An Aspect-Oriented Approach to Handling Crosscutting Concerns in Activity Modeling

Jing Zhang, Yan Liu, Michael Jiang, and John Strassner

***Abstract*—Activity modeling is known as a powerful technique for designing and specifying the flow logic of a process. Due to the complexity of the described process, activity models may involve multiple activities that are tangled with each other. Such activities are known as crosscutting concerns that are difficult to modularize using existing activity modeling constructs. This paper presents an aspect-oriented approach to supporting separation of crosscutting concerns in activity models. An extension to activity modeling is introduced for encapsulating crosscutting activities in well-modularized aspects, which are in turn composed with base activities by a specialized aspect weaver in a systematic way.**

***Index Terms*—Aspect-oriented modeling, Crosscutting concerns, Activity modeling.**

I. Introduction

Activity modeling, as one of the main UML techniques, is frequently adopted in the specification of the behavioral aspects of a system. Activity models are often applied to document workflows in a system, e.g., the logic of a single operation, the scenario of a use case, or the flow logic of a business process. In UML 1.x [5], activity models are defined as a special case of state machines, mainly for describing a computational process in terms of control flow and data flow in state-transition-oriented systems. Since the adoption of the new UML 2.x specification [6], activity modeling is redesigned and based on Petri Nets [16] semantics instead of state machines, which "widens the number of flows that can be modeled, especially those that have parallel flows." [6] It is believed that with such enriched expressive power and well-defined semantics, activity modeling will gain more and more popularity in the design and development of complex software systems.

Nonetheless, due to the increasing complexity of software systems, activity models may involve numerous activities that are tangled within the boundary of a single module. In other cases, a single activity may be scattered across several different activity modules. Such activities are defined as crosscutting concerns that are hard to modularize into separate units using existing software composition techniques. Examples are authentication, logging or error handling activities that spread across the base functionality of the system. The occurrence of tangling and scattering often leads to several impediments to system comprehension and maintenance:

1) Discovering or understanding a specific concern representation that is spread over the system hierarchy is difficult, because the concern is not localized in one single module. This limits the ability to reason analytically about such a concern.
2) Changing a concern requirement is also difficult and time-consuming, because the engineers must go into each relevant module and modify the specific elements one by one. The change process is error-prone and affects productivity and correctness [12].

Aspect-oriented software development (AOSD) [2] offers a powerful technology for handling such concerns, whereby the crosscutting is explicitly specified as an *aspect*. With the intent to support separation of crosscutting concerns involved in activity specification, this paper applies an AOSD approach to activity modeling. An aspect-oriented extension to activity modeling is proposed for encapsulating crosscutting concerns in the constructs of aspects, which are systematically integrated with the base activities by an underlying aspect weaver. Motorola has previously developed an industry-strength weaver for enabling aspect-oriented weaving for UML statecharts that include action semantics [10][18]. This paper extends the Motorola aspect weaver with support for activity models. The goal is to provide designers with more coherent and manageable activity modules through the clean separation of concerns.

The remainder of the paper is structured as follows. Section II gives a brief overview of activity modeling, including the activity metamodel definition. Section III presents the proposed aspect-oriented extension to activity modeling as well as the underlying aspect weaving mechanism. Section IV offers a case study using the proposed aspect-oriented approach to specify a timeout handler aspect for a network fault management system. Finally, the last two sections discuss the related work and conclusions.

II. Activity Modeling

As has been stated in the Section I, activity modeling is about specifying the behavioral aspect of a system. It is typically used to define a computational process in terms of the control flow and data flow among its constituent actions. This section provides a basic background introduction to activity modeling in order to set the context for our

Jing Zhang is with Motorola Autonomics Lab, Motorola Inc., Schaumburg, IL 60196 USA (phone: 847-576-6444; fax: 847-576-0658; e-mail: j.zhang@motorola.com).
Yan Liu is with Motorola Autonomics Lab, Motorola Inc., Schaumburg, IL 60196 USA (e-mail: yanliu@motorola.com).
Michael Jiang is with Motorola Autonomics Lab, Motorola Inc., Schaumburg, IL 60196 USA (e-mail: michael.jiang@motorola.com).
John Strassner is with Motorola Autonomics Lab, Motorola Inc., Schaumburg, IL 60196 USA (e-mail: john.strassner@motorola.com).

aspect-oriented enhancement contribution in activity modeling.

