ReSeagent: A Refactoring Tool for Plan Level Refactoring in MAS Development

Ali Murat Tiryaki and Oguz Dikenelli
Ege University, Department Of Computer Engineering
35100 Bornova, Izmir, Turkey
Email: {ali.murat.tiryaki,oguz.dikenelli}@ege.edu.tr

In our previous work,

Abstract—The need for XP-like agile approaches that facilitate flexible evolutionary development has been widely acknowledged in the AOSE area. Such approaches improve acceptability of agent-technology by the industry. Evolutionary development of multi agent systems-MASs can only be applied successfully, if designs of the MASs being developed are improved throughout the development process. In our previous work, we have defined a refactoring approach that makes evolutionary MAS development possible. In this paper, we mainly aim to identify a development approach for MAS refactoring tools. In order to discuss this approach, we developed a refactoring tool called ReSeagent on the Seagent framework. Although the ReSeagent tool supports plan level refactoring patterns that are to be manually applied by the developers, ideas used in the implementation of this tool are generic ideas that provide a base for different refactoring types and development artifacts.

I. INTRODUCTION

Based on the experiences on agent-based system development, AOSE research community has realized that it is almost impossible to develop complex systems like multi agent systems - MAS in a sequential manner [33], [7]. The solution is the iterative approach which has been accepted as one of the best practices by software development community and integrated to all recent software development methodologies such as Rational Unified Process - RUP [26] and Extreme Programming - XP [3].

Managing the continuous evolution of software architecture and related design is one of the key issues in iterative development. XP introduces two critical practices to manage the evolution of software architectures: test driven development [21] and refactoring [18].

Test driven development produces test code for each class developed during iterations. This test code provides a protection shield against the flaws that can occur as a result of changes made on the working code by guaranteeing the functional accuracy of this code. The other best practice refactoring defines a process for improving the structure of the software system without altering the external behavior.

An iterative and incremental development life-cycle approach is quite appropriate for developing large scale distributed and complex systems such as MASs. XP-like agile processes, that introduce light-weight practices for iterative and incremental development in a controllable way, are needed to improve acceptability of the agent-technology by the industry [7], [33]. However, traditional testing and refactoring approaches and their supporting tools cannot be re-used directly

in MAS development, since MASs are built using different abstractions and techniques. So we need to re-define these practices for MAS development.

In [28], we have proposed a refactoring approach that makes evolutionary MAS development possible. This refactoring approach follows the route of traditional refactoring and provides some new refactoring patterns for MAS development. The proposed approach introduces some common problems called "bad smells" experienced during the development of MAS systems (such as duplicated behaviour structure and big plan) and maintenance strategies called "refactoring patterns" to overcome these bad smells. Each of the refactoring patterns defined in this proposed approach focuses on overcoming one or more than one bad smell(s) encountered during MAS development.

In this paper, we aim to introduce a basic development approach to produce MAS refactoring tools. Then, we implement a refactoring tool called as ReSeagent on Seagent MAS development framework [14], [15] by using the proposed approach. This tool supports refactoring on agent plans supported by almost all MAS development methodologies such as Passi [10], Tropos[5] and MaSe [13]. The tool can be used manually by developers during MAS development activities. ReSeagent focuses on refactoring Seagent plans whose meta model is defined clearly in Seagent. However, software architecture of the tool is generic and it can be used for other planning systems whose meta-models are explicitly defined or other MAS design artifacts such as role, goal and protocols.

II. RELATED WORKS

In the literature, there are some pioneering works which try to apply agile practices to MAS development.

Knublauch [20] used practices of extreme programming - XP [3], which is the one of the most known agile development processes used for MAS development. Although, this work proves the effectiveness of XP practices in terms of MAS development, refactoring is not explained in detail. Since the agent development framework and process meta-model, which are used during development, are very simple, refactoring operations on agents seem as very simple processes and refactoring practice is applied on agents. However, an agent that is developed by using a realistic development framework can play many roles in MAS and those roles have many goals, responsibilities and abilities. So, we believe that agents are not

small; on the contrary they are too big entities for testing and refactoring.

In another important work has been introduced by Chella et. al. [8], well known Passi methodology is transformed to Agile Passi. The testing framework developed by the Agile Passi research team provides an automated testing approach for testing multi-agent systems [6], [11]. Agile Passi approach does not introduce an iterative or evolutionary style for MAS development. Therefore, a refactoring approach that makes agile MAS development possible is not introduced in this work.

In [9], an agile methodology for MAS development is introduced. This methodology is a generic methodology based on the practices such as test driven development and refactoring that come from the agile approaches. The process of SADAAM consists of four key phases: design, test driven implementation, release & review and refactor & enhancement, that are applied iteratively until a finished state is reached. However, any detailed discussion on how the practices called test driven development and refactoring are applied during MAS development. Hence, the proposed methodology is a generic methodology that does not add too many specific ideas to the abstract development process proposed by agile approaches. To concretize how the proposed methodology is used for MAS development, the agile practices in the methodology have to be discussed exhaustively.

