Domain Ontology Description

Process Fragment

Author(s): M. Cossentino, V. Seidita

Last saved on: 23/11/10 15:02

Index

Fragment Description

Fragment Goal

Describing the agent environment in terms of an ontology composed by concepts, predicates and actions.

Fragment Origin

The presented fragment has been extracted from *PASSI* (Process for Agent Societies Specification and Implementation) design process.

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.

PASSI has been conceived in order to design FIPA-compliant agent-based systems, initially for robotics and information systems applications.

Systems designed by using the PASSI process are usually composed of peer-agents (although social structures can be defined). According to FIPA specifications agents are supposed to be mobile, and they can interact by using semantic communications referring to an ontology and an interaction protocol.

PASSI is suitable for the production of medium-large MAS (up to a hundred agent-kinds each one instantiated in an unlimited number of agents in the running platform).

The adoption of patterns and the support of specific CASE tools (PTK) allows a quick and affordable production of code for the JADE platform. This encourages the use of this process even in time/cost-constrained projects or where high quality standards have to be met.

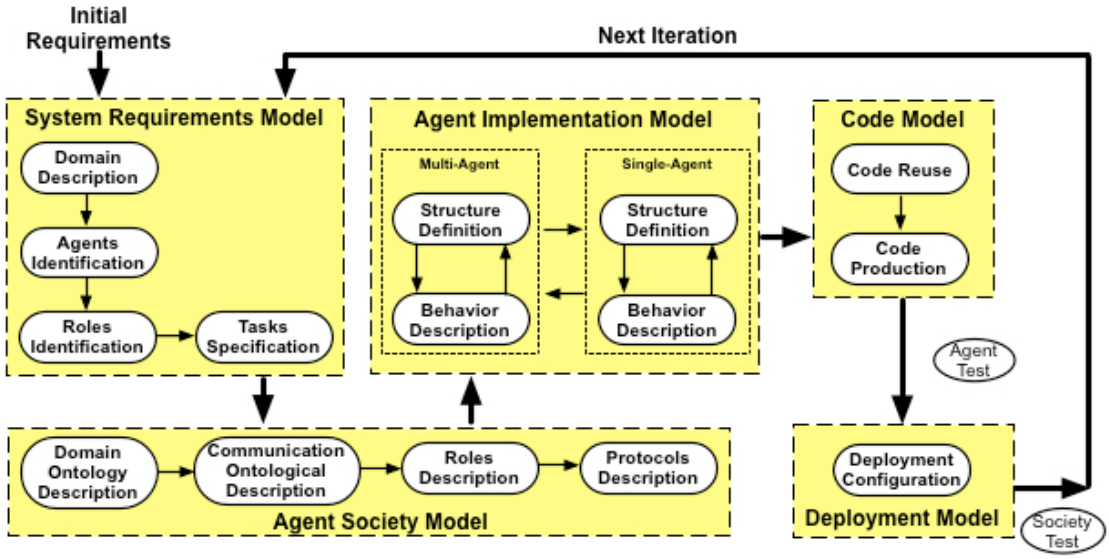


Figure 1. The PASSI design process

The design process is composed of five models (see Figure 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 (i.e. agents) across hardware processing units, and their movements across the different available platforms.

Useful references about the PASSI process are the following:

- M. Cossentino. From Requirements to Code with the PASSI Methodology. In Agent-Oriented Methodologies, B. Henderson-Sellers and P. Giorgini (Editors). Idea Group Inc., Hershey, PA, USA. 2005.
- M. Cossentino, S. Gaglio, L. Sabatucci, and V. Seidita. The PASSI and Agile PASSI MAS Meta-models Compared with a Unifying Proposal. Lecture Notes in Computer Science, vol. 3690. Springer-Verlag GmbH. 2005. pp. 183-192.
- M. Cossentino and L. Sabatucci. Agent System Implementation in Agent-Based Manufacturing and Control Systems: New Agile Manufacturing Solutions for Achieving Peak Performance. CRC Press, April 2004.
- M. Cossentino, L. Sabatucci, and A. Chella. Patterns reuse in the PASSI methodology. In Engineering Societies in the Agents World IV, 4th International Workshop, ESAW 2003, Revised Selected and Invited Papers, volume 3071 of Lecture Notes in Artificial Intelligence. Springer-Verlag, 2004. pp. 294-310
- M. Cossentino, L. Sabatucci, A. Chella - A Possible Approach to the Development of Robotic Multi-Agent Systems - IEEE/WIC Conf. on Intelligent Agent Technology (IAT'03). October, 13-17, 2003. Halifax (Canada)
- Chella, M. Cossentino, and L. Sabatucci. Designing JADE systems with the support of case tools and patterns. Exp Journal, 3(3):86-95, Sept 2003.

