An Aspect-Oriented UML Tool for Software Development with Early Aspects

Yang Wang, Santokh Singh, John Hosking
Department of Computer Science
University of Auckland
Private Bag 92019, Auckland
New Zealand
+64-9-3737-599
{santokh, john}@cs.auckland.ac.nz

John Grundy
Department of Electrical and Computer Engineering
and Department of Computer Science
University of Auckland
Private Bag 92019, Auckland, New Zealand
+64-9-3737-599
john-g@cs.auckland.ac.nz

ABSTRACT

Aspect-oriented Component Engineering uses early aspects to better categorize and reason about provided and required services of individual components in software systems. Our earlier work on AOCE demonstrated an increase in the reusability and understandability of software components and systems via its usage of early requirements and design-phase aspects but lacked adequate tool support. We describe a novel design tool called Aspect-Oriented UML (AO-UML) that can be used to efficiently capture and manage early aspects for software development using the Aspect-oriented Component Engineering methodology. The key benefits of our tool are its use throughout the development lifecycle and its ability to support and take advantage of Aspect-Oriented Component Engineering's features and capabilities.

Categories and Subject Descriptors

D.2.2 **[Software Engineering]**: Requirements/Specifications – *tools.* Design Tools and Techniques – *evolutionary prototyping, modules and interfaces, object-oriented design methods.*

General Terms

Design, Documentation, Management, Reliability, Verification.

Keywords

Early Aspects, Meta-Modeling Tool, AOCE, AO-UML.

1. INTRODUCTION

As software systems have became more sophisticated and complex, traditional software development methodologies which had focused on building software systems from scratch have been replaced by a "build-systems-from-parts" approach [5], [14], [16]. These newer methodologies, collectively called component-based software development (CBSD), have significantly changed how software systems are built by focusing on selection of software components off-the-shelf (COTS) and assembly of those components within an appropriate software architecture [15]. Various software architectures and implementation frameworks have been developed based on the notion of software components, including COM [17], JavaBeans [18] and JViews [12]. In contrast to traditional software systems, component based systems offer potential for better existing or third party component reuse, compositional system development, and dynamic and end user reconfiguration of the applications.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EA'06, May 21, 2006, Shanghai, China.

Copyright 2006 ACM 1-59593-085-X/06/0005...$5.00.

However, CBSD methodologies present their own set of problems, including the common issue of cross-cutting of concerns and the interleaving or "tangling" of common code in software systems. Cross-cutting concerns give rise to designs and implementations that are complicated, difficult to understand and hard to control. In some respects this problem is even more challenging in CBSD than in traditional monolithic software construction as 3rd party components may be assembled, even dynamically, with poor descriptions of the component's requirements and design decisions exposed for developers and other components to understand. We developed a novel CBSD approach called Aspect-Oriented Component Engineering (AOCE) to address the issue of representing cross-cutting concerns in component-based software systems [9], [19]. Our approach makes extensive use of both "early" and "late" aspects throughout the software development lifecycle, with aspect-based characterizations of components used in requirements engineering, software architecture and component design, component implementation and at run-time to support dynamic component discovery, integration, and deployed component testing [9], [11], [13]. However, as AOCE adds extra complexity to component requirements, designs and implementations, using AOCE without adequate tool support is very difficult [11].

In this paper we present our recent work developing integrated tool support for AOCE so that designing and implementing software systems based on early aspects can be performed in a more robust manner. We developed a novel notation and prototype tool set called Aspect-Oriented UML (AO-UML) that supports developers' use of the AOCE methodology for component-based software development throughout the component development lifecycle.

2. MOTIVATION

We have explored the possibility of applying AOCE to a variety of phases of software development in our previous work [9], [11], [13]. However, much of this research has focused on limited phases or areas of component software development, and lacked comprehensive tool support. More specifically, developers may not know how to apply these isolated techniques in a complete

software process, and some techniques such as dynamic discovery and deployment time component testing with aspects cannot be used without specialized tool support [11], [13]. We also found that without tool support we spent considerable time programming and coding aspect oriented applications that were very similar regarding their structure and contents. Therefore a tool to integrate our various AOCE techniques was urgently needed to provide an integrated development environment (IDE).

In this paper, the terms of Software Development Methodology and Software Process Model are based on Boehm's definitions [5] where he maintains a clear distinction between the two.

