Towards Process Lines for Agent-Oriented Requirements Engineering

Fatemeh Golpayegani, Keyvan Azadbakht, Raman Ramsin

Department of Computer Engineering, Sharif University of Technology

Azadi Avenue, Tehran, Iran

golpayegani@ce.sharif.edu, kazadbakht@ce.sharif.edu, ramsin@sharif.edu

Abstract—**Agent-oriented software products are becoming increasingly complicated, and the competitive market is forcing the producers to reduce time-to-market and increase the quality of the software produced. Therefore, developers have come to realize the need for more reliable and efficient agent-oriented software development processes (methodologies) which address the specific needs of each and every project. Software Process Lines provide a solution to this problem by using Process Line Engineering concepts for instantiating bespoke software processes.**

This research focuses on developing a software process line for Requirements Engineering (RE) in the context of agent-oriented software development. Our proposed Agent-Oriented Requirements Engineering Process Line (AOREPL) incorporates a core base which can be directly used for instantiating an Agent-Oriented Requirements Engineering (AORE) process; it also defines variation points and variant method chunks to be added to the core base in order to create variant AORE processes. We also propose a step-by-step process line engineering method which enables process engineers to define and instantiate diverse AORE process lines.

Keywords: Software Development Process; Agent-Oriented Development; Process Line Engineering; Requirements Engineering

I. INTRODUCTION

Software processes are similar to software products [1]; much like software products, they have requirements which can be modeled, developed, tested, and reused. Other similarities include issues regarding reuse; both employ component-based architectures and have repositories for storing reusable components [2]. Therefore, software product reuse techniques can also be used for developing software process reuse mechanisms. *Product* Line Engineering has thus led to *Process* Line Engineering [3]. A process line can reduce the time, cost and risk of process development, and increase its quality, accuracy and predictability.

Typically, developing agent-oriented software requires agent-oriented software development processes. The context of agent-oriented software development has evolved over time, as other software development contexts have; this context now demands sophisticated agent-oriented development methodologies, and sophisticated methodologies require customization to make them applicable in practice, further complicating their use. Various solutions have been proposed for this problem, one of which is the assembly of method chunks according to the specific needs of the target process. To this end, practitioners and researchers have extracted method chunks from existing agent-oriented methodologies, and have organized them in repositories to be used for constructing bespoke processes. However, this solution entails certain problems, in that the person assembling the process must have ample process engineering and agent-oriented knowledge, should manage the relationships between unrelated and fragmented method chunks, and should be able to guarantee the cohesiveness, accuracy, and clarity of the final process.

The above issues have motivated us to explore the applicability of Process Line Engineering for producing bespoke, tailored-to-fit agent-oriented processes. In order to focus our research, we have narrowed its scope to Requirements Engineering (RE) in agent-oriented processes. Requirements engineering is crucial to all software development processes, as minor shortcomings in requirements engineering may lead to major losses in the overall process. Agent-oriented methodologies put special emphasis on RE activities, and include highly specialized RE tasks in their processes. However, process line engineering has not yet been applied to this context, even though Agent-Oriented Requirements Engineering (AORE) practices are mature enough for this purpose. We address this issue through proposing our Agent-Oriented Requirements Engineering Process Line (AOREPL), which is the result of scrutinizing the AORE tasks and practices used in agent-oriented methodologies and frameworks, as well as the important RE activities prescribed in other relevant contexts. AORE processes can be directly instantiated from AOREPL's core base; AOREPL also defines variation points, as well as variant method chunks, which can be added to the core base in order to create variant AORE processes. A further contribution of this research is the step-by-step process line engineering method that we have used for obtaining AOREPL; this method can be reused by process engineers to define and instantiate custom AORE process lines.

The rest of this paper is organized as follows: Section II introduces the research background; Section III introduces our proposed process-line engineering approach, and Section IV presents the proposed Agent-Oriented Requirements Engineering Process Line (AOREPL), focusing on Domain Engineering; Section V introduces the complementary AORE method fragment repository, while Section VI focuses on Application Engineering and validates the proposed AOREPL with a case study; finally, Section VII presents the concluding remarks and discusses possible directions for further research.

978-1-4673-2232-4/13/$31.00 ©2013 IEEE

II. Research Background

Various software process line engineering approaches exist (e.g., [2]–[4]). Since they share many features, a general structure can be defined for them based on their commonalities. Each process line consists of the following general parts: Two distinct processes for domain engineering and application engineering, a repository of reusable method chunks, a reuse process, and a management process [3]. In addition to the above, three stages have been proposed in [4] for software Product Line Engineering (PLE) which can be adapted for use in software process line engineering; these stages include: Scoping the Product Line, Product Line Modeling, and Product Line Architecture. These approaches first focus on domain analysis of product families, and then model the domain knowledge in terms of commonalities and variabilities. Consequently, the common parts are regarded as the core of each product family member. In order to instantiate a product, each variation point must be specified by selecting the most suitable variant. By substituting "product" with "process" in the above stages and tasks, these approaches can be applied to software process line engineering as well.

