Modeling and Integrating Aspects into Component Architectures

Lydia Michotte
Prism CNRS Lab
Versailles University
France
lydia.michotte@prism.uvsq.fr

Robert B. France
Departement of Computer Science Colorado State University, Colorado, USA
france@cs.colostate.edu

Franck Fleurey
IRISA
Rennes
France
ffleurey@irisa.fr

Abstract

Dependable software systems are difficult to develop because developers must understand and address several interdependent and pervasive dependability concerns. Features that address pervasive dependability concerns such as error detection and recovery tend to crosscut application architecture and thus understanding and changing their descriptions can be difficult. Separating these features at the architectural level allows one to better understand and reuse them and thus can lead to better analysis and evolution of the features during design. In this paper we illustrate how an Aspect Oriented Modeling (AOM) technique can be used to model dependability aspects of component architectures separately from other aspects. The AOM architectural model used to illustrate the approach in this paper consists of a component primary view describing the base architecture and a component template aspect model describing a fault tolerance feature that provides error detection and recovery services.

1. Introduction

Developers of critical component-based software systems often must address pervasive fault tolerance concerns in their designs. A concern [1] is a problem and a set of properties determining acceptable solutions. Features that address pervasive fault tolerance concerns may not be conveniently encapsulated in a component. These crosscutting features can be difficult to understand, analyze and change because their descriptions are not localized in a single place.

Support for the separation of fault tolerance concerns early in the design cycle [3] [2] [4]can help reduce late and costly architectural changes. One approach to separation of concerns is Aspect Oriented Modeling (AOM) [1] [6] [17]. We have not encountered any previous work on the aspect oriented modeling of component architectures. In this paper the major contribution is to extend the AOM approach to support separation of crosscutting features in component architectures. A component architecture consists of software components, their structural relationships and their behavioral dependencies [16]. We focus only on modeling structural aspects of component architecture. Each component is associated with one or more provided or required interfaces. A provided interface contains the operations supported by the component, a required interface contains the operations required by the component.

An aspect-oriented component architecture model produced by AOM consists of a base component architecture model called the primary view which reflects the design decisions that determine the core component structure; and a set of aspect views. Each aspect view describes a feature that crosscuts the primary view. An integrated view of the component architecture is obtained by composing the primary and aspect views.

In this paper we show how the AOM approach can be used to model and integrate crosscutting features at component architecture level. To illustrate the approach we model and integrate a fault tolerant feature. A recovery block fault tolerance feature is modeled as an aspect and integrated with a primary view describing the component structure of a health monitoring system. The remainder of this paper is organized as follows. Section 2 presents an overview of the AOM approach as applied to component architectures. Section 3 describes the component-based recovery block feature and presents the aspect describing the feature. In Section 4, we describe how

1541-7719/07 $25.00 © 2007 IEEE
DOI 10.1109/EDOC.2007.49

the models are composed. Section 5 discusses related work, and Section 6 concludes with our plans to extend the approach.

2. Background

Aspect Oriented Modeling (AOM) [1] is an approach to separating crosscutting features from other features in order to ease understanding, analysis and evolution of the features. In this paper a feature addresses a single concern. A feature whose description is tangled with the description of other features and is distributed across a model is said to crosscut the model. An aspect-oriented model in the AOM approach consists of [1]:

1. A *primary view*: The primary view describes the features that determine the dominant design structure.

2. A set of *patterns called aspect models*: The crosscutting features that can be isolated in aspect models are those with distributed elements that have common characteristics. This allows one to describe a crosscutting feature as a pattern.

3. A set of *bindings*: A binding associates an application-specific element to a pattern element. Applying the bindings to an aspect model produces an *aspect view* that describes how the feature is to be realized in the primary view. To incorporate the features described by aspect models into a primary view, the aspect views produced from the aspect models are composed with the primary view.

4. A set of *composition directives*: Composition directives are used to tailor the composition of aspect and primary views [6].

In this paper primary and aspect views are UML descriptions of component architectures. Here, a component describes a deployable unit of implementation [16].

Figure 1 gives an overview of model composition in the AOM approach. Before an aspect model can be composed with a primary design model, the pattern described by an aspect model must be instantiated in the context of the application domain. An aspect view is obtained by binding elements in the aspect model to elements in the application domain. Application specific element names are drawn from an application domain namespace. Aspect and primary views are composed to obtain an integrated design view.

