ADELFE: Using SPEM Notation to Unify Agent Engineering Processes and Methodology

IRIT/2003-10-R

GLEIZES Marie-Pierre
MILLAN Thierry
PICARD Gauthier

Abstract. Designers of multi-agent systems can use several agent-oriented methodologies. They need help to choose the more well-adapted methodology for their applications. Thus, the next challenge of methodologies designers is to provide this kind of tool. In order to do this, it is necessary to find a formalism to describe the different existing methodology. Then, when differences and similarities can be expressed and different tools can be developed such as a decision aided tool to choose the methodology, a tool to design a methodology from parts of different methodologies, an automatic self-design methodology. In this paper, the SPEM (Software Process Engineering Meta-model) is the formalism proposed to describe different methodologies. It is applied on Adelfe methodology. The expression of Adelfe with the SPEM leads to a better clarification of the process followed during the software development. Therefore, an interactive tool supported the process could be developed.

1. Introduction

Nowadays, a lot of agent or multi-agent oriented methodologies exist such as: ADELFE [1], GAIA [13], MESSAGE/INGENIAS [4][7], PASSI [6], TROPOS [5]… This amount of methodologies is due to the evolution of the applications which are more complex, distributed and open. Designers have difficulties and sometimes impossibility to design a centralized control or to list all situations systems have to face. Multi-agent systems bring new paradigms and are well-adapted to solve these problems. Therefore, the development of these systems requires adapted engineering methods. It is now well-known that the agent concept is different than the object concept. The most important discrepancies are the abstraction level offered by agent and the autonomy characteristic of an agent.

Designers of complex and distributed systems can use a lot of agent-oriented methodologies. Each of the available methodologies has specificities and is more or less well-adapted to a given application. For example, TROPOS focuses on the requirements phase, but in GAIA, this phase is not central. ADELFE gives some guide to embed agent with a behavior, but in MESSAGE the agent's behavior is tackled in terms of attributes and methods in a class. MESSAGE/INGENIAS helps the designer to identify agents, but in GAIA the agents are supposed as existing in the application.

Therefore, the next challenge seems to provide designers a decision-based tool to choose the more well-adapted methodology for their application. Three steps must be done. The first step consists in expressing all methodologies with the same formalism. After, the second step identifies the parts or blocks, their differences and their similarity; the last step is the design of a decision-making tool to help the designers to choose or to construct the best methodology for his application. Therefore, the first need is the unification of the language used by designers of agent oriented methodologies.

For this unification work, the SPEM language is proposed since it is a meta-model dedicated to the description of methodology processes and components. Section 2 presents this language. An application of the SPEM formalism to the ADELFE methodology is shown in section 3. Section 4 describes the main guidance and tools associated with the methodology.

2. The Software Process Engineering Meta-Model (SPEM)

The SPEM is a notation for defining processes and their components [14]. It is based on an object-oriented approach to model a family of related software processes. SPEM provides the minimal set of process modeling elements necessary to describe any software development process, without adding specific models or constraints for any specific area or discipline, such as project management or analysis. SPEM uses the UML as a notation.

2.1 SPEM, UML and MOF

As the UML notation, the SPEM processing elements are described in terms of UML concepts. In order to have this description, only some concepts of UML have been

considered as relevant. The set of these concepts constitute the kernel of the definition of the so-called Meta Object Facility (abbreviated MOF). MOF gives a meta-circular definition of this modeling language. The MOF has been adopted and standardized by OMG, and is intended to be the universal meta-model language i.e. the language capable of describing languages such SPEM, UML, relational models and so on. Let us recall that MOF is resting on a four-level standardized architecture; these four levels conventionally denoted M0, M1, M2, and M3 are defined below:

M3	***MOF***						
M2	***SPEM***			***UML***			
M1	***RUP***	***SI Method***	**…**	***User model 1***		**…**	
M0	***Processes as really enacted on a given project***			***User data 1***	***User data 2***	**…**	**…**

A performing process—that is, the real-world production process—as it is enacted, is at level M0. The definition of the corresponding process is at level M1. For example, the Rational Unified Process 2001 [9], DMR Macroscope, the IBM Global Services Method and Fujitsu SDEM are defined at level M1. Both a generic process like RUP and a specific customization of this process used by a given project are at level M1. We focus here on the meta-model, which stands at level M2 and serves as a template for level M1. In order to avoid an infinite number of levels, it was decided by OMG to make M3 reflexive. The consequence is that the MOF must be able to describe itself.

2.2 Process definition elements

Process definition elements help in defining how the process will run. They describe or constrain the overall behavior of the performing process, and are used to assist with planning, executing, and monitoring the process. A *process* can be seen as collaboration between roles to achieve a certain goal or an objective. To guide its enactment, we can constrain the order in which activities must be, or can be, executed. Also there is a need to define the "shape" of the process over time, and its *lifecycle* structure in terms of phases and iterations. A *Process* is a *ProcessComponent* intended to stand alone as a complete, end-to-end process. It is distinguished from normal process components by the fact that it is not intended to be composed with other components. A *ProcessComponent* is a chunk of process descriptions that is internally consistent and may be reused with other *ProcessComponents* to assemble a complete process. A *ProcessComponent* imports a non-arbitrary set of process definition elements, modeled in SPEM by *ModelElements*. A process *Lifecycle* is defined as a sequence of phases that achieve a specific goal. It defines the behavior of a complete process to be enacted in a given project or program.