Our proposed method demonstrates the concrete application of these abstract concepts. None of the extant approaches include a well-defined and step-by-step process for domain and application engineering. Therefore, we have strived to define a clear, concise, and specific method for process-line *domain-* and *application* engineering based on the concepts of software product-line engineering [5], and have used the method to design a process line for requirements engineering in agent-oriented development.

There are various Agent-Oriented Requirements Engineering (AORE) frameworks which provide RE tasks specialized for agent-oriented contexts, albeit at an abstract level (e.g., [6]–[10]). However, there are certain problems with these frameworks: 1) Their abstract tasks cannot be directly incorporated into a concrete development methodology (the Instantiation Problem); 2) they do not fully cover the RE-related tasks required in generic software development methodologies; and 3) customization is rarely possible. We have strived to address these problems in our proposed agent-oriented requirements engineering process line.

III. Proposed Process Line Engineering Approach

A process line can significantly reduce the time, cost and risk of process development, and increase its quality, accuracy and predictability. Thus, designing a process line for a family of processes can be extremely beneficial. As we aim to design a process line for Requirements Engineering in Agent-Oriented methodologies, we first have to define our proposed method for domain and application engineering, then design our process line using this method, and finally validate our process line through a case study.

A. Domain Engineering Process

A domain engineering process focuses on domain identification, specifying the process family members and analyzing their commonalities and variabilities, and constructing a core process which is common for a large group of family members and which can be reused without significant change. On the other hand, all the variation points and variabilities should be defined in this process. We divide this process into eight steps and specify each step in detail throughout the rest of this section. Also, we compare the proposed steps with their counterparts in product line engineering. The activity diagram of Fig.1 shows this process.

1) *Domain Scoping:* In this step, recent processes and methodologies, as well as similar projects and frameworks, are explored, and their features are extracted and prioritized in order to scope the domain. In other words, this step deals with determining an overall view of the processes which can be categorized as a family. There is a fundamental difference between *Process Line* Domain Scoping and *Product Line* Scoping. In Process Line Engineering, we have to carefully explore all the processes which are relevant and similar to the target family to ensure that nothing is overlooked. This means that in some cases, we will use method chunks that are not used in the target family but are known to be necessary for completing the expected functionality. Whereas in Product Line Engineering, we have to abide by the family and cannot deviate from what is defined by family members.

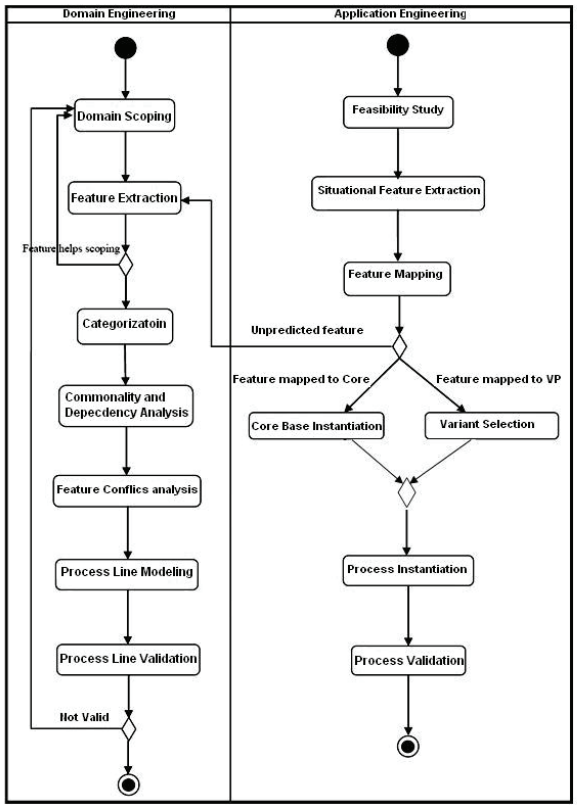

Fig. 1. Proposed Process Line Engineering Approach

2) *Feature Extraction:* In Software Process Lines, features are the buidling blocks of the processes; in addition, there are certain properties which contribute to the building blocks. Process features should be extracted as process method fragments at all levels of abstraction. Features can be extracted from various sources:

- Extracting method fragments from methodologies;
- Extracting phases/stages from related frameworks;
- Analyzing market requirements and adding the required method fragments based on existing and predicted requirements.