The AOM approach in this paper uses a signature based approach applied to component architecture. A signature is a set of syntactic properties of a model element. Model elements with the same signature are merged to form a single model element. The following rules are used to compose component model elements:

1. For components the signatures consist only of their names and thus components with the same name are merged to form a single component in the composed model.

2. Provided interfaces on matching components must have the same name to match. The merged interface contains the union of the operations in the source interfaces and syntactically equivalent operations are included only once in the merged interface.

3. Required interface must have the same name, and the same operations to match.

4. If matching operations (opi) and (opj) are associated with specifications (S1) and (S2), then the result of composing the operations is an operation associated with a specification that is the conjunction of (S1) and (S2). A composition directive can be used to vary how the specifications are logically connected.

5. Unmatched components or operations i.e., components or operations that only occur in either the aspect model or the primary view are included in the composed component diagram.

Composition directives override a subset of the composition rules used by the composition mechanism.

The directives are needed when use of the rules implemented by the composition mechanism is not enough to produce well formed models that have desired properties.

In general, a composition directive can:

1. Determine the order in which multiple aspect models are composed with a primary view.

2. Define precedence or override relationships between matching aspect and primary view components with conflicting properties or definitions.

3. Determine the elements that are renamed (e.g., to resolve conflicts), added, replace or deleted during composition, these directives are called refactoring directives. Adding new components or interfaces or deleting existing components or interfaces may be necessary to correctly compose aspect and primary views.

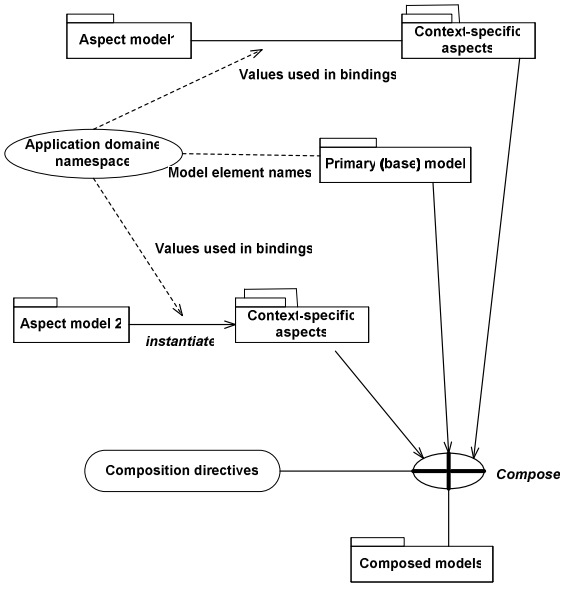


Figure 1 : An AOM overview of Composition in the AOM Approach [1]

3. A fault tolerance architectural aspect

In this section a component-based aspect model for a recovery block fault tolerance feature is presented.

3.1. Modeling recovery block architectural aspect

At the code level, a recovery block [8] consists of:

1. A block of application code or program called the primary block that contains the program primary version.

2. Several alternate blocks that contain program alternate versions executing the same functionality as the primary version.

3. An acceptance test that is the same for the primary and alternate versions.

If the primary block fails, as detected by the acceptance test, the alternates are tried sequentially. This technique uses backward error recovery to return to the state at the start of the execution before executing the next alternate.

We call this state the current component state. In this paper the recovery block technique is adapted to components. Recovery blocks are associated with operations which developers deem as susceptible to failures called the critical operations. Operations and corresponding components to be recovered are structured in a recovery block.

The recovery block aspect consists of five component templates described below and interface templates (see **Figure 2**):

1. A recovery block manager component template which provides several services: execution of the component primary version, saving of the current component state, execution of acceptance tests, performing roll back and alerts on system total failures.

