

Modeling Notation Source

PASSI

Version: April 28, 2003

Document author: Massimo Cossentino, Luca Sabatucci

Index

1	Introduction	2
	Error! Hyperlink reference not valid.	
3	System Requirements Model	4
	Error! Hyperlink reference not valid.	
3.2	Agent Identification	4
	Error! Hyperlink reference not valid.	
3.4	Task Specification	5
	Error! Hyperlink reference not valid.	
4.1	Ontology Description	7
	Error! Hyperlink reference not valid.	
4.1.2	Communication Ontology Description (COD)	8
	Error! Hyperlink reference not valid.	
4.3	Protocols Description	10
	Error! Hyperlink reference not valid.	
5.1	Multi Agent Structure Definition	11
	Error! Hyperlink reference not valid.	
5.3	Multi Agent Behavior Description	12
	Error! Hyperlink reference not valid.	
6	Code Model	14
	Error! Hyperlink reference not valid.	
6.2	Code Model	14
	Error! Hyperlink reference not valid.	
7.1	Deployment Configuration	15
	Error! Hyperlink reference not valid.	

1 Introduction

PASSI (Process for Agent Societies Specification and Implementation) is a step-by-step requirement-to-code methodology for designing and developing multi-agent societies. The methodology integrates design models and concepts from both Object Oriented software engineering and artificial intelligence approaches.

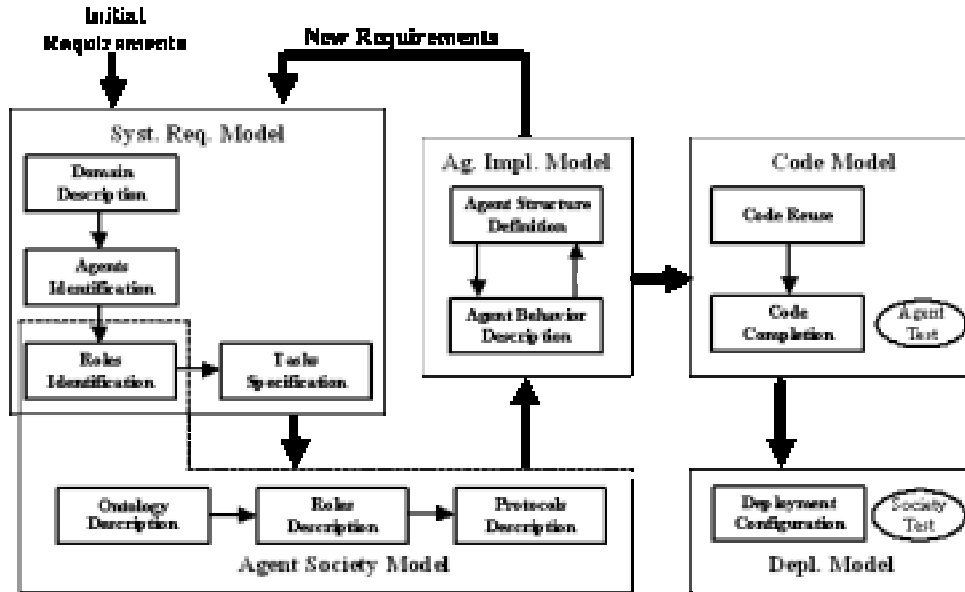


Fig.1: The PASSI methodology

The design process is composed of five models (see Fig. 1): the System Requirements Model is a model of the system requirements; the Agent Society Model is a model of the agents involved in the solution in terms of their roles, social interactions, dependencies, and ontology; the Agent Implementation Model is a model of the solution architecture in terms of classes and methods (at two different levels of abstraction: multi and single-agent); the Code Model is a model of the solution at the code level and the Deployment Model is a model of the distribution of the parts of the system (agents) across hardware processing units, and their movements across the different available platforms.