3) *Categorization:* Extracted method fragments may overlap, or in some cases differ only in input and output artefacts; fragments should therefore be structured and organized through categorization and abstraction. We therefore use frameworks and generic methodologies as a baseline to specify the main phases, then classify method chunks under the phases, and finally add them to a repository.

4) *Commonality and Dependency Analysis:* There are *various* methods for commonality analysis in Product Line Engineering. But we propose the use of Process/Requirements Matrix Analysis. In this method, we first prepare a list of process requirements (e.g. process fragments and features), and then map these requirements or features to process family members. Features which are required in all the processes are considered as common features, and features which are required in some of them are considered as variable features; features which are not absolutely necessary are considered as optional features.

5) *Feature Conflicts Analysis:* The plurality of the variation points results in a wide range of variant configurations, which can lead to complications in configuration management of process-line variants. Therefore, variation points are divided into two types; Encapsulated Variation Points (EVP) and Free Variation Points (FVP). In order to manage variabilities, certain variation points are considered as Encapsulated Variation Points, as they do not have a wide-ranging effect on the process line, and can therefore be encapsulated into their related stages; on the other hand, there are certain variation points which contain variants which can affect the whole process line. In addition, by fragmenting the processes into method chunks, we will have a repository of fragments which contains related as well as unrelated chunks. To form a process line, you have to clearly define the relationships between the chunks; this is especially important when the inconsistent method chunks are located in variation points, and the user can select from among them. Therefore, the process line should automatically reject inconsistent chunks if they are selected. To this aim, we propose a conflict analysis algorithm which uses activity diagrams. This algorithm helps us in variability management and conflict analysis; the algorithm is explained below.

6) *Feature Conflicts Analysis Algorithm:* To analyze the effects of the variabilities, we use communication diagrams to determine their interrelationships. We draw a communication diagram for each stage to analyze the EVPs, and a separate communication diagram for all the stages to analyze the FVPs.

We use the activity diagram to model task dependencies. In order to find related tasks, we follow each artifact's progress, to identify task inconsistencies. The steps are as follows:

- Specify the artifacts: Artifacts can be considered as pre- and post-conditions of each phase, Stage, and Task..
- Draw an activity diagram for each stage: In this step, we aim to analyze the variation points and core tasks to find their dependencies; we can thus identify the tasks whose preconditions involve tasks which do not conform to the current stage, and consequently clarify their interdependencies.
- Draw an activity diagram containing all stages: We model the tasks whose preconditions do not conform to their internal tasks.
- Detect the inconsistencies in method content.

7) *Process Line Modeling:* There are several methods for modeling software product lines, but we prefer the feature tree process line modeling. The feature tree facilitates the modeling of commonalities and variabilities, as well as the dependencies among the method chunks.

8) *Process Line Validation:* In this step, all the method content and the relationships among the method chunks should be verified. The validation method can vary based on the process line construction paradigm. If the process line is identified as a valid process line, the Domain Engineering process is considered as complete; otherwise, the domain engineering process is iterated until the defects are resolved.

B. Application Engineering Process

In order to build specific software processes from the process line, it is instantiated based on the characteristics of the project at hand. In this process, all the project- and process-specific requirements are extracted and mapped to the predefined method chunks; in some cases, the process engineer needs to define the method chunks required at variation points. In this section, we discuss the application engineering process steps to clarify all the required actions that should be performed to ensure a successful instantiation. Process line application engineering steps are as follows:

1) *Feasibility Study:* The suitability of the process line members to be applied in the target context should be investigated. In other words, it should be checked in early steps whether the process family which is modeled in the process line is close to the target domain. To this aim, it is helpful to investigate previous projects that have been executed by this process line, or to map the requirements to the members of the process family.

2) *Situational Feature Extraction:* All the features should be extracted by studying the available resources.

3) *Feature Mapping:* In this step, all the features should be mapped to the predefined process line chunks. Each feature may lead to one of the following three states:

- Core Base Instantiation: The feature is part of the core, so the core base should be instantiated.
- Variant Selection: The feature is mapped to one of the variant method chunks; therefore, the variation point has to be fixed.
- The feature is not mapped to any part of the process line: In this case, there should be a feedback loop to the domain engineering process.

4) *Process Instantiation:* Every configuration of the process line is an instance of the process. By defining and mapping the project-specific features of the process, it can be claimed that the process has been instantiated.

5) *Process Validation:* Once the process is instantiated, it should be validated. Validation results should be returned to domain engineering by a feedback loop.