A *WorkDefinition* is a kind of Operation that describes the work performed in the process. Its subclasses are Activity, Phase, Iteration, and Lifecycle. A *WorkDefinition* can be composed of other *WorkDefinitions*. A *WorkDefinition* is related to the WorkProducts it uses through the *ActivityParameter* class, which specifies whether they

are used as input or output. The work described in the *WorkDefinition* uses the input *workproducts*, and creates or updates the output *workproducts*. Dependency between *WorkDefinition* to another are acted using *Precedes* dependency from one *WorkDefinition* to another, to indicate start-start, finish-start or finish-finish dependencies between the works described. If activity B has a finish-start dependency on activity A, then B can start only after A has. If activity B has a finish-finish dependency on activity B, then B can finish only after A has finished. If activity B has a start-start dependency on activity A, then B can start only after A has started.

A *Phase* is a specialization of *WorkDefinition* that is a kind of operation that describes the work performed in the process. In a phase, a precondition defines the phase entry criteria and a goal defines the phase exit criteria. Phases are defined with the additional constraint of sequentially; i.e., their enactments are executed with a series of milestone dates spread over time and often assume minimal (or no) overlap of their activities in time. An *Iteration* is a composite *WorkDefinition* with a minor milestone. Note that these elements do not describe the enactment itself: they are elements of the process description that are used to help plan and execute enactment of that description.

A *WorkProduct* or artifact is anything produced, consumed, or modified by a process. It may be a piece of information, a document, a model, source code, and so on. It describes one class of work product produced in a process and describes a category of work product, such as Text Document, UML Model, Executable, Code Library, and so on. The range of work product kinds is dependent on the process being modelled. A WorkDefinition has an owning *ProcessPerformer*, representing the primary role that performs that *WorkDefinition* in the process. A *ProcessRole* is responsible for a set of *WorkProducts*. An *Impacts* dependency acts from one *WorkProduct* to another *WorkProduct* to indicate that the modification of a *WorkProduct* could invalidate another. *ProcessPerformer* represents abstractly the “whole process” or one of its components, and is used to own *WorkDefinitions* that do not have a more specific owner. *ProcessPerformer* has a subclass, *ProcessRole*. *ProcessRole* defines responsibilities over specific *WorkProducts*, and defines the roles that perform and assist in specific activities. A ProcessPerformer is the performer of higher level aggregate *WorkDefinitions* that cannot be associated with individual *ProcessRoles*. With each *WorkDefinition* can be associated a *Precondition* and a *Goal*. Preconditions and Goals are Constraints, where the constraint is expressed in the form of a Boolean expression (which is a string) following syntax similar to that of a guard condition in UML. The condition is expressed in terms of the states of the *WorkProducts* that are the parameters of the *WorkDefinition* or of an enclosing *WorkDefinition*.

An *Activity* is the main subclass of *WorkDefinition*. It describes a piece of work performed by one *ProcessRole*: the tasks, operations, and actions that are performed by a role or with which the role may assist. An activity may consist of atomic elements called *Steps*. An Activity is owned by a *ProcessRole* that is the performer of the described *activity*. It may refer to additional *ProcessRoles* that are the assistants in the *activity*. Decomposition within activity is done using *Steps*. A step is described in the context of the enclosing activity in terms of the *ProcessRoles* and *WorkProducts* it uses. In the case of activities carried out by an individual or small group, this will be a *ProcessRole*. As for the *WorkDefinition*, the dependency between an activity to another is acted using *Precedes* dependency.

After the identification of the activities, it is possible to group them into *Discipline*. A *Discipline* is a particular specialization of *Package* that partitions the activities within a process according to a common "theme." Partitioning the activities in this way implies that the associated *Guidance* and output *WorkProducts* are similarly categorized under the theme. The inclusion of an activity in a discipline is represented by the Categorizes dependency, with the additional constraint that every Activity is categorized by exactly one discipline. A *Categorizes* dependency acts from *a Package* to an individual process element in another package, and provides a means to associate process elements with multiple categories. Just as in UML, a package is a container that can both own and import process definition elements. Activities and WorkDefinitions are owned, respectively, by *ProcessRoles* and *ProcessPerformers*; StateMachines are owned by *WorkProducts* and own their internal states and transitions; ActivityGraphs can be owned by Packages, Classifiers, or BehavioralFeatures; other SPEM ModelElements can be owned by packages.

Packages and the Categorizes dependency can be used to implement general *categorization* of process description elements. A package is created to represent each category, and all of the elements linked via a *Categorizes* dependency into that package to represent membership of the category. A package represents a category when it is the source of at least one categorizes dependency. The name of the category is the name of the package. Multiple overlapping categories can be created to serve various purposes in process engineering.. For example, nine disciplines are described in the Rational Unified Process: *Business Modeling, Requirement Management, Analysis & Design, Implementation, Test, Deployment, Project Management, Configuration and Change Management*, and *Environment*.

2.3 Guidance

Guidance elements may be associated with all the SPEM models elements in order to provide more detailed information to practitioners about the associated element. Possible types of *Guidance* depend on the process family and can be for example: Guidelines, Techniques, Metrics, Examples, UML Profiles, Tool mentors, Checklist, Templates. Each Guidance is associated with a *GuidanceKind*, and the name of the *GuidanceKind* indicates what kind of Guidance it is. The following kinds of guidance list provides a basic repertoire; processes based on SPEM may add new kinds if required. A Technique is a detailed, precise "algorithm" used to create a work product. Techniques help to define the skills required to perform specific types of activities. UMLProfile is a kind of Guidance. A UML profile provides mechanisms that specialize UML for a specific target such as C++, Java, and CORBA or for a specific purpose such as analysis, design, and so on. Every development activity using UML can be ruled by a profile that dictates those UML consistency rules that need to be applied or which UML model element is relevant for the current context and focus of the activity. A *Tool Mentor* shows how to use a specific tool to accomplish an activity. Each *Tool Mentor* is associated with a single Tool and inherits the association with the Activity it supports from Guidance. Guideline is a kind of Guidance. A Guideline is a set of rules and recommendations on how a given work product must look or must be organized. Template is a kind of Guidance. Estimate is a kind of Guidance. An Estimate describes

an effort associated with a particular element. The description associated with an Estimate gives a context and interpretation for the effort.

