Assembling Components with Aspect-Oriented Modeling/Specification*

Fei Cao[1], Barrett R. Bryant[1], Rajeev R. Raje[2], Mikhail Auguston[3], Andrew M. Olson[2], Carol C. Burt[1]

[1]Department of Computer and Information Sciences
University of Alabama at Birmingham
{caof, bryant, cburt}@cis.uab.edu

[2]Department of Computer and Information Science
Indiana University Purdue University at Indianapolis
{rraje, aolson}@cs.iupui.edu

[3]Computer Science Department
Naval Postgraduate School
auguston@cs.nps.navy.mil

Abstract:
Component-Based Software Development (CBSD) offers a cost-effective means of software production with reduced time-to-market. Integration of heterogeneous components poses a non-trivial challenge in realizing this vision, which is further complicated in a distributed environment as a result of blurred functional and non-functional aspect[1] representation and management. We propose a two-level approach, i.e., to apply aspect-oriented component modeling/specification to handle the problem.

Keywords:
Aspect Orientation, Component Modeling/Specification, UniFrame, Weaving

1. Introduction

1.1 Background

Recent development in software component technology enables the production of complex software systems by assembling off-the-shelf components. This not only boosts productivity attributed to the reusability of components, but also improves cost-control and maintenance of software systems. Meanwhile, another hallmark of current software components is the heterogeneity in environment, language and application over distributed systems.

UniFrame [Raje01] is a framework for seamless interoperation of heterogeneous distributed software components. It is based on the Unified Meta-component Model (UMM) [Raje00] for describing components. A Generative Domain Model (GDM) [Czar00] is used to describe the properties of domain specific components and to elicit the rules for component assembly. Systems constructed by component composition should meet both functional and non-functional requirements such as the Quality of Service (QoS) [Brah02]. Towards the realization of the vision of the UniFrame project, an appropriate means for component modeling/specification is needed, which should be capable of:

* This research is supported by the U. S. Office of Naval Research under the award number N00014-01-1-0746.

[1] In this paper, "non-functional aspect", "non-functional-property" and "Quality of Service (QoS)" may be used interchangeably.

- representing the functional properties (including not only syntactic structure but also semantic behaviors) and requirements (pre/post condition, dependency, temporal constraints, etc.).
- representing the non-functional properties and requirements [Brah02].
- specifying the heterogeneity in terms of representing domain knowledge, e.g., technology domain, business domain, etc.

1.2 Current Issues

Assembly of heterogeneous distributed components will require glue/wrapper code to fuse them together. General practice leverages vendor-specific bridging products or applies hard coding, and both the functional and non-functional aspects of the assembled system tend to be blurred by this ad hoc treatment. We have applied Two-Level Grammar (TLG) as a formalism to specify various aspects of components [Brya02] based on UMM. Meanwhile, it has been brought to our attention such aspects of components as functional pre/post conditions and non-functional properties crosscut component modules and handling of these aspects spread across component modules. This poses some problems:

- reduced reusability of components. Component behavior may change in different contexts. The inter-relationship between components may also change under different business rules. The "Hard-coded" modeling/specification will be inadequate to capture the dynamics of components and component representations may have to be revised upon different environments
- blurred representation and management of functional and non-functional aspects of components. As those aspects are entangled with other aspects of components, reasoning for the integrated system based on those aspects will be hard to be carried out.

Aspect Orientation [Kicz97] provides a means to capture crosscutting aspects in a modular way with new language constructs. This makes us believe that augmenting our existent specification approach with aspect orientation can separate those crosscutting aspects intervening components, loosen the coupling between components, which will contribute to not only the reusability and evolution of component without changing the component itself, but also the manageability of component assembly. On the other hand, by using weaving technology, dynamic concerns can be "glued" into the composition of components. This paper will investigate the application of aspect orientation in the modeling/specification of components, in particular, the handling of their exported service and QoS of heterogeneous distributed components in the context of the UniFrame project.

This paper is organized as follows: Section 2 first gives an analysis of component assembly models. Section 3 presents our two-level, model-based, aspect-oriented approach for heterogeneous distributed component representation. Section 4 draws the conclusion.