IV. Agent-Oriented Requirements Engineering Process Line (AOREPL): Domain Engineering

Using the proposed process line engineering approach, we hereby develop the Agent-Oriented Requirements Engineering Process Line (AOREPL). To this aim, the first stage is domain engineering; therefore, all the steps of domain engineering are performed, as delineated below:

The first step is domain scoping. In this step, we have considered most of the Agent-Oriented Requirements Engineering frameworks available as sources of information. (e.g., [6]–[10]), the analysis phases of prominent agent-oriented methodologies (e.g., [11]–[17]), and have also inspected generic Requirements Engineering frameworks [18]. As a result, all the stages which are important in the AORE domain have been scrutinized.

The next step is extracting the process features. To this aim, we have first carefully examined all the methodologies, frameworks, and method repositories available, and all the relevant process fragments have been extracted (including stages, activities and tasks). We have then analyzed the use of AO methodologies in industry and added additional method fragments which are related to this domain. Finally, an investigation has been conducted to predict the future needs of AO processes, such as agility and formalism, and the method chunks required to realize these needs have been added. The process fragments extracted from different sources may have over lapping in their definitions or in their output artifacts; so we have to separate them and define them clearly. To this aim, we have categorized all the related method chunks under their related framework or methodology steps, and have defined new abstract categories for method chunks which are similar to one another but which cannot be easily separated.

Once the method chunks are categorized, it's time to manage the method chunks and form the process line. We therefore have to form the core base, and specify the variation points and the variants for each variation point. To this aim, we have to analyze the necessity of each method chunk in every agent-oriented methodology. We use the commonality matrix for this purpose, with agent-oriented methodologies listed on the horizontal axis, and the extracted method chunks on the vertical axis. A cell is marked with a "Yes" if its corresponding method chunk is used in the intersecting methodology. The commonality matrix showing the correspondence of categorized method fragments to the selected set of agent-oriented methodologies is shown in Table I.

Following commonality analysis, the core base is formed with all the mandatory method chunks and variation points, but the method chunks still need to be analyzed more precisely. Thus, we follow the conflict analysis algorithm defined above to identify inconsistent method chunks and specify preconditions and post-conditions for each and every method chunk.

TABLE I
COMMONALITY ANALYSIS MATRIX

				Selected Set of Agent-Oriented Methodologies						
Agent-Oriented Requirements Engineering	**AORE Stages**	**AORE Tasks and Sub-Tasks**		GAIA [17]	TROPOS [11]	ADELFE [19]	MASE [15]	Message/uml [12]	Prometheus [16]	ASPECS [14]
	Domain Specification	Resource Planning		Y	Y	Y	Y	Y	Y	Y
		Environment Modeling		Y	Y	Y				Y
		Organization Modeling		Y	Y			Y	Y	Y
		Stakeholder Modeling			Y					
	Requirements Elicitation	Prediction								
		Project-Specific Requirements		Y	Y	Y	Y	Y	Y	Y
		Organization-Specific Requirements		Y	Y	Y	Y	Y	Y	Y
	Requirements Modeling	Scenario-based		Y	Y			Y	Y	Y
		Use-Case-based				Y	Y			Y
	Requirements Specification	Agent Identification	Goal Identification		Y	Y	Y	Y	Y	Y
			Role Identification	Y			Y	Y		Y
			Plan Identification		Y					Y
		Interaction Modeling	Agent Dependency Modeling	Y	Y	Y	Y	Y	Y	Y
			Acquaintance Modeling	Y	Y			Y	Y	