2 Notation Overview

PASSI models use the following modeling elements:

Agent – an autonomous entity that is composed by roles and has a knowledge. An agent can be seen from different level of abstraction. In the System Requirement Model agents are a logical aggregation of functionalities (Use Case diagrams). In the Agent Society Model agents are composed by role and interact with other agents. In the Agent Implementation Model each agent is a package containing an agent class and one or more task classes.

Role – an agent could play one or more roles in the system. Each role describes an aspect of agent life cycle and it is often related to a service offered by the agent to the society or to the achievement of one of its goals.

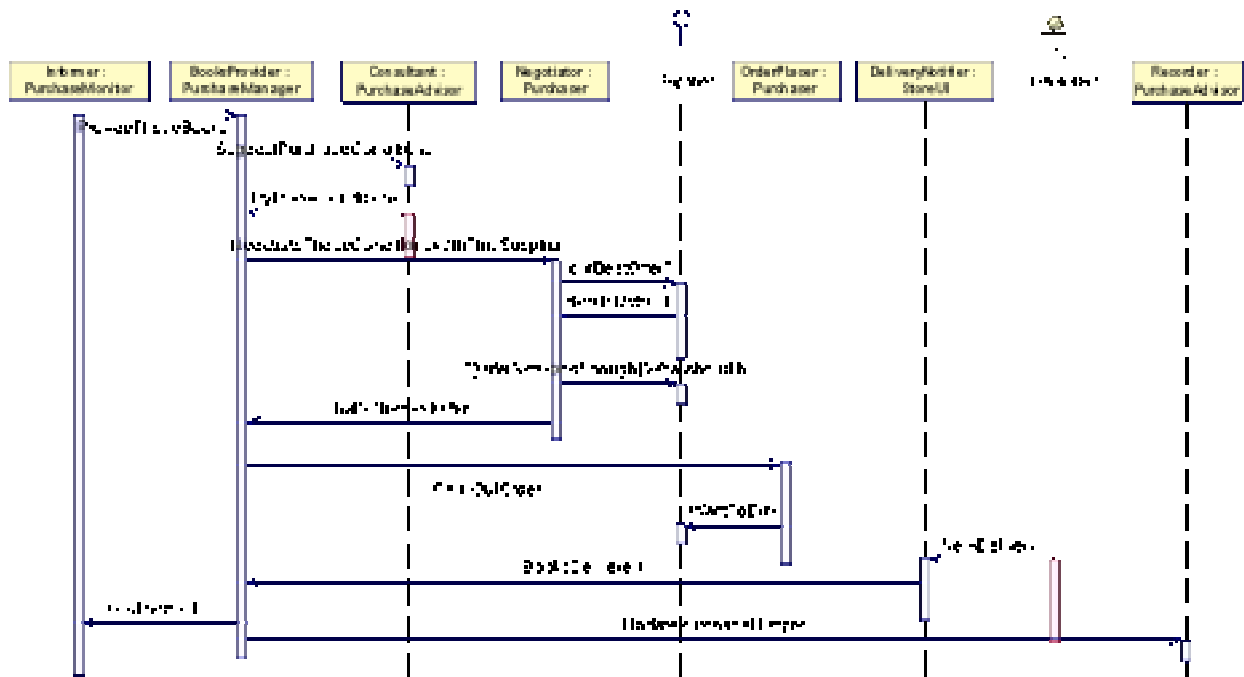
Task – An agent uses tasks to execute its plan(s). Each task is an entity that aims to reach a sub-goal (for example dealing with a communication or executing some transformations on a specific resource).

Communication — a communication is an interaction between two agents. Each communication is described in terms of: ontology (related to the part of knowledge exchanged by the agents), content language and interaction protocol.

Dependency - a relation between two roles of different agents, which indicates that one agent depends, for some reason, on the other in order to attain some goal or deliver a resource.