2 Component Assembly Model Analysis

In [Shaw97], *component* and *connector* are proposed as building blocks of software architecture. The examples of component include clients, servers, databases; the examples of connector include procedure call, event broadcast, database protocols. The various kinds of combination patterns of component and connector form the collection of architecture styles.

From the perspective of component assembly, we use the connector concept as an abstraction for glue/wrapper codes necessary for component assembly, and analyze how the use of this abstraction makes the assembly process scalable. The approach of removing assembly logic from the component into the connector can increase the reusability of the component, reduce the complexity and boost maintainability. Meanwhile, assembly model analysis will contribute to the automation of this process. Based on the hierarchical relationship between component and connector in the assembly process, the assembly models can be categorized as follows:

1) the connector and component reside at the same level (Figure 1).
   This is the most common and simple assembly model, and conforms to most architecture styles listed in [Shaw97], such as *pipes and filter,* and *event system.* The connector here may be remote

method call, or event/message based communication for client/server architecture. This model is mostly seen in distributed component assembly.

Figure 1: Component & Connector: Same Level

2) the components are contained in the connector. Figure 2 provides a COM[2] model.

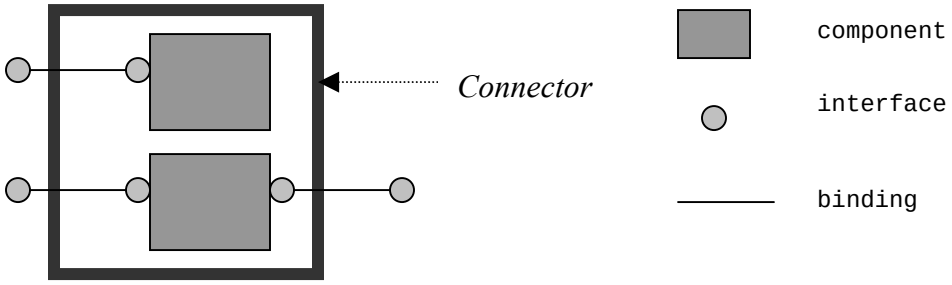

Figure 2: Connector as a Container

The connector acts as an infrastructure in the form of framework, which assembles components via *inversion of control*, such as in EJB[3], CCM[4]; or a package, using such way as manifest file to package components, such as in JavaBeans[5]. Also such connector in some cases plays the role as a container providing extra services for the components to leverage, such as security, transaction, life cycle management, persistence.

3) mixed form of the above two cases.
In this case, component assembly is comprised of a hierarchical process, the father assembly is derived from the assembly of the output of each child assembly process, in the form as described in either (1) or (2). Each child assembly process further is derived from their own child assembly process in either (1) or (2).

3. Two-level Component Modeling/Specification with Aspect Orientation

In light of prior assembly analysis, we propose a *two-level* approach toward an effort of component assembly by handling the modeling of the component and the specification of their interaction (aka. connector) separately: the first level is the modeling of heterogeneous components (their functional as well as non-functional properties [Brah02]) in graphical forms using some advanced CASE tools such as the Generic Modeling Environment (GME) [GME01]; the specification of inter-relationships between components and manipulations of the component model are included in the second level, which constitutes the connector module. The assembly of components for the production of the final system will be in an automatic fashion using an aspect weaver based on the modeling and specification. Figure 3 illustrates the process.

[2] COM: Component Object Model, *http://www.microsoft.com/com.*
[3] EJB: Enterprise Java Beans, *http://java.sun.com/products/ejb*
[4] CCM: CORBA® Component Model, *http://www.omg.org/cgi-bin/doc?orbos/99-07-01*
[5] *http://java.sun.com/beans/*

3.1 Level 1: Component Modeling

One of the Object Management Group (OMG) [6] initiatives is Model Driven Architecture (MDA®) [OMG01], i.e., by reverse engineering legacy systems and Commercial-Off-The-Shelf (COTS) components, software can be transformed into Platform Independent Models (PIMs). PIMs, in turn, will be mapped to Platform Specific Models (PSMs), such as CORBA[7], EJB, SOAP[8] and .NET[9]. In this way, legacy systems