Another work [30] focuses on to define which and how traditional refactoring practice can be used for agent based simulation systems on a multi agent simulation systems modeling paradigm called MASim. At the end of the working, a catalog that consists of the refactoring patterns used to improve the system designs based MASim is introduced. Moreover, some of the proposed refactoring patterns has been implemented and integrated to an agent based simulation platform called SeSam. This working does not introduce a general refactoring approach and its main characteristics for MAS development. The proposed refactoring patterns can be used to only the agent systems that have a specific type. To define the refactoring patterns that can be used during MAS development, firstly a general refactoring approach based on the characteristics of these systems has to be defined.

In [29], an iterative and incremental development approach called agent oriented test driven development - AOTDD is proposed to handle the complexity and continuously changing nature of the requirements in MAS development. In AOTDD, developers follow the development cycle with adding the new functionalities to the system between iterations, just like all other agile & iterative development approaches. Also, the life cycle of proposed test driven approach and a testing tool that supports the proposed test driven approach are introduced in this work. Since this work is focused on the testing part of test driven development, the refactoring step that is very critical for iterative and incremental development is not discussed in detail.

These works neither propose a systematic approach for MAS refactoring nor introduce a refactoring tool support for these approaches. In this paper, we focus on these two points.

III. A Development Approach for MAS Refactoring Tolls

Refactoring is directly dependent on the executable artifacts of the developed system. In AOSE area, it is not possible to define a set of executable artifacts that can be agreed on, since there are several agent architectures such as BDI, re-active, self-organized and several development approaches such as Gaia [32], Tropos [5] and Adelfe [4] that aim to develop agents based on these agent architectures. Naturally, different executable artifacts may emerge based on the used agent architecture and/or development frameworks that support these architectures. As a conclusion, it is impossible to develop a refactoring tool that is usable for all kinds of MAS development artifacts. Instead of this, we need a generic approach to develop MAS refactoring tools. This approach can be specified for different MAS architectures and/or different artifacts to produce a suitable refactoring tool.

The proposed development approach has three steps listed as follows:

1) Define the meta-model of the target executable development artifacts,
2) Define the bad smells encountered during the development of the artifacts specified in the previous step,
3) Define the refactoring patterns that overcome the bad smells defined for the target development artifact.

To illustrate how the proposed approach can be applied to refactoring tool development, we chose the agent plans as the target development artifact. Agent plan abstraction is one of the most common artifacts in MAS development methodologies. Almost all of the MAS methodologies use plan abstraction (it is named with different terms in different methodologies) to model agents' internal behaviours. For example, in the goal oriented development approach, each agent goal is achieved by one or more agent plan(s). Several approaches can be used to build plan structures such as HTN [31], [23] based on the agent infrastructure. Developers have to specify the planning approach before they apply first step of the proposed refactoring tool development approach (defining the meta-model of the target artifacts).

In our case, we aim to develop a refactoring tool for the Seagent framework that was developed by our research group. So, our refactoring tool called ReSeagent aims to refactor agent plans that are directly dependent on the Seagent planning infrastructure.

The rest of this section explains how the steps of the proposed approach were applied for the plan artifact in the Seagent framework.

A. Define the meta-model of the target executable development artifacts

Performing a refactoring pattern requires a clear understanding of the abstract syntax and semantics of both the source and target models. Meta-modeling is a common technique for defining the abstract syntax of models and the interrelationships between model elements. To identify refactoring patterns and bad smells for an executable design artifact, we have to

know the meta-model used to build structures of this type of artifacts.

In Seagent framework, the hierarchical task network - HTN formalism [23], [31] is used to build and store agent plans. Hence, the plan level refactoring patterns introduced in this paper are dependent on the HTN formalism. ReSeagent refactoring tool supports to apply the mechanics of these HTN dependent refactoring patterns on HTN plans that are developed through Seagent. The structure of the HTN meta model used in the current Seagent version is shown in the figure 1.
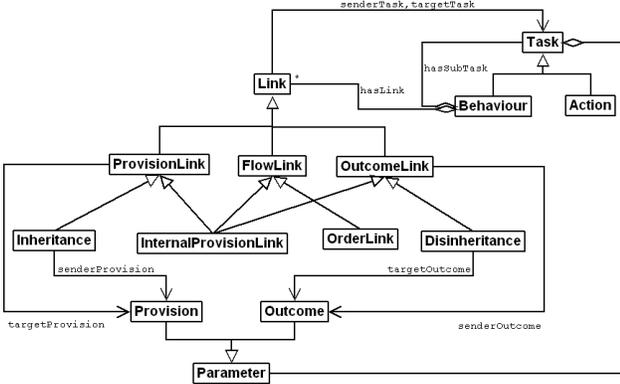

