Designing a multi-agent solution for a bookstore with the PASSI methodology

Piermarco Burrafato[1], Massimo Cossentino[2]

[1] DINFO - Dipartimento di Ingegneria Informatica, Università degli Studi di Palermo
Viale delle Scienze, 90128 Palermo, Italy
burrafato@csai.unipa.it
[2] CERE-CNR - Centro di studio sulle Reti di Elaboratori-Consiglio Nazionale delle Ricerche
c/o CUC, Viale delle Scienze, 90128 Palermo, Italy
cossentino@cere.pa.cnr.it

Abstract. PASSI (a Process for Agent Societies Specification and Implementation) is a step-by-step requirement-to-code methodology for designing and developing multi-agent societies integrating design models and concepts from both OO software engineering and artificial intelligence approaches using UML notation. The models and phases of PASSI encompass anthropomorphic representation of system requirements, social viewpoint, solution architecture, code production and reuse, and deployment configuration supporting mobility of agents. The methodology is depicted making use of a well-known bookstore case study. A comparison with the Gaia and MaSE methodologies is also provided.

1 Introduction

At present, several methods and representations for agent-based systems have been proposed [1][2][3][4][22][23]. Robbins et al. [21] have defined the concept of *fidelity*, which is the distance between a model and its implementation. Thus, low fidelity models are problem-oriented, whilst high fidelity models are more solution-oriented.
Since agents are still a forefront issue, some researchers have proposed methods involving abstractions of social phenomena and knowledge [1][3][4] (low fidelity models); others have proposed representations involving implementation matters [2][22][23] (higher fidelity models).

There exists one response to these headways, which is to treat agent-based systems the same as non-agent based ones. However, we refuse this idea because we think it is more natural to describe agents using a psychological and social language. Therefore we believe that there is a need for specific methods or representations tailored for agent-based software. This belief originates from the related literature. To give an example, Yiu and Li [11] say: "an agent is an actor with concrete, physical manifestations, such as a human individual. An agent has dependencies that apply regardless of what role he/she/it happens

to be playing". On the other hand, Jennings [1] defines an agent as "an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives". Also, Wooldridge and Ciancarini see the agent as a system that enjoys autonomy, reactivity, pro-activness, and social ability [12].

Therefore, multi-agent systems (MAS) differ from non-agent based ones because agents are meant to be autonomous elements of intelligent functionality. Consequently, this requires that the agent-based software engineering methods need to encompass standard design activities and representations as well as models of the agent society.

Two more responses exist. They both argue that agents differ from other software but disagree about the differences. The first, proposed by supporters of low-fidelity representations is that what distinguishes agents are their social and epistemological properties; only these need to require different abstractions. The second, proposed by supporters of high-fidelity representations is that what makes the difference is the deployment and interaction mechanisms. DeLoach [2] argues that "an agent class is a template for a type of agent in the system and is analogous to an object class in object-orientation. An agent is an actual instance of an agent class", and "… agent classes are defined in terms of the roles they will play and the conversations in which they must participate".

We also reject these two views in their extreme forms. A designer may want to work at different levels of detail when modelling a system. This requires appropriate representations at all levels of detail or fidelity and, crucially, systematic mappings between them. Because such issues are at present not addressed by any of the existing MAS analysis and design methodologies, we have decided to think at a brand new one.

The methodology we are going to illustrate is named PASSI ("a Process for Agent Societies Specification and Implementation" or "steps" in the Italian language) [17]. It is a step-by-step requirement-to-code methodology for designing and developing multi-agent societies integrating design models and concepts from both the object-oriented software engineering and the MAS using the Unified Modeling Language (UML) notation. It is closer to the argument made above for high-fidelity representations, but addresses the systematic mapping between levels of detail and fidelity. The target environment we have chosen is the standard widely implemented FIPA (Foundation for Intelligent Physical Agents) architecture [9][10]. PASSI is the result of a long period of theoretical studies and experiments in the development of embedded robotics applications (see [6][7][8]). Its precursor, AODPU has been applied in the synthesis of embedded robotics software [5].

In PASSI, an agent is a significant software unit at both the abstract and concrete levels of design. According to this view, an agent is an instance of an agent class. So it is the software implementation of an autonomous entity capable of going after an objective through its autonomous decisions, actions and social relationships. An agent may undertake several functional roles during interactions with other agents to achieve its goals. A role is a collection of tasks performed by the agent in pursuing a sub-goal. A task, in turn, is defined as a purposeful unit of individual or interactive behaviour.

The remainder of this article is structured as follows. Section 2 gives a quick presentation of the methodology's models and provides a justification for PASSI. Section 3 presents the application of PASSI to the "Juul Møller Bokhandel A/S" case study [15] giving a detailed description of the steps and the use of UML notations within each of them. A comparison of PASSI with the methodologies Gaia [4] and the MaSE [2] is discussed in section 4, and some conclusions are presented in section 5.

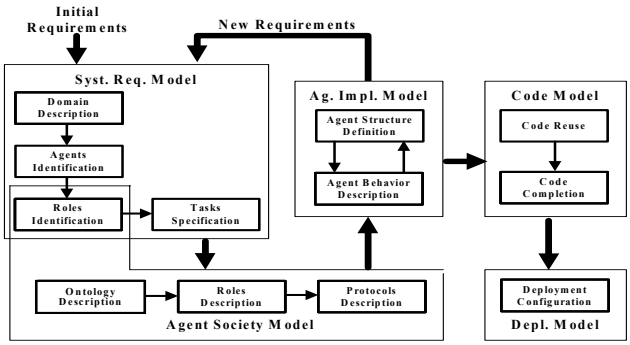
2 A Quick Overview of the PASSI Methodology

The PASSI methodology is made up of five models (see Figure 1) concerning different design levels, and twelve steps in the process of building multi-agent systems.