3. ADELFE Process Modeling with SPEM

In this section, the SPEM notation is illustrated with an example, the ADELFE Process, which is briefly expounded in the first subsection. The four main Work Definitions—or sets of Activities—of the process, the Preliminary Requirements, the Final Requirements, the Analysis and the Design are presented. Each of them has been modeled with SPEM activity diagrams.

3.1 Background

ADELFE [3] is a toolkit for designing adaptive multi-agent systems. This methodology is devoted to software engineering of adaptive multi-agent systems. Adaptive software is used in situations where either the environment is unpredictable or the system is open. ADELFE guarantees that the software is developed according to the AMAS theory [8]. It aims to help any software developer –not only those specialized in adaptive multi-agent systems. So, it must guide designers without making explicit hypothesis on the fact that he plainly knows what such systems are. ADELFE consists in a process, a notation and some tools.

The process in ADELFE is based on the Rational Unified Process [9], a classical process of object-oriented methodology. To take into account the specificity of agents, some steps has been added to this classical process. Nowadays, ADELFE covers the phases of a classical software design from the requirements to the design but it aims to cover from the requirements to the deployment. ADELFE uses UML notation and extension of UML already done in AUML in particular the AIP notations [11]. The UML choice is justified because UML is a *de facto* standard for object-oriented modeling and then designers can rapidly take-up it. Our aim is not to add one more methodology to existing ones but to work on some aspects not already taken into account by existing methodologies such as complex environment, dynamic, software adaptation. The most important Guidances are: (1) the interactive interface which guides designers throughout the process. (2) The adequacy AMAS software which helps designers to decide if the AMAS is well adapted to their applications. And (3) The OpenTool© which supports UML and AUML notations and some adds. These tools are described in [1].

3.2 Preliminary Requirements Work Definition

The Preliminary Requirements Work Definition consists of five Activities (see fig. 1).

It is nearly the same as in the RUP. The participants of this Work Definition are the Requirement Analyst, the End User, and the Client. The activities are:

- Activity 1: Define User Requirements. Its objective is to produce a preliminary version of a document in which requirements are expressed, named Requirements Set.

- Activity 2: Validate User Requirements. This activity aims to validate, by the End User, the last document.
- Activity 3: Define Consensual Requirements. This activity aims to regroup in the Requirements Set document the requirements expressed by both the End User and the Analyst.
- Activity 4: Establish the Keyword Set. From the Requirements Set document, the Analyst can extract keywords and list them in the Keyword Set document.
- Activity 5: Extract Limits and Constraints. The Analyst has to define limits for the system to be, in terms of operating system, languages, technology and so on. These constraints are added to the Requirements Set document.