The Process lifecycle

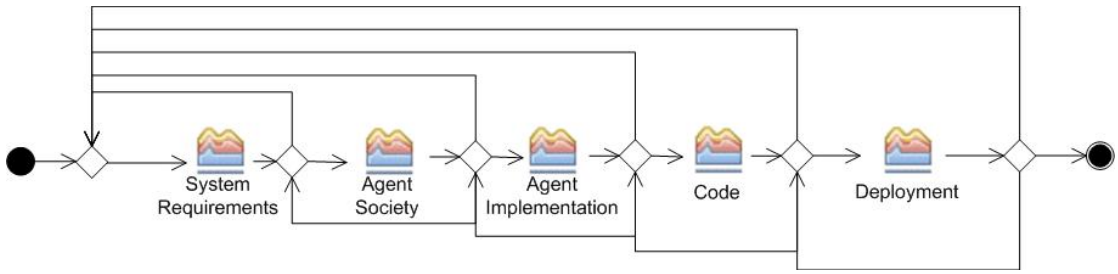


Figure 2. The PASSI process phases

PASSI includes five phases (see Figure 2) arranged in an iterative/incremental process model:

- System Requirements: It covers all the phases related to Req. Elicitation, analysis and agents/roles identification
- Agent Society: All the aspects of the agent society are faced: ontology, communications, roles description, Interaction protocols
- Agent Implementation: A view on the system's architecture in terms of classes and methods to describe the structure and the behavior of single agent.
- Code: A library of class and activity diagrams with associated reusable code and source code for the target system.
- Deployment: How the agents are deployed and which constraints are defined/identified for their migration and mobility.

Each phase produces a document that is usually composed aggregating UML models and work products produced during the related activities. Each phase is composed of one or more sub-phases each one responsible for designing or refining one or more artefacts that are part of the corresponding model. For instance, the System Requirements model includes an agent identification diagram that is a kind of UML use case diagrams but also some text documents like a glossary and the system use scenarios.

Fragment Overview

Consider the PASSI process (see Figure 2) and the phase “Agent Society” with its outcome “Agent Society Model”. Now, let us consider the work definition “Domain Ontology Description” (red colored in Figure 3) and the consequent outcome (the “Domain Ontology Description” composite document). This is a fragment whose aim is to design the ontology of the system.