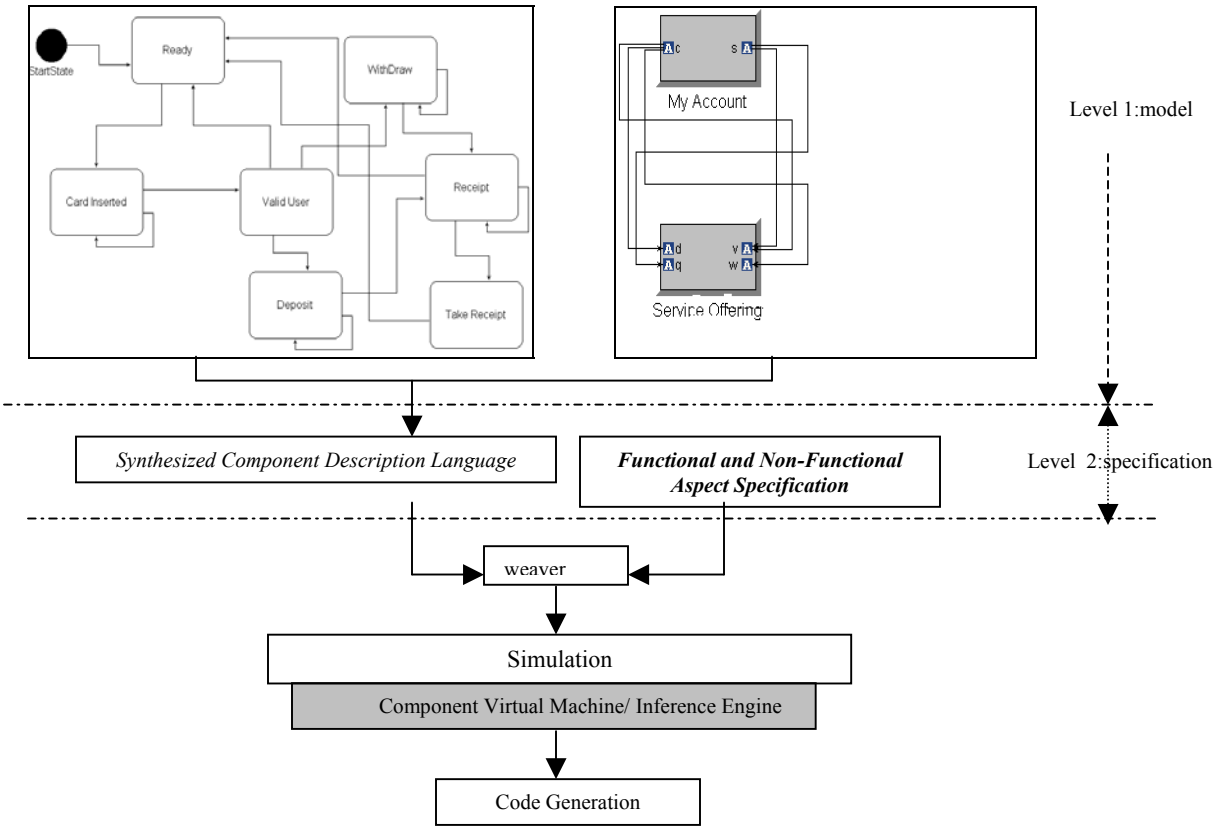

Figure 3: Process of Aspect-Oriented Component Modeling/Specification

and COTS can be reintegrated into new platforms efficiently and cost-effectively. We embrace the same vision here by representing the software components with a model-based approach. However, such PIM model envisioned here is derived by creating meta-models specific to component modeling. In other words, we need to formulate the building blocks for describing component models. This includes the meta-model for business and technology domains [Zhao03]. But these are out of our scope here, which are actually the concerns of some organization such as OMG. Additionally, there should be a means in the component modeling level to represent the *join point* [Kicz97] in a component, which denotes the points that are affected by a particular crosscutting concern. In an AOP language such as AspectJ [Kicz01], *join points* are represented by referring to the syntactical constructs of the base program source. [Stei02] explores the representation of *join points* in UML models by marking affected model elements using UML tags. Here we may denote the *join points* by referring to the meta-information of model constructs. In that sense the

[6] *http://www.omg.org*
[7] *http://www.CORBA.org*
[8] SOAP: Simple Object Access Protocol, *http://www.w3.org/TR/SOAP*
[9] *http://www.microsoft.com/net*

join points here also represent domain knowledge and can serve as query parameters in search of specific components.