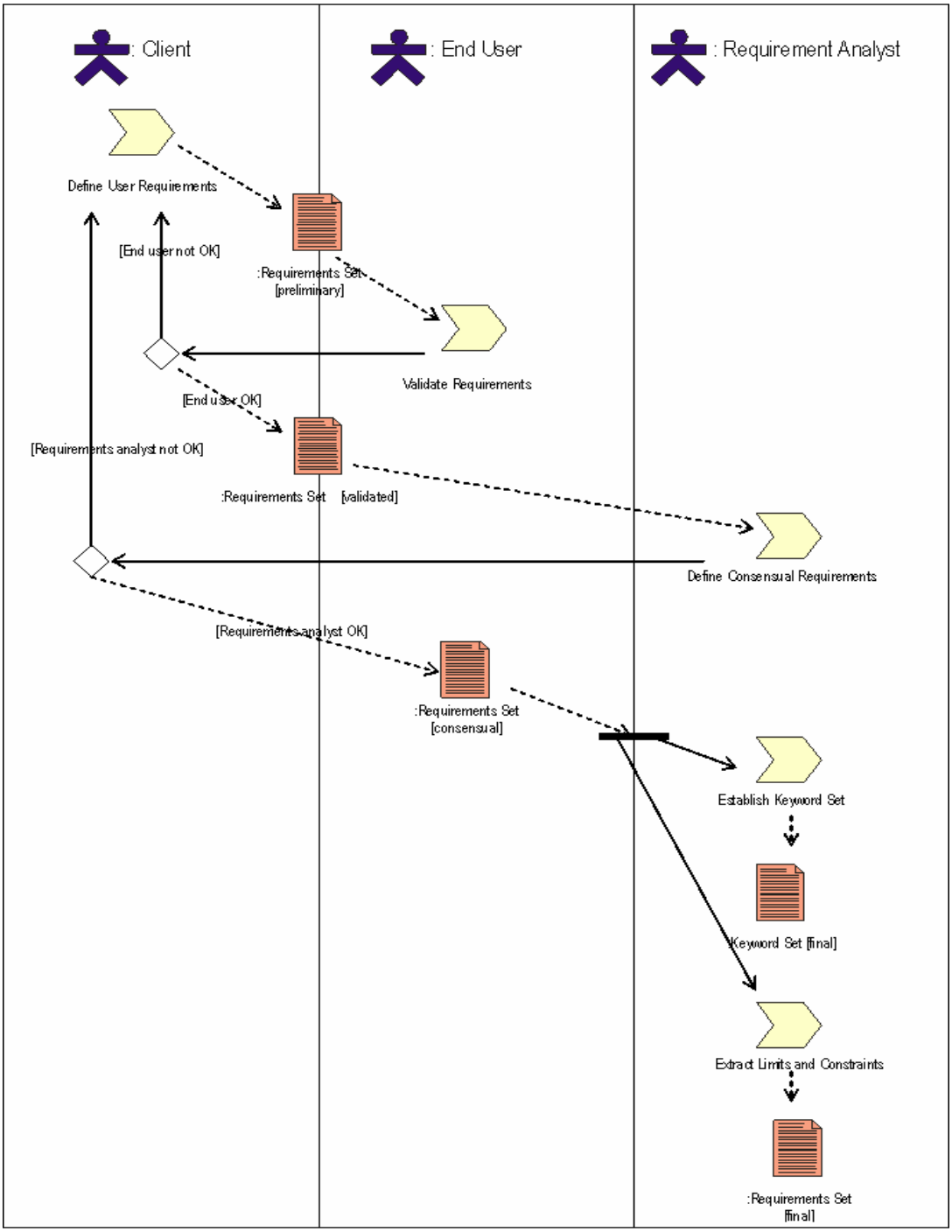


Figure 1. The Preliminary Requirements Work Definition SPEM Specification. Three Participant Roles, the Client, the End User and the Requirement Analyst, execute the activities of this Work Definition: Define User Requirements, Validate User Requirements, Define Consensual Requirements, Establish Keyword Set and Extract Limits and Constraints. Two documents are produced: the Requirements Set, which evolves from initial to final state. and the Kevword Set.

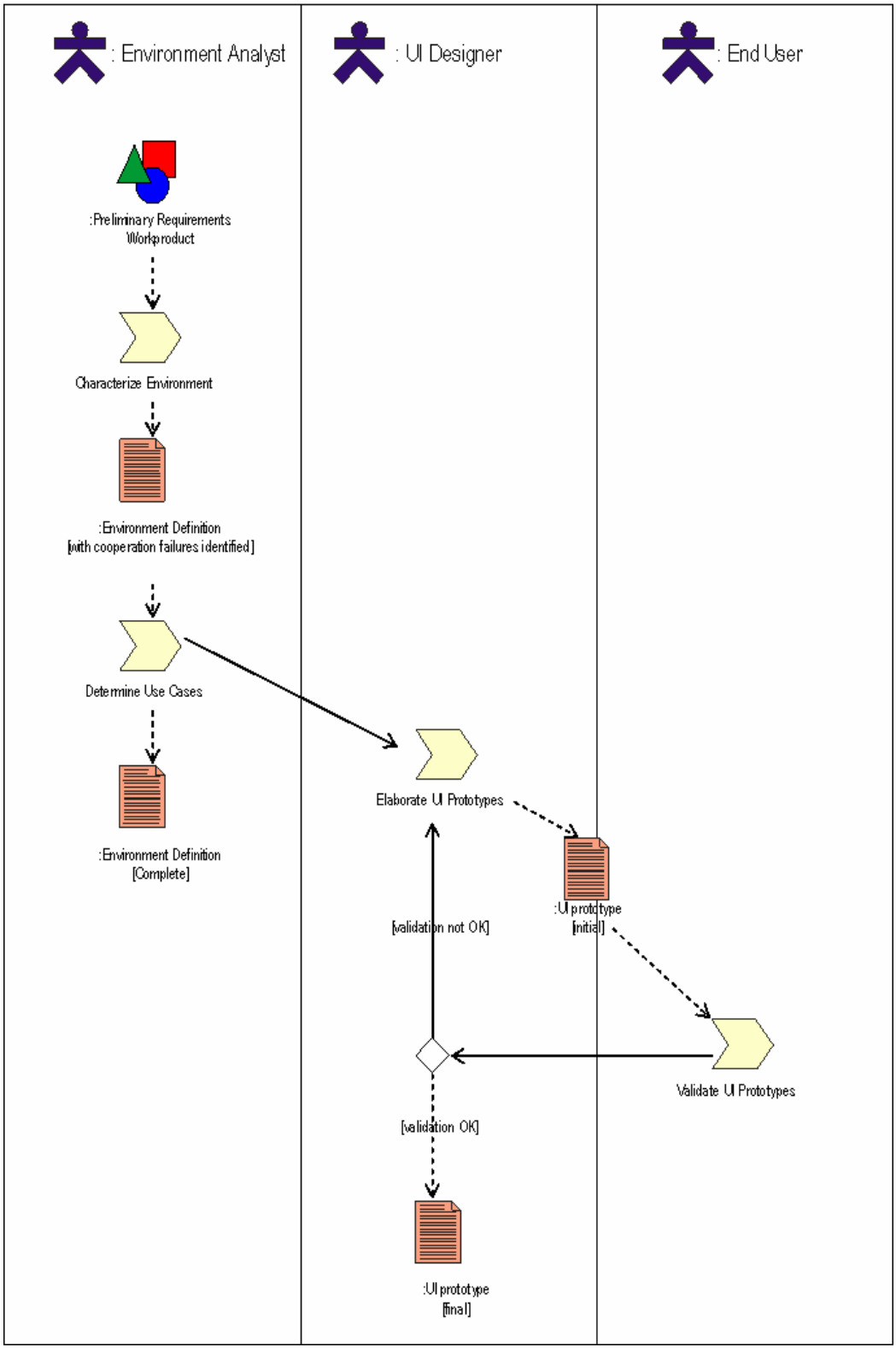


Figure 2. The Final Requirements Work Definition SPEM Specification. Three Participant Roles, the Environment Analyst, the End User and the UI Designer, execute the activities of this Work Definition: Characterize Environment, Determine Use Cases, Elaborate UI Prototype, and Validate UI Prototypes. The Preliminary Requirements WorkProduct, representing the set of previously produced document, is considered as the input of this Work Definition.