In PASSI we have adopted UML as the modelling language since it is widely accepted both in the academic and industrial environments. Its extension mechanisms (constraints, tagged values and stereotypes) helps in customizing the representations of agent-oriented designs so as to avoid the adoption of a totally new modelling language.

The models of PASSI are the following:

– *System Requirements Model*. An anthropomorphic model of the system requirements in terms of agency and target. It comprises four steps.
– *Agent Society Model*. A model of the social interactions and dependencies between the agents playing a part in the solution. It necessitates four steps.



Key

D.D. – Domain Description
A.Id. – Agents Identification
R.Id. – Roles Identification
T.Sp. – Tasks Specification
A.S.D. – Agents Structure Definition
A.B.D. – Agents Behaviour Description
O.D. – Ontology Description
R.D. – Roles Description
P.D. – Protocols Description
C.R. – Code Reuse
C.C. – Code Completion
D.C. – Deployment Configuration

Fig. 1. The models and phases of the PASSI methodology

– *Agent Implementation Model*. A model of the solution architecture in terms of classes and methods.
– *Code Model*. A model of the solution at the code level.
– *Deployment Model*. A model of the distribution of the system's parts across hardware processing units, and of their migration.

3 The Steps of the PASSI Methodology

Throughout the following subsections we refer to the "Juul Møller Bokhandel A/S" case study [15] that describes the problems of a small bookstore coping with rapidly expanding Internet-based book retailers. The bookstore has a strong business relationship with the Norwegian School of Management. Nevertheless, there are communication gaps between them. As a consequence the bookseller is in trouble, for example, when pricing the books (due to a lack of information about the number of attendees of some course) or when the School changes the required literature. Besides, there are also problems with the distribution chain. This requires a strong knowledge of distributors' and publishers' processes and practices.

3.1 Domain Description Phase

Although some authors make use of goals in requirements engineering (e.g. [20]), we prefer the approach coming from Jacobson [16] and we describe requirements in terms of use case diagrams. Domain Description Phase, as a result, is a functional description of the system composed of a hierarchical series of use case diagrams. Scenarios of the detailed use case diagrams are then explained using sequence diagrams. Figure 2 shows a Domain Description diagram depicting our analysis for the bookstore case study. Stereotypes used here come from the UML standard. The convention adopted for relationships between the external actors and the system is to direct arrows from the communication's initiator to the participant.

Throughout this paper, we will only examine one scenario. That is the one that takes place every time that the bookstore needs to purchase some books. This may happen, for example, before the beginning of every semester, so as to provision the store with the requested books and therefore anticipate the students' needs; or when some faculty has been known to change the required literature [15], or switch a book from "recommended" into "required". The scenario is triggered from the *Predict Students Needs* functionality (see Figure 2) that uses the *Search Store's Archive* functionality in order to establish whether there are a sufficient number of items of that book in the store, or not. If not, and the book is needed, a new purchase is to be made and therefore the *Provide Books* functionality is invoked.

3.2 Agent Identification Phase

If we look at a MAS as a heterogeneous society of intended and existent agents that in Jackson's terminology can be "bidden" or influenced but not deterministically controlled [13], it is more reasonable to locate required behaviours into units of responsibility from the start. That is why we have put this phase in the *System Requirements Model*.

Agents' identification starts from the use case diagrams of the previous step. According to our definition of agents given in section 1, it is possible to see an agent as a use case or a package of use cases in the functional decomposition of the previous phase. Starting from a sufficiently detailed diagram of the system functionalities (Figure 2), we group one or more use cases into stereotyped packages so as to form a new diagram (Figure 3). In so doing, each package defines the functionalities of a specific agent. Figure 3 shows the part of the Agent Identification diagram that comprises only the agents involved in the scenario we investigate throughout this paper.

Relationships between use cases of the same agent follow the usual UML syntax and stereotypes (see PurchaseMonitor and PurchaseAdvisor agents in Figure 3), whilst relationships between use cases of different agents are stereotyped as "communication". The convention adopted for this diagram is to direct communication relationships between agents from the initiator towards the participant.

