
</tr>
</tbody>
</table>


### Table 5 Customer Agent Triggers Model

<table>
<thead>
<tr>
<th>Reservation rejected</th>
<th>Event</th>
<th>Car rental clerk agent</th>
<th>Notify real customer of a rejected reservation (plan).</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

4.2 MAS Architecture Stage

The MAS architecture stage is concerned with constructing the multi-agent system. It starts with building the interaction model, which captures all interactions between agents in the system. This is followed by constructing agent relationships model, which captures the relationships between agents in the system. Finally, the agent services model is constructed. This model exhibits the services that each agent makes available to other agents in the system. It facilitates the access to services that are offered by each agent, and organizes the cooperation between agents in the system.

4.2.1 Agent Interaction Model

The agent interaction model describes the process in which agents exchange information with each other and with their environment. This model is considered as an initial model to represent interactions between agents in a high-level view. In the design phase, an interaction protocol is used to represent interactions in more detail.

The agent interaction model is represented by a notation called interaction diagrams. This notation is suggested by the MASD methodology to describe such interactions. The main function of interaction diagrams is to transform the use case maps scenarios that are developed in the system scenario model into communication messages between the agents. These communications should be comprehensible by the system’s agents. Therefore, a simple agent conversation language is developed based on the speech act theory by Tsohatzidis [45] to help agents understand each other. This speech act theory consists of a communicative act called performative which means purposeful actions performed during conversations between the communicators. For example, the request performative means that the sender requests the receiver to execute some action/actions. On the other hand, the receiver can recognize which type of response is expected from the contents of the conversation. Table 6 illustrates each performative and its description that are used in the agent conversation language.
<table>
<thead>
<tr>
<th>Request</th>
<th>The sender requests the receiver to execute some actions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>The sender asks the receiver for information.</td>
</tr>
<tr>
<td>Inform</td>
<td>The sender gives the receiver.</td>
</tr>
<tr>
<td>Provide</td>
<td>The sender provides the receiver with information already requested by the receiver.</td>
</tr>
<tr>
<td>Call for proposal (cfp)</td>
<td>The sender calls for proposals.</td>
</tr>
<tr>
<td>Propose</td>
<td>The sender proposes to execute specific actions under some conditions.</td>
</tr>
<tr>
<td>accept-proposal</td>
<td>The sender accepts the proposal to execute some actions presented in advance.</td>
</tr>
<tr>
<td>reject-proposal</td>
<td>The sender rejects the proposal presented in advance.</td>
</tr>
<tr>
<td>Agree</td>
<td>The sender agrees to execute some actions.</td>
</tr>
<tr>
<td>Reject</td>
<td>The sender rejects to execute some actions.</td>
</tr>
<tr>
<td>Failure</td>
<td>The sender notifies that it tried to execute some actions but the execution has failed for some reason.</td>
</tr>
<tr>
<td>not-understood</td>
<td>The sender notifies the receiver that it cannot understand the message the sender received.</td>
</tr>
</tbody>
</table>

Table 6  **Performatives for Agent Conversation Language**

Interaction diagrams are developed from the system scenario model by capturing the lines that connect agents (components) in the use case maps diagram and transfer them into conversations (communication acts) between the agents.

Fig. 9 shows the notation of interaction diagrams. The black circle refers to the starting point of the interaction. The path indicates the flow of events in the interaction. The arrow indicates the direction of the flow. The title indicates the performative or the event being exchanged. The symbol “X” indicates the end of the interaction. The agent life bar indicates the life of the agent in the interaction.
Fig. 9  **Notation of Interaction Diagrams**

Fig. 10 illustrates the mapping from UCMS scenarios to interaction diagrams and shows how interaction diagrams are derived from UCMs scenarios. It describes the request of the customer agent to the car rental clerk agent, which then requests more detailed information about the customer. The customer agent replies to the car rental clerk agent who checks the rules of the car rental company and then replies either by acceptance or by rejection.