Even though our previous research has evaluated our AOCE techniques in realistic scenarios, the evaluations have only concerned a specific development phase independent of other phases and its impact on the whole development life cycle. Thus our AO-UML IDE project is aimed at investigating the capability of the AOCE techniques to support whole lifecycle aspect-oriented component engineering in an efficient manner. Furthermore, although we believe AOCE has strong potential for improvement of component-based software engineering, it is only an abstract methodology without proper tool support. For developers who wish to explore AOCE, our AO-UML tool provides a systemic and recipient way for them to learn about AOCE techniques and apply them in their own scenarios.

3. EARLY ASPECTS AND AOCE

Aspect oriented software development (AOSD) [1], [7] has become an important new approach to software engineering. AOSD addresses the problem of overlapping, horizontal cross-cutting concerns across multiple classes (and components) that exist in traditional software development. The fundamental idea here is to use aspects to represent concerns that cut modules, and implement aspects at a programming level separately from the modules. This enables aspects to be easily managed and controlled since they are isolated from the modules, and once defined, modules can be reconfigured by weaving aspect code in.

Up until the last few years, most work using aspects has been limited to the implementation phase of software development, i.e. finding cross-cutting concerns that implementation units have in common and factoring those out as aspects. Many current applications of aspects, such as in Aspect-Oriented Programming (AOP) [19],[22], mainly concentrate on the implementation phase of the life cycle. These aspects are actually code blocks that can specify different concerns in modules. However, much recent work has tried to generalize the concept and apply it to different phases of the life cycle. A new direction of AOSD is to identify and categorize aspects, called early aspects, in early phases of the life cycle, and convert them into programming level aspects for modules in the implementation phase. Identifying aspects at an early stage helps to achieve separation of crosscutting concerns in the initial system analysis instead of deferring such decisions to later stages of design and code, hence avoiding costly redesigning and refactorings. Several approaches have been introduced to try to assist in the identification of early aspects for AOSD, these includes Theme [4], Early-AIM [22], and EA-Miner [20].

However, most such techniques are still in initial stages of research. Our whole of lifecycle AOCE methodology [9], [11] is one such technique. AOCE uses a concept of different cross cutting systemic capabilities (clearly defined early aspects such as persistency, security, user interface, transaction, configuration, collaboration, resource utilisation etc.) which are identified in the early phases of the System's Development Life Cycle (SDLC). These aspects are used to categorize and reason about provided and required services of individual candidate components in a software system, whether these may be existing or to-be-developed components. AOCE supports the identification, description and reasoning about the component's high-level functional and non-functional requirements grouped by different early aspects, with "aspect details" and "detail properties" providing an ontology and constraint language to express constraints on the provided and required relationship between components and component compositions. AOCE component requirements are refined into design-level software component services implementing these aspects but which are also characterized by more detailed design and implementation-level aspects, tracable back to the requirements-level aspects. Components are implemented using aspect characterizations to support dynamic component description, discovery, adaptation, reconfiguration and deployment-time testing [11]. Using AOCE, components can be automatically indexed by their early aspects, and users can formulate high-level queries about the component's capabilities. Moreover, some pre-defined properties of early aspects in AOCE provide validation functions for sensible configurations of any retrieved components [11].

In the past we have redesigned and redeveloped some complex traditionally implemented component-based software systems using AOCE for proof of concept purposes [12] using manual application of AOCE techniques (i.e. without tool support). These results were very encouraging and show that re-engineering using AOCE can produce significantly better characterized component requirements and more easily reused and reconfigured components. The early aspect information provides advantages during component requirements analysis and design, such as rich multiple perspectives for components, better structuring of components and design, better dynamic configuration and de-coupled component interaction, and run-time access to detailed component knowledge [12], [20]. Having proven the concept our aim now is to develop usable, comprehensive and novel tools to support the whole AOCE software development process. The AO-UML tool presented here is the first of these and is novel in its comprehensive support for aspects across the SLDC.

4. AO-UML

Our prototype AO-UML tool has the capability to support several perspectives for software development using AOCE. These include a component-based system's functional requirements, architectural designs and detailed component specifications and characteristics. An exemplar application developed using AO-UML is used to illustrate the tool's capabilities. We implemented AO-UML with the Pounamu meta-CASE tool which was used to specify and generate the AO-UML IDE.

The Pounamu [27] meta-CASE tool supports specification and generation of multiple view visual tools. The tool permits rapid specification of visual notational elements, underlying tool information model requirements, visual editors, the relationship between notational and model elements, and behavioral components. Pounamu can generate visual modeling tools automatically and the tools used for modeling immediately.