Figure 1 shows the simplified activity metamodel in the Meta-Object Facility (MOF) [4] specification. An activity contains various kinds of nodes connected by edges to form a complete flow model. The sequencing of actions is controlled by control flow and object flow edges. An activity node can be an action, an object node or a control node. Some of the common kinds of actions are listed in Figure 1. An operation action may reference an activity specification, which means that the invocation of the operation involves the execution of the referenced activity. Send signal action and accept event action deal with event and signal transmission. An object node holds data that flow through the activity model. A pin is an object node that can be attached to actions for expressing inputs and outputs. Control nodes are responsible for routing control and data flows in an activity. For instance, decision node and merge node are used to designate conditional behavior, while fork node and join node are used to delineate parallel behavior. Activities can be divided into different partitions that represent different kinds of activity groups for identifying actions that have some characteristics in common. Activity actions can also be grouped into an interruptible region, within which all execution can be terminated if an interrupting activity edge is leaving the region.

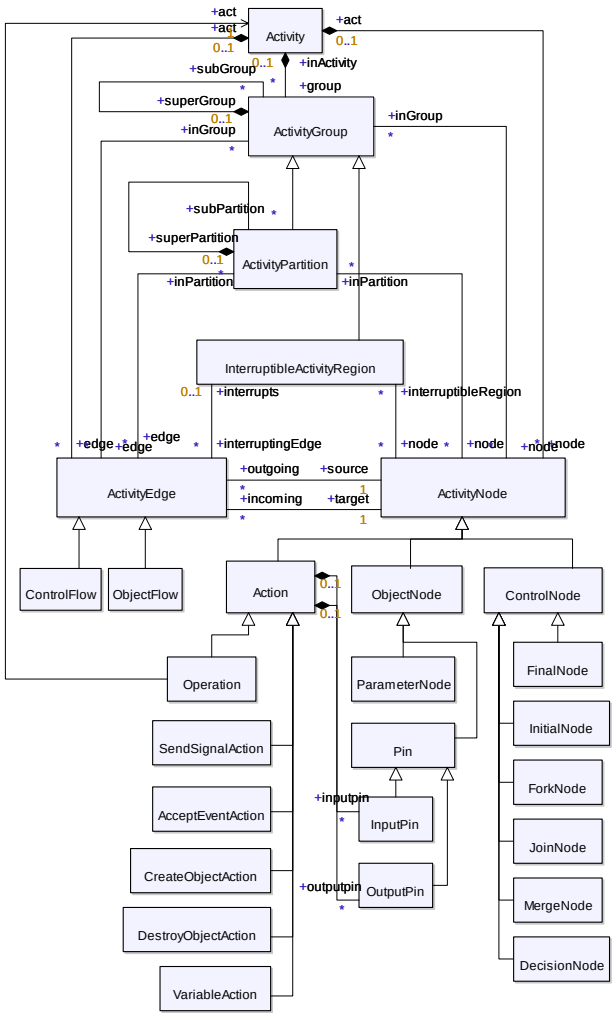

Figure 1. Simplified Activity Metamodel

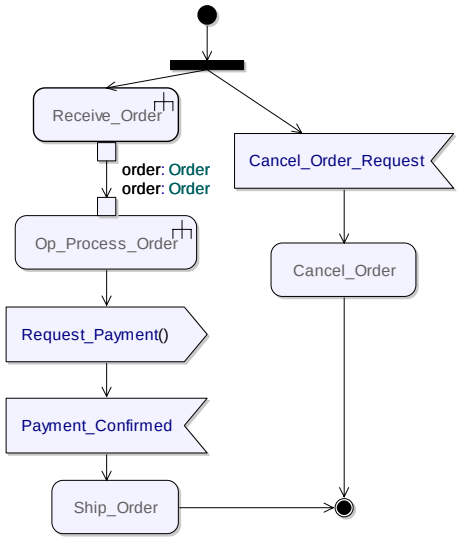

Figure 2. An Order Processing Activity Model

An example of an order processing activity model is illustrated in Figure 2 (adapted from [6]). Two concurrent flows are involved. One focuses on the normal procedure for order processing, including order receiving, order processing, payment handling and order shipment. Another flow indicates that during the same period of time that the first flow proceeds, the order will be cancelled whenever a cancellation signal is received.