2. A client component template that calls for execution of the critical operation via the recovery block manager.

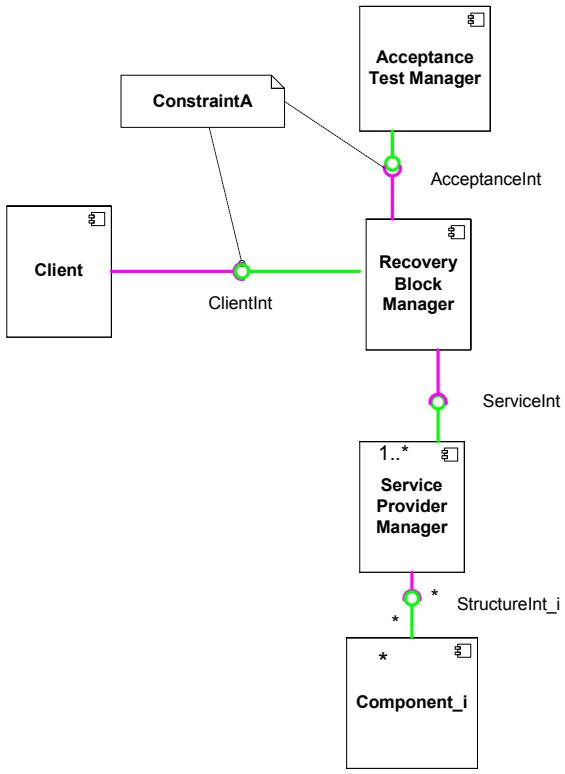
3. A Service provider manager template that provides the component primary version and alternate versions.

4. An acceptance test manager template which evaluates the critical operation execution results for the primary and alternate component versions.

5. The Component_i template used to instantiate the structure of components to which the component primary service provider and its alternates are connected to in the component architecture.

The client component via its (ClientInt) interface calls the recovery block manager to execute a critical operation (opi). The recovery block manager then saves the current component state which consists of the operation (opi) arguments values, and executes via the (ServiceInt) interface, the operation (opi). At the end of (opi) execution, the operation result is recorded by the recovery block manager which then executes the acceptance test on that result via its (AcceptanceInt) interface.
If the acceptance test result is acceptable then the operation ends successfully and the recovery block sends the operation result to the client component. If the acceptance test result is not acceptable, the first service alternate version provided is executed and operation (opiAlt) is executed with the recorded current component state.
If all service alternate versions failed when executed then the system fails and the recovery block manager sends an alert of system failure.

Figure 2 shows the component diagram template of the recovery block aspect.



ConstraintA: #opi= #testopiopiAlt

Figure 2: Recovery Block Aspect Model

The services provided at the interfaces of the recovery block aspect components are described below, (see **Figure 3**):

1. The ServiceInt interface template contains operation (opi) that invokes the service primary version. This interface template can be instantiated one to (n) times to invoke the service alternate versions with the operation opialtj. The operations (opialtj) and (opi) can have respectively one to (n) arguments.

2. The clientInt interface template contains operation (opi) that is called through the recovery block manager. This interface template can be instantiated one to (n) times and the operation (opi) can have one to (n) arguments.

3. The AcceptanceInt interface contains an operation (testopiopialt) that executes the acceptance test the result of operation (opi) or operation (opialtj).

4. The StructureInt_i interface contains the operation (opstructi) that is invoked between the component_i of the components structure and the component primary service version and its alternates. This interface template can be instantiated one to (n) times and the operation (opstructi) can have one to (n) arguments.

Each operation has a corresponding acceptance test operation. This is expressed in the aspect model as a constraint (ConstraintA in Figure 2).

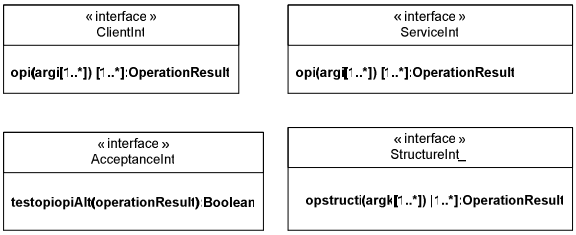


Figure 3 : Recovery Block Aspect Interfaces Class Diagram

The recovery block aspect is applied to a component called the recovery target component and to operations called target or critical operations. The recovery block aspect can be used in a centralized way or a distributed

way. In the centralized way, the different recovery target components share the same recovery block manager. In the distributed way, each recovery target component has its own recovery block manager. The use of one of these variations (centralized, distributed) is a design decision. In section 4 we give an example of how the aspect model can be instantiated.