4.1 FUNCTIONAL REQUIREMENTS

The SDLC literature introduces and defines various phase models [1], [3], [9], [15]. These models, however, while using different terminology, have very similar phases in terms of the activities defined in the phases. We have identified six core phases that define an SDLC based on the descriptions in these papers; these six phases are listed in Table 1 together with the overall requirements for those phases. Table 1 also lists the functional requirements of AO-UML for each of the SDLC phases. For example, in the Software Requirements Analysis phase shown, we list out the services that the system should provide as aggregate early aspects and this identification of aspects is done very early in the life cycle.

Phase	Requirements	Functions provided by AO-UML
Planning	System engineering and modeling, which involves in setting up the required resources such as hardware, people and software.	Initializing the project in AO-UML and the AOCE component database*.
Software Requirement Analysis	Analyzing the requirements of the software system.	Listing out services that the system should provide as aggregate early aspects.
System Analysis and Design	Defining overall software structure.	Invoking components that might be reused from the AOCE component database.
		Defining components and their aspects information.
		Aspects mapping across components.
		Deploying components to UML design.
Implementation	Translating the design into a platform-readable format such as Java, C# or PHP.	Generating code for various platforms. (Current version only supports Java)
Testing	After the code is generated, developers should start testing the code in a planned manner.	Dynamic Validation Agents to support deployment of component testing.[11]
Maintenance	Software will definitely be updated for some reason, e.g. for bug fixing or functional enhancement after it is delivered to customer.	Storing and querying components from AOCE database through AO-XML standard. [13]
		Reverse engineering with aspect information.

*AOCE component database: A component database which supports the storage and query of aspect information of the component.

Table 1: Functional Requirements of AO-UML

Even though we used the system development life cycle (SDLC) for the software process model (SPM) to introduce the functions of AO-UML, these functions can in fact be applied to any SPM to support development using Early Aspects. One of these models is our eXtreme AOCE [22] which extends the features of both AOCE and eXtreme Programming (XP) to support cross-cutting concerns of the components with Agile techniques. Instead of developing the software system as a whole in SDLC, eXtreme AOCE tends to slice the system into small pieces containing aspect information, and work on developing them separately. Through this method, the development team can handle any change of requirements more rapid and easily, at the same time achieve a high level of productivity. Also, eXtreme AOCE drives the team to produce high quality reusable aspect-oriented software components based on AOCE concepts. For employing eXtreme AOCE as the SPM in AO-UML, each small piece of new program that needs to be developed is treated as an independent mini-project that is then merged with the remainder of the program once it has been tested. Therefore, developers can do the requirements analysis, aspect identification and component design for each small piece as if they are working on independent projects without disrupting the rest of the system that is already developed. As such, though the AO-UML is specially built for software development using AOCE, it is versatile and can also be used to fully support other AOCE related techniques like eXtreme AOCE because of the tool's ability to enable the design and construction of high quality aspect-oriented components and address issues concerning Early Aspects during the development of the software.

4.2 Architecture of AO-UML

The architecture of AO-UML, shown in Figure 1, consists of three main parts, the Graphical User Interface (GUI), a repository and an implementation factory. The GUI of AO-UML is composed of three view types, which can be switched between: the component, aspect and UML views. Each view type has its own visual symbols and semantics for describing the design in multiple perspectives. The aspect view manages information concerning component aspects and their mapping (the mapping is presented as aggregate aspects).

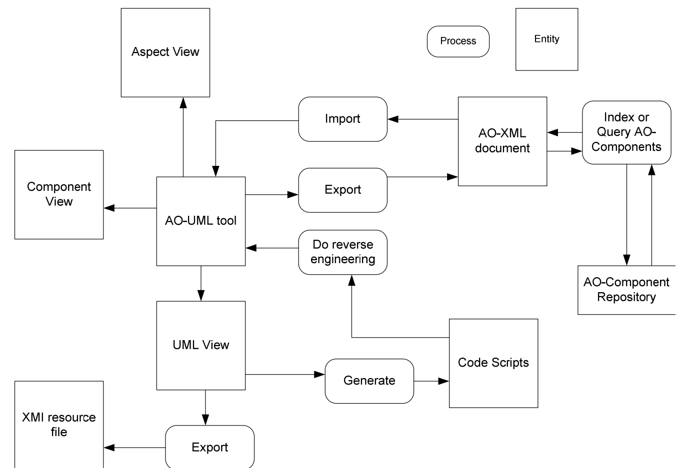

Figure 1: The overall architectural design of AO-UML

The component view is used to gather information about components including their child classes, aspects and the mapping between methods and aspects. The UML view co-operates with the component view. It allows users to import the classes defined in the component views and use additional notations provided by the UML view to complete the UML class diagram for implementation. Even though these three view types have their

own concepts and definition, some of their visual symbols such as classes and aspects share the same meaning and properties. The AO-UML thus provides a way of sharing these kinds of concerns and symbols across different view types and to synchronize the entities relating to these.