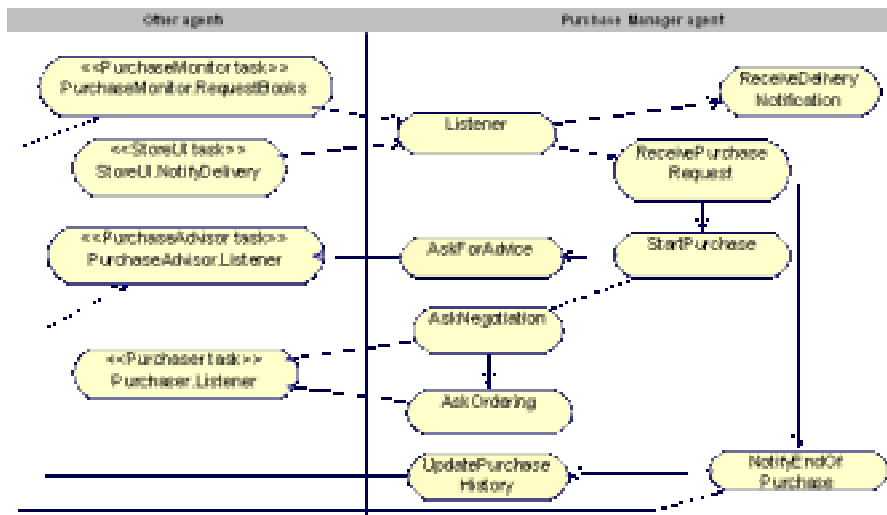
Ontology —An ontology is composed of concepts, actions and predicates.

3.3 Roles Identification



Scenarios describing the interactions among different agents and their roles. The name of each class is in the form: <role name>:<agent name>

3.4 Task Specification



One different activity diagram is drawn for each agent. This diagram describes how the agent can use its tasks to execute its plans.

Each diagram is composed of two swimlanes and contains activities that usually represent tasks of the agent. The right swimlane contains tasks of the agent we are describing (Purchase Manager in the figure above), in the left one we can find tasks of other agents that interact with this one.

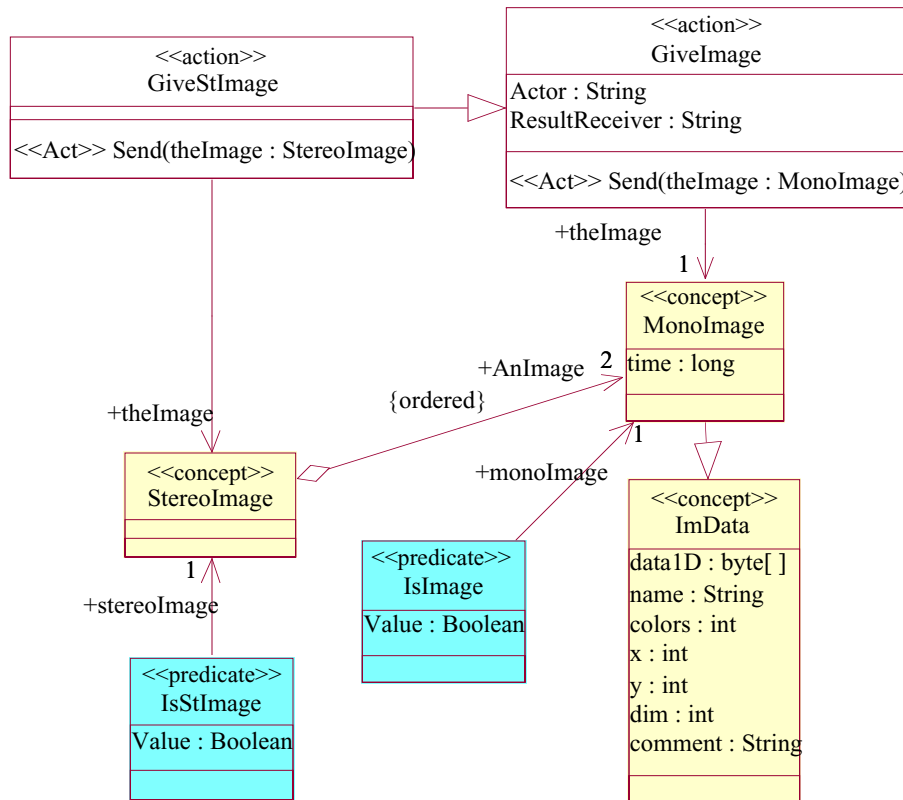
Transitions in the same swimlane describe the flow of control from different tasks while transitions from one swimlane to the other represent communications.