3.2. The recovery block aspect sequence diagram

The recovery block execution sequence is shown **Figure 4**:

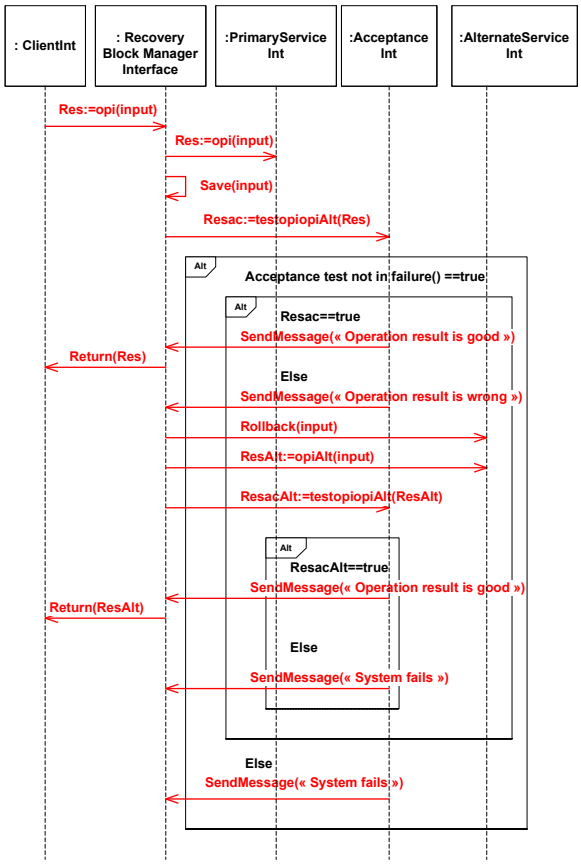


Figure 4: Recovery Block Aspect Sequence Diagram Using One Alternate

The client component via its interface (clientInt) calls the recovery block manager for the execution of a critical operation (opi).

Then the recovery block manager executes the operation via its interface (PrimaryServiceInt) for the target recovery component primary version saving first the current component state which are here the operation set of argument values called here (input).

At the end of (opi) execution, the operation result is recorded by the recovery block manager which executes the acceptance test operation (testopiopiAlt) on that result.

Two cases are possible:

1. If the result is acceptable then it is returned to the client.

2. If the result is not acceptable then an error has occurred.

   2.1. The system rolls back to the initial state of the current executing component using the saved data called (input).

   2.2. An alternate component version is executed.

If all Alternate versions fail, the acceptance test may be in failure or all the system fails.

4. Composition with fault tolerance architectural aspect views and primary view

In this section we describe a health watcher component architecture that is a primary view (see **Figure 5**) and with the recovery block pattern or aspect model component template (see **Figure 2**) we instantiate aspect views (see **Figure 7** and **Figure 8**) defining bindings and generating corresponding composed model (see **Figure 9** and **Figure 10**).

The AOM composition is very flexible we can use the same pattern or aspect model and the same primary view for different recovery block mechanisms (distributed, centralized or mixed) by applying different merging rules, what is detailed below.

4.1. The health watcher system primary view

The health watcher system is a web based system for collecting and managing public health related complaints and notifications. The system also notifies people by mail and news on web pages about important information regarding the health system. The system users are citizens or health department employees. The system must be highly available for users 24 hours a day and 7 days a week.

The primary component architecture (the AOM primary view) is shown in **Figure 5** consists of six components that interact via interfaces, (see **Figure 6**). The components are: the user citizen manager, the user employee manager, the complaint manager, the request manager, the login manager, the notification manager.

The user citizen manager component interacts with the login manager to be authenticated, with the complaint manager component in order to enter new complaints also it interacts with the citizen request manager to send queries to the system and receives notifications from the notification manager component about requests and complaints or general health information.

The user employee manager component interacts with the complaint manager component to handle the complaint, with the request manager component to handle requests, with the login manager component to authenticate employees and to change login.

