Supporting Evolving Multi-Agent Systems with a System Evolution Directory

Bin Lu, Leon Sterling, Kuldar Taveter
Department of Computer Science and Software Engineering
the University of Melbourne
Vic 3010, Australia
{bin,leon,kuldar}@csse.unimelb.edu.au

Abstract

In order to describe how a structure - System Evolution Directory (SED), can be used to facilitate the evolution of multi-agent systems, we first introduce a case study on citation finding. We use this case study to explain the components within a SED and the process to build a SED. A unified and simple model directory is used to describe the life cycle of a system and a goal model to describe the meaning among system artefacts. Developers can not only get all necessary information ranging from high level system requirements to low level source code from the model, but also can understand semantic connections between system artefacts. Different versions of a system can be recorded easily in the structure. We believe the structure has potential for further speeding up and automating the process of system evolution. All in all, the structure complements current agent-oriented methodologies.

1 Introduction

Software agents have attracted a lot of attention recently [30]. It is believed to be a good way to add flexibility and intelligence to current software. Although there is no standard definition of agent, we think of an agent as an intelligent piece of software which acts reactively as well as proactively on behalf of a person to perform a complicated or time-consuming job, or some task requiring intelligence.

In order to help developers develop agent-oriented systems efficiently and effectively, agent-oriented software engineering (AOSE) methodologies such as Tropos [13], Prometheus [21] and ROADMAP [15] have been introduced. The core idea of AOSE is to use the experience accumulated from traditional software engineering to build software agents and to utilize agent concepts including goal, role, and organization to build systems.

Not only has agent-oriented technology provided us a new perspective of software system designing, it has also brought challenges to multi-agent system evolution.

A released multi-agent system includes many artefacts, such as goal models, role models, design models, source code, and user manuals. These artefacts reflect different aspects of the system and various system abstraction levels. Any omission or inconsistency of these artefacts can make system evolution difficult and even impossible.

In this paper, we present a structure for multi-agent software evolution. This structure will handle and improve the following issues.

The first issue is to organize all system artefacts logically and effectively in a single place. System artefacts are created and maintained by different developers, namely analysts, designers, programmers etc, who have different responsibilities in the software life cycle. In order to fulfil their responsibilities, they need to share information. However, lack of a unified mechanism makes storing and sharing information difficult and even impossible.

The second issue is to record and reflect the meaning and relationship between system artefacts. The most significant example is the lack of mechanism to record and reflect the meaning of source code explicitly. The meaning of source code is stored in the memory of programmers and is reflected through system execution. Reverse engineering attempts to solve this problem through source code analysis, but we do not think it can solve it completely because its mechanism is based on guesswork. However, the meaning and relationship are important in software evolution. Without them, understanding a system is difficult. Therefore, we need a mechanism to record and reflect the meaning and relationship between system artefacts.

In order to handle the first issue, we provide a goal-centric way implemented by directory models to organize and manage system artefacts. All directory models are connected together to form a goal-tree. For the second issue, we introduce a goal model to record the meaning and semantic relationships between artefacts. All these models are connected together to form a new structure, the system evolution directory (SED), to facilitate the development of...
an evolvable agent-oriented system.

Three case studies of system evolution have been conducted; the results indicate that the System Evolution Directory is a promising new strategy to facilitate the evolution of agent-oriented systems.

A tool was designed and implemented to help the establishment and automation of the directory.

We organize this paper as follows. Section 2 introduces the system evolution directory and a case study to show how the structure works. A tool designed and implemented for helping to establish the system evolution directory is introduced. Related works is introduced in section 3. Section 4 concludes and gives direction for our future work.

2 The System Evolution Directory

A system evolution directory is made up of directory models tightly connected together to form a tree structure.

We use a case study to demonstrate the system evolution directory. A concrete case study is more easily accepted than an abstract theory and figures can simplify a complicated issue compared with literal description. Since the directory is based on models made up of figures, we use some pictures to show our directory.

In this section, a case study - Citation Finder - is first introduced. The case study is then used to demonstrate a system evolution directory: how to establish it and how it can help developers.

2.1 Case Study - Citation Finder

Anyone who is writing an academic paper has to refer to other academics’ papers; however, managing such citations is a complex and boring job. We can discover citations of a paper from the Internet, but it is time-consuming and unproductive. It would be a great relief for authors if there were software to complete this task. The case study was first introduced by Loke et al.[18]. We re-designed it with current agent-oriented methodologies, implemented it with the language Java, and evolved it with a system evolution directory. We name the system Citation Finder.

The aim of Citation Finder is to design and implement a multi-agent system to fulfill the following tasks: Firstly, the system should be able to accept users’ input: the title of a journal article and author’s name. Secondly, it should search the Internet according to the users’ input. Thirdly, the system should be able to identify at least two kinds of citation format: Latex format and normal format; the system also should be able to show the results of a search, and lastly, it can provide some functionality, such as screen clearing.