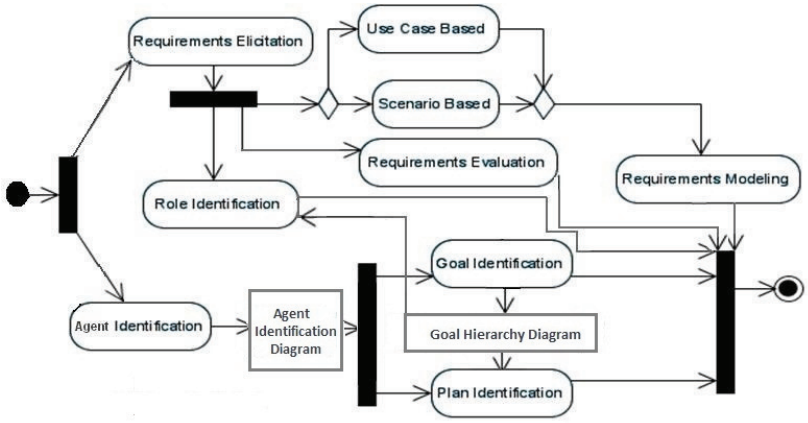

Fig. 2. Conflict Analysis Algorithm: Activity Diagram

Fig. 2 is a partial depiction of this analysis. In this sample, we have supposed that the process engineer has decided to select Role Identification and Plan Identification; Goal Identification is also instantiated through core base instantiation, since it is a mandatory method fragment. We now have to analyze the relationships between these tasks and their techniques. Therefore, we specify each task's precondition and post-condition and define the dependencies among these tasks according to their precondition artifacts. After dependency analysis, the process line is modeled in FeatureIDE, a feature modeling tool which is used for modeling product lines. Using this tool is not mandatory, but it has been chosen because of its powerful feature-tree modeling and product instantiation capabilities. In this tool, the AO process line feature tree is modeled at four levels: The zero level is the root, the next level is for the stages observed in requirements engineering, the second level is for the tasks, and the last level is where the sub-tasks reside (Fig. 3). In addition, we have techniques under each task and sub-task too, but we refrain from modeling these techniques in the feature tree to avoid excessive complexity.

In order to validate the designed process line, we have configured the process line for all the methodologies that have been used for defining it. The results show that nothing has been left out, and that process coverage has been adequately observed.

V. Agent-Oriented Requirements Engineering Method Fragments Repository

As mentioned before, one part of a process line is a repository of method fragments. In this repository, we have placed a proposed set of AORE method chunks. This repository includes AORE stages, tasks, sub-tasks and techniques. It can also be completed and elaborated in future research. We have summarized the method chunks of the repository throughout the rest of this section.

A. Stage: Domain Modeling

A step in the Requirements Engineering process, in which the problem domain is identified and modeled. The documents of this step provide a better view for eliciting and specifying the requirements.

1) *Task: Environment Modeling (EnvModeling):* According to [20], Agents, unlike objects, are situated in an environment, with which they interact by observing and changing it. So we should model the environment housing the system, as well as the interactions between the system and its environment. This task helps better identify, elicit, and specify the requirements.

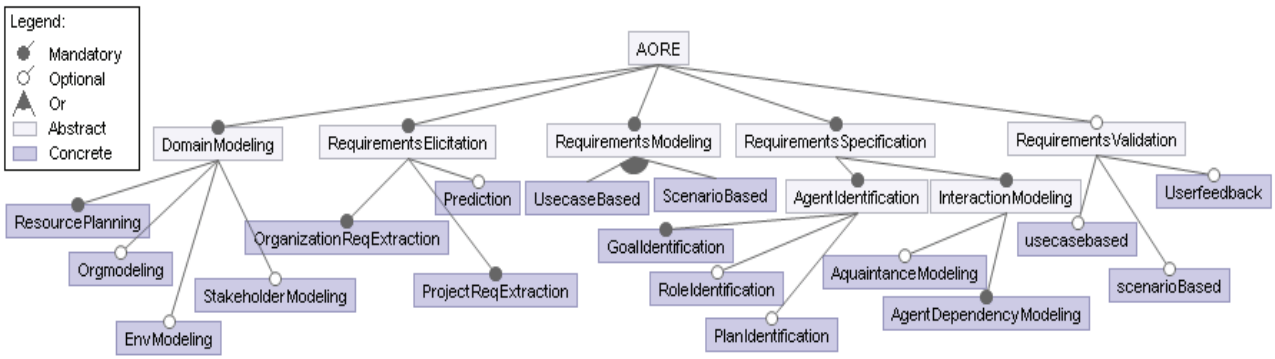

Fig. 3. Agent-Oriented Requirements Engineering: Feature Tree

Also, when identifying system-level functionalities, it can help us state the Why, in addition to the What and How, of the system functionality.

- *Technique 1:* In this approach, the environment is modeled as interactive actors in the early steps before system modeling. Then the system is added as an actor and its interactions with environmental actors are modeled [6], [11].
  Output artifact(s): Actor Diagrams and Goal Diagrams at a high level of abstraction
  Precondition (input artifact(s)): Preliminary specification of the system and its environment
- *Technique 2:* After modeling the agents and presenting them in class diagrams, environmental actors that interact with the system are added to this model [13].
  Output artifact(s): Multi-agent structure definition diagram
  Precondition (input artifact(s)): Definition of requirements, roles, agents, and protocols

2) *Task: Organization Modeling (OrgModeling):* An organization is a group of Agents (or roles) working together to a common purpose [12]. At a high level of granularity, we can consider the system as a set of organizations that interact with one another. Organizational view to the system and its environment can provide relatively straightforward mappings from organizational definitions to the agent structure through subsequent stages of the development process.