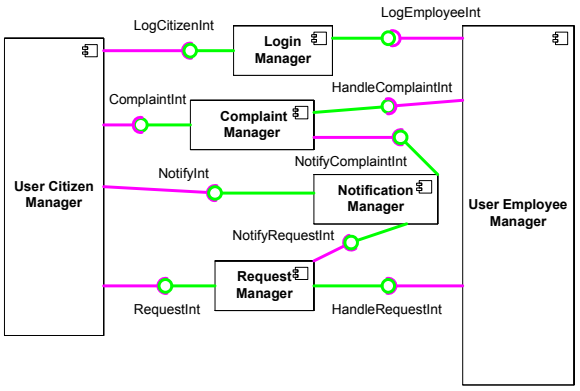


Figure 5 : The Health Watcher System Primary View Component Diagram

The interfaces of the health watcher primary view are detailed **Figure 6**:

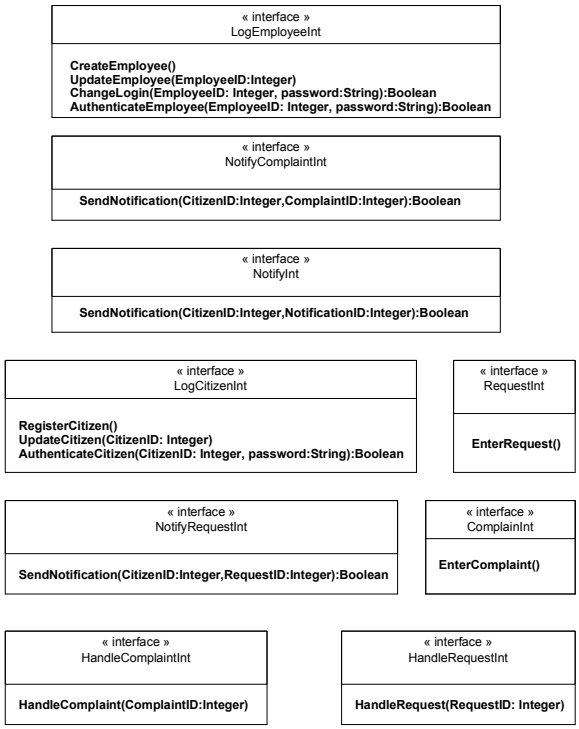


Figure 6: Health Watcher Primary View Interfaces Class diagram

4.2. The Recovery Block Aspect Views

The recovery block context specific aspect model or aspect view is an instantiation of the recovery block aspect model used in the centralized and/or distributed way in the context of the health watcher application.

In the health watcher primary view we have decided to apply the recovery block aspect model to the notification manager component, the complaint manager component and the request manager component which becomes the target recovery components. A given target recovery component has only one alternate component version in a recovery block. The target operations are the ones that send complaint and request notification to citizens:

1. op1= SendNotification(CitizenID: Integer, NotificationID: Integer) : Boolean.

2. op2= SendNotification(CitizenID: Integer, ComplaintID: Integer) : Boolean.

3. op3= SendNotification(CitizenID: Integer, RequestID: Integer) : Boolean.

Aspect views describing distributed and centralized recovery block features are described below.

4.2.1. The distributed recovery blocks aspect view

In the distributed recovery block aspect view each target recovery component has its own instantiation of the aspect model in the context of the health watcher primary view as shown in
Figure 7**.**

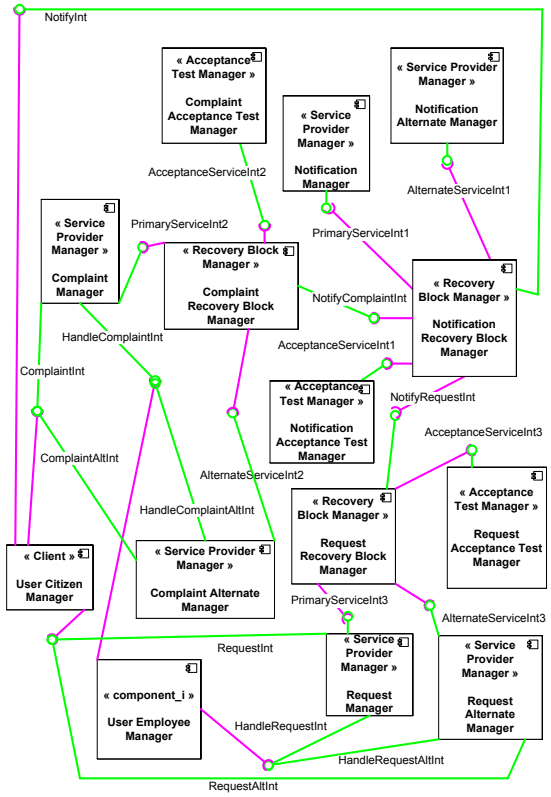


Figure 7: Aspect View for Distributed Recovery Blocks

4.2.1.1. Notification aspect view

The notification aspect view is the instantiation of the notification manager component recovery block aspect model in the context of the health watcher primary view.