III. Aspect-Oriented Activity Modeling

As the complexity of the described system grows, activity specifications also grow in complexity. This growth requires lifecycle maintenance for the concerns that crosscut different activity modules. For instance, a new requirement asking for a tracing capability that logs all the information of all operation actions results in appending a trace activity to every operation action. The trace activity is a crosscutting concern that is hard to modularize into a single activity or action unit using existing activity modeling techniques. Such a concern can be extremely difficult to comprehend and change due to its scattering nature.

The application of aspect-oriented approaches [1][2] to activity modeling provides a solution to support this kind of modularization by encapsulating crosscutting concerns in a specialized unit called *aspect*. Following the aspect-oriented programming (AOP) [3][13] terminology, two fundamental constructs are involved in an aspect model.

First, we need to specify “where” (i.e., the locations, or *join points*) in the models the crosscutting behavior emerges. Based on the activity metamodel definition, which defines an activity as being composed of a sequence of actions, join points refer to various kinds of actions that are allowed in activity modeling. A group of particular join points are represented in a special construct called *pointcut*, which defines a pattern to identify matching join points.

Second, we need to specify “what” (i.e., the behavior) makes up the crosscutting concern. In activity modeling, the concern behavior is implemented using an activity model

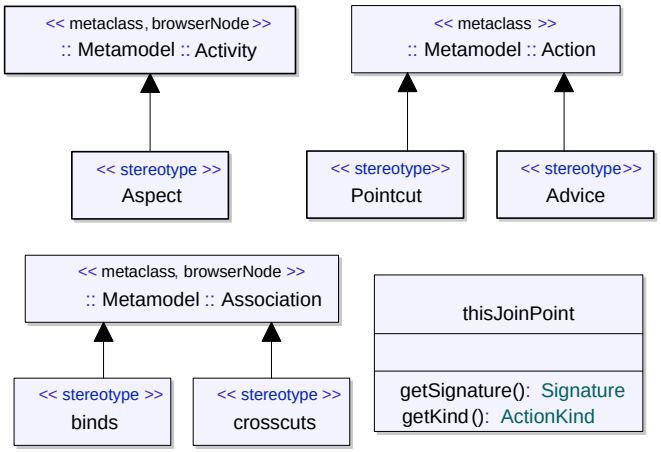

Figure 3. Aspect-Activity Modeling Profile

referenced by a special action called *advice*. An advice may contain a *proceed* operation action that refers to the current join point action. An advice can also obtain the join point information through a set of predefined reflective APIs.

These aspect-activity modeling concepts are defined upon a light-weight extension of UML through profiles and stereotypes [6]. As shown in Figure 3, an aspect is a special activity that encapsulates a crosscutting concern. Pointcuts and advice are denoted as special actions. An aspect-activity model contains a binding diagram that defines which advice is bound to which pointcuts. Those bindings are realized by a stereotype named *binds*. Aspects are deployed to the base activity models through a special association stereotyped by the name *crosscuts*. The *thisJoinPoint* class defines a set of APIs that are used to retrieve the reflective information of the matched join points (e.g., the signature of an operation action, or the kind of the join point action).

As an illustrative example, Figure 4 specifies a trace aspect model, with an aspect called `Aspect_Trace` applied to a base activity model. The purpose is to keep track of certain actions involved throughout the execution of the base activity flow. This aspect contains one advice that is bound to four different pointcuts. The pointcut `Cancel_*` denotes all of the actions whose name starts with `Cancel_` (e.g., the `Cancel_Order` action in Figure 2). The underneath pattern matching is based on the regular expression mapping against the pointcut name. The pointcut `Op_Process_Order` refers to an operation action that has one parameter of the type `Order`. `Request_Payment` and `Cancel_Order_Request` match to a send signal action and an accept event action individually. The advice action `Trace` is implemented by an activity model, which extends the original join point action (denoted by `proceed`) with a log action that stores the join point information to an external file. An aspect model can also introduce inter-type members [3] that are to be inserted to the join point action implementations (e.g., the integer `flag` declared in `Aspect_Trace`).