Figure 1.   Seagent HTN Meta Model

In HTN formalism, there are two kinds of task; complex task (we call behaviour) and primitive task (we call action). Complex tasks hold the structure of its sub-tasks and links between these tasks. Primitive tasks have directly executable code. Information requirements of tasks are illustrated as provisions. Outcomes are result states of tasks. Data is transferred to other tasks through the several kinds of link. An inheritance link is used to transfer a provision of a parent complex task to a sub task. Disinheritance links are used to transfer outcomes of sub-tasks to parent complex task. And finally, the information flow between outcomes and provisions of the tasks in the same level is provided by internal provision links. As an addition of the links come from the HTN formalism, there is one more type of link called order link in the Seagent behavioural PSM. Order links do not pass any information between tasks. They are used only to order the execution of tasks.

B. Define the bad smells encountered during the development of the artifacts specified in the previous step,

To make decision about applying refactoring patterns on an artifact, we need to define the common design problems that are encountered by developers during the system development process frequently. Fowler named these common problems as bad smells in [18] and introduced refactoring patterns to overcome these bad smells in software design. Each of the defined refactoring patterns is introduced to overcome one or many bad smell(s).

Similarly, we have to define bad smells for MAS development. Based on our experiences on the MAS development, we have identified following bad smells for plan level refactoring: Duplicated behaviour structure, execution decision in a plan, big behaviour, incoherent behaviour, redundant task group.

Detailed definitions of all defined bad smells are accessible through the Seagent web site[1].

C. Define the refactoring patterns that overcome the defined bad smells for the target development artifact.

To develop a refactoring tool that can be used during MAS development, the refactoring patterns that define the order of the mechanics for each refactoring pattern has to be defined by a standard way. Each pattern includes pre-defined mechanics in order to overcome one or more bad smell(s). These mechanics has to be defined by using a machine understandable semantic. The tool uses the pre-defined machine understandable pattern definitions to overcome bad smells.

In ReSeagent, we have defined seven refactoring patterns for HTN based plan refactoring. These plan level refactoring patterns that were defined based on our experiences with its initiator bad smell(s) are shown in table I.

Refactoring	Bad Smell
Extract Plan	Execution Decision in Plan
Extract Behaviour	Big Behaviour Duplicated Behaviour Structure
Behaviour to Plan	Execution Decision in Plan
Rename Behaviour	Incoherent Behaviour
Replace Tasks with a Task	Redundant Task Group
Deliver from Behaviour	Incoherent Behaviour
Extract Super-Behaviour	Duplicated Behaviour Structure

Table I
REFACTORING PATTERNS AND THEIR INITIATOR BAD SMELLS IN THE PLAN LEVEL

The detailed definitions of all refactoring patterns defined are accessible on the refactoring patterns part of the Seagent web side.

IV. A HYBRID APPROACH FOR DEFINITION OF REFACTORING PATTRENS

To implement a refactoring tool, firstly the refactoring techniques supported by the tool have to be defined in a usable form for the tool. We used a hybrid approach based on the mechanisms used for defining transformation rules in the Model Driven Enginnering - MDE to define the refactoring tecniques mentioned in the previous section for our refactoring tool. MDE is a discipline in software engineering that relies on models as first class entities and that aims to develop, maintain and evolve software by performing model transformations [22], [25]. This section introduces our hybrid approach that is based on the model transformation techniques [19] and used to define refactoring techniques in ReSeagent refactoring tool.

---

[1]http://etmen.ege.edu.tr/wiki/index.php/refactoring_agent_system

From the MDE perspective, a refactoring operation is a model transformation between the initial and the improved models. Model is a structure of design artifacts in the refactoring context. Like each model transformation has some teransformation rules, each refactoring has some mechanics that achieve main goal of the refactoring by working together. It is natural to think that the refactoring approaches and tools can be based on the model transformation approaches and techniques. To develop a refactoring tool by using the model transformation techniques, mechanics of refactorings should be defined as the transformation rules. In this manner, the refactoring tool operates the refactoring mechanics that are defined as transformation rules according to the refactoring pattern in order. This is not a new idea and there are some works about rule based refactoring in the literature such as [2], [24]. All of these works aim to define refactoring operations by rules for their own target development sytle.

Transformation rules are the focal point for model transformation. The techniques and languages that are used to identify these rules specify the maim characteristics of model transformation. There are two main approaches to define trasnformation rules; declarative and imperative approaches [12]. Declarative approaches (e.g., [1]) are attractive because particular services such as source model traversal, traceability management and automatic bidirectionality can be offered by an underlying reasoning engine. On the other hand, imperative approaches (e.g., [27]) may be required to implement transformations for which declarative approaches fail to guarantee their services. Especially when the application order of a set of transformations needs to be controlled explicitly, an imperative approach is more appropriate thanks to its built-in notions of sequence, selection and iteration.