For the notification manager component:
1. The target operation is op1= SendNotification(CitizenID: Integer, NotificationID: Integer) : Boolean.

2. The alternate operation is op1Alt= SendNotificationAlt(CitizenID: Integer, NotificationID: Integer) : Boolean.

3. The acceptance test operation is testop1op1Alt.

4.2.1.2 Complaint Aspect View

The complaint aspect view is the instantiation of the complaint manager component recovery block aspect model in the context of the health watcher primary view.

For the complaint manager component:
1. The target operation is op2= SendNotification(CitizenID: Integer, ComplaintID: Integer) : Boolean.

2. The alternate operation is op2Alt= SendNotificationAlt(CitizenID: Integer, ComplaintID: Integer) : Boolean.

3. The acceptance test operation is testop2op2Alt.

4.2.1.3. Request aspect view

The request aspect view is an instantiation of the request manager component recovery block aspect model in the context of the health watcher primary view.

For the request manager component:
1. The target operation is op3= SendNotification(CitizenID: Integer, RequestID: Integer) : Boolean.

2. The alternate operation is op3Alt= SendNotificationAlt(CitizenID: Integer, RequestID: Integer) : Boolean.

3. The acceptance test operation is testop3op3Alt.

4.2.2. Centralized Complaint_Request, Distributed Notification Recovery Blocks Aspect View

In this scenario, the target complaint and request recovery components share the same recovery block manager instantiation with the name Complaint_Request Recovery Block Manager (see **Figure 8**).

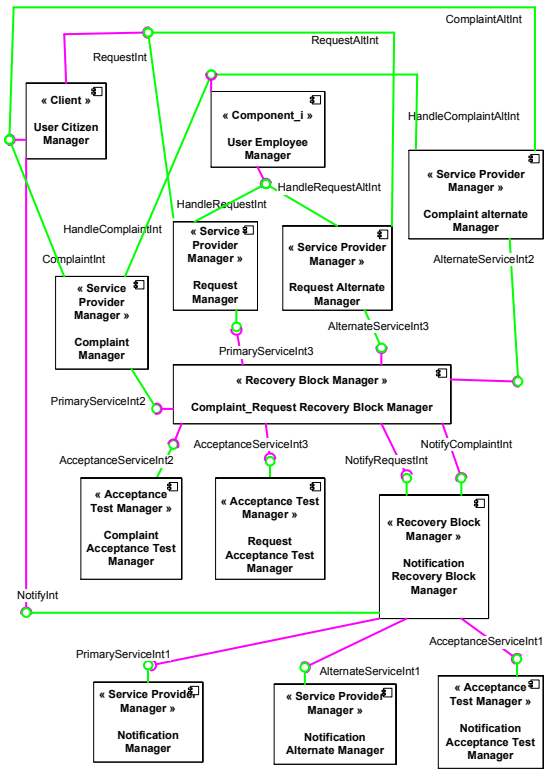


Figure 8: Aspect View for Centralized Complaint_Request, Distributed Notification Recovery Blocks

4.3. The Recovery Block Composed Model

The composed model is the composition of the recovery block aspect view and the health watcher primary view.

The set of rules and directives used to compose the views are given below:

1. Components match if they have the same name.

2. Required interfaces match if they have the same name and the same operations.

3. Provided interfaces match if they have the same name.

4. If the matching components have operations with operation specifications, the operation specification in the composed model is the conjunction of the operation specifications associated with the matching operations. A composition directive can be used to vary how the specifications are logically connected.

5. Unmatched components i.e., component that only occur in either the aspect model or the primary view are included in the composed component diagram.

6. Composition directives can be used to add, delete, replace and rename components and interfaces.

Two composed model have been generated: The distributed recovery block composed model and the centralized Complaint_Request distributed notification recovery block composed model. In our models the required and provided interfaces have the same name and same operations.

4.3.1. The Distributed Recovery Blocks Composed Model

In the distributed recovery block composed model shown in **Figure 9**:

The unmatched components added are: “Login Manager”, “Notification Recovery Block Manager”, “Complaint Recovery Block Manager”, “Request Recovery Block Manager”, “Notification Acceptance Test Manager” “Complaint Acceptance Test Manager” “Request Acceptance Test Manager”, “Notification Alternate Manager”, “Complaint Alternate Manager” and “Request Alternate Manager” components.