The aspect and the base models are automatically composed together through a specialized aspect weaver for activity models, as indicated in Figure 5. The weaving procedure starts with instantiating advices based on the pointcuts they are bound to. All of the calls to the reflective API are resolved based on the current join point. The `proceed` actions are replaced by the original join point

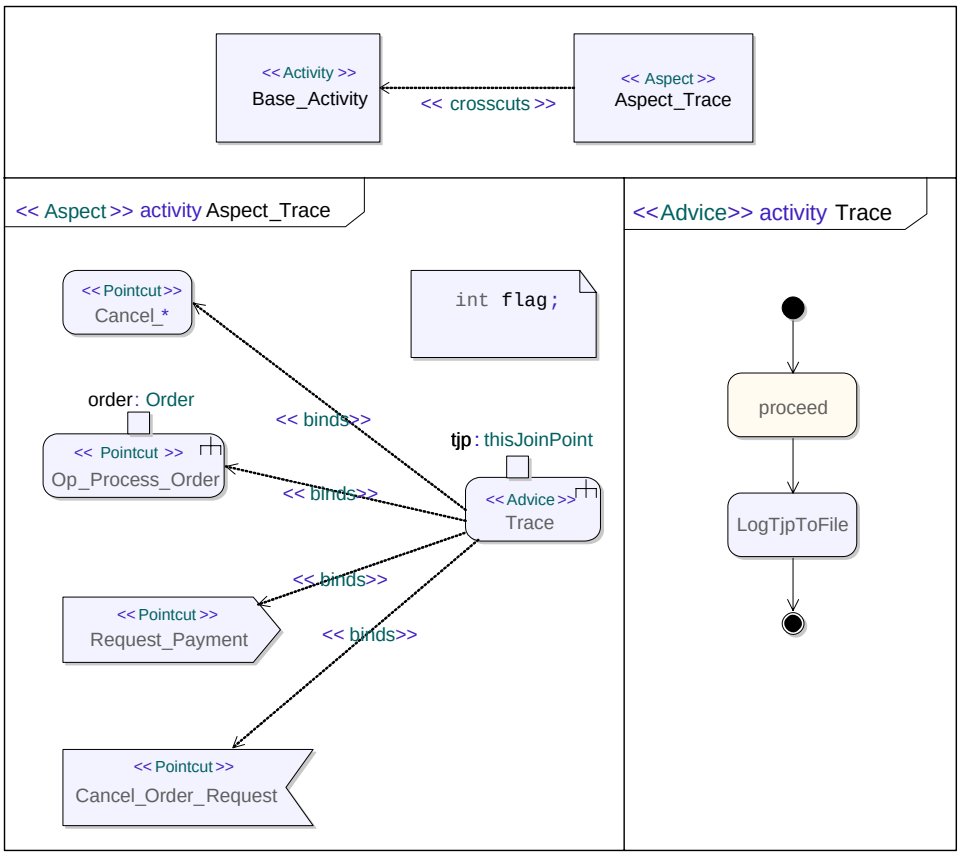

Figure 4. Trace Aspect, Pointcuts and Advice

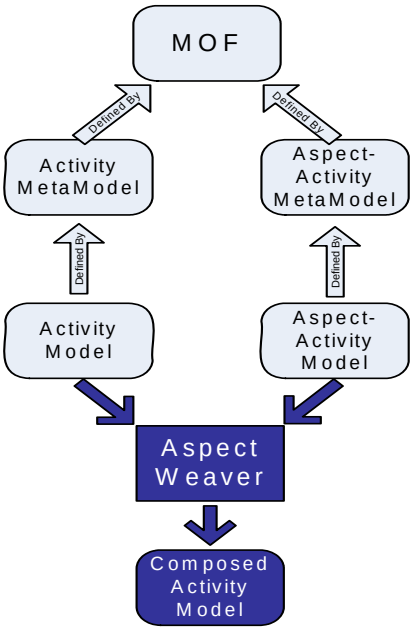

Figure 5. Aspect Weaving on Activity Models

action. These advice instances are in turn woven into the base models in one of the following two ways: wrapping or inlining. In the wrapping mode, the original join point action is replaced by an operation invocation to the corresponding advice instance. For the inlining version, the contents of all the advice instances are directly embedded into the base activity models. The aspect-activity weaving process conforms to the one developed for aspect-statecharts in the current Motorola aspect weaver. For more details about the Motorola aspect weaver, please refer to [10][18].

IV. Case Study: Applying Timeout Handler to a Network Fault Management System

In order to illustrate the proposed approach, this section provides a case study on applying a timeout handler aspect to a real world network fault management system using aspect-oriented activity models.