In Seagent framework, plan models are stored in ontologies that are built by using a description logic based language called Web Ontology Language - OWL[2]. Refactoring operations on these plan models can be considered as model transformations between initial plan ontologies and improved plan ontologies. So, a logic based declerative approach looks like appropriate for building rules on the these models. A refactoring tool that supports transformation between OWL ontologies using logic based rules is useful for refactoring Seagent plan models. However, logic based rules are not enough for defining refactoring mechanics because of the two handicaps of logic based declarative approaches listed belove:

1- Almost all of the logic languages such as Prolog have been developed to extend their target models. By using these languages, new definitions can be made on the existent elements in the model. These languages do not support to remove and change the existent elements in the model. Refactoring techniques require removing and changing of the existent elements in the model besides extending of the model.

2- Another handicap of the declerative approach for creating transformation rules of refactorings is that the mechanics of refactorings should be operate in sequence. As mentioned above, such a sequence operation can be controlled explictly by an imperative approach.

Because of these handicaps, the declarative approach cannot be used alone to define refactoring mechanics. To define refactoring mechanics as rules, imperative or hybrid approaches should be used.

ReSeagent uses a hybrid approach that combines the advantages of declarative and imperative rule definition approaches. The declarative side of this approach is achieved by using Semantic Web Rule Language - SWRL[3]. SWRL is a logic based rule language that supports defining rules on OWL ontology models. Refactoring mechanics that extend the initial models are defined as SWRL rules. Refactoring mechanics defined using the HTN paradigm forms the imperative side of our hybrid approach[31], [23].

In this hybrid approach, each refactoring pattern is defined as an agent plan by using the HTN paradigm and each refactoring mechanic is implemented as an executable action in refactoring plans. Refactoring mechanics that extend the initial models are implemented as rule actions. Rule actions have the responsibility of operating a SWRL rule on the OWL model of the related Seagent plan. Refactoring mechanics that change and/or remove the existing element in the initial model are implemented as normal actions. Normal actions implement change and removal activities by means of Java code that handles the ontology explicitly. All of the actions that are implemented to realize the mechanics of a refactoring pattern compose a refactoring plan that achieves the main goal of the refactoring pattern. Sequential execution of these actions is controlled by the plan structure.

Many different methods such as finite state machine can be chosen to implement the imperative side of our hybrid rule definition approach. However, defining each refactoring pattern as an agent plan by using the HTN paradigm in ReSeagent has some advantages listed below:

- **Simplicity:** In Seagent, HTN paradigm is used for testing and implementation of plans. Defining refactoring patterns by using the same method simplifies the addition of new refactoring patterns into the refactoring tool.
- **Reusability:** HTN paradigm makes it possible to reuse other pre-defined plan structures in higher level plan structures. Thanks to this, big refactorings can be simply implemented by re-using the pre-defined refactoring plans in a high level refactoring plan.
- **Generality:** Since the refactoring plans are agent plans like domain dependent user plans, these refactoring plans can also be refactored by applying refactoring plan(s) on these plans.

The software architecture of ReSeagent that executes the refactoring patterns defined by our hybrid approach is explained in the following section.

V. Overal Software Architecture of ReSeagent

ReSeagent refactoring tool was implemented as a plug-in on the Seagent plan editor in the Seagent Development Environment - SDE like the refactoring support of Eclipse. ReSeagent gives suport for refactoring the plan models developed using

---

[2]http://www.w3.org/TR/owl-features/

[3]http://www.w3.org/Submission/SWRL/

SDE. The tool applies pre-defined refactoring patterns on the related plan models to fulfill the refactoring requests received from Seagent plan editor.

ReSeagent focuses on the refactoring of Seagent plans whose meta-model is clearly defined in Seagent. Additionally, since the software architecture of the tool is generic and it can be used for other planning systems whose meta-models are different or for other MAS design artifacts such as role, goal and protocols. To add support for the artifacts that have different meta-models, you have to add new refactoring plans that work on these meta-models into the refactorer role of ReSeagent, and make some additions to the initiator module for initiating these new refactoring plans.
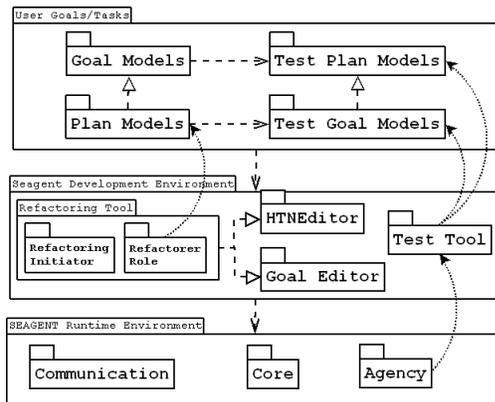

Figure 2.   Overal Software Architecture of ReSeagent

SDE has the responsibility of developing executable artifacts such as plan models and goal models that can be executed by Seagent. For this purpose, SDE includes a plan and a goal editor for developing plan and goal models. It includes also a testing tool called SeaUnit that verifies the functionalities of these development artifacts [16]. The overall software architecture of ReSeagent and its dependencies on other modules in SDE is shown in figure 2.