Agent interaction diagrams give only a partial picture of the system behavior. In order to have a precisely defined system, it is necessary to progress from interaction
diagrams to interaction protocols. Interaction protocols define precisely which interaction sequences are valid within the system.

4.2.2 Agent Relationship Model

The agent relationships model describes relationships between the agents. It helps the agents to make the necessary decisions when cooperation between agents takes place. It also establishes an official framework of duties and responsibilities. The agent relationships model consists of a set of system components (agents, objects, resources, etc.) that are connected together to satisfy and pursue a common goal. The model assists in organizing the coordination between the system agents. This coordination is achieved through a set of commitments realized by formal agreements and contracts that guarantee rights for both parties. Each commitment is directed from one agent giving this commitment towards its contracting partner, who receives this commitment. In addition, this model helps to identify the proper communication protocols that will be chosen for the conversation between agents in the design phase.

The concept of dependency relationships was inspired from Elammari et al. [25], and Yu [50,51]. Agents’ dependency relationships are represented as diagram, where each square represents an agent, and each link between two agents represents the relationship. The link between two agents indicates that one agent (dependant) depends on the other (dependee). The dependency relationship object is called the (dependum). Three types of agent dependency relationships are identified: goal, task, and resource dependency. These relationships can be established either by runtime negotiation or advanced commitment. The model distinguishes among the types of restrictions based on the type of the required relationship between dependant and dependee dependencies. Fig. 11 illustrates the symbols that are used for agent dependency relationships. An arrow represents dependency that is going from a dependee agent to a dependant agent.

Fig. 11 Dependency Relationship Symbols
**Task dependency** represents a relationship in which an agent requires a specific task to be performed. **Goal dependency** represents the relationship in which an agent is dependent on another agent to achieve a specific goal. **Resource dependency** represents the relationship in which an agent is dependent on a supplying agent to provide it with a specific resource. A resource can be physical or informational. These three types of dependencies can be either a **negotiated** or a **committed** relationship. **Negotiated** relationships represent a relationship where an inter-agent negotiation is required to fulfill the dependency. **Committed** relationships indicate that an agent is obligated to provide a service to fulfill the dependency.

Fig. 12 shows dependencies between the customer agent and the car rental clerk agent. The customer agent depends on the car rental clerk agent to handle reservation requests. This dependency is classified as “goal dependency” because the customer agent depends on the reservation agent to achieve this goal, which is called “request reservation”. It also depends on the car rental clerk agent to achieve the canceling reservation goal when the customer wishes to cancel the reservation, or to provide it with reservation information. The car rental clerk agent depends on the customer agent to provide it with personal information.

![Dependency Diagram between Customer Agent and Reservation Agent](image)

Fig. 12 **Dependency Diagram between Customer Agent and Reservation Agent**

### 4.2.3 Agent Services Model

The agent services model provides a standard mean of interoperation between agents in the system. This model is intended to provide a common description of agent services. The model is also intended to define the location of the agent services within a multi-agent system. This guides the agent community to find those services easily. Agent services are captured by means of the messages exchanged between requester agents and provider agents. The main goal of the agent services model is to facilitate access to services that are offered by each agent. Moreover, it organizes the cooperation between agents through constructing formal agreements. An agreement maintains the agents’ rights by providing them the ability to obtain those services in time.
This model is composed of the following five parts: service, agent, expiry date, time of availability, and cost. The service represents the service title. The agent represents the agent offering the service. The expiry date represents the end date of the service. The time of availability is the time that the service should be available to be exploited by other agents. The cost represents the service cost. The agent service model is derived from the use case diagrams that were developed in the system scenario model. Agent services can be derived directly from use case diagrams where each use case can be identified as service. In addition, agent services can be identified as a set of use cases that are compounded into one service. Table 7 illustrates the agents’ services model including the car rental clerk agent services.