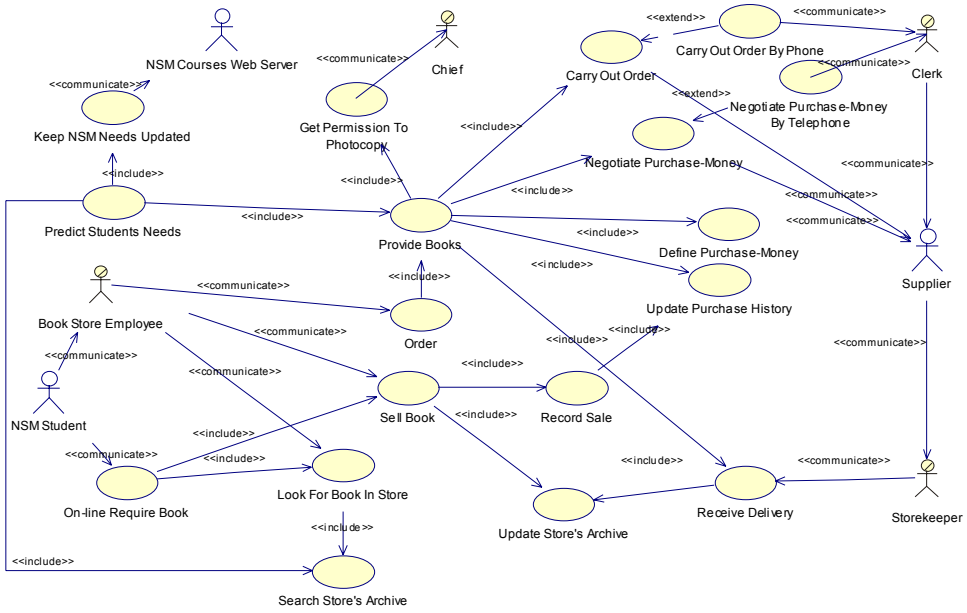


Fig. 2. The Domain Description diagram for the bookstore case study

3.3 Roles Identification Phase

This phase occurs early in the requirements analysis since we now concern more with an agent's externally visible behaviour rather than its structure – only approximate at this step. Because a role is also a social concept, we consider this phase to be also part of the *Agent Society Model* (see Figure 1). Roles identification is based on exploring all the possible paths of the Agents Identification diagram involving inter-agent communication. A path describes a scenario of interacting agents working to achieve a required behaviour of the system. It is composed of several communication paths. A communication path is simply a "communicate" relationship between two agents in the above diagram. Each of them may belong to several scenarios, which are drawn by means of sequence diagrams in which objects are used to symbolize roles.

Figure 4 shows the scenario discussed in section 3.1, arising when a new purchase is required from the role Informer of the PurchaseMonitor agent to the role BooksProvider of the PurchaseManager agent. Each object in the diagram represents a role and we name it with the syntax: <role_name> : <agent_name>.

An agent may participate in different scenarios playing distinct roles in each. It may also play distinct roles in the same scenario (e.g. the Purchaser and the PurchaseAdvisor agents in Figure 4). The messages in the sequence diagram may either signify events generated by the external environment or communication between the roles of one or more agents. A message specifies what the role is to do and possibly the data to be provided or received.

We can describe the scenario as follows:

– The *Informer* informs the *BooksProvider* that the bookstore needs to purchase a specified stock of books.
– Given a list of suppliers for the needed books, the BooksProvider requests the *Consultant* to suggest purchase conditions (number of stocks, purchase-money, etc) on

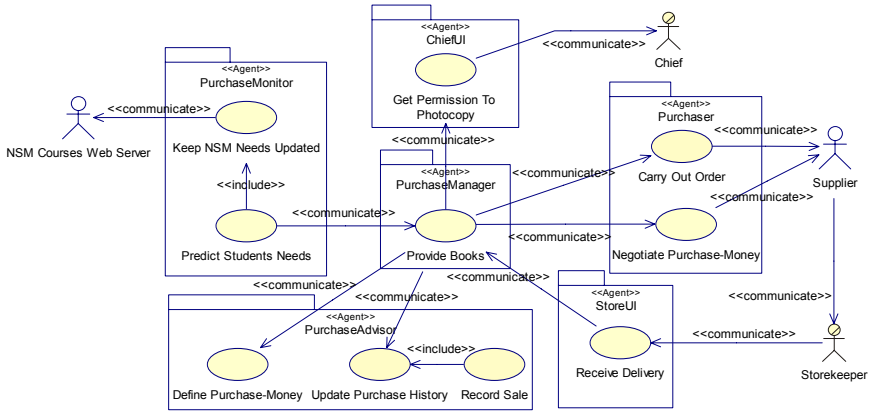


Fig. 3. Part of the Agent Identification diagram obtained from the D.D. phase

the basis of past business.

– When the *Consultant* has returned an advise, the *BooksProvider* gives the *Negotiator* the data about the supplier to negotiate with and the conditions to be negotiated; at the same time it requests the negotiation to be started. The *BooksProvider* is then ready to take care of other requests that may come from the cooperating agents' roles.
– The *Negotiator* negotiates via fax or e-mail (this is the case of the present scenario) and gets the best offer. It then returns it to the *BooksProvider*.
– The *BooksProvider* establishes whether the offer is good enough or not, according to its budget and considerations such as the pricing of the book and the number of students that would then buy it. In this scenario we assume that the offer be good enough and so the *BooksProvider* proposes the *OrderPlacer* to buy the books. Therefore the *BooksProvider* is then ready to take care of other requests.
– When the books are delivered a notification is then forwarded from the *DeliveryNotifier* to the *BooksProvider*.

The rest of the scenario is straightforward. Data contained in the messages of the above sequence diagram are specified in more detail later in the Ontology Description phase (section 3.5).

3.4 Task Specification Phase

At this step, for each agent we focus on its behaviour in order to decompose it into tasks. Tasks generally encapsulate some functionality that forms a logical unit of work.