ReSeagent consist of two sub-packages: refactoring initiator and refactorer role. Refactoring initiator package was developed as a plug-in on the Seagent Plan Editor like the refactoring support of the well known Eclipse environment. It has the responsibilities of capturing the refactoring requests in the plan editor, and initiating a refactoring operation for each of these refactoring requests. On the other hand, refactoring role holds the refactoring plans that are defined to provide the mechanics of the refactorings.

A. Refactoring Initiator

Refactoring initiator module of ReSeagent has a simple structure. This module has the responsibilities of capturing the refactoring requests from the plan editor and the user preferences that are needed to fulfill these requests, and initiating a refactoring operation that is suitable for each of the captured refactoring requests.

When the developer wants to start a refactoring operation on a pre-defined plan model by using the refactoring plug-in of the Seagent plan editor, the initiator initiates an agent that plays the refactorer role and then passes the refactoring request with its inputs to this refactorer agent. This agent maps the received refactoring request to a suitable refactoring plan situated in the plan library of the refactorer role, passes the input values to this plan and executes it. During the plan execution, the input values of the refactoring requests are passed to the sub-tasks of the plan via the links in the HTN structure. Then, these subtasks are executed depending on their order in the HTN structure. At the end of the plan, updated plan models are returned to the initiator via the outcomes so that the results can be shown to the plan editor user.

There can be many plans that realize the same refactoring operation by means of different ways. In such a situation, refactorer agent decides which refactoring plan has to be executed, according to inputs of the refactoring request and its internal state. This ability of the refactorer agent comes from Goal Mapping Engine in Seagent framework [17]. Thanks to this engine, Seagent agents can map a request to most suitable of many plans that achieve same goal in different ways. When this decision is made, some criteria such as inputs and outputs of the goal are considered by the Goal Mapping Engine. To add more than one refactoring plan that can realize the same refactoring operation to Seagent, it is enough to add correct mapping definitions to the knowledge base that are used for goal mapping (called Goal Mapping Ontology in Seagent) by the refactorer agent .

Another responsibility of the initiator is passing the updated plan model(s) to the plan editor. Each refactoring plan has some outcomes that return the plan models that have been changed during the execution of the refactoring plan. There is one outcome for each updated plan model. Initiator listens to the planner of the refactorer agent after it initiates this agent. When a PlanFinished event is thrown by the planner, the initiator captures the updated plan models returned by the outcomes and updates the model(s) in the plan editor. Hereby, new structures of the refactored plans are shown to the user.

B. The Refactorer Role

Refactorer role is a special role that has refactoring goals and refactoring plans that achieve these goals. This role can access plan models and action definitions in the system. This role has the responsibility of applying plan level refactorings on plan models during MAS development.

The refactorer role has refactoring goals that aim to apply a specific refactoring pattern on the Seagent plan models. Each of these refactoring goals is achieved by one or more than one refactoring plans. So, the plan library of this role has to include at least one refactoring plan for each refactoring pattern supported by ReSeagent tool.

The current version of ReSeagent refactoring tool supports the following plan level refactoring patterns: *Replace Tasks with a Task, Extract Behaviour, Behaviour to Plan, Extract Plan.*

The detailed definitions of these refactoring plans can be found on the ReSeagent plans page of the Seagent web site.

Also, the OWL ontologies of the plans and Java codes of the actions in these plans are downloadable on this page.

To give an insight about the refactoring plans in ReSeagent, one of the refactoring patterns and the refactoring plan developed to achieve this refactoring in ReSeagent are explained in the following section.

Replace Tasks with a Task Refactoring and Its Implementation in ReSeagent

Replace Tasks with a Task refactoring can be used in such a case: a functionality achieved by more than one tasks in a plan structure can be achieved by only one task. This refactoring removes these tasks from the plan structure and adds the new task to the plan structure instead of the removed tasks.

When the *Redundant Task Group* bad smell is realized in a plan structure, the plan can be made more readable and simpler by replacing the task group with a task that can achieve same functionality.

To apply *Replace Tasks with a Task* refactoring, the new task has to have all of the provisions and outcomes of the tasks in the replaced task group that are linked to other tasks in the plan structure.