3) *Task: Resource Planning:* There are usually various types of resources in every system. Resource modeling and planning can have an important role in the analysis and design of systems. Resources cause certain additional dependencies among the entities existing in the system, or between the system and its environment. Also, constraints generated by the resource plan may necessitate applying changes to the definition of requirements.

- *Technique 1:* Resource usage can be modeled by assigning permissions to roles. Roles use the resources for fulfilling their responsibilities. For information resources, permissions are of three types: read, change, and generate [17].
  Output artifact(s): Roles Model
  Precondition (input artifact(s)): Requirements Statement
- *Technique 2:* Resources can be modeled as elements in actor models. By applying this technique, the dependencies of actors to resources for realizing goals and plans are modeled. These dependencies can be modeled at different levels of abstraction [11].
  Output artifact(s): Actor Diagrams and Goal Diagrams at various levels of abstraction
  Precondition (input artifact(s)): Appropriate level of knowledge about system resources
- *Technique 3:* Resources can be modeled as individual elements besides organizations and roles. Dependency of the resources to organizations can be determined in such a model. In addition to structural models, they can also be represented in interaction models [12].
  Output artifact(s): Structural and Acquaintance Relationships
  Precondition (input artifact(s)): Appropriate level of knowledge about system resources

B. Stage: Requirements Elicitation

Any activity that results in the exploration, identification, and/or determination of the requirements of the system is included in this stage. Some of these activities are common in all types of software development processes, and some of them are particularly agent-oriented.

C. Stage: Requirements Modeling

In this step, models are produced which refine, specify and document the initial requirements.

D. Stage: Requirements Specification

In this step, initial requirements, which were characterized earlier, are refined and modeled in a form that is usable for agent-oriented design and implementation.

1) *Task: Agent Identification:* In this step, initial requirements are transformed and modeled based on agent-oriented notions. Results of this task help base agents on correct assumptions. Any activity which helps achieve this aim is part of this task.

2) *Sub-Task: Role Identification:* This feature focuses on determining the roles played by agents. A Role describes the external characteristics of an Agent in a particular context [12]. Because of the conceptual cohesion of their functionalities, the roles can help in better structuring of these functionalities.

- *Technique 1:* After receiving the organizational specifications defined in earlier steps, the organization's overall behavior is decomposed into smaller collaborative organizations. Every one of these fine-grained behaviors will be demonstrated through one role. Regarding the system as an organizational structure at a high level of abstraction, this technique defines the roles at lower levels of abstraction [14], [17].
  Output artifact(s): Interactions and Role Identification diagram
  Precondition (input artifact(s)): Organization Identification Diagram
- *Technique 2:* After determining the agents by functionality classification, the agents are placed in sequence diagrams and role-related scenarios are applied to them. Thus, agents participate in collaborations in different roles; the preliminary structure of the roles is thereby determined, to be further refined in subsequent stages [13].
  Output artifact(s): Role Identification Diagram,
  Precondition (input artifact(s)): Agent Identification Diagram
- *Technique 3:* In this approach, goals are mapped to roles. Goal to role mapping is "one to one". But it may

be changed to "many to one", due to issues related to performance or execution platform [15].
Output artifact(s): Role Model
Precondition (input artifact(s)): Goal Hierarchy Diagram

3) *Sub-Task: Goal Identification:* A Goal is a different view of the requirements which is expressed from an Entity's point of view. An entity can be an agent, or an actor (at a higher level of abstraction. The concept of goal is more abstract than functionality. This feature focuses on goal analysis. This analysis can consist of determining the dependencies among functional and/or nonfunctional goals, and also the dependencies of goals to resources or plans.

- *Technique 1:* In this approach, an actor goal is analyzed from the point of view of the actor, through the use of different reasoning techniques; examples include means-end analysis, contribution analysis, and AND/OR decomposition [6], [11].
  Output artifact(s): Goal Diagrams
  Precondition (input artifact(s)): Preliminary specification of the system
- *Technique 2:* In this approach, system goals are defined and organized. For organizing the goals, a hierarchy of goals is defined, modeling the goals at different levels of abstraction and from different points of view [15].
  Output artifact(s): Goal Hierarchy Diagram
  Precondition (input artifact(s)): Functionality Descriptor

4) *Sub-Task: Plan Identification:* The plan, at different levels of abstraction, presents the procedure for performing a task. In Agent-Oriented Requirements Engineering, the plan presents a procedure for realizing a goal. Designing the plan may necessitate the definition of new goals or the modification of existing ones based on the defects detected in other elements with which the plan interacts.