As is illustrated in the diagram, the first-level model will be transformed into the second level using a model-based approach consistent with the vision of MDA. This can be achieved easily using the meta-model information of the component models. In GME [GME01], this is realized by using the Builder Object Network (BON) framework for building interpreters, which traverses objects in the model tree by calling methods within the BON API and generates the Component Description Language (CDL), which also includes associated meta-model information to be used as the anchor of the join point.

3.2 Level 2: Component Specification

This level involves the creation of an Aspect Specification Language (ASL[10]) for describing crosscutting concerns in a separate way. Also a weaver is built to weave the ASL with CDL to generate targeted executable specification of components.

3.2.1 Constructs of ASL

In AspectJ [Kicz01], the aspect specification includes three elements: *pointcuts* to pinpoint the affected location of applications; *advice* to describe the actions that are applied to the *pointcuts*; the condition which governs how/when to apply *advice* to *pointcuts* using "before", "after", etc. To generalize for ASL, we need a means to specify:

1) join points.
2) behavior specification describing the actions to be performed.
3) policy on how the behavior is applied to join points.

(1) is as mentioned in 3.1, and is supposed to be specified in CDL. (2) and (3) will be provided in ASL.

3.2.2 Concerns Involved

This part will eventually evolve into a catalog of concerns to be handled in heterogeneous distributed component specification. For now the most distinct concerns involved will be:

1) gluing/wrapping of components.
   The gluing/wrapping of components is generally influenced by such aspects as platform and distribution. The component assembly process will be subject to evolution if components are deployed on a different platform/location. This dynamism can be well embraced by policy description in ASL. The pre/post condition as well as other constraint checking necessitated for the components to perform interaction (here, assembly) can be represented in the behavior specification under the corresponding policy. Obviously here the join points are contained in the involved components to be assembled.
2) QoS measurement.
   We also embed the non-functional aspects such as QoS measurement at the higher level specification of ASL, which will contribute to the measurement of QoS of the generated system at run-time. This is especially desired in a dynamic distributed environment, where a large amount of existent components may be exported for use, overall system QoS serving as the criteria to the filtering of service offerings among peer components. In [Augu95], event grammar is proposed to perform the system testing. We believe the introduction of the aspect-oriented approach will provide support to this effort, i.e., we can treat the QoS probing code as a behavior specification; the policy will govern how the probing code will be called at join points for dynamic measuring of QoS. The probing code will not be manually embedded in the points of interest, but rather using the weaver for dynamic instrumentation.

3.2.3 Simple Assembly Example using Aspect Orientation

To help clarify the aforementioned concepts, we give a simple example demonstrating how aspect orientation can be applied to component assembly. The ideas are adapted from aspectual components [Lieb99], in which aspects are decoupled from the base program by being defined as a generic aspectual

[10] Note this is nothing to do with the *Action Semantics Language* of OMG.

component, which is instantiated later over a concrete data-model. In this way, an aspect definition can be reused. Here we define aspectual component by capturing join points at the meta-model level of components.

Assume the component A is a banking domain client component hosted on Java RMI requesting some banking service from some server side. Below is the partial specification of its CDL:

```
A.0   Component A
A.1   Bankoperation:: Service.
A.2   Bank::BusinessDomain.
A.3   Platform::TechDomain.
A.4   Platform= “RMI”.
A.5   Requires Bankoperations .
A.6   end Component A.
```

Note that right hand side of “::” denotes the *meta-type* of the left hand side. Line A.4 and A.5 are *hyper-rules*. *Meta-type* and *hyper-rule* are Two-Level Grammar notations. For more details of TLG, see [Brya02].