Mechanics:

1) Remove the task group from the plan and add the new task.
2) Find inheritance and provision-outcome links that are attached to the provisions of the replaced tasks and attach such links to the suitable provisions of the new task.
3) Find disinheritance and provision-outcome links that are attached to the outcomes of the replaced tasks and attach such links to the suitable outcomes of the new task.
4) Scan the provision-outcome links whose source or target task(s) is the reference to the replaced tasks. If there are such links, remove these links from the plan structure.

Replace Tasks with a Task refactoring plan in ReSeagent refactoring tool is developed to realize the goal of "replacing a task group with a task that can fulfill same functionality" which is the aim of *Replace Tasks with a Task* refactoring pattern. HTN structure of this refactoring plan that implements the mechanics of *Replace Tasks with a Task* refactoring pattern is shown in figure 3.
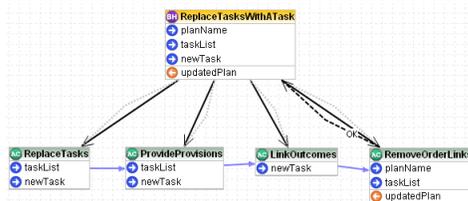

Figure 3.   HTN structure of the *Replace Tasks with a Task* refactoring plan in ReSeagent

The plan takes the name of the plan whose structure is refactored, the list of the tasks that are to be removed from the plan structure and the task that is to be added into the plan structure as provisions, and transfers these provisions

to its sub-tasks. The plan has four actions. Each of these actions achieves one of the mechanics of the *Replace Tasks with a Task* refactoring pattern. These actions which are called *ProvideProvisions* and *LinkOutcomes* extend the model of the target plan structure by adding new links. Hence, each of these actions operates a SWRL rule to fulfill its responsibility. At the end of the plan execution, updated plan structure is returned through an outcome.

VI. CASE STUDY

In this section, we introduce a case study that shows the usage of our refactoring approach and ReSeagent refactoring tool during the development of an actual MAS application which is a conference management system that has been developed by Seagent group.

At the beginning of the development, we did not intervene the developers and let them to follow a sequential development process that does not impose evolutionary development. The developers developed some of the goals such as "building the program committee", "sending call for paper" and "initiating a conference" by applying the activities of their development process.

After developing a few of the system goals, some bad smells emerged in the design of the MAS that was developed: some plan structures were duplicated in many plans. Furthermore, the developers were disappointed from the unmanageable structures of the plans developed.

We can give an example for these bad smells on a simple plan structure from the conference management MAS. This plan structure achieves the "sending call for paper" goal of the "organization" role in this system. The initial HTN structure of this plan that was obtained at the end of the sequential development process for "sending call for paper" goal is shown in figure 4.
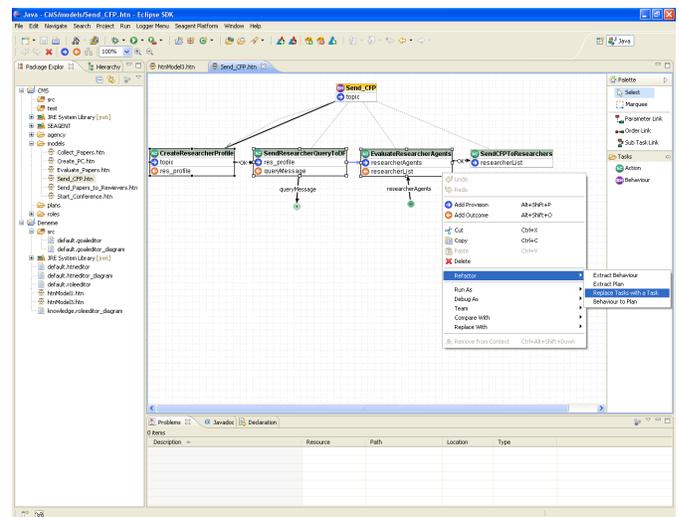

Figure 4.   The initial plan structure of the Send_CFP plan

The simple plan in the figure 4 takes the conference topic as a provision. This provision is passed to the "create suitable researcher profile" action through an inheritance link. In this action, a researcher profile object is created, the interested_topic

field of this profile is set with the topic that is received as a provision and this researcher profile is returned through the "OK" outcome. The other action called "prepare and send query message to DF" takes the researcher profile, creates a query message by using this profile and sends this message to directory facilitator - DF. The "evaluate incoming researchers" action has an external provision called researcherList. This provision includes agent descriptions of the researcher agents that are sent by the DF. In this action, the description of the researcher agents are filtered according to the preferences and suitable researchers are selected. The final action called "send CFP to selected researchers" has the responsibility of sending call for paper of the conference to selected researchers using the agent descriptions that are received as a provision.