The dynamics of (part of) the system represented in the left swimlane is not complete since the focus of this diagram is the agent represented in the right swimlane; only incoming and outgoing communications from/to the detailed agent are shown in the left swimlane

4 Agent Society Model

4.1 Ontology Description

4.1.1 Domain Ontology Description (DOD)



The ontology is described (using a class diagram) in terms of concepts (fill colour : yellow), predicates (fill colour: light blue) and actions (fill colour: white), see [6][7].

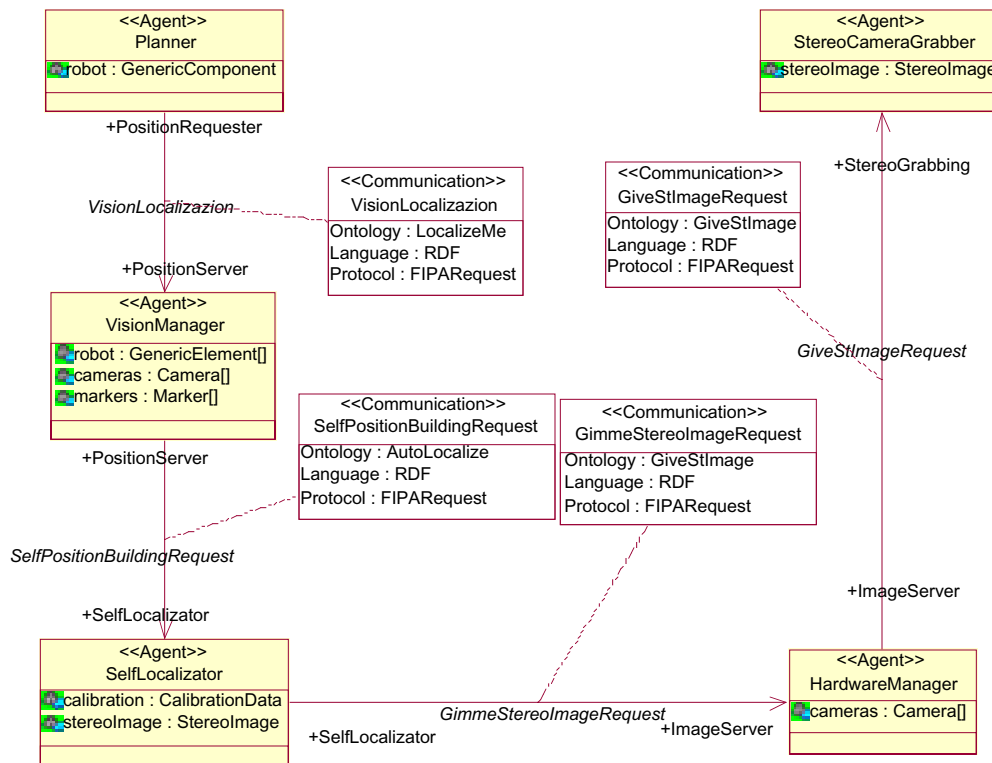
Elements of the ontology can be related using three UML standard relationships:

- Generalization: it permits the generalize/specialization relation between two entities that is one of the fundamental operator for constructing an ontology.
- Association: it models the existence of some kind of logical relationship between two entities. It is possible to specify the role of the involved entities in order to clarify the structure.
- Aggregation: it can be used to construct sets where value restrictions can be explicitly specified; in the W3C RDF standard [6] three types of container objects are enumerated: the bag (an unordered list of resources), the sequence (an ordered list of resources) and the alternative (a list of alternative values of a property). We choose of considering a bag as an aggregation without an explicit restriction, a sequence is qualified by the *ordered* attribute while the alternative is identified with the *only one* attribute of the relationship.