An aspect-oriented component repository is used by AO-UML users to store and share information concerning all AO-components developed so far. This supports fast search and retrieval of these components using aspect-enhanced queries. AO-UML can also be used to visualize the components as notations on its views and allow users to reuse them in their designs and analysis. In order to communicate with the repository, a schema is required for both sides to interpret and understand the conversation. AO-XML [23] is a novel schema which we developed to define the grammar for exchanging aspectual information and stores information related to the components and their aspects in a well-structured way.

The final part of the architectural design is an implementation factory. This provides tool support for the implementation phase such as transferring the design into a platform-readable form, implementing reverse engineering and exporting XMI files. Based on the UML design diagrams drawn, AO-UML can be used to generate code scripts for various platforms to minimize the developer's effort in writing code. The current AO-UML version generates Java code. AO-UML can also support reverse engineering to allow recovery of a design with aspect information from code scripts.

4.3 AO-UML NOTATION

Figure 3 describes the main visual elements of the component view type. These include the component, class, aspect, connector and event flow symbols. In a software system developed using AOCE, all system components are aspect oriented by categorizing their operations with early aspects.

Visual Symbol	Explanation
ComponentName	A software component is a reusable piece of software which has some certain functions, and can be integrated with other components.
ClassName Class attributes methods	A class is a reference type that encapsulates data and defines its behaviors using methods, properties, events etc. A class is one of the elements of a component.
AspectName aspect details parameters	Aspects are system capabilities that can be cross-cut between components to identify, describe and reason about the system's high level functional and non-functional requirements.
⟶	A connector is used to connect related entities to expose their relations e.g. the component this class belongs to.
⟶	The event flow describes what a component does to another component in a particular case. It is used in the component use case diagram at the early stage to reason about the components at a very abstract level.

Figure 3: Sample of modeling capabilities in component views.

An AO-component may consist of several classes which are also aspect oriented, meaning that each and every method in a class of this type of component is also categorized as belonging to aspects that either provide services to other methods or require services from other methods across components. These services are defined as aspect details which are categorized into particular aspect types [9] (e.g. Persistency, User Interface type of aspects). An AO-component can be composed of one or more aspect types, and in our AO-UML tool, each aspect type can be depicted as having several "provided" (prefixed "+") or "required" (prefixed "-") aspect details. The purpose of the component view is to allow users to design the components and their classes with aspect information.

Figure 4 shows the main visual elements of the aspect view and their explanations. Unlike the component view which concentrates on individual components and their constructs, the aspect view focuses on components' aspects and the system's aggregate aspects. This view type shows the details of all kinds of aspects and the aspect mapping information across components and is vital for understanding the aspects' effects and impact across the components and system.

Visual Symbol	Explanation
AggregateAspectName aspect details	An aggregate aspect is a set of services provided by the software system, these service are from several provided or required aspects of the same aspect type of different components
AspectName aspect details parameters	Aspects in this view are imported from the component view, so they correspond to the same element but shown on different representation layers. Aspects in this view are automatically synchronized with the aspects in the component view.
⟶	A connector is used to connect related entities to expose their relations such as what aspects this aggregate aspect consists of.

Figure 4: Sample of modeling capabilities in aspect views.

The visual elements used in the UML view type are shown in Figure 5. The UML view is used to show details of how the classes are constructed and related to each other, and hence has UML specific symbols. The current prototype implements all the essential notational elements of UML class diagrams.

4.4 AN EXEMPLAR APPLICATION

This section provides an example to demonstrate the usage of the AO-UML through the whole SDLC. The scenario is a simple online banking system called "Simple Bank". To develop this software system we shall assume the following requirements:

- The system must be developed using Java (Java Servlets for the website).
- The bank's customers are able to login to the system through the website.
- Customers can view their account balances and deposit or withdraw money through the website.
- All accounts information is stored in a database.