The interfaces are the same as described in the aspect view and in the primary view but composition directives are used to delete interfaces:

1. Interface NotifyInt between Notification Manager Component and User Citizen Manager Component.

2. Interface NotifyComplaintInt between Complaint Manager Component and Notification Manager Component.

3. Interface NotifyRequestInt between Request Manager Component and Notification Manager Component.

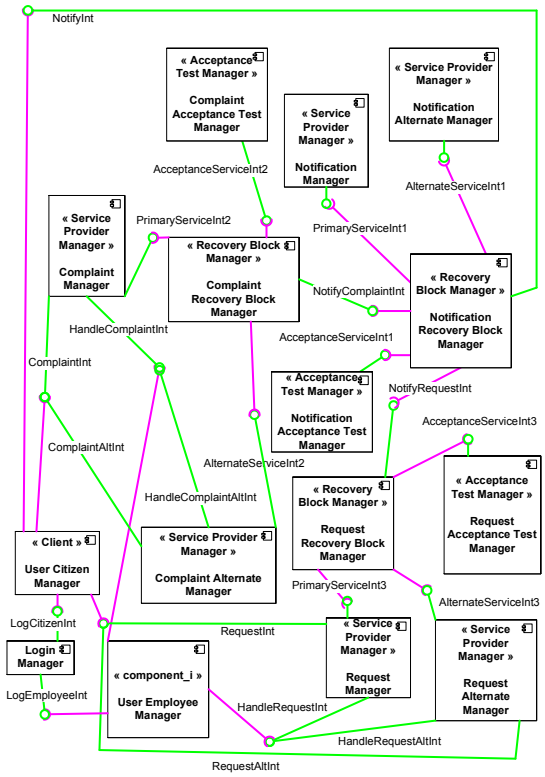


Figure 9: Composed Model for Distributed Recovery Blocks

4.3.2. The Centralized Complaint_Request, Distributed Notification Recovery blocks Composed Model

In the Centralized Complaint_Request, Distributed Notification Recovery blocks Composed Model shown in **Figure 10**:

The unmatched components added are: "Login Manager", "Notification Recovery Block Manager", "Complaint_Request Recovery Block Manager", and "Notification Acceptance Test Manager" "Complaint Acceptance Test Manager" Request Acceptance Test Manager, "Notification Alternate Manager", "Complaint Alternate Manager" and "Request Alternate Manager" components.

The interfaces are the same as described in the aspect view and in the primary view but composition directives are used to delete interfaces:

1. Interface NotifyInt between Notification Manager Component and User Citizen Manager Component.

2. Interface NotifyComplaintInt between Complaint Manager component and Notification Manager Component.

3. Interface NotifyRequestInt between Request Manager component and Notification Manager component.

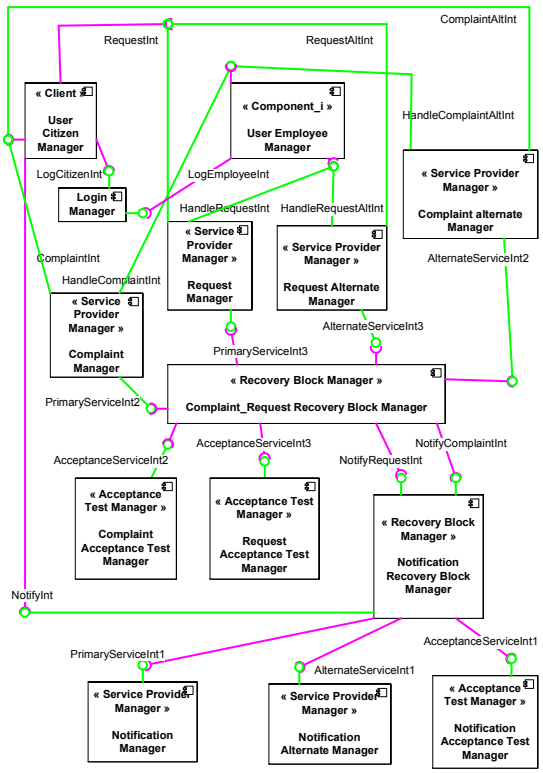


Figure 10: Composed Model for Centralized Complaint_Request, Distributed Notification Recovery Blocks

5. Related work

We are not aware of any other work that specifically addresses the component oriented modeling of recovery block fault tolerant feature and its integration with a given application using the AOM approach.