In the previous figure we have a small portion of a robotic vision ontology. *MonolImage* is a specialization of the *ImData* concept with a time stamp (grabbing time). The ordered aggregation of two mono images gives the *StereolImage*. We define the *GiveImage* action in order to allow a robot to ask for an image. The image should be provided by the *Actor* and sent to the *ResultReceiver* (both agents). Predicates are also defined in relation to some existing concepts (*IsImage*, *IsStImage*)

4.1.2 Communication Ontology Description (COD)

The COD diagram is a class diagram and it is mainly composed of two elements: agents and communications.



Each agent (fill colour: yellow) is described in terms of its knowledge (pieces of the ontology described in the previous diagram). There is one relationship between two agents for each communication they are involved in. In each relationship the roles played by the agents during the communication are also reported.

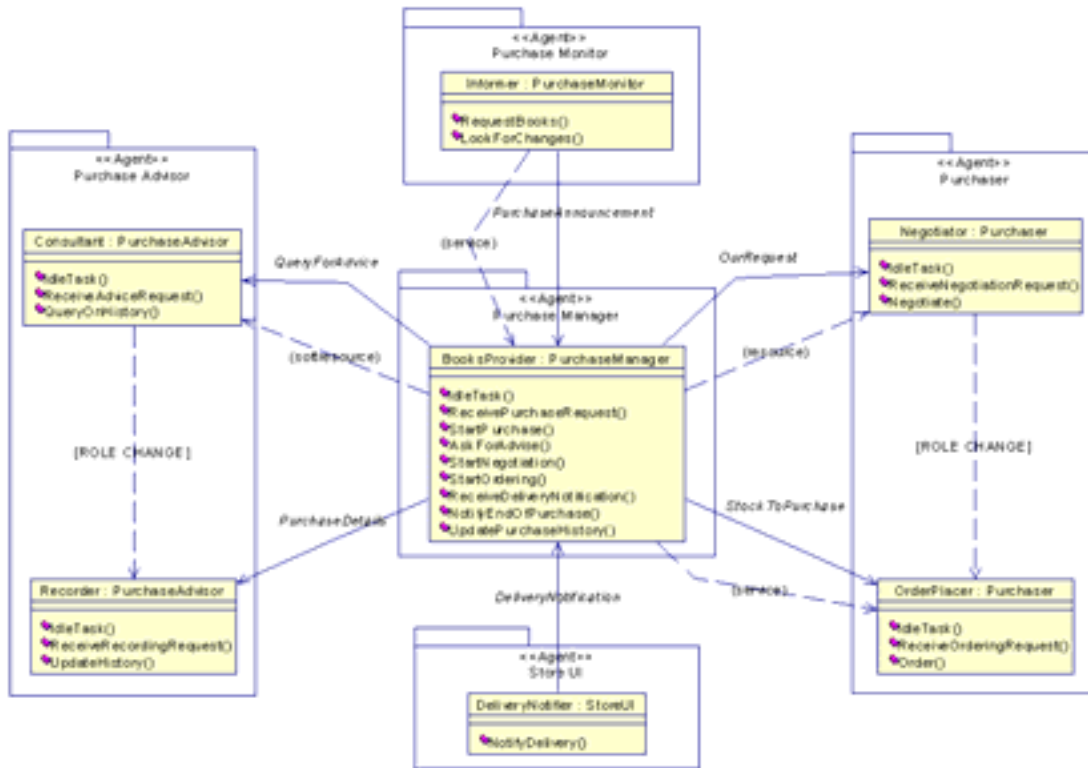
Each communication (fill colour: white) is represented by the relationship among the two agents and it is detailed in the relationship attribute class. The class is identified by a unique name (also reported in the relationship among the two agents) and it is described by the *ontology*, *language* and *protocol* fields.