3.3 Final Requirements Work Definition

The Final Requirements Work Definition consists of four Activities (see fig. 2):

- Activity 6: Characterize the Environment. It consists in reasoning about the nature of the environment. The analyst has to characterize the environment by using a specific vocabulary [12] and to check the input or output interfaces of the system-to-be in which cooperation failure may appear, according to the AMAS theory. This activity produces an initial Environment Definition document.
- Activity 7: Determine Use Cases. This activity, by using the use case UML notation, aims to clarify the functionalities the system-to-be has to provide. It results on the production of the Functional Description model that is added to the Environment Description document.
- Activity 8: Elaborate UI Prototypes. This classical software engineering activity produces UI prototypes for each previously defined use cases.
- Activity 9: Validate UI Prototypes. This activity aims to validate the last work product by the End User.

3.4 Analysis Work Definition

The Analysis Work Definition consists of four Activities (see fig. 3):

- Activity 10: Analyze the Domain. In order to model a static view of the system, by using actors and classes, the Domain Analyst has to index identified entities in the Software Architecture document.
- Activity 11: Verify the AMAS Adequacy. During this activity, the Agent Analyst has to use the AMAS Adequacy Tool [1] to decide if the AMAS technology is necessary to design the system-to-be. This tool may also guide the Agent Analyst to detect a possible recursive decomposition of the system. The results of this analysis appeared in the AMAS Adequacy Synthesis document.
- Activity 12: Identify Agents. The Agent Analyst has to examine previously identified entities—during Activity 6—in their context, use case or sequence diagrams, to decide if some of them may be represented as cooperative agents in the system—in the sense of the AMAS theory—in terms of their interactions or the possibility to be in interaction with cooperation failures. This activity enhances the Software Architecture document.
- Activity 13: Study Interaction between Entities. The Domain Analyst has to reason on relations between active and passive entities, between active entities, and between agents. This activity produces models: sequence diagrams and AIP protocol diagrams. It aims to finalize the Software Architecture and the Environment Description documents.

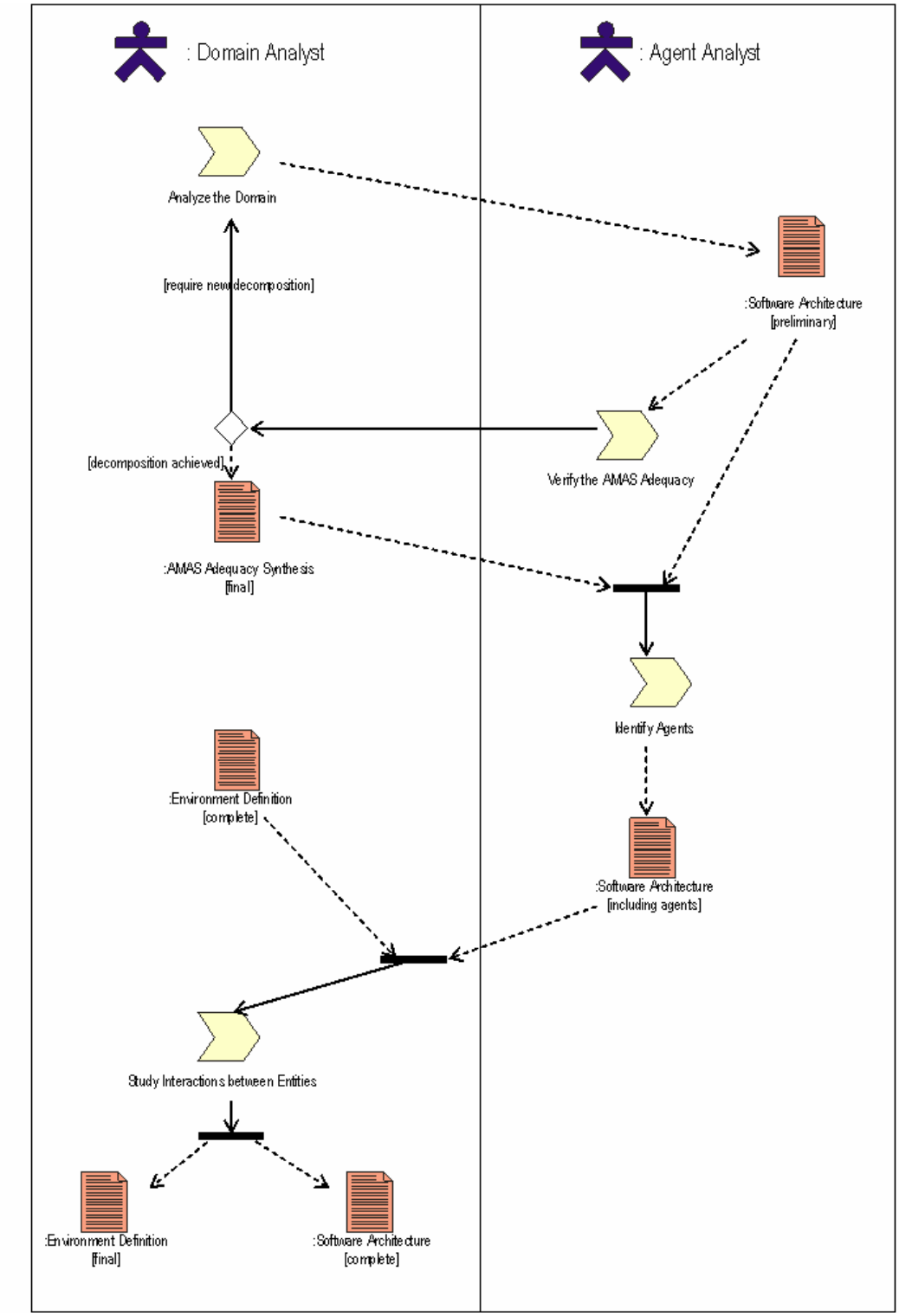


Figure 3. The Analysis Work Definition SPEM Specification. Two Participant Roles, the Domain Analyst and the Agent Analyst, execute the activities of this Work Definition: Analyze the Domain, Verify The AMAS Adequacy, Identify Agents, and Study Interactions between Entities. This Work Definition outputs three documents: the final AMAS Adequacy Synthesis, The final Environment Definition and the complete Software Architecture.