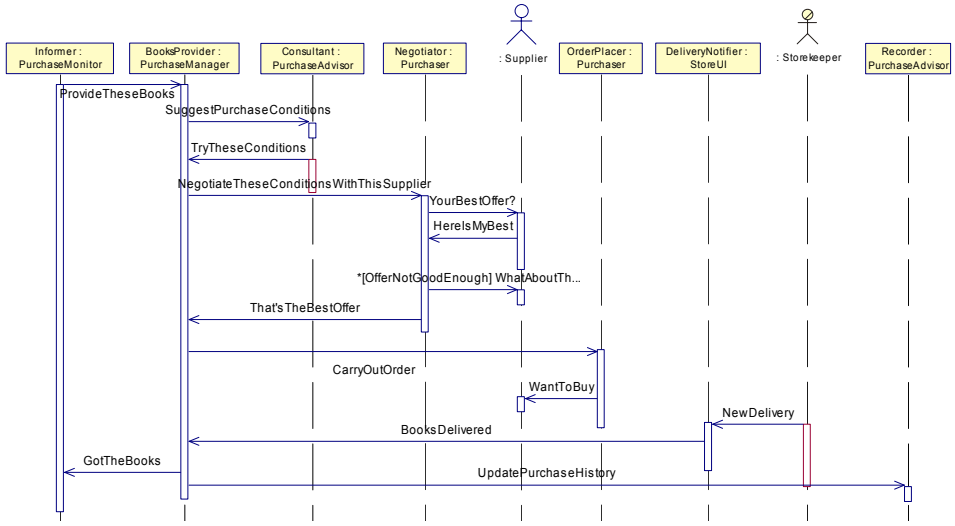


Fig. 4. The Roles Identification diagram for the scenario in which the Purchase Monitor announces the need for a books purchase

For every agent in the model, we draw an activity diagram that is made up of two swimlanes. The one from the right-hand side contains a collection of activities symbolizing the agent's tasks, whereas the one from the left-hand side contains some activities representing the other interacting agents.

A Task Specification diagram (see Figure 5) summarizes what the agent is capable of doing, ignoring information about roles that an agent plays when carrying out particular tasks. Relationships between activities signify either messages between tasks and other interacting agents or communication between tasks of the same agent. The last are not *speech* acts, but rather signals addressing the necessity of beginning an elaboration, that is triggering a task execution or delegating another task to do something. Each activity may be further specified by sequence diagrams, activities diagrams or semi-formal text with fields for preconditions, triggers, etc. In order to yield an agent's T.Sp. diagram we need to look at all of the agent's R.Id. diagrams (i.e. all of the scenarios it participates to). We then explore all of the interactions and internal actions that the agent performs to accomplish a scenario's purpose. From each R.Id. diagram we obtain a collection of related tasks. Grouping them all together appropriately then results in the T.Sp. diagram.
Because drawing a Task Specification diagram for each agent would require much space in the paper, we only depict the one for the Purchase Manager agent (Figure 5). A Listener task is needed in order to pass incoming communication to the proper task. Further tasks are needed to handle all the incoming messages of the R. Id. scenario (see ReceivePurchaseRequest and ReceiveDeliveryNotification tasks in Figure 5 that correspond to the R. Id messages coming from the PurchaseMonitor and StoreUI agents respectively in Figure 4). Likewise, a task is introduced for each outgoing message of the R. Id. scenario (see AskForAdvice, AskNegotiation, AskOrdering, UpdatePurchaseHistory, NotifyEndOfPurchase in Figure 5). Also, extra tasks may be introduced to face with a better decomposition of the agent (see StartPurchase task in Figure 5).

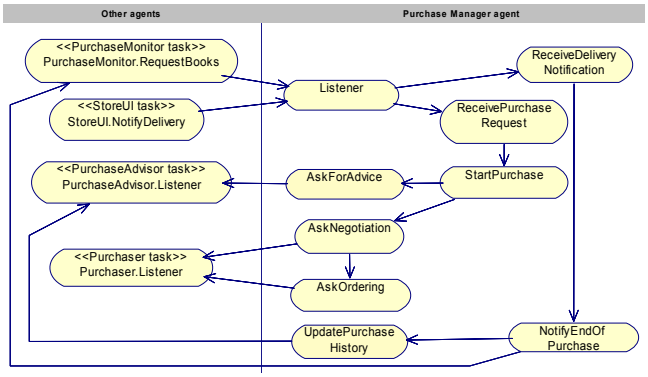


Fig. 5. The tasks of the PurchaseManager agent

3.5 Ontology Description Phase

In this phase we describe the society of agents taking into account the ontological point of view. The Ontology Description yields two diagrams: the Domain Ontology Description (D.O.D.) diagram and the Communication Ontology Description (C.O.D.) diagram; in the former the domain ontology is represented as an XML schema whereas in the latter the communication is explained by a class diagram. Figure 6 shows the D.O.D. diagram; the elements defined here will be used to define the pieces of domain's knowledge. Figure 7 (C.O.D. diagram) depicts the agents with both their knowledge (represented as attributes) and the ontology of their communication.

According to FIPA standards, communications consist of speech acts [18]. Since these are grouped by FIPA in several interaction protocols, according to the intention they respond to, we deduct that in order for a message to make sense it does need to specify a protocol. Furthermore, without a language and an ontology the content of a message would not be understood. So we have the necessity to use association classes in the C.O.D. diagram in order to link the above three elements (protocol, language and ontology) to each communication (see Figure 7). A communication is drawn from the initiator to the participant of the conversation. Each one is deducted from the R.Id. diagram. In Figure 7, for example, the PurchaseManager agent starts a conversation (see QueryForAdvise association class) with the PurchaseAdvisor agent. The Conversation contains the Course ontology, the query protocol and the RDF language. This means that the PurchaseManager wants to perform a speech act based on the FIPA's query protocol in order to ask the PurchaseAdvisor an advice on how to purchase (supplier, number of stocks, number of items per each, purchase-money) provided the Course information.