The *ontology* field refers to an element of the DOD (Domain Ontology Description); the *language* addresses for the content language of the communication while the *protocol* points out the adopted FIPA Interaction Protocol.

In the previous diagram we can see that the *HardwareManager* agent asks for a stereo image to the *StereoCameraGrabber* agent with the *GiveStImageRequest* communication.

This communication refers to the *GiveStImage* action defined in the previous seen DOD diagram, uses the RDF content language and the FIPA Request interaction protocol.

4.2 Roles Description



We represent the Role Description diagram as a class diagram where roles are classes grouped in packages representing the agents.

Roles can be connected by relationships representing changes of role, dependencies for a service or the availability of a resource and communications. Each role is obtained composing several tasks for this reason we specify the tasks involved in the role using the operation compartment of each class.

More in details:

- Classes represent roles of the agent. They are grouped in packages that stand for the agent.
- Relationships among roles can be of 3 different kinds:
 - Communications. Represented by a solid line directed from the initiator to the participant. Names of communications come from the Communication Ontology Description diagram.
 - Dependencies. Like in i* [9], we can have service or resource dependencies. A *service* dependency shows that a role depends on another to bring about a goal (indicated by a dashed line with the *service* stereotype). In the *resource* dependency, a role depends on another for the availability of an entity (indicated by a dashed line with the *resource* stereotype). We can also have *soft-service* and *soft-resource* dependencies; in this case the requested service/resource is helpful or desirable, but not essential to achieve a role’s goal.
 - Role changes. This connection is depicted as a dependency relationship because we want to signify the dependency of the second role on the first. Sometimes 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. We use OCL or semi-formal text to express the trigger condition.

4.3 Protocols Description

We use sequence diagram to describe interaction protocols like in AUML.

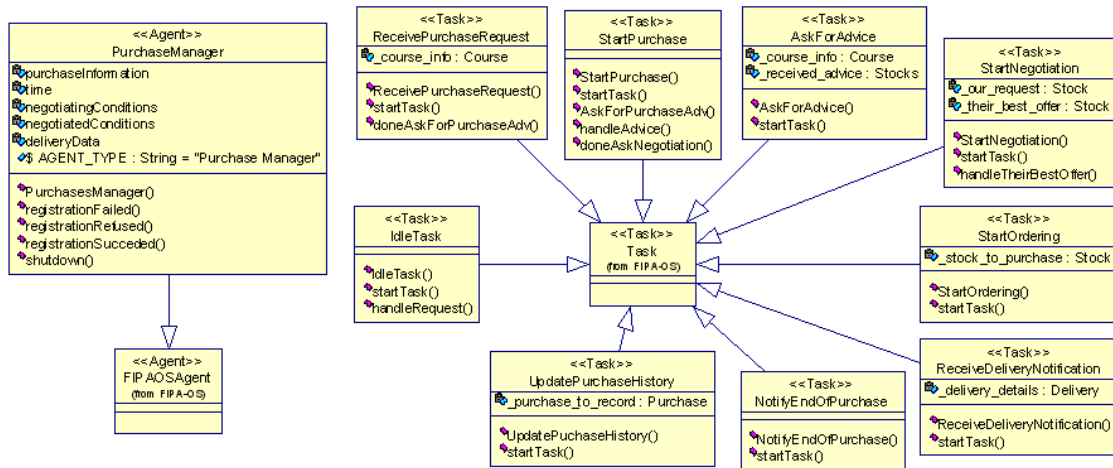
5 Agent Implementation Model

5.1 Multi Agent Structure Definition



The class diagram contains classes and actors. Each class symbolizing one agent of the system. Attributes compartments can be used to represent the knowledge of the agent (referring to entities defined in the Domain Ontology Description), whereas operations compartments are used to signify the agent's tasks. The relations indicates the flow of exchanged information (communications)