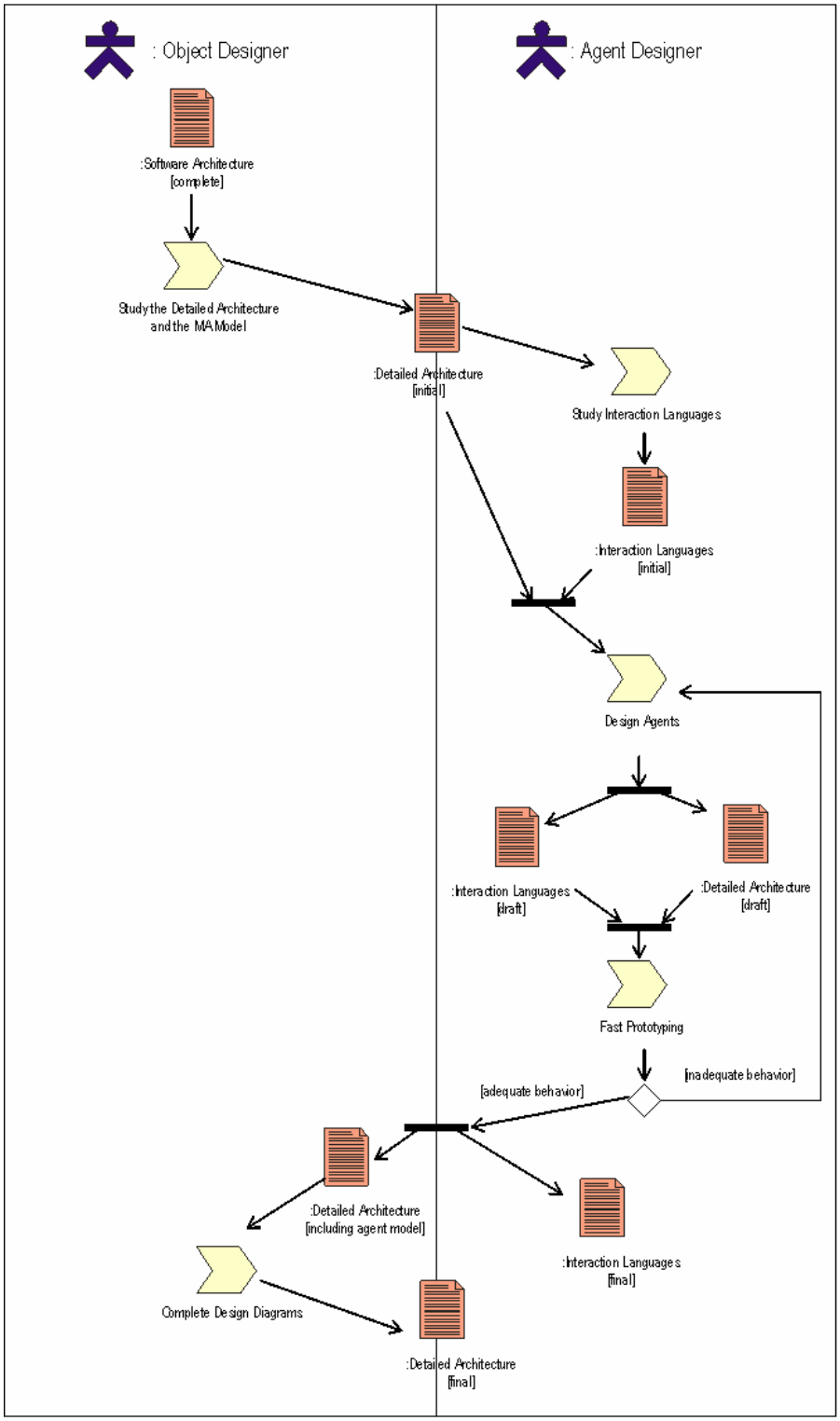


Figure 4. The Design Work Definition SPEM Specification. Two Participant Roles, the Object Designer and the Agent Designer, execute the activities of this Work Definition: Study the Detailed Architecture and MA Model, Study Interaction Languages, Identify Agents, Design Agents, Fast Prototyping and Complete Design Diagrams. This Work Definition mainly outputs the final Detailed Architecture and achieves the specification works.

3.5 Design Work Definition

The Design Work Definition consists of five Activities (see fig. 3):

- Activity 14: Study the detailed Architecture and the Multi-Agent Model. The aim of this activity is to define the different components (packages, classes, etc.) that appear in the system, and to possibly re-use pre-existing components (i.e. *design-patterns*) in order to design the architecture of the system and to produce the Detailed Architecture document.
- Activity 15: Study Interaction Languages. Agents generally interact by using specific protocols. This activity aims to define the different interaction protocols agents may use by using AIP model. This notation has been integrated to the OpenTool software. This activity produces an initial Interaction Languages document, and its models.
- Activity 16: Design Agents. For each previously identified agent, the Agent Designer has to specify its architecture, i.e. specify its skills, its aptitudes, its interaction language(s), its world representations and its non-cooperative situations. So, the Detailed Architecture and Interaction Languages documents are enhanced and then completed.
- Activity 17: Fast Prototyping. This activity enables the Agent Designer to test the agents' behaviours by simulating instances of agents in the OpenTool platform. During this activity, the Agent Designer may modify agents' components by backtracking to the previous activity while agents' behaviours are not correct.
- Activity 18: Enhance Design Models. This last design activity aims to complete previous specifications models to close the design work definition and to design dynamical behaviours of the different entities appearing in the system by using state-chart diagrams.