The system works in the following way: When the system receives a search request, it requests search engines, such as Google, to provide web pages which include the keywords inputted by the users. Once the system receives these web pages, it analyses them to check whether one of them includes citations that match the users’ input. Finally, a result about the search will be provided to its users.

Figure 1 is a snapshot of the system we implemented. From the next section, we will enter our evolution trip; the entire process of evolution will be introduced step by step.

2.2 Evolution

At the beginning of the process of multi-agent system evolution, we first have to answer the questions: what does evolution mean? What benefits can we get from evolution and where to start?
Since there is no unanimously agreed definition of evolution, we argue that evolution is a process rather than a phase. During this process, changes happen gradually and systems undergoing the changes are improved towards an advanced and mature stage. Evolution happens in all phases of the software life cycle, from requirements elicitation, analysis, design, to implementation.

Evolution of software is different from evolution in the natural world in that we can supervise and control the process. The first step to supervise and control software evolution is to record the evolution.

What are the benefits of using the evolution concept to describe software change instead of the concept of maintenance? We believe that the concept of evolution reflects the essence of software change and pushes us to think how to improve this process.

Evolution begins from requirements elicitation.

2.3 Form a Goal

A software system is designed to solve a problem or get something done. In order to do this, we first have to set up a goal.

A goal is something to be achieved, often vaguely stated [8]. In goal-oriented requirements engineering (GORE), it can be considered as a function or a quality to be achieved. A goal not only has attributes, but also has a life cycle.

The attributes of a goal include name and description. The name attribute is a short definition of a goal and it can be used to identify a goal uniquely. The description attribute is a detailed description of a goal and it is used to explain why the goal is proposed.

A goal also has its life cycle. Similar to the life cycle of a system, a goal’s life cycle can be divided into three phases: the elicitation, analysis and design phase, the implementation phase and the evolution phase.

We name goals originated by customers as customer-oriented goals. In our case study, the goal "Find Citation" is a customer-oriented goal. It is obtained by analysts from customers through some elicitation techniques such as interview and observation.

Once we get goals from the customers, we then begin our analysis of them so that we can fully understand them. Each goal is represented by a goal model and a corresponding directory model. The directory model is used to manage the life cycle of a goal and the goal model is used to record and analyse a goal.

In the following two sections, we will introduce how the “Find Citation” goal is managed by a directory model and how it is analysed by a goal model.

2.4 Directory Model

What is a directory model? A directory model works as an information aggregator and an interface and has three purposes. First of all, it aggregates information of a goal, including its attributes, its life cycle, its sub-directory models and its supporting information into one model. By us-
Figure 6. Find Citation Goal Evolution Diagram

Figure 7. Find Citation Goal Model Version 1

Figure 8. Find Citation Source Code Block View Diagram
ing this model, customers and developers can get a holistic view of a goal. Secondly, a directory model also works as an interface to access other models. Thirdly, it reflects the relationships between models and artefacts.

Figure 2 is the directory model of the “Find Citation” goal. From the figure we can see that a directory model has a tree structure, which includes various nodes.

Nodes can be divided into three types. The first type is the root node, the second type is the classification node, and the third type is the instantiation node.

The “Find Citation” node is a root node, which is the root of the tree. Each directory model has only one root node. It is used to store the attributes of a goal.

The second type of node is a classification node; it is used to classify nodes into different purposes. It can be divided into five categories. The children nodes of a classification node are instantiation nodes. They are instantiations of classification nodes.

The first category is the “Analysis and Design” node which has a child node named as “Find Citation Goal Mode”. This category gathers models associated with goal analysis and design together and reflects part of the life cycle of a goal. We will introduce what is a goal model and how to use a goal model to analyse a goal in the next section.

The second category is the “Implementation” node which has a child node named as “Source Code Block View”. This category gathers models and artefacts which are associated with goal implementation together and reflects part of the life cycle of a goal. We will introduce this model in the following sections.

The third category is the “Evolution” node which has a child node named as “Evolution Diagram”. This category gathers models and artefacts which are associated with goal evolution together and reflects part of the life cycle of a goal. We will introduce it in the following sections.

The fourth category is the “Directory” node which has three children nodes. They are “Search Citation”, “Setup User Interface” and “Set Up Internet Parameters”. This category gathers models which represents “Find Citation” goal’s child goals together. The “Directory” node represents the current goal’s sub-goals’ directory models. For example, figure 4, shows the “Setup User Interface” goal’s directory model.