- *Technique 1:* In this approach, plan modeling is used as a complementary technique for goal modeling, through using certain reasoning techniques. This technique uses plan diagrams, a diagram similar to activity diagrams, for addressing certain details [11].
  Output artifact(s): Goal Hierarchy Diagram
  Precondition (input artifact(s)): Functionality Descriptor
- *Technique 2:* In this approach, the plan is further decomposed into finer-grained parts, and is defined at the intra-agent level. As mentioned before, plans are procedures for performing tasks; in this technique, they are triggered by goals and events, and are modeled by Descriptors [16].
  Output artifact(s): Agent Overview Diagram, Plan Descriptor
  Precondition (input artifact(s)): System Overview Diagram, Agent Descriptor

5) *Task: Interaction Modeling:* This feature focuses on determining the interactions recurring among roles or agents. The interaction protocol can then be defined. Also, the message structure can be determined through this feature. The scenarios and interaction patterns thereby extracted can provide valuable feedback, which can be used for refining and adjusting the requirements.

6) *Sub-Task: Agent Dependency Modeling:* This feature focuses on the dependencies among entities for realizing goals, performing plans, and furnishing resources.

7) *Sub-Task: Acquaintance Modeling:* An acquaintance relationship indicates the existence of at least one interaction involving the entities concerned. This feature relates to determining and modeling this type of relationship between entities.

E. Stage: Requirements Evaluation

Any activity that the specified requirements are evaluated by, and whose results can be used to enhance the determination and specification of requirements, can be counted as part of this step.

VI. Agent-Oriented Requirements Engineering Process Line: Application Engineering Case Study

Application engineering is the process of producing concrete processes from the process line by defining the exact process situation and feature. To this aim, we consider PASSI as our target process and try to instantiate the requirements engineering part of it using the AORE process line.

PASSI [15] is a well-known AO methodology and covers the requirements engineering tasks, therefore it is feasible to instantiate it from the designed process line. The next step is to extract PASSI's features and map them to the process line features, and then instantiate the process from the process line and validate it through comparison with the original PASSI methodology.

The process instantiated from the process line is shown in Fig. 4. The instantiated process has all the method chunks that the RE part of the PASSI methodology has, yet it also has a number of additional method components, such as use-case-based validation and plan modeling, as it should be applicable independently.

The instantiation process was fast and helped us customize the main process to reach the target process. In addition, instantiation did not need special knowledge of method engineering; the person who instantiates the process is not required to be concerned with method-chunk dependencies and conflicts. Furthermore, the method chunks were well-defined in that they delineated the pre- and post-conditions of each fragment, thus providing the means to attach the constituent chunks together perfectly.

On the other hand, due to the lack of mapping procedures, the process engineer may get confused while mapping the *project* requirements to *process* requirements. In addition, because the proposed process line does not fully cover the software development lifecycle, connecting the instantiated process to the rest of the process lifecycle may prove problematic at the connection points.

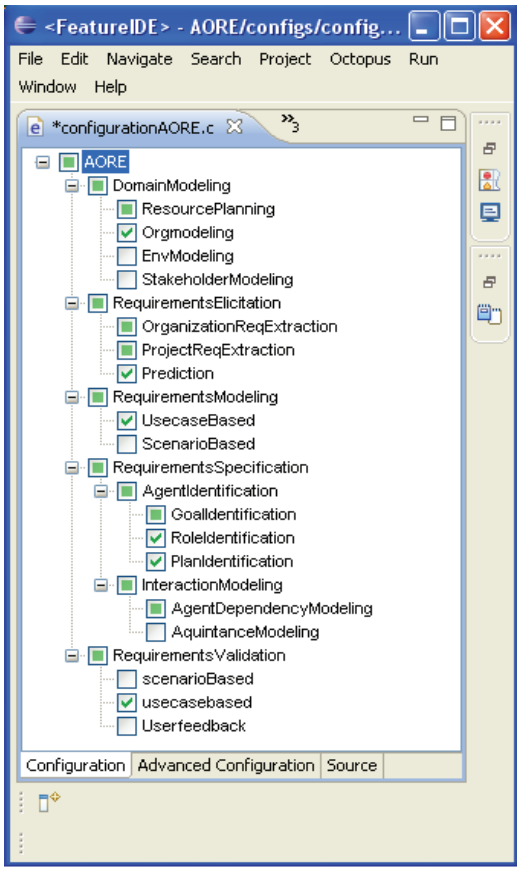

Fig. 4. AOREPL configuration for PASSI's requirements engineering process

VII. Conclusions and Future Work

We have proposed an Agent-Oriented Requirements Engineering Process Line (AOREPL), which covers all the method chunks required for successful application in relevant contexts. We also define variant method chunks to help process engineers in instantiating bespoke agent-oriented requirements engineering processes.