The above specification will be translated into a corresponding aspectual component:

```
B.0   aspect A
B.1   Bankoperation:: Service.
B.2   Bank::BusinessDomain.
B.3   expect Bankoperations.
B.4   expect wrap Argument. //usage interface
B.5   replace  Bankoperation:   //modification interface
B.6         if expected().getComponent().getPlatform()== “CORBA”
B.7         then  return expected().wrap(“RMI”).
B.8   end aspect A
```

Note those lines prefixed by **expect** denote operation signatures that are expected to be supplied with *advice*. In that sense the operation signatures here correspond to the join points in AOP. In the proposed approach here we only use meta-level types for the operation signature definition. Also the above **expected** keyword denotes something to be bound to join points. In line B.3, *Bankoperation* itself is meta-type in the banking business domain. Expected operations are either used (usage interface) or modified (modification interface, preceded with **replace**) in the aspectual component definition. For details please see [Lieb99]. Also lines B.6-B.7 provide *advice* (reimplementation) for the associated operations to be specified in the *connector* part below.

Assume the component B is a banking domain server component implemented in CORBA providing some banking services.

```
C.0   Component B.
C.1   Withdraw, Deposit:: Service;Port.
C.2   Bank::Domain.
C.3   Platform::TechDomain .
C.4   Platform= “CORBA”.
C.5   end Component B.
```

Note in line C.1, the two types denoted in the right hand side of “::” means both withdraw and deposit are not *Services*, but also *Ports*, which means they are component services offered to external components.

The following is an ASL specification for component assembly.

```
D.0   connector A-B
D.1   Bankoperation=Withdraw, Deposit.     //join points
D.2   wrap(Argument): if (Argument.getname==”RMI”)
D.3                {
D.4                     //provide wrapping specification for
```

```
D.5                    //RMI-CORBA inter-operation
D.6               }
D.7   end  connector A-B
```

Note that lines D.2-D.6 further implement the *advice* part for the join points (here, *Withdraw and Deposit* operation). The body of *wrap* is ignored without loss of generality.

From the example illustrated in this section, we can see the interactions of two components can be separated by being handled in a module (here in the aspectual component definition, i.e. the "aspect A" module). Consequently the assembly process can be implemented by using a weaver to weave *advice* together with component specifications. As we can see in the body of "aspect A", it is straightforward for us to apply other concerns in between, e.g., we can call expected().precondition() wherever applicable in the **replace** function body to enforce some preconditions.

3.3 System-Level Simulation

We are investigating such program transformation tool as DMS[11] for building a weaver to weave CDL and ASL together, the output of which will be fed into the simulation phase to validate the functional system behavior against requirements before implementation code is generated and deployed. This simulation may be carried out by building a component virtual machine [Ducl02], which serves as an interpreter to interpret the weaved specifications; or by building rule sets based on requirement and then use some inference engine to validate the functional requirements. In this way, the assembled system will be functionally sound at an early phase. On the other hand, the generated applications, as they are probed with non-functional aspect related codes, are amenable to be benchmarked over the specific QoS parameters [Brah02] in the system deployment time.

4. Summary and Future Work

We have presented a two-level approach for handling the crosscutting concerns of functional/non-functional concerns in integrating heterogeneous distributed components. This approach has a close tie to MDA in the sense that we leverage component modeling at the first level and then map the component models into the CDL in the second level. The CDL and ASL will be weaved together to generate the executable specification for system simulation. The approach also applies to model weaving in MDA.

We have applied modeling techniques for enriching semantics of Web Services and to generate semantically enriched Web Service Description Language (WSDL) [Cao03]. We have also prototyped CDL for component assembly [Cao02]. Future efforts will be to apply modeling experiences to describing the semantics of component cases of some specific domain, and to build ASL together with its associated weaver for the synthesis of executable specifications.

References:

[Augu95] M. Auguston. Program Behavior Model Based on Event Grammar and its Application for Debugging Automation. *Proceedings of the 2nd International Workshop on Automated and Algorithmic Debugging*, pp. 277-291, 1995.