4. ADELFE Guidance and Tool Mentor

4.1 AMAS Adequacy Tool

ADELFE guides developers in deciding if and where the adaptive multi-agent system technology is required in the system-to-be. In certain cases, this kind of programming is completely useless; for example, if the algorithm required to resolve the task is already known, if the task is not complex or if the system is closed and nothing unexpected can occur. For an industrial, it is very important to know very early, if the system to develop justifies some investment in a new methodology or technique. A piece of software, called adequacy tool, helps designers to take this decision.

This adequacy is studied both at the global level (the system level) and at the local one (the level of the different parts composing the system). To study the adequacy of the AMAS theory several criteria have been defined and the answers given by designers are then analyzed by the support decision tool to inform him if using an AMAS to implement the system is useful. The adequacy tool is linked with the activity 11 in the process.

4.2 ADELFE Stereotypes

AMAS theory introduces specific notions such as cooperation, aptitudes, or skills. These notions have particular semantics and to ensure their adequate use, ADELFE guides developers by defining new stereotypes. This set of stereotypes is modeled as a Guidance in the ADELFE process that may help in designing agents.

Here are the ADELFE-specific stereotypes:

- <<cooperative agent>> characterizes a class as an agent one in the sense of the AMAS theory. An example of rule on this stereotype is: a <<cooperative agent>> stereotyped class must own cooperation rules (i.e. <<cooperation>> stereotyped features or associations); in order to represent cooperative agent.

The following stereotypes can only be attributed to features appearing in a <<cooperative agent>> stereotyped class:

- <<cooperation>> characterizes features that manages the cooperative behavior of an agent (e.g. non-cooperative situation detection rules).
- <<perception>> characterizes features which represent agents' perceptions (e.g. sensors).
- <<action>> characterizes features which represent agents' actions (e.g. wheel control).
- <<skill>> characterizes features which represent agents' skills (e.g. a knowledge base about brokerage).
- <<aptitude>> characterizes features which represent agents' aptitudes (e.g. an inference engine).
- <<interaction>> characterizes features which represent agents' interactions, notably those appearing in AUML protocol diagrams.
- <<world representation>> characterizes features which represent agents' representations and beliefs on the other agents for example.
- <<characteristic>> represents intrinsic characteristics of an agent (e.g. an identification number, a color, a weight,…).

4.3 Modeling with OpenTool

During the ADELFE process, designer has to construct UML or AUML model. These specification tasks are eased by using dedicated editors. OpenTool, developed by TNI-Valiosys, is a UML 1.4 editor and checker. This software is considered as a Guidance in the sense it eases designers in developing adaptive multi-agent systems because:

- all the ADELFE-specific stereotypes and their rules are integrated to OpenTool;
- OpenTool already manipulates AUML protocol diagrams;
- OpenTool owns a simulation functionality which is a useful tool for the Fast Prototyping activity. This tool simulates objects by executing their state-machines to observe the dynamical behaviors.

Of course, designers may use different tools to model their systems if they integrate all the required notations and functionalities.

5. Conclusion and Future

The SPEM meta-model leads a better clarification of the process followed during the software development. The expression of activities, steps and associated work products facilitates the manipulation of the process. Therefore, an interactive tool supported the process could be developed. Moreover, the prototype ADELFE V1.0 is operational[1].

The SPEM can be useful to describe several methodologies in a unified way, in order to compare them. This comparison can be used first, by designers to choose the more well-adapted methodology for their applications and second, to design a computer aided methodology engineering. The possibility of defining an automatic tool to choose and use methodologies is the next challenge for software engineering designers. A methodology can self-design in using parts of several methodologies. This tool could be based on adaptive multi-agent systems [8]; each component of a methodology should be "agentified" and all these components will cooperate to build at runtime the adequate methodology for the dedicated application.

Acknowledgments

We would like to thank the support of the French Ministry of Economy, Finances and Industry as well as our partners: TNI-Valiosys Ltd., for their customization of OpenTool©, ARTAL technologies Ltd and L3I La Rochelle.

References

[1] Bernon C., Camps V., Gleizes M.-P., Picard G. - *Tools for Self-Organizing Applications Engineering* - The First International Workshop on Engineering Self-Organising Applications (**ESOA'03**) Melbourne, Australia, July 2003.

[2] Bernon C., Gleizes M-P., Peyruqueou S., Picard G. – ADELFE, a Methodology for Adaptive Multi-Agent Systems Engineering – *In Third International Workshop on Engineering Societies in the Agents World (ESAW-2002)*, Madrid, 16-17 September 2002.

[3] Bernon C., Gleizes M-P., Picard G., Glize P. – The Adelfe Methodology For an Intranet System Design – *In Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002)*, Toronto (Ontario, Canada) at CAiSE'02, 27-28 May, 2002.

[4] G. Caire, F. Leal, P. Chainho, R. Evans, F. Garijo, J. Gomez, G. Pavon, P. Kearney, J. Stark & P. Massonet - Agent Oriented Analysis using MESSAGE/UML - *AOSE 2001*.

[5] J. Castro, M. Kolp & J. Mylopoulos – A Requirements-driven Development Methodology – In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, Stafford, UK – June, 2001.

[6] Cossentino M., Different Perspectives in Designing Multi-Agent System*, AgeS'02 (Agent Technology and Software Engineering) Workshop at NodE'02,* Erfurt, Germany, October 2002.