In the Planning phase, developers receive the project from their customers and after the resources and working environments in the AO-UML are set up, developers start gathering requirements. This proceeds into the analysis of the requirements phase, where requirements are identified as the functional services of the system, and are stated as aggregate aspects in AOCE.

Visual Symbol	Explanation
name *interface* attributes methods	An interface in UML is represented by a rectangle with a sign of "interface".
ClassName *Class* attributes methods	A class in UML is represented by a rectangle with a sign of "class; this entity can be imported from the component view.
———◇	An aggregation represents "whole/part" or "has-a" relationships between classes.
———	An association in UML indicates the relationship between an object of one type with an object of another type.
———▷	A generalization represents a relationship in which one class is of a more specialized version of the other.
- - - - - - ▷	A dependency is a relationship in which an object of one type must rely on an object of another type.

Figure 5: Basic modeling capabilities of the UML view type

Figure 6 shows the aggregate aspects and their aspect details as listed in AO-UML's aspect view. In this figure, the tool also lists out the aspect details (services) of the system. In the following phase, developers will commence analysing the gathered requirements to design the components. To design and structure the components, AO-UML provides various icons and functions in its component view to help developers design the system step by step.

Figure 6: Aggregate aspects of the "Simple Bank" example.

A component use case diagram for the "Simple Bank" system is shown in Figure 7. These types of diagrams are very useful for developers to think and reason about all the possible components required and assign tasks to them at an early stage of the development process to increase efficiency.

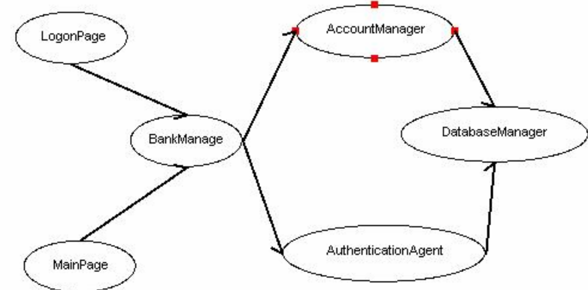

Figure 7: A component use case diagram for SimpleBank.

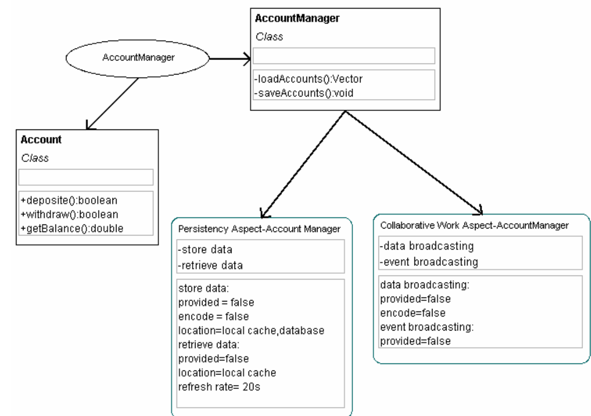

Figure 8: The component diagram of AccountManager.

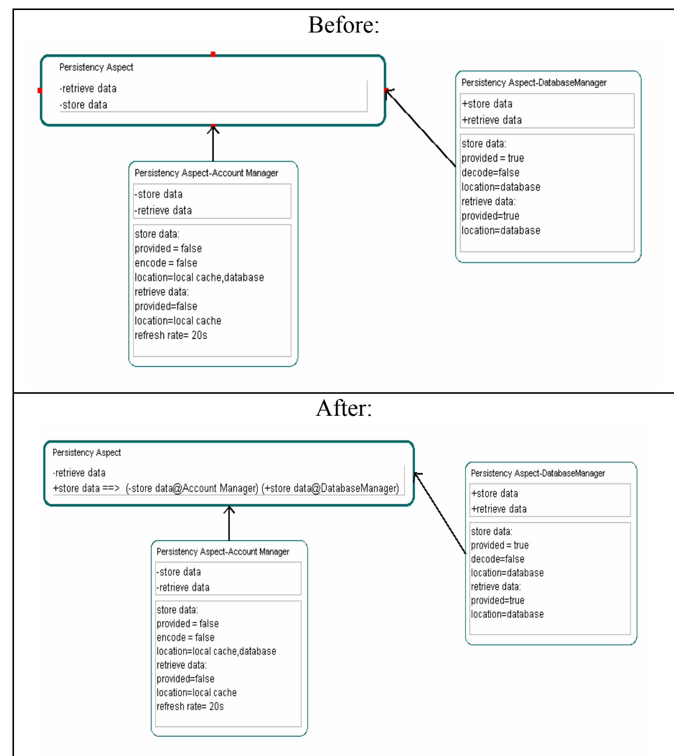

Figure 9: An aspect diagram before and after being mapped.