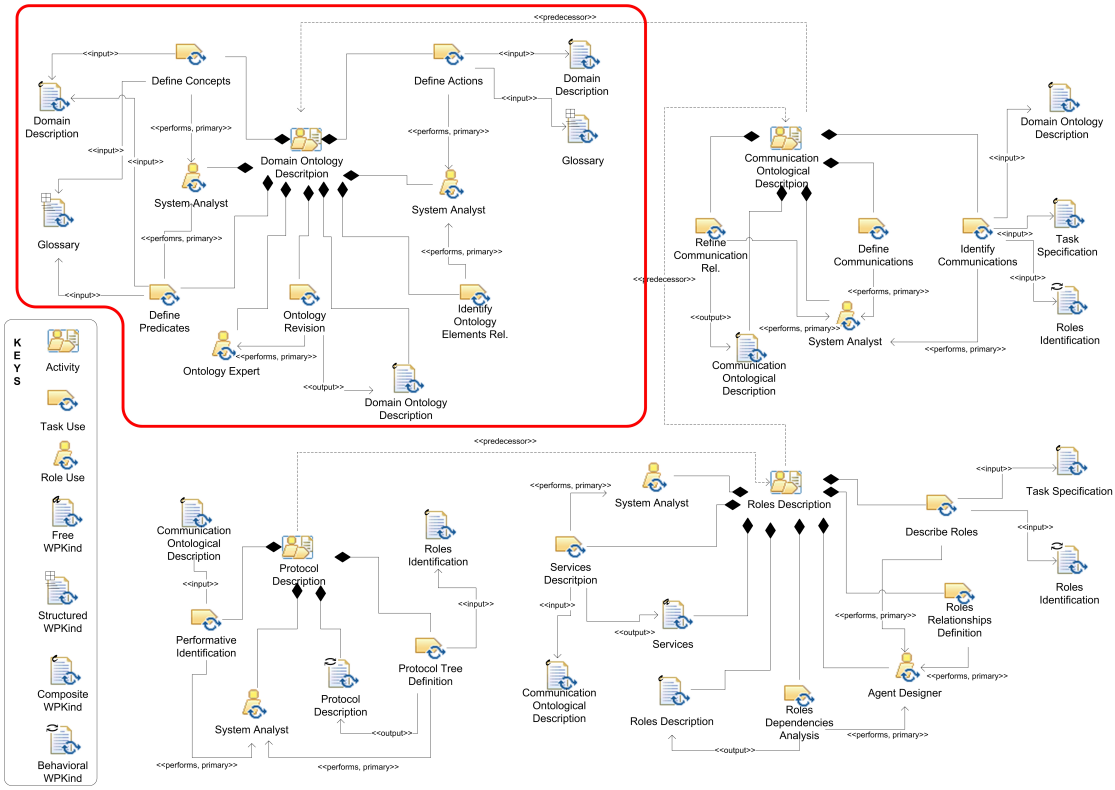


Figure 3. ***The Agent Society Phase (structural view)***

Fragment System metamodel

The portion of metamodel of this fragment is:

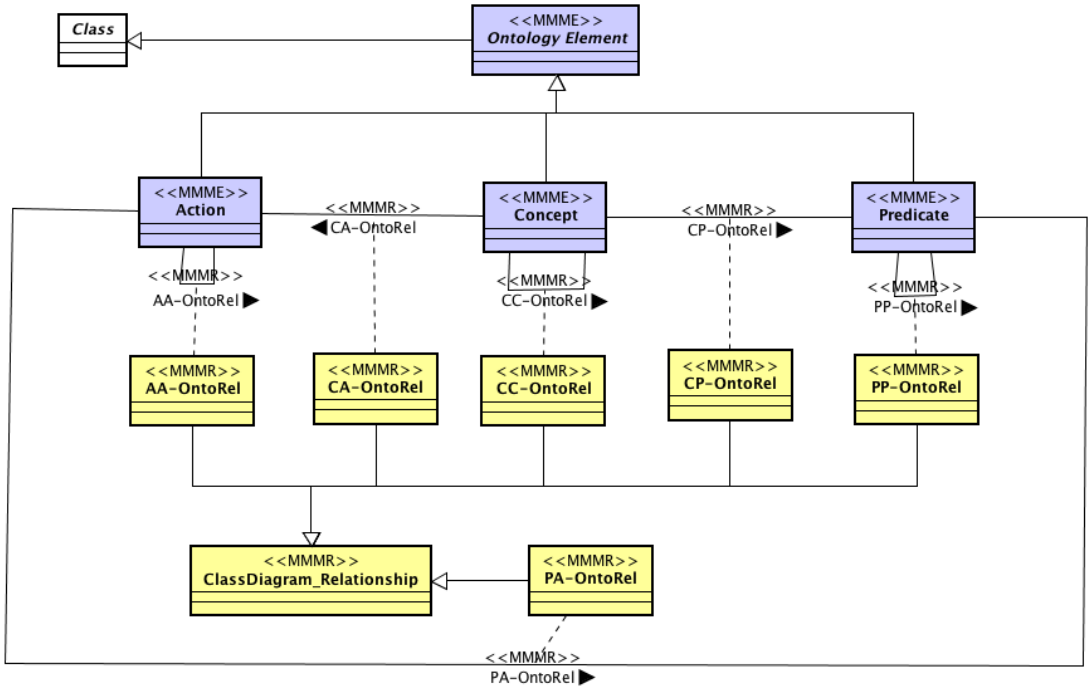


Figure 4. The fragment MAS metamodel

This fragment refers to the MAS meta-model adopted in PASSI and contributes to define and describe the elements reported in Figure 4.