<table>
<thead>
<tr>
<th>Service</th>
<th>Agent</th>
<th>Expiry Date</th>
<th>Time of Availability</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reply to customer inquiries</td>
<td>Car rental clerk agent</td>
<td>Open</td>
<td>always</td>
<td>Free</td>
</tr>
<tr>
<td>Handle reservation request</td>
<td>Car rental clerk agent</td>
<td>Open</td>
<td>8:00 am to 8:00 pm</td>
<td>Free</td>
</tr>
<tr>
<td>Handle rental</td>
<td>Car rental clerk agent</td>
<td>Open</td>
<td>8:00 am to 8:00 pm</td>
<td>Free</td>
</tr>
<tr>
<td>Handle car service</td>
<td>Car rental clerk agent</td>
<td>Open</td>
<td>8:00 am to 4:00 pm</td>
<td>Free</td>
</tr>
</tbody>
</table>

Table 7 Agent Services Model

Providing agent services allows agents to search for a certain service that it requires in order to complete its goals or tasks. This model may be updated at runtime by new agents with their services or by new services for agents that already exist.

5. Design Phase

The design phase introduces the detailed representation of the models developed in the analysis phases and transforms them into design constructs. These design constructs are useful for actually implementing the new multi-agent system. The models that were developed in the analysis phase are revised according to the specifications of implementation environment. The main objective of the design phase is to capture the
agent's structural design and system design specifications. The design phase has two steps:

- Creating an agent container.
- Defining inter-agent communications.

### 5.1 Agent Container Model

The first step of the design phase is to construct the agent container, which can be seen as a type specification for a class of instantiated agents. An agent container represents agent behavior, which can be modularized and decomposed into role specifications that are used by an agent. The core part of the agent specification is to define beliefs, goals, plans and capabilities of the agent and place them in the appropriate agent part.

The agent behavior is defined by a container that represents agent roles and its conversations. The agent container simply contains all the important aspects that are needed by the agent to start working. The agent container is composed of several components (beliefs, goals, and plans) where each is represented by a certain model. Each model and its programming aspects will be designed in order to fit with the programming environment.

### 5.1.1 Beliefs

The first part of the agent container is the agent beliefs. The beliefs are considered as “agent beliefbase” which represents the agent knowledge about the environment or the world in which the agent works. The agent's beliefbase can be considered as simple data-storage, responsible for creating new beliefs, belief sets, or removing old ones. This beliefbase is shared among all agents’ plans. The beliefs are classified into two types: beliefs that allow the storage of exactly one fact and beliefs that allow the storage of a related set of facts.

More details are added to the beliefs during the design stage. A new field called `class` is added to indicate the type and the possible values are: Integer, String or Boolean. The `initial value` for the field depends on the type of belief. The `category` field was also added which has “F” for the beliefs that store exactly one fact and has “S” for the belief that store set of facts. These additional fields assist the designers to specify the way in which these beliefs are implemented correctly. Table 8 provides a detailed description of all the additions of customer agent’s belief model from the analysis stage.


<table>
<thead>
<tr>
<th>Belief</th>
<th>Type</th>
<th>Purpose</th>
<th>Class</th>
<th>Initial value</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent _Id</td>
<td>Constant</td>
<td>Storage</td>
<td>String</td>
<td>Customer</td>
<td>F</td>
</tr>
<tr>
<td>Customer wants to rent a car</td>
<td>Variable</td>
<td>Storage</td>
<td>Boolean</td>
<td>True</td>
<td>F</td>
</tr>
<tr>
<td>Customer decides to reserve by phone</td>
<td>Variable</td>
<td>Storage</td>
<td>Boolean</td>
<td>True</td>
<td>F</td>
</tr>
<tr>
<td>Customer decides to reserve by E-mail</td>
<td>Variable</td>
<td>Storage</td>
<td>Boolean</td>
<td>True</td>
<td>F</td>
</tr>
<tr>
<td>Customer decides to reserve car online</td>
<td>Variable</td>
<td>Storage</td>
<td>Boolean</td>
<td>True</td>
<td>F</td>
</tr>
<tr>
<td>Reservation confirmed</td>
<td>Variable</td>
<td>Storage</td>
<td>Boolean</td>
<td>True</td>
<td>F</td>
</tr>
<tr>
<td>Reservation rejected</td>
<td>Variable</td>
<td>Storage</td>
<td>Boolean</td>
<td>True</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 8 Revised agent beliefs model