[7] Eurescom - Project P907-GI - MESSAGE: Methodology for Engineering Systems of Software Agents, Deliverable 1 - Initial Methodology - *http://www.eurescom.de/~pub-deliverables/P900-series/P907/D1/P907D1.zip*

[1] The prototype is now accessible on the website http://perso.unvi-lr.fr/vcamps/Adelfe/Adelfe.html

[8] M-P. Gleizes, V. Camps, P. Glize - A Theory of Emergent Computation Based on Cooperative Self-Organization for Adaptive Artificial Systems - *4th European Congress of Systems Science*, Valencia, 1999

[9] I. Jacobson, G. Booch & J. Rumbaugh – The Unified Software Development Process – *Addison-Wesley,* 1999.

[10] N.R. Jennings, M. Wooldridge - Agent-oriented software engineering - In *J Bradshaw (Ed.), Handbook of Agent technology*, AAAI/MIT Press 2000

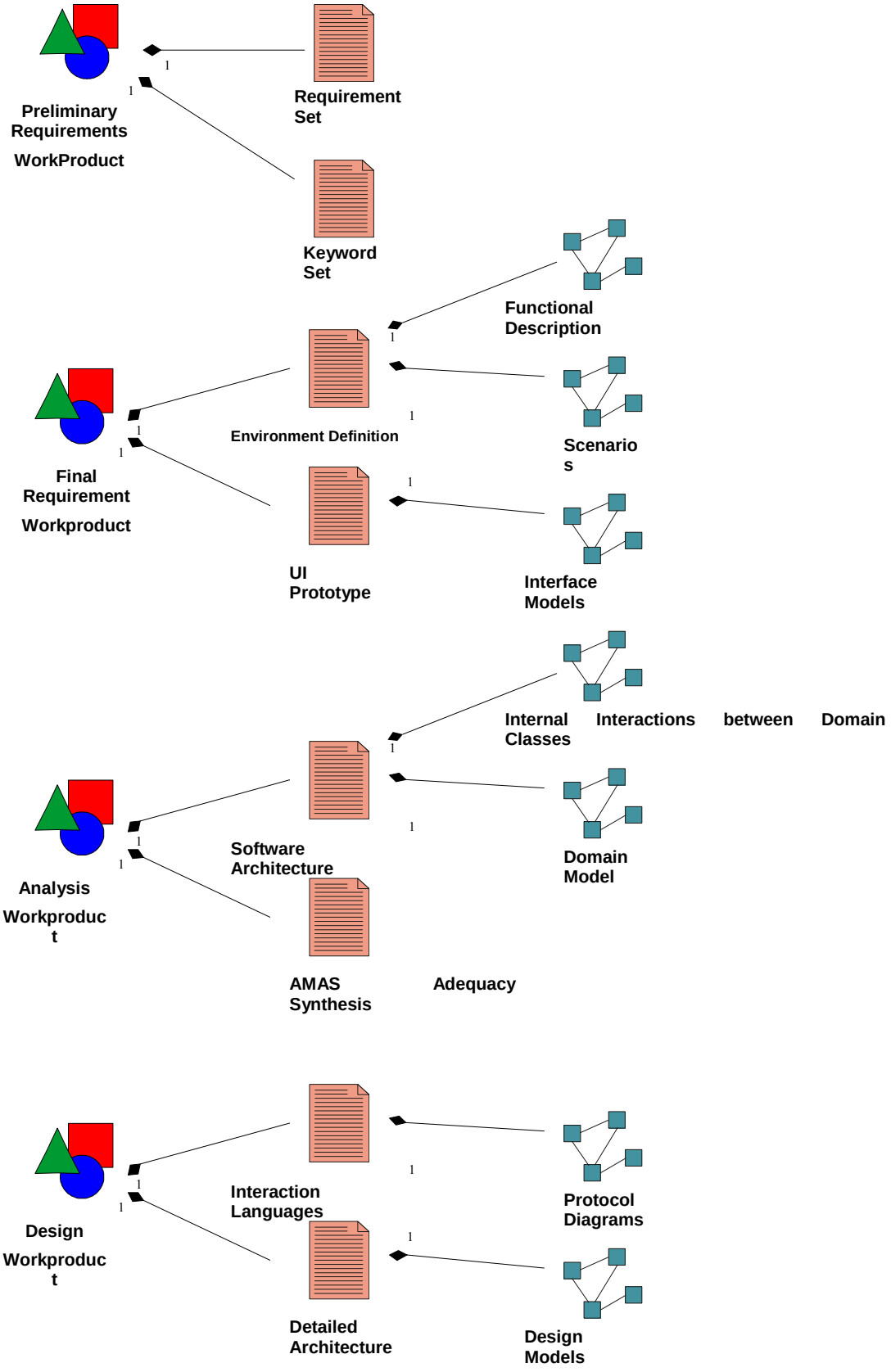
[11] J. Odell, H.V. Parunak, & B. Bauer - Extending UML for Agents - In *Proceedings of the Agent Oriented Information Systems (AOIS) Workshop at the 17th National Conference on Artificial Intelligence (AAAI)*, 2000.

[12] Russel S. and Norvig P. – *Artificial Intelligence: a Modern Approach* – Prentice-Hall, 1995.

[13] M. Wooldridge, N. R. Jennings & D. Kinny - A Methodology for Agent-Oriented Analysis and Design - In *Proceedings of the 3rd International Conference on Autonomous Agents (Agents 99)*, pp 69-76, Seattle, WA, May 1999.

[14] Object Management Group - Software Process Engineering Metamodel Specification Version 1.0, formal/02-11-14, November 2002. http://www.omg.org/docs/formal/02-11-14.pdf

A. ADELFE Workproducts



B. ADELFE Participants