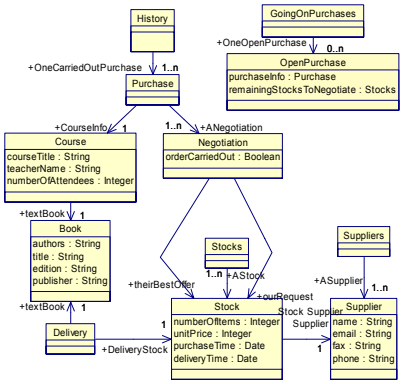


Fig. 6. The Domain Ontology Diagram

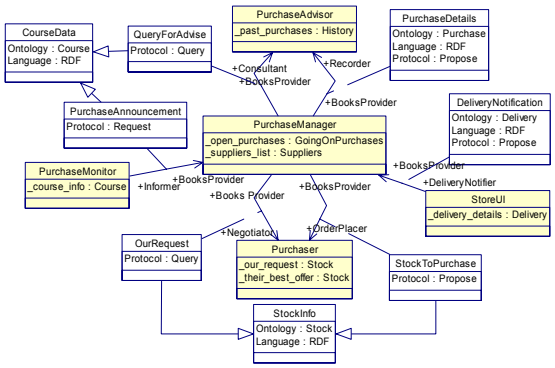


Fig. 7. The Communication Ontology diagram

3.6 Roles Description Phase

This phase models the lifecycle of an agent taking into account its roles, the collaborations it needs and the conversations it is involved in. In this phase we can also introduce the social rules of the society of agents (organizational rules, [3]) and the behavioural laws as considered by Newell in his "social level" [19]. These laws may be expressed in OCL or other formal, or semi-formal manner depending on our needs.

The R.D. phase yields a collection of class diagram in which classes are used to represent roles. Agent is symbolized by a package containing roles' classes (see Figure 8). Each role is obtained composing several tasks in a resulting behaviour. This is the reason why we put tasks in the operation compartment of the related role's class. This makes clear what a role is able to and it can be helpful in the identification of reusable patterns. A R.D. diagram can also show connections between roles of the same agent, representing a changes of role (dashed line with the name [ROLE CHANGE]). This connection is depicted as a dependency relationship because we want to signify the dependency of the second role on the first. Sometime the trigger condition is not explicitly generated by the first role but its precedent appearance in the scenario justifies the consideration that it is necessary to prepare the situation that allows the second role to start. Conversations between roles are indicated by solid lines as we have done in the Communication Ontology Diagram, using exactly the same relationships names.

We have also considered dependencies between agents. Agents are autonomous, so they could refuse to provide a service or a resource. For this reason, the design needs a schema that expresses such matters so as to explore alternative ways to achieve the goals.

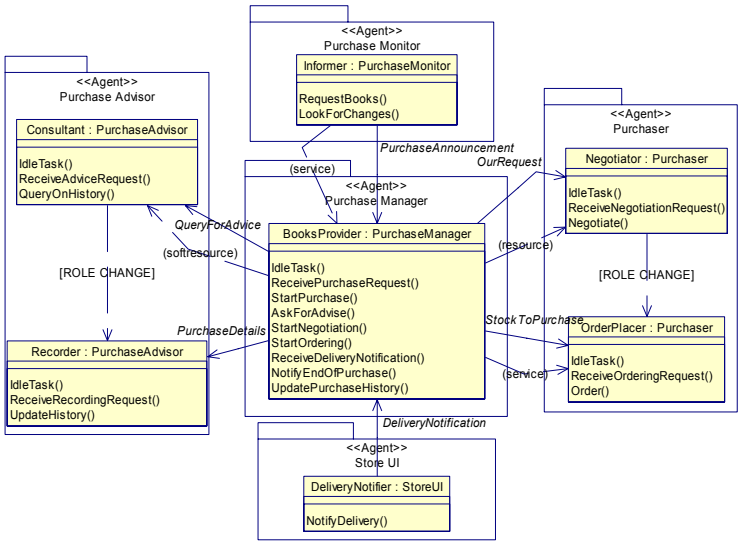


Fig. 8. The Roles Description diagram for our scenario

In order to realize such a schema, we have introduced in Roles Description diagram some additional relationships that express the following kinds of dependency:

- *Service dependency*. A role depends on another to bring about a goal (indicated by a dashed line with the '(service)' name).
- *Resource dependency*. A role depends on another for the availability of an entity (indicated by a dashed line with the '(resource)' name).
- Soft-Service and Soft-Resource dependency. The requested service/resource is helpful or desirable, but not essential to bring about a role's goal (indicated by a dashed line with the '(soft-service)' and '(soft-resource)' names).

In the example of Figure 8 the dependency between the *BooksProvider* role and the *Consultant* role is named (softresource) as the *BooksProvider* would be happy to get an advice from the *Consultant*. But, if the *Consultant* somehow can not satisfy the query the *BooksProvider* will do without.

3.7 Protocols Description Phase

As we have seen in the Ontology Description phase and as specified by the FIPA architecture, a protocol has been used for each communication. All of them are standard FIPA protocols in our example. Usually the related documentation is given in form of AUML sequence diagrams [14]. Hence the designer does not need to specify protocols by their own. In some cases, however, FIPA's existing protocols are not adequate and subsequently some dedicated ones need to be properly designed; this can be done using the same FIPA documentation's approach.