5.1.2 Goals

The second component in the agent container is the agent goals. The goals that were developed during the analysis stage are classified into four types of goals: perform, achieve, query, and maintain goal types. **Perform goal** is a type of goal where some action is required to be performed. The results of the goal depend on specific actions. Naturally, when no actions could be performed, the goal has failed. **Achieve goal** is a goal where an agent wants to achieve a certain state (target state) of affairs. This target state is represented by a target condition. When an agent gets a new achieve goal (e.g. no waste at given location) that shall be pursued, the agent starts activities for achieving the target state. When the target state is reached then the goal has been achieved. Otherwise, for a yet unachieved goal, plans are selected for execution. Whenever during the plan execution phase, the target condition is reached, then all running plans of that goal can be aborted. **Query goal** is used to enquire information about a specified issue. Therefore, the goal is used to retrieve a result for a query and does not necessarily cause the agent to engage in actions. When the agent has sufficient knowledge to answer the query the result is obtained instantly and the goal succeeds. Otherwise, applicable plans will try to gather the needed information. **Maintain goal** is the goal that has to keep a specific desired
state (its maintain condition) satisfied all the time. When the condition is not satisfied any longer, plans are invoked to re-establish the given state. The maintain goal stays idle until the maintained condition is violated. An example for a maintain goal is to keep the temperature of a nuclear reactor below some specified limit. When this limit is exceeded, the agent has to act and normalize the state.

Based on this classification, a type field is added to the goal model as shown in the following goal model. Table 9 provides a detailed description of the added type field of the agent goals model that was obtained in the analysis phase.

<table>
<thead>
<tr>
<th>Goal</th>
<th>Type</th>
<th>Priority</th>
<th>Preconditions</th>
<th>Postconditions</th>
<th>Plans</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request reservation</td>
<td>Achieve goal</td>
<td>High</td>
<td>• Customer wants to rent a car</td>
<td>• Reservation confirmed</td>
<td>• Reserve car by phone</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Reservation rejected</td>
<td>• Reserve car by E-mail</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Reserve car online</td>
</tr>
<tr>
<td>Cancel reservation request</td>
<td>Achieve goal</td>
<td>Normal</td>
<td>• Customer wants to cancel reservation</td>
<td>• cancellation confirmed</td>
<td>• Cancel reservation by phone</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Cancel reservation by Email</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Cancel reservation online</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 9 Revised agent goals model

5.1.3 Plans

In this section, the plans that have been developed in the analysis phase are refined in order to meet the design specifications. The plans are classified into two types. The first type is called the service plan, a plan that has service nature. An instance of the plan is usually running and waits for service requests. It represents a simple way to react on service requests in a sequential manner without the need to synchronize different plan
instances for the same plan. The second type is called the *passive plan*. This type can be found in all other procedural reasoning systems. Usually, the passive plan is only run when it has a task to achieve. For this kind of plan, triggering events and goals should be specified to let the agent know what kinds of events the plan can handle (as represented in the agent triggers model). When an agent receives an event, the candidate plan(s) should be selected and instantiated for execution. A field called *type* is added to the plan. It identifies the type of the plan and helps developers to decide the suitable mechanism for plan implementation. Table 10 shows the same table that was shown in the analysis phase plus an additional field called type.