The proposed AOREPL leads to facilitated instantiation, since it has been implemented in the FeatureIDE tool and allows the process engineer to instantiate his target process easier, faster, with a higher quality and lower risk.

Furthermore, we use a conflict analysis algorithm to identify the relationships among method chunks, which results in knowledge-independent process lines. In other words, the process engineer or the one who is going to instantiate the process from the process line will not need to know the details of method chunk interdependencies, since we encapsulate our knowledge in every relationship and in the definition of every method chunk.

Future research in this context can focus on completing the proposed repository, and extending the proposed process engineering approach with procedures for facilitating requirements mapping. A separate strand can concentrate on extending the proposed process line to provide full coverage of the generic agent-oriented software development lifecycle, instead of just the AORE activities.

References

[1] L. Osterweil, "Software processes are software too," in *Proc. ICSE'87*, 1987, pp. 2-13.

[2] H. Washizaki, "Building Software Process Line Architectures from Bottom Up," in *Proc. PROFES'06*, 2006, pp. 415-421.

[3] O. Armbrust, M. Katahira, Y. Miyamoto, J. Munch, H., Nakao, A. Ocampo, "Scoping Software Process Models - Initial Concepts and Experience from Defining Space Standards," in *Proc. ICSP'08*, 2008, pp. 160-172.

[4] D. Rombach, "Integrated Software Process and Product Lines," in *Proc. ISPW'05,* 2005, pp. 83-90.

[5] P. Clements, L. Northrop, *Software Product Lines*, Addison-Wesley, 2002.

[6] S. K. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *Proc. ISRE'97*, 1997, pp. 226-235.

[7] E. Y. Lesperance, T. G. Kelley, J. Mylopoulos, and E. S. K. Yu, "Modeling Dynamic Domains with ConGolog," in *Proc. CAiSE'99,* 1999, pp. 365-380.

[8] X. Wang and Y. Lesperance, "Agent-oriented requirements engineering using ConGolog and i*," in *Proc. AOIS'01*, 2001, pp. 59-78.

[9] P. Donzelli, "A goal-driven and agent-based requirements engineering framework," *Requirements Engineering*, vol. 9, pp. 16-39, Feb. 2004.

[10] J. N. Mazon, J. Pardillo, and J. Trujillo, "A Model-Driven Goal-Oriented Requirement Engineering Approach for Data Warehouses," in *Proc. ER Workshops*, 2007, pp. 255-264.

[11] P. Bresciani, P. Giorgini, F. Giunchigilia, J. Mylopoulos, A. Perini, "TROPOS: An agent oriented software development methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, pp. 203-236, May 2004.

[12] G. Caire et al., "Agent oriented analysis using Message/UML," in *Proc. AOSE'01*, 2001, pp. 119-135.

[13] M. Cossentino, "From requirements to code with the PASSI methodology," in *Agent-oriented Methodologies*, B. Henderson-Sellers, P. Giorgini, Eds. Idea Group Publishing, 2005, pp. 79-106.

[14] M. Cossentino, N. Gaud, V. Hilaire, S. Galland, A. Koukam, "ASPECS: an agent-oriented software process for engineering complex systems," *Autonomous Agents and MultiAgent Systems*, vol. 20, pp. 260-304, Mar. 2010.

[15] S.A. Deloach, M. Wood, C. Sparkman, "MultiAgent Systems Engineering," *Software Engineering and Knowledge Engineering,* vol. 11, pp. 231-246, Jun. 2001.

[16] L. Padgham, M. Winikoff, "Prometheus: A Methodology for Developing Intelligent Agents," in *Proc. AOSE'03*, 2003, pp. 174-185.

[17] M. Wooldridge, N.R. Jennings, D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design," *Autonomous Agents and MultiAgent Systems,* vol. 3, pp. 285-312, Sep. 2000.

[18] P. Loucopoulos and V. Karakostas, *System Requirements Engineering*, McGraw-Hill, 1995.

[19] C. Bernon, M.p. Gleizes, P. Migeon, G. Serugendo, "ADELFE: A Methodology for Adaptive Multi-agent Systems Engineering," in *Proc. ESAW'02*, 2002, pp. 156-169.

[20] J. Debenham, B. Henderson-Sellers, "Designing agent-based process systems - extending the OPEN Process Framework," in *Intelligent Agent Software Engineering*, V. Plekhanova, Ed. Idea Group Publishing, 2003, pp. 160-190.