3.8 Agents Structure Definition Phase

This phase influences and is influenced by the Agent Behaviour Description phase as a double level of iteration occurs between them. The Agent Structure Definition phase

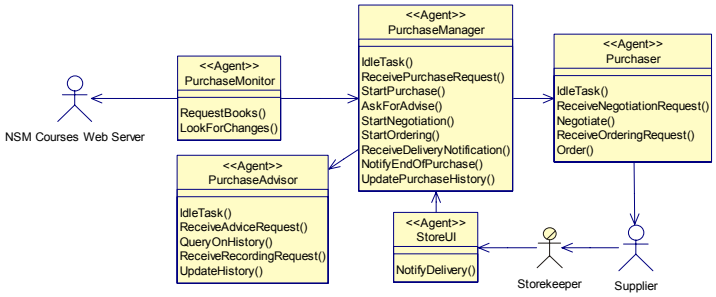


Fig. 9. Part of the Multi-Agent Structure Definition diagram for the bookstore case study

produces several class diagrams logically subdivided into two views: the multi-agent and the single-agent. In the former, we call attention to the general architecture of the system and so we can find agents and their tasks in it. In the latter, we focus on each agent's internal structure, revealing all the attributes and methods of the agent class together with its inner tasks' classes. In the Multi-Agent Structure Definition (MASD) view, one class diagram represents the MAS as a whole (Figure 9). The diagram shows classes, each symbolizing one of the agents identified in the A.Id. phase. Attributes compartments can be used to represent the knowledge of the agent as already discussed in the Communication Ontology diagram, whereas operations compartments are used to signify the agent's tasks. In the Single-Agent Structure Definition (SASD) view, one class diagram (Figure 10) is used for each agent to illustrate the agent's interior structure through all of the classes making up the agent, which are: the agent's main class together with the inner classes identifying its tasks. At this point, we set up attributes and methods of both the agent class (e.g. the constructor and the shutdown method required by FIPA-OS environment) and the tasks' classes (e.g. methods required to deal with communication events such as the handleRequest method of the IdleTask invoked when the agent gets a request communicative act). The result of this stage is to obtain a detailed structure of the software, ready to be implemented almost automatically.

3.9 Agents Behaviour Description Phase

The Agent Behaviour Description phase produces several diagrams that are subdivided into the multi-agent and the single-agent views. In the former, we draw the flow of events by methods invocation and the messages exchange. In the latter, we feature the above methods. At the Multi-Agent Behaviour Description (MABD) view, one or more activity diagrams are drawn to show flow of events between and within both the main agents classes and their inner classes (representing their tasks). We depict one swimlane for each

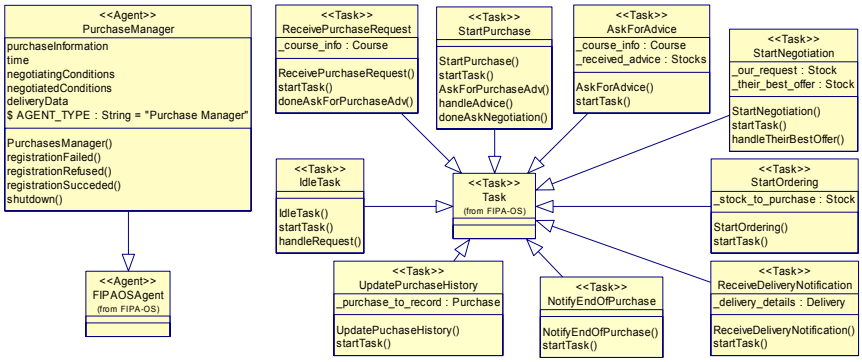


Fig. 10. The Single-Agent Structure Definition diagram for the PurchaseManager agent

agent and for each task. The activities inside the swimlanes indicate the methods of the related class. Unlike DeLoach [2], we need not introduce a specific diagram for concurrency and synchronization since UML activity diagrams' syntax already support it.

Usual transitions of the UML standard are here depicted to signify either events (e.g. an incoming message or a task conclusion) or invocation of methods. A transition is drawn for each message recognized in the preceding phases (e.g. from the R.Id. diagram). In this kind of transition we indicate the message's performative as it is specified in the Communication Ontology Description diagram and the message's content as described in the Domain Ontology Description diagram. This results in having a comprehensive description of the communication including the exact methods involved. Figure 11 shows an example of agents behaviour description. The StartPurchase task of the PurchaseManager agent instantiates the StartNegotiation task by invoking the newTask super-class method. This has to be done in order to ask the Purchaser agent to perform a negotiation with a supplier. The invocation of the StartNegotiation task implies its startTask method to be invoked (according to the FIPA-OS implementation platform we have used). What the startTask method does is just to send a message to the Purchaser agent. This contains the performative request (as required by the FIPA Request protocol) and the content OurRequest (coming from the D.O.D. diagram). The handleRequest method of the Purchaser's IdleTask task receives the incoming communication and sends it to the ReceiveNegotiationRequest task after this one has been instantiated as above. When a task completes its job the done method is invoked.

The Single-Agent Behaviour Description (SABD) view is quite a common one as it involves methods implementation, exactly the ones introduced in the SASD diagrams. We are free to describe them in the most appropriate way (for example, using flow charts, state diagrams or semi-formal text descriptions).