Some tasks in the plan structure have a common goal called "finding the suitable researcher agents" that should be tested independently from the other goals. This goal can also be part of the other plan structures such as "create program committee" in the system. This was a bad smell called *Duplicated Behaviour Structure*. So, we decided to collect the actions called *CreateResearcherProfile, SendResearcherQueryToDF* and *EvaluateResearcherAgents* into a new plan that achieves the common goal by applying *Extract Behaviour* refactoring on these actions.

To initiate an *Extract Behaviour* refactoring operation on these three actions, we selected these actions and right clicked the mouse. In the opened menu, we chose the Refactor —> Extract Behaviour as shown in figure 4. Then, a window releated with the *Extract Behaviour* refactoring appeared on the screen. This window had three spaces that had to be set by the input values of the plan. These values were plan name, tasks list and new task. When we set all of the spaces by values and then clicked to the start button, the refactoring operation was initiated. After, a short duration, the improved structure of the plan appeared in the Seagent plan editor.
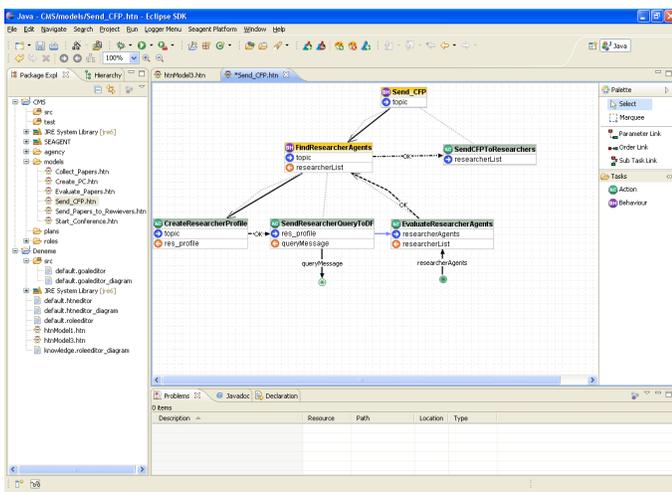

Figure 5.   Final structure of the Send_CFP plan

At the end of the *Extract Behaviour* refactoring, we obtained a new plan called *FindResearcherAgents* that can be re-used in the other plan structures. This plan has the responsibility of finding agent descriptions of the researchers that work on

the conference topic according to the topic provision. The *FindResearcherAgents* plan was simply used in some other plans in conference management system. The plan structure of our "send CFP" plan after the *Extract Behaviour* refactoring is shown in figure 5.

After the refactoring operations on the developed plan structures, we obtained more readable and manageable plans for the conference management system. Reusable plan structures such as *FindResearcherAgents* obtained during the refactoring operations simplified to development of other system goals that include the goals achieved by these plans.

VII. CONCLUSION

In this paper, we define a development approach for refactoring tools that can be used during the development of MASs. To discuss this development approach on a refactoring tool implementation, a refactoring tool called ReSeagent has been implemented on the Seagent framework by following the process proposed by this approach. This tool supports the manual application of pre-defined refactoring patterns stored as agent plans on Seagent plan models during the development activities. So far, ReSeagent refactoring tool was used in the development of the several MAS applications such as e-barter and conference management systems developed by the Seagent group. The experiences we obtained during the development of these systems show that ReSeagent refactoring tool facilitates evolutionary MAS development by simplifying the refactoring process.

REFERENCES

[1] David H. Akehurst and Stuart J. H. Kent. A Relational Approach to Defining Transformations in a Metamodel. In *The Unified Modeling Language: Model Engineeing, Concepts, and Tools*, volume 2460 of *Lecture notes in computer science*. Springer, 2002.

[2] Joachim Baumeister and Dietmar Seipel. Verification and refactoring of ontologies with rules. In Steffen Staab and Vojtech Svatek, editors, *EKAW*, volume 4248 of *Lecture Notes in Computer Science*, pages 82–95. Springer, 2006.

[3] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[4] Carole Bernon, Valérie Camps, Marie P. Gleizes, and Gauthier Picard. Engineering Adaptive Multi-Agent Systems: The ADELFE Methodology . In Brian H. Sellers and Paolo Giorgini, editors, *Agent-Oriented Methodologies*, pages 172–202. Idea Group Pub, NY, USA, juin.

[5] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[6] G. Caire, M. Cossentino, A. Negri, A. Poggi, and P. Turci. Multi-agent systems implementation and testing. In *From Agent Theory to Agent Implementation, Fourth International Symposium (AT2AI-4)*, 2004.

[7] L. Cernuzzi, M. Cossentino, and F. Zambonell. Process models for agent-based development. *Journal of Engineering Applications of Artificial Intelligence*, 18 (2), 2005.