<table>
<thead>
<tr>
<th>Plan-name</th>
<th>Reserve cars online</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Passive plan</td>
</tr>
<tr>
<td>Preconditions</td>
<td>Customer decides to reserve car online</td>
</tr>
<tr>
<td>Postconditions</td>
<td>Reservation confirmed</td>
</tr>
<tr>
<td>Successful internal actions</td>
<td>Inform the real customer to pickup the car</td>
</tr>
<tr>
<td>Failed internal actions</td>
<td>Try with another car rental company</td>
</tr>
<tr>
<td>Plan body</td>
<td></td>
</tr>
</tbody>
</table>

Table 10 Reserve car online plan with Plan Type Field
5.1.4 Capabilities

In many situations, goals, beliefs and plans become a common part of a specific task. The agent may need to achieve more than one goal, use more than one belief, and plan. Capabilities are simply a group of goals, beliefs and plans grouped together in a package in order to be used when they are needed to do a certain task in the system. These capabilities are captured from the identified beliefs, goals and plans that are required by the agent capability to implement a specific task. The agent capability is derived from the roles model that has been developed in the analysis phase. The following structure shows how the reservation capability is structured.

**Reservation capability:**

**Begin** // Reservation capability,

**Beliefs:**

- Customer wants to rent a car,
- Reservation confirmed,
- Reservation rejected.
- Customer wants to cancel reservation
- Cancellation confirmed.

**Goals:**

**Begin** // goals

Request reservation goal

**Plans:**

- Reserve by phone call.
- Reserve by E-mail.
- Reserve car online.

Cancel reservation request goal.
5.2 Inter-Agent Communication Model

This model describes in detail possible interactions between agents. To establish communication between agents, agreed on and accepted protocols have to be deployed. The most established standard is the FIPA Agent communication language [27].

This model is derived by transforming the interaction diagrams that were developed in the agent interaction model in the analysis phase into conversation messages according to FIPA protocol patterns. The flow of interaction diagrams is transferred into messages according to the FIPA ACL protocols. The interaction diagrams are classified according to FIPA ACL protocols that the agents should follow to realize successful conversations. Determining the proper protocol for each interaction diagram is considered as the bases in the process of selecting the proper message between agents. Fig. 13 illustrates how an interaction diagram is transferred into messages according to FIPA ACL protocols.
In the following example, the customer agent requests the car rental clerk agent to reserve a car.

(request
  :sender (agent-identifier :Customer_agent)
  :receiver (agent-identifier :Car_rental_clerk_agent)
  :content
  "Reserve group B car for rent"
  :reply-with reserve-car
  :language sl
  :ontology e-Rent
  :protocol fipa-request interaction)
In the following message the car rental clerk agent answers the customer agent that it agrees to the request.

```
(agree
 :sender (agent-identifier :Car_rental_clerk_agent)
 :receiver (agent-identifier :Customer_agent)
 :content
 "((action (agent-identifier :Car_rental_clerk_agent)
     (agree))")
 :protocol fipa-agree
 :language fipa-sl)
```

6. Implementation Phase

The implementation phase is the point in the development process when the program code actually is written. During the implementation phase, the system is built according to specifications from previous phases, as the previous phases provided models ready to be transferred into an implementation. The produced models have a set of design specifications showing how the agent system and its components should be structured and organized. There are several agent frameworks and platforms proposed to develop multi-agent systems. MASD supports some of them (such as JADE [30], JACK [15], MADKIT [35], Jason [10], and Jadex [11]) as tools in the development process. The Jadex platform is recommended because it is Java based, has a FIPA compliant agent environment, and allows the development of goal-oriented agents following the BDI model. The customer agent is used as an example to show how agents are implemented in Jadex, and this section describes in brief how the customer agent is implemented.

Starting up an agent begins with the creation of the agent, and the agent is created according to the agent container developed in the design phase. Each agent container represents an Agent Definition File (ADF) in Jadex.