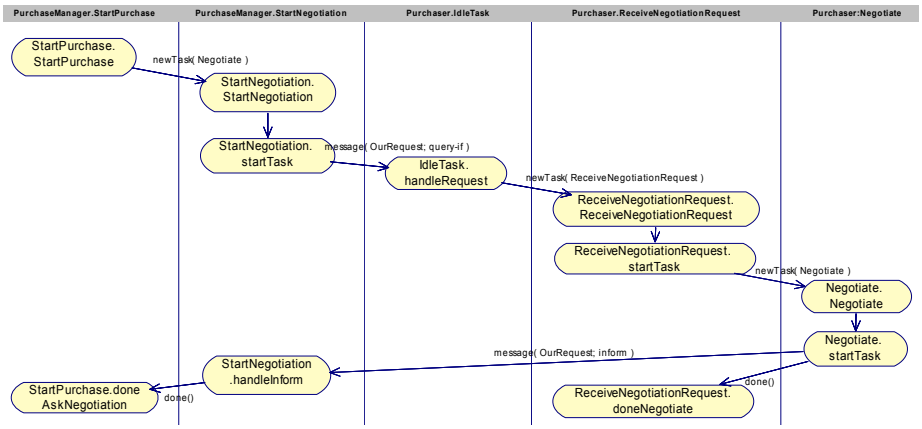


Fig. 11. An example of Multi-Agent Behaviour Description diagram

3.10 Code Reuse Phase

In this phase we try to reuse predefined patterns of agents and tasks. With the term pattern not only do we mean code but also design diagrams. As a matter of fact, the reusing process typically takes place in some CASE tool environment, where the designer looks more at diagrams detailing pattern's libraries rather than rough code. So we prefer to look at patterns as pieces of design and code to be reused in the process of implementing new systems.

We have extended the Rational Rose UML CASE tool by developing an Add-In supporting PASSI. This has proven quite flexible in producing patterns, thanks to its binding of design elements to code. The result has been a collection of reusable code's pieces documented by MABD and SASD diagrams; the former describing behaviours through flow of events and methods invoked; the latter describing software structures (e.g. an agent main class together with its tasks).

Due to the particular FIPA's language structure that delegates a specific task for each specific communication, it has turned out that in our applications, which are FIPA-OS based, some of the most useful patterns are the ones that could be categorized as interaction patterns.

3.11 Code Completion Phase

This phase is the classical work of the programmer, who just needs to complete the body of the methods yielded to this point, by taking into account the design diagrams.

3.12 Deployment Configuration Phase

This phase represents one key aspect in the evolution of PASSI from the previous methodology developed by one of the authors (AODPU, [5]). The D.C. phase has been thought to comply with the requirements of detailing the agents' positions in distributed systems or more generally in mobile-agents' contexts.

The Deployment Configuration diagram illustrates the location of the agents (the processing units where they live), their movement and their communication support. The standard UML notation is useful for representing processing units (by boxes), agents (by components) and the like. What is not supported by UML is the representation of the agent's mobility, which we have done by means of a syntax extension consisting of a dashed line with a "move_to" stereotype.

4 Related Work

The Gaia methodology [4] is a general and comprehensive approach to the agent-oriented analysis and design. It is intended to be applied to a wide range of multi-agent systems, and comprehensive, as it faces the societal and the agency aspects of systems.
The authors of Gaia reject the idea that existing software engineering techniques could be applied to agents' societies. So they have developed a totally new methodology that is specifically tailored to multi-agent systems.

Agents in Gaia are "heterogeneous, in that different agents may be implemented using different programming languages, architectures, and techniques." [4]. Hence, no assumptions about the delivery platform are made.

MaSE [2] stands for "Multiagent Systems Engineering". It is a general-purpose methodology for developing heterogeneous multiagent systems. Authors of this research view their methodology as a further abstraction of the object-oriented paradigm where agents are meant to be a specialization of objects. Therefore, in this view, agents do not communicate by method invocation as objects, but "coordinate with each other via conversations and act proactively to accomplish individual and system-wide goals" [2]. The authors of MaSE see agents "as a convenient abstraction, which may or may not posses intelligence" [2]. This allows intelligent and non-intelligent components to be treated the same way in the framework. The MaSE methodology does not depend on any particular agent architecture, programming language or communication framework.

The first difference between PASSI, Gaia and MaSE is in the analysis and design paradigm. Gaia is a mixed top-down and bottom-up approach, whilst MaSE is implicitly iterative and our approach is instead explicitly iterative. Nowadays, a widely accepted result in software development is the iterative approach as it allows to refine initial requirements and knowledge of systems, and also permits to add new requirements at design time. This is a clear advantage compared to other more rigid approaches.

A second difference is that of commitments about the target agent architecture. PASSI is a requirement-to-code analysis and design methodology characterized by an iterative step-by-step refinement of the system, producing at its final stage a concrete design and implementation based on the FIPA architecture. Gaia, by contrast, regards the output of the analysis and design process as an abstract specification that necessitates being further developed by extra lower-level design methodologies. So does MaSE but, on the other hand, it goes further in the design process if compared with Gaia. Now, one might think that a general approach such as GAIA is more advantageous, given the present proliferation of agent technologies. However, PASSI does not lead to a narrow scope concrete technology but instead it actually yields executable code for a concrete more and more employed standard architecture such as FIPA.