The components can then be specifically designed and structured one by one in AO-UML's component view. Figure 8 shows the constructs of the AccountManager component from our exemplar application. This component consists of two classes, Account and AccountManager. The aspects and aspect details of the classes are also depicted. For example the AccountManager class has two types of aspects,. Collaborative Work and Persistency. Each of these aspect types are further shown to contain several aspect details, including "provided +" or "required -" (i.e. this aspect provides or requires this service) attributes.

After the components' design is completed, their aspects can be imported into the aspect view where the required and provided aspect details can be matched together. The first row in the table illustrated in Figure 9 shows the aspect diagram for the persistency aspects from the various different components of our exemplar system. Aspects can be mapped using AO-UML's "Aspect Mapper" to match particular aggregate aspect details. Through this approach, we can obtain all the aggregate aspect details mapped with their corresponding provided and required aspects. The lower diagram in figure 9 shows the persistency aspect-diagram that was produced after mapping the component's aspects. In this, we see that the "store data" service required by the Account Manager persistency aspect is provided by the DatabaseManager Persistency aspect "store data" service. A further mapping could be used to match "retrieve data" services.

A snippet of the AO-XML document generated from the sample project is shown in Figure 10. To export the information of the components and aspects that has been designed, users can use AO-UML's export function to generate the AO-XML [14] shown. This document is very useful and can be read by the AO-UML tool to regenerate the design diagrams in the tool.

```
<?xml version="1.0" encoding="utf-8"?>
<aoxml:application language="Java" name="SimpleBank" xmlns:aoxml="http://www.cs.auckland.ac.nz/">
  <aoxml:components>
    <aoxml:documentation information=""/>
    <aoxml:description description=""/>
    <aoxml:component description="" name="AuthenticationAgent">
      …………
      <aoxml:object description="" name="AccountManager" type="Class">
        <aoxml:aspects>
          <aoxml:aspect description="" name="Collaborative Work Aspect">
            <aoxml:detail name="data broadcasting" provided="false"/>
            <aoxml:detail name="event broadcasting" provided="false"/>
          </aoxml:aspect>
          <aoxml:aspect description="" name="Persistency Aspect">
            <aoxml:detail name="store data" provided="false"/>
            <aoxml:detail name="retrieve data" provided="false"/>
          </aoxml:aspect>
        </aoxml:aspects>
      </aoxml:object>
      …………………..
    </aoxml:component>
    <aoxml:component description="" name="DBAgent">
      <aoxml:object description="" name="Database" type="Class">
        <aoxml:aspects>
          <aoxml:aspect description="" name="Persistency Aspect">
            <aoxml:detail name="store data" provided="true"/>
            <aoxml:detail name="retrieve data" provided="true"/>
          </aoxml:aspect>
        </aoxml:aspects>
      </aoxml:object>
    </aoxml:component>
    …………………
  </aoxml:components>
</aoxml:application>
```

Figure 10: Example of the AO-XML "Simple Bank" project.

AO-UML's UML view provides icons and connectors for developers to draw UML class diagrams with pre-populated classes from the component view. An example of the UML class diagram of the "Simple Bank" application is shown in Figure 11. During this phase, developers are working on the static/structural design of the software. Classes in this view need to be completed with all the necessary parameters and connected to each other using UML connectors. Java skeleton code is generated by AO-UML after all the aspect-oriented UML class diagrams are completed. Full business logic etc. can be inserted into the skeleton code based on the specifications of the system being built. All code and component-interfaces are generated in objects constructed within their respective components with their namespaces clearly defined to aid developers.