5.2 Single Agent Structure Definition

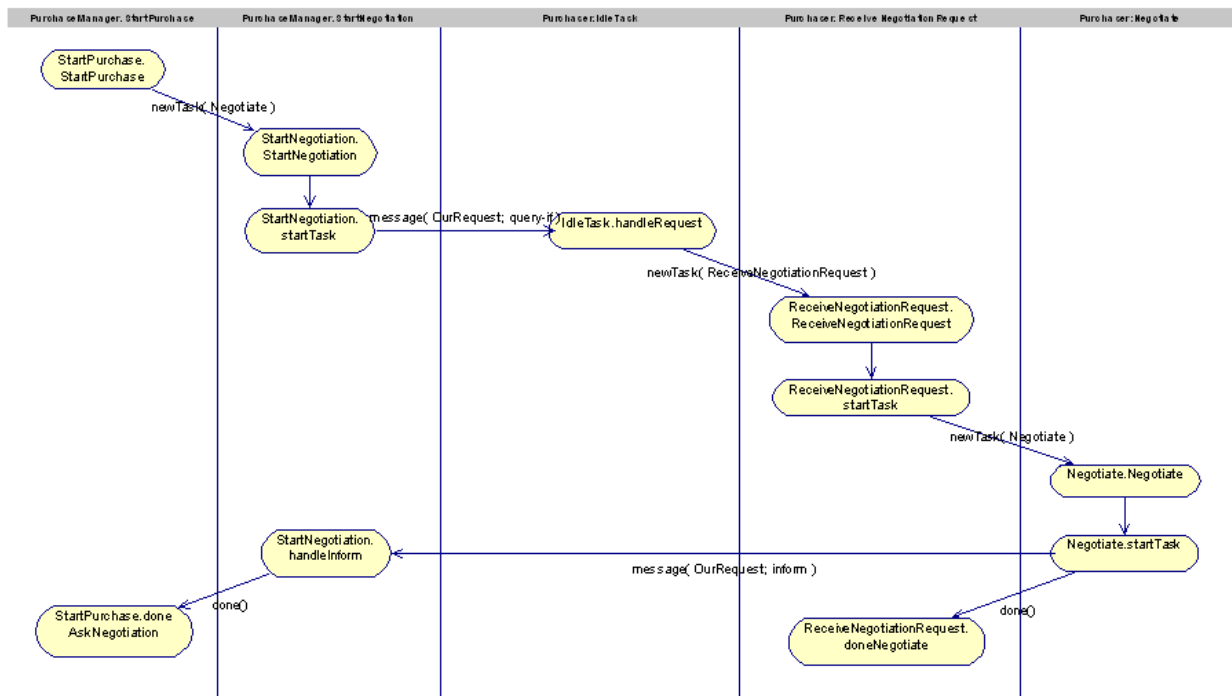


One different class diagram is drawn for each agent. This diagram describes the structure of the agent and all of its tasks.

Each class represents the agent or one of its task. The agent base class and the tasks are obtained specializing the base agent and task classes of the implementation platform (FIPA-OS in the figure above).

Attributes and methods are the elements that will constitute the real (code) implementation of the system. It is possible to automatically produce code from this diagram with many commercial tools.

5.3 Multi Agent Behaviour Description



This activity diagram can be used to show the flow of events among and within both the main agents classes and their inner classes (representing their tasks).

We use one swimlane for each agent and for each of its tasks. The activities inside the swimlanes indicate the methods of the related class.

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.

If the transition is related to a conversation, the label reports the message's performative and content.

5.4 Single Agent Behaviour Description

This phase is quite a common one as it involves methods implementation. We are free to describe them in the most appropriate way (for example, using flow charts, state diagrams or semi-formal text descriptions).