Finally, PASSI and (partially) MaSE make use of a standard modelling language such as UML. In PASSI, UML has been adopted because it is widely accepted both in the academic and industrial environments. Its extension mechanisms (constraints, tagged values and stereotypes) help in customizing the representations of agent-oriented designs

so as to avoid the adoption of a totally new modelling language. Our use of UML is therefore intended to establish a common language such that it can be easily understood and in so doing we kept in our minds experiences coming from the AUML standardization effort. In our approach, arguments that are typical of the object-oriented approach (such as classes, attributes, etc) are only used in the implementation phases as this could not be made in a different way.

5 Conclusions and Further Work

The methodology here proposed proved successful with multi-agent and distributed systems both in robotics and information systems. It has been used in several research projects ([7] and [8] among the others) and in both the course of Robotics at the University of Palermo and the course of Software Development Process at the Georgia Institute of Technology for final assignments. Students appreciated the step-by-step guidance provided by the methodology and have found it rather easy to learn and to use. The implementation environments that we have used was based on the FIPA architecture [9], [10]. We have also applied it in distributed systems [7].

We are currently investigating how multiple perspectives of agency, society, design architecture and physical deployment coexist and inform each other in MAS design

Also, we are working on the development of a CASE tool supporting PASSI and code reuse by means of reusable patterns. This has resulted in a high rate code reusing.

6 Acknowledgements

This research was partially supported by grants from Engineering Ingegneria Informatica S.p.A, Rome (Italy).

References

1. Jennings, N.R. On agent-based software engineering. In Artificial Intelligence, 117 (2000), 277-296.
2. DeLoach, S.A., Wood, M.F., and Sparkman, C.H. Multiagent Systems Engineering. International Journal on Software Engineering and Knowledge Engineering 11, 3, 231-258.
3. F. Zambonelli, N. Jennings, M. Wooldridge. Organizational Rules as an Abstraction for the Analysis and Design of Multi-agent Systems. Journal of Knowledge and Software Engineering, 2001, 11, 3, 303-328.
4. Wooldridge, M., Jennings, N.R., and Kinny, D. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems. 3,3 (2000), 285-312.

5. Chella, A., Cossentino, M., and Lo Faso, U. Designing agent-based systems with UML in Proc. of ISRA'2000 (Monterrey, Mexico, Nov. 2000).
6. Chella, A., Cossentino, M., Infantino, I., and Pirrone, R. An agent based design process for cognitive architectures in robotics in proc. of WOA'01 (Modena, Italy, Sept. 2001).
7. Chella, A., Cossentino, M., Infantino, I., and Pirrone, R. A vision agent in a distributed architecture for mobile robotics in Proc. Of Worskshop "Intelligenza Artificiale, Visione e Pattern Recognition" in the VII Conf. Of AI*IA (Bari, Italy, Sept. 2001).
8. Chella, A., Cossentino, M., Tomasino, G. An environment description language for multirobot simulations in proc. of ISR 2001 (Seoul, Korea, Apr. 2001).
9. O'Brien P., and Nicol R. FIPA - Towards a Standard for Software Agents. BT Technology Journal, 16,3(1998),51-59.
10. Poslad S., Buckle P. and Hadingham R. The FIPA-OS Agent Platform: Open Source for Open Standards. Proc. of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents (Manchester,UK, April 2000), 355-368.
11. Yu, E., Liu, L. Modelling Trust in the i* Strategic Actors Framework. Proc. of the 3rd Workshop on Deception, Fraud and Trust in Agent Societies at Agents2000 (Barcelona, Catalonia, Spain, June 2000).
12. M. Wooldridge, P. Ciancarini. Agent-Oriented Software Engineering: The State of the Art. Agent-Oriented Software Engineering, P. Ciancarini and M. Wooldridge eds., Springer-Verlag, Berlin (2001), 1-28.
13. Jackson, M. Problem Frames: Analyzing and structuring software development problems. Addison Wesley, 2001.
14. J.Odell, H. Van Dyke Parunak, B. Bauer. Representing Agent Interaction Protocols in UML, Agent-Oriented Software Engineering, P. Ciancarini and M. Wooldridge eds., Springer-Verlag, Berlin (2001), 121–140.
15. Espen Andersen (Norwegian School of Management). Juul Møller Bokhandel A/S, 1997. http://www.espen.com/papers/jme.pdf
16. Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley (1992).
17. Cossentino, M., Potts, M. A CASE tool supported methodology for the design of multi-agent systems in Proc. of the 2002 International Conference on Software Engineering Research and Practice (SERP'02) (Las Vegas, USA, June, 2002). (accepted paper).
18. Searle, J.R., Speech Acts. Cambridge University Press, 1969.
19. Newell, A. The knowledge level, Artificial Intelligence, 18 (1982) 87–127.
20. Potts, C. ScenIC: A Strategy for Inquiry-Driven Requirements Determination in proc. of IEEE Fourth International Symposium on Requirements Engineering (RE'99), (Limerick, Ireland, June 1999), 58-65.
21. Robbins, J.E., Medvidovic, N., Redmiles, D.F., and Rosenblum, D.S. Integrating Architecture Description Languages with a Standard Design Method. II EDCS Cross Cluster Meeting in Austin, Texas. www.ics.uci.edu/pub/arch/uml/research/.
22. Aridor, Y., and Lange, D. B. Agent Design Patterns: Elements of Agent Application Design. In Proc. of the Second International Conference on Autonomous Agents (Minneapolis, May 1998), 108–115.
23. Kendall, E. A., Krishna, P. V. M., Pathak C. V. and Suresh C. B. Patterns of intelligent and mobile agents. In Proc. Of the Second International Conference on Autonomous Agents, (Minneapolis, May 1998), 92–99.