[Brah02] G. J. Brahnmath, R. R. Raje, A. M. Olson, M. Auguston, B. R. Bryant, and C. C. Burt. A Quality of Service Catalog for Software Components. *Proceedings of (SE)2 2002, the Southeastern Software Engineering Conference*, pp. 513-520, 2002.

[11] DMS: Design Maintain System™, *http://www.semdesigns.com/*

[Brya02] B. R. Bryant, B.-S. Lee. Two-Level Grammar as an Object-Oriented Requirements Specification Language. *Proceedings of 35th Hawaii Int. Conf. System Sciences*, 2002, http://www.hicss.hawaii.edu/HICSS_35/HICSSpapers/PDFdocuments/STDSL01.pdf.

[Cao02] F. Cao, B. R. Bryant, R. R. Raje, M. Auguston, A. M. Olson, C. C. Burt. Component Specification and Wrapper/Glue Code Generation with Two-Level Grammar using Domain Specific Knowledge. *Proceedings of 4th International Conference on Formal Engineering Methods (ICFEM'02)*, LNCS 2495, Springer-Verlag, pp. 103-107, 2002.

[Cao03] F. Cao, B. R. Bryant, C. C. Burt, J. G. Gray, R. R. Raje, A. M. Olson, M. Auguston. Modeling Web Services: Toward System Integration in UniFrame, to appear in *Proceedings of 7th World Conference on Integrated Design and Process Technology (IDPT'03)*, 2003.

[Czar00] K. Czarnecki, U.W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.

[Ducl02] F. Duclos, J. Estublier, P. Morat. Describing and Using Non Functional Aspects in Component Based Applications. *Proceedings of Second International Conference on Aspect-Oriented Software Development, AOSD'02*, 2002.

[GME01] *GME 2000 User's Manual, Version 2.0*, ISIS, Vanderbilt University, 2001.

[Kicz97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. *Proceedings of European Conference on Object-Oriented Programming (ECOOP)*, LNCS 1241, Springer-Verlag, pp. 220-242, 1997.

[Kicz01] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold An Overview of AspectJ. *Proceedings of European Conference on Object-Oriented Programming (ECOOP)*, LNCS 2072, Springer-Verlag, pp.327-353, 2001.

[Lieb99] K. Lieberherr, D. Lorenz, M. Mezini. Programming with Aspectual Components. *Technical Report, NU-CCS-99-01*, 1999, http://www.ccs.neu.edu/research/demeter/papers/aspectual-comps/aspectual.ps.

[OMG01] Object Management Group (OMG). Model Driven Architecture: A Technical Perspective. *Technical Report. Document # ormsc/2001-070-1,* Framingham, MA, Object Management Group, 2001.

[Raje00] R. R. Raje. UMM: Unified Meta-object Model for Open Distributed Systems. *Proceedings of ICA3PP, 4th IEEE Int. Conf. Algorithms and Architecture for Parallel Processing*, pp. 454-465, 2001.

[Raje01] R. R. Raje, B. R. Bryant, M. Auguston, A. M. Olson, C. C. Burt. A Unified Approach for the Integration of Distributed Heterogeneous Software Components. *Proceedings of Monterey Workshop Engineering Automation for Software Intensive System Integration*, pp. 109-119, 2001.

[Shaw96] M. Shaw, D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[Stei02] D. Stein, S. Hanenberg and R. Unland. On Representing Join Points in the UML. *Aspect Modeling with UML Workshop at the Fifth International Conference on the Unified Modeling Language and its Applications*, 2002, http://www-stud.uni-essen.de/~sw0136/wissensArbeiten/UML02Workshop.pdf.

[Zhao03] W. Zhao, B. R. Bryant, C. C. Burt, J. G. Gray, R. R. Raje, A. M. Olson, M. Auguston. A Generative and Model Driven Framework for Automated Software Product Generation. *Proceedings of CBSE 6, the 6th Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction*, 2003, http://www.csse.monash.edu.au/~hws/cgi-bin/CBSE6/Proceedings/papersfinal/p31.pdf.