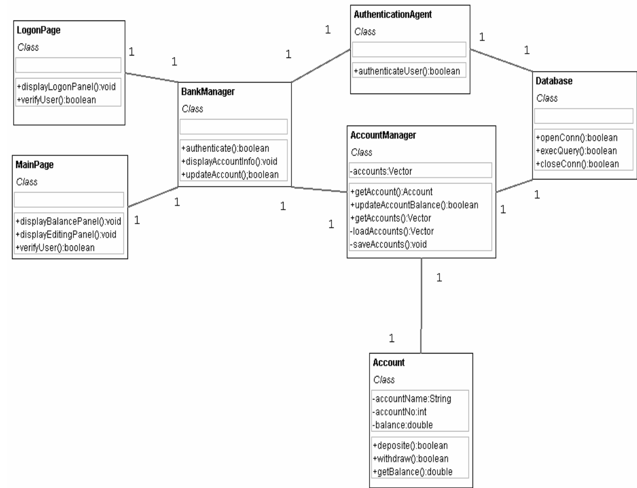

Figure 11: A UML class diagram for "Simple Bank"

```
<?xml version="1.0" encoding="utf-8"?>
<XMI xmi.version="1.0">
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>Novosoft UML Library</XMI.exporter>
      <XMI.exporterVersion>0.4.20</XMI.exporterVersion>
…………
        <Foundation.Core.Class xmi.id="Class$AccountManager">
          <Foundation.Core.ModelElement.name>AccountManager
                </Foundation.Core.ModelElement.name>
          <Foundation.Core.ModelElement.visibility xmi.value="public"/>
          <Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
          <Foundation.Core.GeneralizableElement.isRoot xmi.value="false"/>
………….
```

Figure 12: Portion of XMI document from the "SimpleBank".

In addition to system code generation, the AO-UML can also export this UML information in the design diagrams as an XMI document, an example is shown in Figure 12. It is part of the AccountManager object from our example of the SimpleBank project with its major elements included in the figure. The tool can also be used to store, inspect and retrieve all XMI and AO-XML documents, designs, generated code and aspect-oriented components from its repository, as in [13]. Other software engineers may also use the AO-UML to locate these objects from the repository and may reuse any or all of them.

5. DISCUSSION

Developing the AO-UML tool and using it for software development based on AOCE techniques has given us valuable hands-on experience on the significance and application of AOCE and early aspects in real world scenarios. We have shown a sample application (an extension of "SimpleBank") developed using AO-UML. While using AO-UML we have noted some features of the prototype tool which can be improved upon. Currently there is a lack of navigation support for the individual elements across views, e.g. for navigation involving aspects and classes. Users of the tool currently have to manually search

through the diagrams to find related components. A search and indexing mechanism would mitigate the hidden dependencies that otherwise result. Formatting of detailed information as text inside the visual icon of the corresponding component has room for improvement. For example, in figure 9 above, the string "+retrieve data ==> (-retrieve data@AccountManager)(+retrieve data@Database) () etc." could be better rendered. In addition, capability to elide some of this detail is needed. Though AOUML provides quite comprehensive functions to support AOCE development phases it still needs to be improved to provide services and functions in the requirements phase. Currently users identify their own aspects and design components manually in the tool, and this can be quite laborious and time consuming. One solution is to provide a comprehensive library that can be used to look-up, identify, reuse or store aspects and aspect details. Another possible improvement is to provide specific visual UML icons and code generation for various target systems. More substantial future directions include the following:

- Designing and developing a more comprehensive AO-component repository, possibly on top of a traditional component database with aspect-based indexing.
- We are currently migrating Pounamu to the Eclipse IDE [9] in the form of a new meta tool (Marama). This opens up the possibility of migrating AO-UML into Eclipse as well. Integration as an Eclipse plug-in will potentially permit us to integrate with Eclipse UML tools (offering improved UML support), code generators, and code views.
- The current version of AO-UML lacks adequate tool support for the testing phase. We plan to re-develop validation agents that can automatically test the components with their non-functional constraints and properties [11].
- Reverse engineering is another area that we are very interested in. One challenge here is how to recover aspectual information from code scripts. Information of aspects and components can be retrieved from AO-UML's component repository, so one solution may be to use an indexing and query framework to retrieve this information.

6. SUMMARY

Aspect Oriented Component Engineering or AOCE is a methodology that uses early aspects to develop aspect-oriented software components. These components are the basic building blocks of our aspect-oriented software systems. We successfully designed and developed a novel tool called the Aspect-Oriented UML (AO-UML) that can be used to efficiently capture and manage early aspects for software development using AOCE. We also showed how the AO-UML can be used throughout a system's development life cycle based on capturing and using early aspects in aspect-oriented components.

7. REFERENCES

[1] Araujo, J., Baniassad, E., Clements, P., Moreira, A. and Tekinerdogan, B. Early Aspects: The Current Landscape, Feb 2005.

[2] Arthur, L. J. Software Evolution: *The Software Maintenance Challenge.* New York: Wiley & Sons (1988)

[3] Bader, J., Edwards, J., Harris-Hones, C., & Hannaford, D. Practical engineering of knowledge-based systems. *Information and Software Technology* (1988), 5, 266-277.