However there is considerable research on recovery block modeling [7][8][9][13][14] and AOM application to features like security [10], performance [12][10], access control [11].

The recovery block modeling basic approach is procedure oriented [8][7][9] but some object oriented solutions [14] also exist. None of the work we encountered deal with use of recovery blocks at the component level design. A description of an "idealized component" is given in [13] [15]but it is not detailed in term of operations.

6. Conclusion

Current AOM research addresses the problem of crosscutting concerns by providing support for separating the concerns, composing aspect and primary views, analyzing composed models to identify and resolve conflicts that may arise as result of composition.

In this paper our major contribution is to extend the AOM approach to component architectures.
To illustrate the approach we use aspects to describe a fault tolerance recovery block feature at the architectural level, we developed a component based recovery block aspect model and we have showed how it can be integrated with component architecture and we apply our approach to a health monitoring system.

Composing models by hand can be tedious and thus we plan to build a tool to automate the process. There is a tool for class diagram composition [18] that can be extended to take into account component diagram. This work may be extended also to support composition of behavioral models.

7. References

[1] R.B. France, I. Ray, G. Georg, S. Ghosh , " An Aspect-Based Approach to Early Design Modeling". IEEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, 151(4), pp 173-185, August 2004.

[2] Garlan, D., Perry, D., Introduction to the Special Issue on Software Architecture. IEEE Transactions on Software Engineering, 21(4), April 1995.

[3] Garlan, D., Shaw, M., An Introduction To Software Architecture. In V. Ambriola and G. Tortora, editors, Advances in Software Engineering, Vol 1, pages 1-40. World Scientific Publishing Company, 1993.

[4] Garlan, D., Software Architecture: a Roadmap. In A. Finkelstein, editor, The Future of Software Engineering, International Conference on Software Engineering (ICSE'2000), pp 93-101, Limerick, Ireland, June 2000. IEEE Computer Society Press / ACM Press.

[5] http://www.omg.org/uml.

[6] Straw, G., Georg, G., Ghosh, S., France, R., Model composition Directives. In Proceedings of the International Conference on the UML, October 2004, pp 84-97. Springer, 2004.

[7] Randell, B., System Structure for Software Fault Tolerance, IEEE Transactions on Software Engineering, vol SE-1, no.2, pp 220-232, June 1975.

[8] Randell, B, Xu, J., The Evolution of the Recovery Block Concept, In Sofware Fault Tolerance book edited by Lyu, M.R.,, Wiley, 1995.

[9] Randell, B., Horning, J.J., Lauer, H.C., Melliar-Smith, P.M., A Program Structure for Error Detection and Recovery. In Proceedings Conference Operating Systems: Theoretical and Practical Aspects, pp 23-25, IRIA, April 1974.

[10] Petriu, D.C., Woodside, C.M., Petriu, D.B., Xu, J., Geri, G., France, R., Bieman, J.M., Houmb, S.H., Performance Analysis of Security Aspects in UML Models.

[11] Ray, I., France, R., Li, N., Geri, G., An Aspect-Based Approach to Modeling Access Control Concerns, Information and Software Technology, 40(9), pp 557-633, 2004.

[12] Shen, H., Petriu, D.C., Performance Analysis of UML Models Using Aspect-Oriented Modeling Techniques

[13] Xu,J., Randell, B., Rubira-Calsavara, C.M.F., Stroud, R.J., Towards an Object-Oriented Approach to Software Fault Tolerance

[14] Miller, J., Wood, M., Brooks, A., Roper, M., Applying Objet Oriented Construction to Fault Tolerant Systems, In IEEE, 1994.

[15] Anderson,T., Lee, P.A., Fault Tolerance Principles and Practice. Prentice Hall International, 1981.

[16] Daniels, J., Cheesman, J., UML Components, A simple Process for Specifying Component-Based Software. Addison Wesley, 2001.

[17] Clarke,S., Baniassad, E., Aspect-Oriented Analysis and Design: The Theme Approach, Addison Wesley April 2005.

[18] Aspect Composition tool, http://www.cs.colostate.edu/puml/kompose2.html.