First, a new agent definition file (ADF) called `customer.agent.xml` is created. In this file, all important agent startup properties are defined in a way that complies with the Jadex schema specification. The first attribute of the agent is its type name, which must be the same as the file name (similar to Java class files). In this case, it is set to Customer. Additionally, one can specify a package attribute, which has a similar meaning
as in Java programs and serves for grouping purposes only. All plans and other Java classes from the agent's package are automatically known and need not to be imported via an import tag. The following XML code describes in brief the customer ADF:

```xml
<!-- CustomerAgent -->

<agent xmlns="http://jadex.sourceforge.net/jadex"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://jadex.sourceforge.net/jadex http://jadex.sourceforge.net/jadex-0.96.xsd"
       name="Customer"
       package="jadex.tutorial">
  <beliefs>
    <belief name="agent_Id" class="String">
      <fact>"customer1"</fact>
    </belief>
    <belief name="customer_wants_to_rent_a_car" class="Boolean">
      <fact>"true"</fact>
    </belief>
    <belief name="customer_decides_to Reserve_car_by_phone" class="Boolean">
      <fact>"true"</fact>
    </belief>
  </beliefs>
  ...
  <goals>
    <achievegoal name="request_reservation">
      <creationcondition>
        $beliefbase.Customer_wants_to_rent_a_car.
      </creationcondition>
      <unique/>
      <deliberation>
        <inhibits ref="cancel_reservation_request"/>
      </deliberation>
      <targetcondition>
        $beliefbase.reservation_confirmed || $beliefbase.reservation_rejected
      </targetcondition>
    </achievegoal>
    <achievegoal name="cancel_reservation_request">
      <creationcondition>
        $beliefbase.customer_wants_to_cancel_reservation.
      </creationcondition>
      <unique/>
    </achievegoal>
  </goals>
  <plans>
    <plan name="reserve_car_by_phone">
      <body> new reserve_car_by_phonePlan() </body>
    </plan>
  </plans>
</agent>
```
In the implementation, a Directory Facilitator (DF) must be built. The DF is an extension of the agent services model, which was developed in the analysis phase. The DF model serves as the “Yellow Pages” for the system agents. The DF allows agents to publish the services they provide so that other agents can find the services and successively use them. Agents may register their services with the DF, or query the DF to find out which services are offered by which agents. An agent is responsible for providing service-related information, e.g. service type, service name, etc. Furthermore, an agent can also deregister or modify its service details. Any agent can interact with a DF to make its services public and to identify agents that provide particular services through the yellow pages. In addition, agents can ask (search) the DF for agents that provide desired services. The DF should provide the agents in the system with the following functions: register, deregister, modify, and search.

7. Future Work

This section lists several topics that are not addressed in this paper. Each topic would clearly benefit from further investigation and, hopefully, would strengthen the MASD methodology. Topics for future investigation include:

- How to utilize the methodology with special domains such as web-base applications, real time systems, etc.
- How to perform testing for the resulting agent system software.
• How to deal with issues related to agent project management, such as: metrics, estimation, schedule, risk, and quality.
• How to deal with agent mobility.

8. Conclusion

Agent-oriented approaches represent an emerging paradigm in software engineering; and there has been a strong demand to apply the agent paradigm in large and complex industrial applications and across different domains. Therefore, the availability of agent-oriented methodologies that support software engineers in developing agent based systems is very important. In recent years, there have been an increasing number of methodologies developed for agent-oriented software engineering. However, none of them are mature and complete enough to fully support the industrial needs for agent-based system development. For these reasons, it is useful to begin gathering together the work of various existing agent-oriented methodologies with the aim of developing a new, more comprehensive, methodology. Thus, this research paper focused on developing a design methodology to assist multi-agent system developers through the entire software development lifecycle, beginning from system requirement phase, and proceeding in a structured manner towards working code. There are few principle strengths of the methodology developed through this research work. It is based on three important aspects—concepts, models, and process—and it is focused toward the specific capabilities of multi-agent systems. At the commencement of research, MASD combined several techniques and concepts into a single, simple, traceable, and structured methodology. These concepts and techniques are represented through a set of models. Most of these models used within the methodology have therefore already been justified and validated within the domain of agents and multi-agent systems. MASD provides extensive guidance in the process of developing the design and for communicating the design within a work group. It is very clear that the existence of this methodology provides great assistance in thinking about and deciding on design issues, as well as conveying design decisions.