The fifth category is the “Support” node; it has one sub-category node named as “File Directory”. This kind of node gathers models and artefacts which reflect resources used by the goal. For example, in order to implement the “Find Citation” goal, many files are produced; the “File Directory” model reflects the file system directory of the goal. Figure 9 shows the file system directory of the “Find Citation” goal.

2.5 Analyse and Design a Goal - Goal Model

Now that we have a goal “Find citation” from a customer, we will use a goal model to record and analyse this goal.

What is a Goal Model? A Goal Mode is a model used
to analyse and record the relationships between goals and its sub-goals. It is also used to distribute responsibility to a role. A role represents who will take the responsibility to implement this goal. Our goal model is derived from the ROADMAP methodology [15].

Figure 3 shows the goal model diagram of the “Find citation” goal. In the goal model diagram, parallelograms represent goals and stick figures represent roles.

A goal might be made up of its sub-goals and actions. The process of goal analysis is a process to elaborate a goal into sub-goals and actions. The rule of elaboration is using sub-goals and actions to define their parent goal. If sub-goals and actions of a goal can clearly answer the question how to implement the goal, we think this goal model is complete and the process of analysis is completed.

The “Find Citation” goal can be elaborated into two sub-goals: “Setup User Interface” sub-goal and “Handle User Request” sub-goal. The “Handle User Request” goal can be elaborate into its four sub-goals. The model diagram shows us that in order to implement the “Find Citation” goal, first we have to set up a user’s interface for users to put the title and authors of a paper; we then have to handle users various requests such as clearing window, searching, inputting, etc.

The process of goal analysis also includes distribution of responsibilities to role. This distribution is the basis of implementation. For example, through analysis, we decided that “Find Citation” should be implemented by the “Requester” role.

Some goals are so complicated that we cannot analyse them clearly with only one goal model diagram, under such circumstance; we can use another goal model diagram to analyse its sub-goals instead. Of course, this goal model has a corresponding directory model. For example, we can use another goal model diagram to analyse the “Setup User Interface” sub-goal.

Figure 5 is the “Setup User Interface” goal model diagram. It is made up of five sub-goals: “Setup Status Bar”; “Setup Toolbar”; “Listen to Enter Key”; “Setup Text Area for Results”; “Setup Layout”. Figure 4 is the “Setup User Interface” goal directory model diagram.

Different goal model diagrams represent different level of abstractions. The “Find Citation” goal model diagram has a higher abstraction level than the “Setup User Interface” goal model diagram. It also means that we can call the “Setup User Interface” as an analyst-oriented goal, because it originated from analysists. Designers will consider the best solution for it.

The “Setup Status Bar” goal can be called as a designer-oriented goal, because it originated from the designer. Programmers will use computer languages to implement the goal.

### 2.6 Implement a goal

Once programmers get design-oriented goals from designers, they will transform these design-oriented goals into source code which can be represented by a source code block view diagram. In order to share information with other kinds of developers, programmers have to connect the source code block view diagram into a directory model. From figure 2, we can see that there is a child node under “Implementation” classification node. This child node is named as “Source Code Block View”. From this child node we can get access to the real source code block view diagram.

Figure 8 is the source code block view diagram. Some nodes in this diagram can be mapped to its corresponding nodes in goal model diagrams. For example, the “setUpUserInterface” can be mapped to “Setup User Interface” sub-goal in figure 3. That means we can get source code’s semantic meaning through goal model. By using models in system evolution directory, we can make a system understandable. This will improve the productivity of system evolution.

In the next section, we will introduce how to evolve and record the “Find Citation” goal.

### 2.7 Evolve a Goal

Figure 6 is an evolution diagram of the “Find Citation” goal. It includes a root node named as “Find Citation - Evolution”. This root node has a child node named as “Goal Model Version 1”. The “Goal Model Version 1” is an interface. Through it, we can access the real model which is represented by figure 7.

The evolution diagram is used to record the evolution history of a goal. For example: figure 7 recorded the old version of the goal model of the “Find Citation” goal. These kinds of record give developers an opportunity to understand the evolution process of a goal.

### 2.8 Support a Goal

Supporting information is indispensable for the evolution of a goal. For example: in order to get access to source codes, we have to know information about the file system directory; in order to store data, we have to know information about database system; in order to understand a goal we have to get access to documents.

Through directory model, developers can connect any supporting information into this model. This will save developers an immense deal of time for searching stuffs. Figure 9 is a diagram which depicts the file system directory used by the “Find Citation” goal. From this diagram we can get a clear image about the structure of the file directory and what kinds of files are used.
Table 1. The SED Views and Models.