Definition of System metamodel elements

This fragment underpins the following model elements:

Ontology Element *(abstract class)* – An ontology is composed of concepts, actions and predicates. An Ontology element is an abstract class used as a placeholder for the ontology constituting elements (either concepts, predicates or actions).
Concept - Description of a certain identifiable entity of the domain
Action – It expresses an activity, carried out by an agent.
Predicate – Description of a property of an entity of the domain

Definition of System metamodel relationships

This fragment underpins the following relationships among the model elements:

ClassDiagram_Relationship (abstract class) – the abstract representation of a relationship in a class diagram
AA-OntoRel – A relationship connecting two actions. It can be further qualified by using a stereotype.
CA-OntoRel – A relationship connecting an action to the concept(s) it affects. It can be further qualified by using a stereotype.
CC-OntoRel – A relationship connecting two concepts. It can be further qualified by adopting classical UML relationship types (es. Aggregate, generalize, …)

CP-OntoRel – A relationship connecting a predicate to the concept(s) it is applied to. The roles of the concepts in the predicate can be specified if needed. It can be further qualified by using a stereotype.
PP-OntoRel – A relationship connecting two predicates. It can be further qualified by using a stereotype.
PA-OntoRel – A relationship connecting a predicate to the action(s) it is applied to. It can be further qualified by using a stereotype.

System metamodel Input/Output

Input, output system metamodel elements to be designed in the fragment are detailed in the following tables.

As regards system metamodel elements:

Input		To Be Designed		To Be Refined		To Be Quoted	
MMME	**MMMR**	**MMME**	**MMMR**	**MMME**	**MMMR**	**MMME**	**MMMR**
Functional Requirement		Concept	AA-OntoRel				
Non Functional Requirement		Predicate	CA-OntoRel				
Actor		Action	CC-OntoRel				
		Environment	CP-OntoRel				
			PP-OntoRel				
			PA-OntoRel				

Definition of input system metamodel elements and relationships

Functional requirement - Functional requirements describe the functions that the software is to execute. (from IEEE SEBOK 2004)
Non-Functional requirement - Non functional requirements constrain the solution and are sometimes known as constraints or quality requirements. (from IEEE SEBOK 2004)
Actor - An external entity (human or system) interacting with the multi-agent system.

Stakeholders

Roles involved in this fragment are:

- Ontology Expert,
- System Analyst

Their responsibilities are described in the following subsections.

Ontology Expert

(S)He is responsible for:

1. Concepts definition. It consists in the identification of the concepts describing the system domain.
2. Predicates definition. The identification of predicates (the assertions relating concepts to the system domain).
3. Actions definition. The identification of activities an agent may perform.
4. Ontology element relationships refinement. It consist in relating the previous three elements with ontological relationships

System Analyst

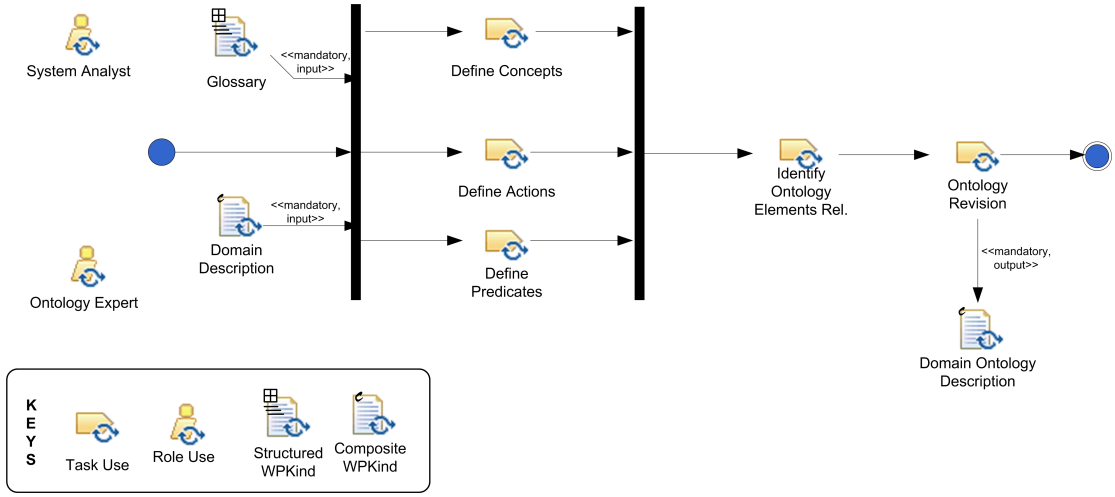
(S)He is responsible for:

1. Ontology revision. The revision of ontological elements in order to define the pieces of knowledge of each agent and their communication ontology.