[8] A. Chella, M. Cossentino, L. Sabatucci, and V. Seidita. From passi to agile passi: Tailoring a design process to meet new needs. In *IEEE/WIC/ACM International Joint Conference on Intelligent Agent Technology (IAT-04)*, 2004.

[9] Neil Clynch and Rem Collier. Sadaam: Software agent development an agile methodology. In *Proceedings of the Workshop of Languages, methodologies, and Development tools for multi-agent systems*, 2007.

[10] Massimo Cossentino, Luca Sabatucci, and Antonio Chella. Patterns reuse in the passi methodology. In *Proceedings of the Fourth International Workshop Engineering Societies in the Agents World (ESAW'03*, pages 29–31. Springer-Verlag, 2003.

[11] Massimo Cossentino and Valeria Seidita. Composition of a new process to meet agile needs using method engineering. In *SELMAS*, pages 36–51, 2004.

[12] Krzysztof Czarnecki and Simon Helsen. Classification of model transformation approaches. In *OOPSLA-03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.

[13] DeLoach S. A. Multiagent Systems Engineering A Methodology and Language for Designing Agent Systems. In *Proc. of Agent Oriented Information Systems*, pages 45–57, 1999.

[14] Oguz Dikenelli, R. C. Erdur, O. Gumus, E. E. Ekinci, O. Gurcan, G. Kardas, Inanc Seylan, and Ali Murat Tiryaki. Seagent: a platform for developing semantic web based multi agent systems. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1271–1272, New York, NY, USA, 2005. ACM Press.

[15] Oguz Dikenelli, Riza Cenk Erdur, Geylani Kardas, Ozgr Gumus, Inanc Seylan, Onder Gurcan, Ali Murat Tiryaki, and Erdem Eser Ekinci. Developing multi agent systems on semantic web environment using seagent platform. In *Engineering Societies in the Agents World VI*, volume 3963 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2006.

[16] Erdem Eser Ekinci, Ali Murat Tiryaki, and Oguz Dikenelli. Goal oriented agent testing revisited. In *Agent Oriented Software Engineering 2008*. Springer Verlag, 2008.

[17] Erdem Eser Ekinci, Ali Murat Tiryaki, Onder Gurcan, and Oguz Dikenelli. A planner infrastructure for semantic web enabled agents. In *OTM Workshops*, volume 4805 of *Lecture Notes in Computer Science*, pages 95–104, Vilamoura, Algarve, Portugal, 2007. Springer.

[18] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA, 1999.

[19] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture–Practice and Promise*. Addison-Wesley Professional, April 2003.

[20] Holger Knublauch. Extreme programming of multi-agent systems. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 704–711, New York, NY, USA, 2002. ACM Press.

[21] Johannes Link and Peter Frolich. *Unit Testing in Java: How Tests Drive the Code*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[22] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, March 2006.

[23] Massimo Paolucci, Dirk Kalp, Anandeep S. Pannu, Onn Shehory, and Katia Sycara. A planning component for retsina agents. In *Lecture Notes in Artificial Intelligence, Intelligent Agents VI*, 1999.

[24] Ivan Porres. Rule-based update transformations and their application to model refactorings. *Software and System Modeling*, 4(4):368–385, 2005.

[25] Shane Sendall and Wojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Software*, 20:42–45, 2003.

[26] Rational Software. The rational unified process, 1998.

[27] Jonathan Sprinkle, Aditya Agrawal, Tihamer Levendovszky, Feng Shi, and Gabor Karsai. Domain model translation using graph transformations. In *ECBS*, pages 159–167. IEEE Computer Society, 2003.

[28] Ali Murat Tiryaki, Erdem Eser Ekinci, and Oguz Dikenelli. Refactoring in multi agent system development. *Lecture Notes in Artificial Intelligence*, 5244:183–194, 2008.

[29] Ali Murat Tiryaki, Sibel Öztuna, Oguz Dikenelli, and Riza Cenk Erdur. Sunit: A unit testing framework for test driven development of multi-agent systems. In *AOSE*, volume 4405 of *Lecture Notes in Computer Science*, pages 156–173. Springer, 2006.

[30] Cornelia Triebig and Franziska Klugl. Refactoring of agent-based simulation models. In *Multikonferenz Wirtschaftsinformatik*, 2008.

[31] M. Williamson, K. Decker, and K. Sycara. Unified information and control flow in hierarchical task networks. In *Theories of Action, Planning, and Robot Control: Bridging the Gap: Proceedings of the 1996 AAAI Workshop*, pages 142–150, Menlo Park, California, 1996. AAAI Press.

[32] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12(3):317–370, 2003.

[33] Franco Zambonelli and Andrea Omicini. Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent Systems*, 9(3):253–283, 2004.