[4] Baniassad, E. and Clarke, S. Theme: An approach for aspect-oriented analysis and design. *Proc. of the 26th International Conference on Software Engineering*, IEEE 2004.

[5] Boehm, B. W. *A spiral model of software development and enhancement. Computer*, (1988) 21, 61-72.

[6] Bosch, F. v. e., Ellis, J. R., Freeman, P., Johnson, L., McClure, C.L., Robinson, D., Scacchi, W., Scheff, B., Staa, A.v., Tripp, L.L.. Evaluation of Software Development Life Cycle. Software Engineering Notes (1982) Vol 7 no1, Page 45-59, *ACM SIGSOFT*

[7] Chitchyan, R. and Rashid, A. Survey of Aspect-Oriented Analysis and Design Approach. *AOSD-Europe* (May 2005).

[8] Connors, D. T. Software Development Methodologies and Traditional and Modern Information Systems. Software Engineering Notes (1992) Vol 17 no2, Page 43, *ACM SIGSOFT*

[9] Eclipse.org, *Eclipse Modeling Framework (EMF)*, at URL: http://www.eclipse.org/emf/

[10] Grundy, J. Multi-perspective specification, design and implementation of software components using aspects. *International Journal of Software Engineering and Knowledge Engineering* (2000) vol.10, No. 6.

[11] Grundy, J.C., Ding, G., and Hosking, J.G. Deployed Software Component Testing using Dynamic Validation Agents, *Journal of Systems and Software*, vol. 74, no. 1, Jan 2005, Elsevier, pp. 5-14.

[12] Grundy, J.C. and Hosking, J.G. Engineering plug-in software components to support collaborative work, *Software – Practice and Experience,* vol. 32, Wiley, pp. 983-1013, 2002.

[13] Grundy, J.C. Storage and retrieval of Software Component using Aspects. *Proc. Of the 2000 Australasian Computer Science Conference, Canberra, Australia (Jan 30-Feb 3 2000.*

[14] Henderson-Sellers, B.S. and Edwards, J.M. Object-Oriented Systems Life Cycle. *CACM*, (1990) Vol 35, No9, Page 142-159

[15] Pour, G. Moving toward Component-Based Software Development Approach, *TOOLS 27*, Sept. 1998, pp. 296 – 300.

[16] Pour, G. Component-based software development approach: new opportunities and challenges. *TOOLS 26*, Aug. 1998, pp. 376 – 383

[17] Sessions, R, *COM and DCOM: Microsoft's vision for distributed objects*, Wiley, 1998

[18] O'Neil, J. and Schild, H. *Java Beans Programming from the Group up*, Osborne McGraw-Hill, 1998

[19] Panas, T., Andersson, J. and Assmann, U. The editing aspect of aspects. In I. Hussain, editor, *Software Engineering and Applications (SEA 2002)*, Cambridge, Nov 2002. ACTA.

[20] Sampaio, A., Chitchyan, R., Rashid, A. and Rayson, P. EA-Miner: a Tool for automating aspect-oriented requirements identification, *2005 Conf. Automated Software Engineering*, ACM, Nov 2005.

[21] Singh, S., Grudy, J.C. and Hosking, J.G. Developing .NET web service-based applications with aspect-oriented component engineering. *Proc. AWSA*, April 2004, Australia.

[22] Singh, S., Chen, H., C., Hunter, O., Grundy, J., C. and Hosking, G., J. Improving Agile Software Development using eXtreme AOCE and Aspect Oriented CVS, *APSEC 2005*, Taiwan, Dec 2005.

[23] Singh.S, *AO-XML Specification Version 1.0*, *PhD Thesis,* Department of Computer Science, 2006.

[24] Sampaio, A., Rashid, A. and Rayson, P. Early-AIM: An approach for identifying aspects in requirements, *RE'05,* IEEE.

[25] Suzuki, J. and Yamamoto, Y. "Extending UML with Aspects: Aspect Support in the Design Phase," ECOOP 1999 Workshop on Aspect-Oriented Programming, 1999.

[26] W3C, *Extensible Markup Language (XML)*, at URL: http://www.w3.org/TR/xslt

[27] Zhu, N, Grundy, J, Hosking, J, Pounamu: a meta-tool for multi-view visual language environment construction, in *Proc IEEE VL/HCC'04*, Rome, Italy, September 2004.