Fragment workflow

Workflow description

The process that is to be performed in order to obtain the result is represented in the following as a SPEM 2.0 diagram.



Activity description

The fragment encompasses the following *work breakdown elements*:

Name	Kind	Description	Roles involved
Define Concepts	Task	The identification of the concepts describing the system domain	Ontology Expert (perform)

Define Predicates	Task	The identification of predicates, the assertions relating concepts to the system domain.	Ontology Expert (perform)
Define Actions	Task	The identification of the activities that an agent may perform.	Ontology Expert (perform)
Identify Ont. Elem. Relationships	Task	It consists in relating the previous three elements with ontological relationships.	Ontology Expert (perform)
Ontology Revision	Task	The revision of ontological elements in order to define the pieces of knowledge of each agent and their communication ontology.	System Analyst (perform), Ontology Expert (assist)

System metamodel elements and relationships input/output

The above described *work breakdown elements* have the following input/output in terms of system metamodel components.
In the Input column, system metamodel components utilization is completed by the name of the input document reporting them in the original design process.

	Input		**Output**	
Activity/Task Name	**MMME**	**MMMR**	**MMME**	**MMMR**
Define Concepts	Functional Requirement, Non Functional Requirement, Actor (System Requirements document)		Concept	
Define Predicates	Functional Requirement, Non Functional Requirement, Actor (System Requirements document)		Predicate	
Define Actions	Functional Requirement, Non Functional Requirement, Actor (System Requirements document)		Action	
Identify Ont. Elem. Relationships	Functional Requirement, Non Functional Requirement, Actor (System Requirements document)			AA-OntoRel, CA-OntoRel, CC-OntoRel, CP-OntoRel, PP-OntoRel, PA-OntoRel
Ontology Revision	Functional Requirement, Non Functional Requirement, Actor (System Requirements document)			

WP Input/Output

Input, output work products to be designed in the fragment are detailed in the following tables.

Input	Output
System Requirements Document	Domain Ontology Description (DOD) Document

Deliverable

Domain Ontology Description Document

This fragment produces a composite document composed by a class diagram (whose classes represent concepts, actions and predicates) and a text document describing the elements reported in the diagram with the following details:

- Concepts are described in terms of their attributes,
- The returned type is specified for predicates,
- Actions have an *Actor* (that is responsible to do the job), a *ResultReceiver* (that is to be notified of the action results) and an *Act* that describes the action to be done with the required input and prescribed outcome.

As already said, information described in the class diagram is (optionally) completed by a text document reporting the following data for each element (concept, action, predicate) of the ontology:

Concept	Description	
Attribute	**Type**	**Description**

Predicate	Return Type	Description

Action	Description
Actor	**Description**
ResultReceiver	**Description**
Act	

Parameter	Type	Description
Result	**Type**	**Description**

Domain Ontology Description Diagram: example of notation

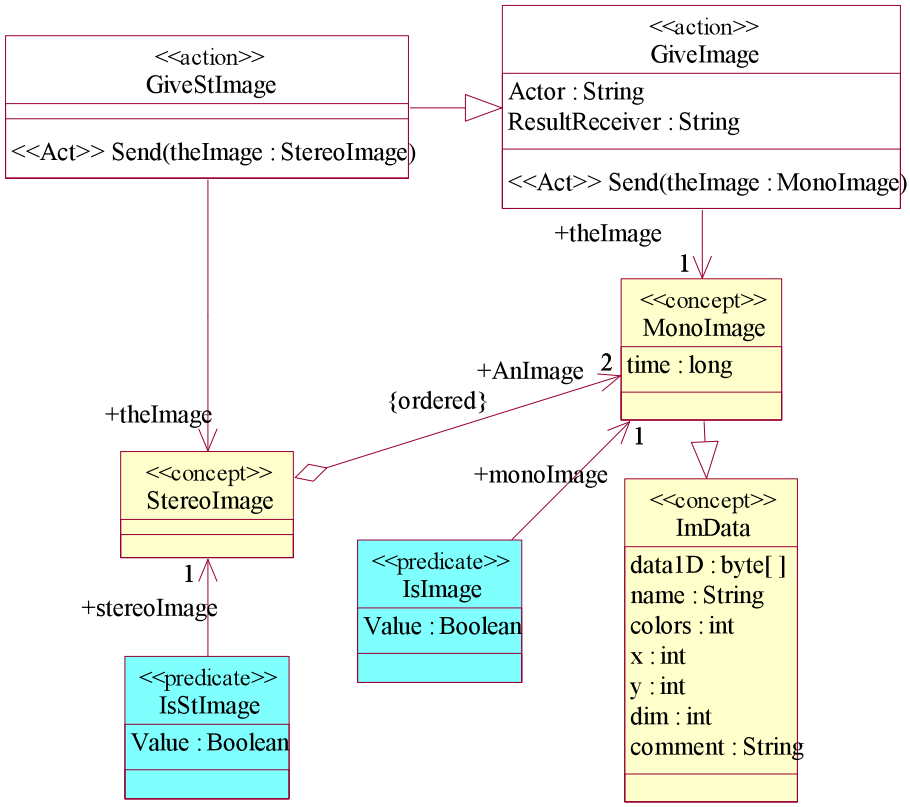


Figure 5. ***an example of Domain Ontology Description diagram***

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).
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 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. *MonoImage* is a specialization of the *ImData* concept with a time stamp (grabbing time). The ordered aggregation of two mono images gives the *StereoImage*. 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*).

Deliverable relationships with the system metamodel

The following figure describes the structure of this fragment work products in relationship with the MAS model elements:

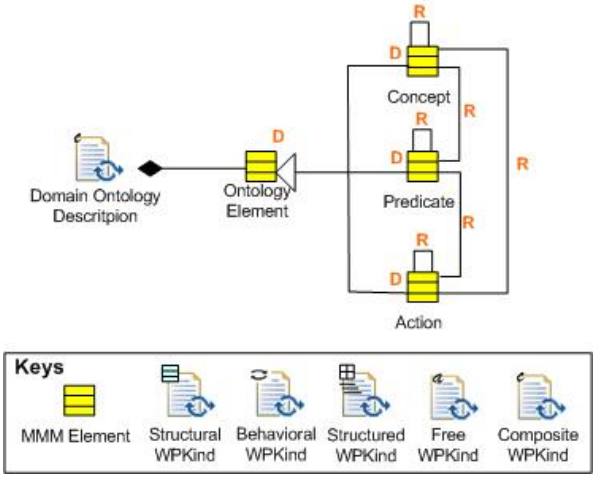


Figure 6. ***Structure of the fragment work-product in terms of MAS meta-model elements***

Guidelines

Enactment Guidelines

As it is obvious, in order to design an ontology we need to know the application domain described in the System Requirements document and we need a glossary of terms. The ontology expert is supposed to be an expert of the domain who can produce a formal representation of its categories according to the adopted notation.

Consider that the suggested structure is based on the RDF specification and therefore different approaches are not recommended.

Ontology elements may be identified starting from requirement descriptions and communication needs of agents (for instance, in PASSI, described in the Task Specification document).

A convenient approach consists in looking at use case (textual) descriptions to find recurrent nouns. These concepts are then arranged by means of inheritance and abstraction. New higher level concepts will be so identified. When concepts have been identified, the third task consists in determining concepts relationships and especially composition ones. There are two ways available to determine these relations. The first consists in looking into existing and well known ontologies to determine if a reuse is possible. The second is interviewing

domain experts and looking into the textual description of usage scenario and other available documentation.

Reuse Guidelines

Composition

This fragment is conceived to produce (possibly with the support of an automatic code generation tool) an RDF description of ontology categories. This makes it enough general but it could not be appropriate in some conditions.

Ontology designed with this fragment is supposed to be 'static'. It supports some kind of a-priori (design-time) ontology and no type of dynamic discovery (at run-time) of new categories/relationships.

This fragment is suitable for describing ontology in an RDF-like way. If the desired outcome should be expressed in a different structure, this could not be the appropriate method to obtain it

Dependency Relationship with other fragments

This fragment requires requirements and agent communications (even at a preliminary-analysis stage level of detail) as useful inputs for ontology elements identification. For this reason this may be fruitfully reused together with the following PASSI fragments:

- Domain Requirements Description
- Task Specification

References

[1] FIPA RDF Content Language Specification. Foundation for Intelligent Physical Agents, Document FIPA XC00011B (2001/08/10). http://www.fipa.org/specs/ fipa00011/XC00011B.html

[2] S. Cranefield and M. Purvis. UML as an ontology modelling language. In Proc. of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99), 1999

[3] Modeling XML applications with UML. D. Carlson. Addison-Wesley. 2001.