<table>
<thead>
<tr>
<th>Major Area</th>
<th>View</th>
<th>Model</th>
<th>Main Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis and Design</td>
<td>Goal View</td>
<td>Goal Model</td>
<td>Goal, Sub-goal, Role, Action</td>
</tr>
<tr>
<td>Implementation</td>
<td>Component View</td>
<td>Directory Model</td>
<td>Goal Life Cycle, Block</td>
</tr>
<tr>
<td>Evolution</td>
<td>Evolution View</td>
<td>Evolution Model</td>
<td>Evolution</td>
</tr>
<tr>
<td>Support</td>
<td>Support View</td>
<td>File Directory Model</td>
<td>Resource</td>
</tr>
</tbody>
</table>

2.9 Views on Goal Tree

From a system evolution directory, we can get different views. The reasons for providing different views lie in the fact that different developers have different needs for information. For example: an analyst only cares about information comes from customers, while a programmer only cares about information comes from designers. Table 1 shows views that can be obtained from the SED.

The Goal view reflects the semantic meanings and relationships of system artefacts. It provides a high-level view to the users of the directory.

Component view reflects the logical division of the system and it made up of components of a released system. Through this view we can know what really makes up a system.

Evolution view reflects the process of system evolution. It is a history of system development and maintenance. It can help a maintenance team to understand differences between versions.

Supporting view reflects other aspects of a system which comprises other designs (database design, data structure design, file system design, requirements documents, design documents, user manual, etc.) and documents of a system.

2.10 A tool for building SED

Although almost all agent-oriented methodologies provide their tools. However these tools are mainly used for modelling and they did not provide the support needed for the effective management of the relationships among system artefacts.

To compensate for this insufficiency, we designed and implemented a tool to help our research work. We named this tool as System Evolution Directory Manager. It was implemented with Java language and it has realized the following functionalities: First, the tool can be used to draw various figures interactively which model the system evolution process. Second, the tool is extensible, which means we can add new functionalities according to our needs. Third, not only developers but also software agents can make use of figures created by the tool. Fourth, it can automatically extract information from the models and use it to establish, update or remove relationships among models.

3 Related Work

At present, research on software change focuses on source code level, such as reverse engineering and refactoring.

“Reverse engineering is the process of analysing a subject system to create representations of the system at a higher level of abstraction” [5]. Reverse Engineering attempts to revive a system with little documents or no documents and it is useful for reviving some legal systems. It tries to get new representations of an existed system according to low level information of the system, such as source code or user menus. It can be used in a circumstance where there is no high level abstraction of the system: due to the absence of high level information, the working rationale of the reverse engineering is guess; therefore, the reverse engineering is unable to find out comprehensive higher level of abstraction. I think that the reverse engineering can be used to decrease difficulty of source code maintenance or automate the maintenance of source code, but it does not solve the root cause of evolution.

Refactoring is defined as “the process of changing a software system in such a way that it does not alter the external behaviour of the code, yet improves its internal structure” [19]. Source code refactoring is a new technique for programmers. It provides strategies to help them optimize their source code and make it easier to understand. I think it is an important technique for system evolution and it can be used in the implementation phase of system development. But it does not address the issue how to link source code to system design phase. It does not always convey a clear image about the reason that led the developers to perform such changes. Its strategies are also subjective. For example, software refactoring demands we should refactor classes so that they are understandable, but it does not provide a clear way to refactor and it just can provide developers some examples.

Model driven development is another method to improve the changeability of software. It attempts to make software systems comprehensive by using various models from the
high level of requirement elicitation to the low level system implementation. Models are used to extract the essence of an issue without considering the details. In my research, the principles of model driven development are to be used to help the evolution of multi-agent systems.

4 Conclusion and Future Work

We proposed a structure to facilitate the evolution of agent-oriented systems. Directory model and goal model are the core of the structure and are connected together to form a tree which is used to record the process of system evolution. All information supporting the implementation of a goal is gathered together under the goal. Maintenance team can use the tree to understand the underlying mechanism of a system development. Meanwhile, the tree also provides a way to hide unnecessary information. We can obtain different views from this tree including goal view, component view and supporting view. These views satisfy different requirements of different developers. For example: based on the directory, programmers can get source code they need, and analysts can get goals they need.

The proposed structure can be used for the development and evolution of agent-oriented systems. We are currently applying it in three case studies with agent-oriented system development. Although this structure is designed for the purpose of evolution of multi-agent systems, we believe it can also be successfully used in the evolution of object-oriented systems.

We have at least three future research directions ahead. The first one is to verify the system evolution structure through more case studies. The second is to think about the issue of distributing the directory. At present, the directory can only be used in one computer by one user. It will be more useful if it is distributed and can be used by multiple developers from different locations. The third research direction is to think about how to identify and locate information from the directory easily and rapidly. A real project usually includes a huge amount of information, if such information is stored in a system evolution directory, how to identify and locate this information becomes a critical issue for the success of the system evolution directory.

References