6 Code Model

6.1 Code Reuse

The repository of patterns (see also [3],[8]) is described as reported below:

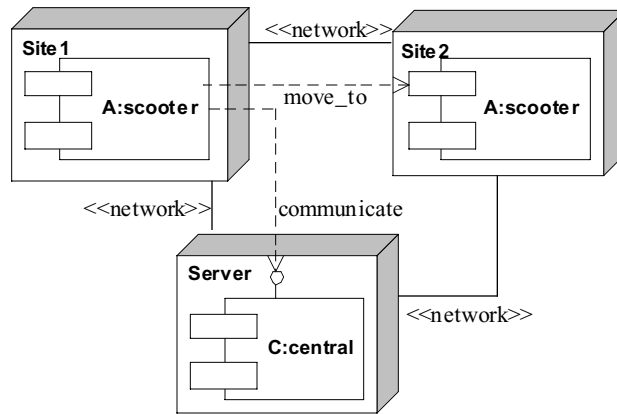
Name	The name of the pattern
Classification	The classification of the pattern according to the following criteria and related categories: <ul style="list-style-type: none"> • Application context: Action, Behavior, Component and Service pattern • Functionality: Access to local resource, Communication, Elaboration, Mobility
Intent	A description of what the pattern does and its rationale and intent
Motivation	A scenario that illustrates a design problem and how the agents and their tasks in the pattern solve the problem.
Pre-conditions	The initial situation in which the pattern can be applied.
Post-conditions	The consequences of the application of the pattern: what changes the pattern introduces into the system
Structure	A graphical representation of the structure of the agent and its tasks (usually done with a class diagram)
Participants	A description of the agents involved in the pattern and their roles
Collaborations	A (graphical) representation of the collaborations of the agents involved in the pattern (if any)
Implementation availability	Availability of the implementation code for the FIPA-OS/JADE platforms. Availability of the UML diagrams of the solution (XML) for importing them in the existing system design
Implementation description	Comments on the most significant code fragments to illustrate the pattern implementation in the specific agent platforms
Implementation Code	FIPA-OS/JADE code of the solution
Related Patterns	Patterns that should be used in conjunction with this one

6.2 Code Model

Just the code of the solution

7 Deployment Model

7.1 Deployment Configuration



We use a deployment diagram to describe the dissemination of agents across the available platforms and their movements.

Platforms are described as processing nodes, agents as components and their communications (if they are not too many) connects the initiator with the interface symbol of the participant.

We represent the agent movements with a relationship (with the *move_to* label) from the agent in the previous position (processing node) to the agent in the target (after moving) position. We often attach notes to these relationships to specify the pre-conditions for the movement of the agent.

8 References

- [1] PASSI documentation website: www.csai.unipa.it/passi
- [2] M. Cossentino - "Different perspectives in designing multi-agent systems" - AGES '02 workshop at NODE02 - 8-9 October 2002 - Erfurt, Germany (www.csai.unipa.it/cossentino/paper/AGES02.pdf)
- [3] M. Cossentino, P. Burrafato, S. Lombardo, L. Sabatucci - "Introducing Pattern Reuse in the Design of Multi-Agent Systems" - AITA'02 workshop at NODE02 - 8-9 October 2002 - Erfurt, Germany (www.csai.unipa.it/cossentino/paper/AITA02.pdf)
- [4] M. Cossentino, C. Potts - "A CASE tool supported methodology for the design of multi-agent systems" - The 2002 International Conference on Software Engineering Research and Practice (SERP'02) - June 24 - 27, 2002 - Las Vegas (NV), USA (www.csai.unipa.it/cossentino/paper/SERP02.pdf)
- [5] P. Burrafato, M. Cossentino - "Designing a multi-agent solution for a bookstore with the PASSI methodology" - Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002) - 27-28 May 2002, Toronto (Ontario, Canada) at CAiSE'02 (www.csai.unipa.it/cossentino/paper/AOIS02.pdf)
- [6] Resource Description Framework. (RDF) Model and Syntax Specification. W3C Recommendation. 22-02-1999. <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>
- [7] FIPA RDF Content Language Specification. Foundation for Intelligent Physical Agents, Document FIPA XC00011B (2001/08/10). <http://www.fipa.org/specs/fipa00011/XC00011B.html>
- [8] AgentFactory website: <http://mozart.csai.unipa.it/af>
- [9] 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).