The methodology has captured all the requirements of the system in a proper way by combining well-known techniques (UCMs and UCDs) into one extensive model called the system scenarios model. MASD methodology has proposed the trigger concept, which has allowed the representation of agent reactivity. MASD has also proven its ability to support organizational aspects by utilizing the role concept, which provides the work at different levels of abstraction.
9. References


20. K. H. Dam, and M. Winikoff, *Comparing Agent-Oriented Methodologies*, the International Bi -

2004.


http://www.springer.com/978-3-540-40700-3

24. K. R. Dittrich, S. Gatziu and A. Geppert, *The Active Database Management System Manifesto A
Rulebase of ADBMS Features*, Lecture Notes in Computer Science 985 ISBN 3-540-60365-4,


26. EU-Rent, EU-Corporation.: In http://www.businessrulesgroup.org/egsbrg.shtml

27. FIPA ACL.: In http://www.fipa.org/repository/aclspe.cs.html

desire-intention model of agency*, In Möller, J. P., Singh M. P., and Rao, A. S., editors, ATAL,

29. IEEE Std 830-1998: In

of the 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures,


47. *UML Use case diagrams*: In http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_t- utorial/use_case.htm


Authors' Bios

Dr. Tawfig M. Abdelaziz (B.Sc, M.Sc and Ph.D) is a lecturer at the Software Engineering Department, Faculty of Information Technology, Garyounis University, in Benghazi, Libya. He is a qualified Software Engineer and has studied computer science in his first degree (B.Sc) (1989) at the University of Garyounis, and received a M.Sc. (1997) from Technical university of Brno, Brno, Czech Republic and Later Ph.D. (2008) in for Computer Science and Business Information Systems (ICB) from the University of Duisburg-Essen, Essen, Germany, specializing in Multi-agent systems. Currently, he leads the Software Engineering department at the Information Technology Faculty in Garyounis University. His general research interests include Software Engineering and Multi-agent systems.

Dr. Elammari is currently Dean of the Faculty of Information Technology at the University of Garyounis, where he is also a professor in the Department of Software Engineering. He received a B.Sc. and M.Sc. degrees in computer science from Acadia University in Nova Scotia, Canada and a Ph.D. degree in computer science from the Carleton University in Ottawa, Canada. His research interests include Software Engineering, Agent Systems, and e-government.

Rainer Unland is a full professor in computer science at the Institute for Computer Science and Business Information Systems (ICB) at University of Duisburg-Essen where he heads the chair Data Management Systems and Knowledge Representation. He has authored, co-authored and edited more than 120 publications, journals and (text)books in the areas of non-standard/object-oriented database management systems, XML and database systems, object-oriented software development, component-based and aspect-oriented software engineering, advanced transaction management, computer supported cooperative work, (distributed) artificial intelligence, especially Multi-Agent Systems, and industrials informatics. Moreover, he has served as Chair and/or PC member for more than 180 national and international conferences, workshops, and symposia. He is co-founder of the annual International German Conference on Multi-Agent Systems Technology (MATES) and the annual International conference SABRE that serves as an umbrella conference for topics related to software engineering, multi-agent system, Grid computing, and Web-Services and the Internet.

Professor Branki has an enviable record of research and development in the provision of innovation and technological support and advice to numerous organisations including global/international concerns and also Scottish SMEs. Professor Branki has developed a track record in research and development as well as recent and current management of several international projects: COSMOS- Construction On Site Mobile
Operation Support (a 4.5 Million Euros ESPRIT European Funded project), PATRICIA-Performance and Theory of Remote Intelligent Cooperative Agents on the Internet (DAAD: German-British Council supported project), MAST- Multi-Agents in Structural Design (A Polish-British Council Project). He has worked as senior investigator in several national and international projects such FOCUS (Front-end for Open and Closed User Systems), the first UK multimedia project: MUMS (Multi Media User Modeling Systems).