A. Background

The Intelligent Network Fault Management (INFM) [14] system is being developed within Motorola for providing solutions to manage faults in a CDMA cellular network. One of the most important features of a fault management system is alarm correlation, which provides functionalities to filter out informational alarms, report meaningful alarms that are regarded as actionable or as requiring operator attention and provide assistance in troubleshooting. Network operators rely on the alarm correlation feature to reduce the number of alarms to a limited number that could be handled within the required time constraints. The significant reduction, usually greater than 80% on average, is achieved by correlating the alarms using patterns and hidden correlations discovered by machine learning algorithms.

INFM carries out the alarm correlation by applying frequent pattern discovery algorithms, which use certain parameters to control how candidate patterns are constructed from the learning data (i.e., alarm instances). When the number of alarm instances is fairly large, the time complexity of the algorithm increases dramatically, which causes a violation of real-time constraints and fails to provide prompt correlation operation. This motivates us to add a timeout handling capability to resolve such failures and maintain the healthy state of the fault management system.

Figure 6 illustrates a fragment of the simplified activity model for specifying the flow of the alarm correlation process logic in the INFM system. The control flow starts with configuring the data source for streaming event data from either a database server or an FTP site. Once the connection is established, the process then initiates the parameters for the pattern discovery algorithm. After the algorithm execution is completed, a special operation will be invoked to process the patterns that are generated.

B. Modeling Timeout Handler Activity Aspect

Timeout is among the most common software failures that can occur in almost every operation or service invocation. The timing failure is usually associated with certain time constraints, which can be a real time constraint or a relative deadline with respect to certain events. For example, a time limit can be imposed on the alarm correlation process in a fault management system, as opposed to the relative deadline which states that “the alarm correlation must be completed before the next batch of alarms is received.”

Figure 7 shows an aspect-activity model for managing the timeout failure for the INFM alarm correlation system. The `proceed` action refers to an operation that has sensitive timing concerns and needs to be analyzed upon the timeout failure. The aspect model intercepts and wraps this action

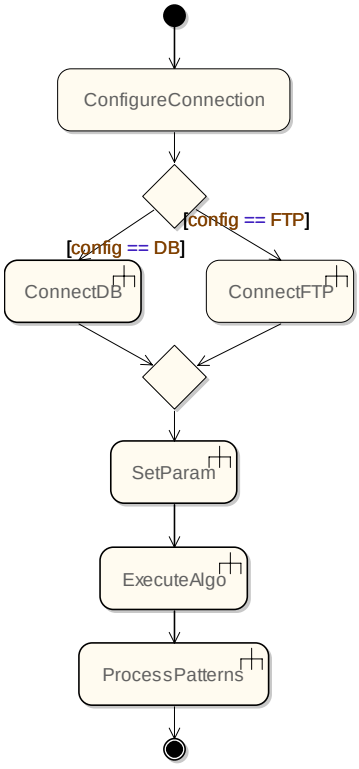

Figure 6. A Simplified Alarm Correlation Activity Model

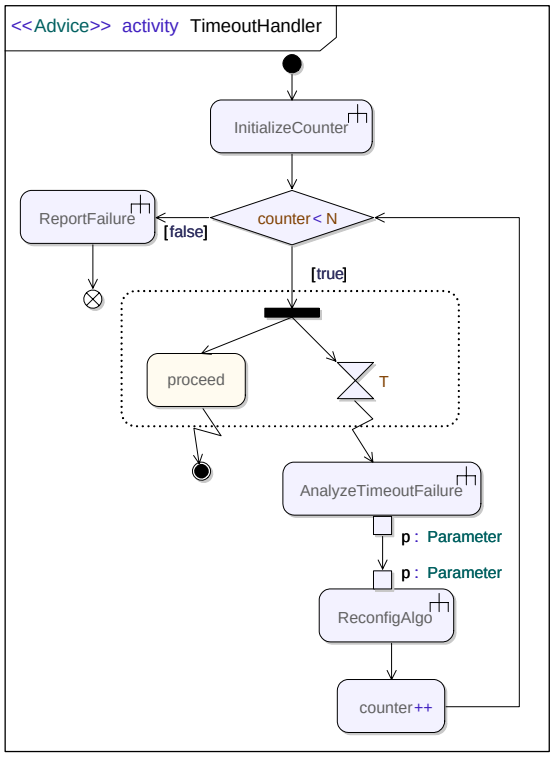

Figure 7. A Timeout Handler Aspect

with a sequence of failure management activities. The aspect process first initiates the counter for the allowed number of iterations. If the counter already exceeds the allowed number of iterations, it means that the failure cannot be resolved and has to be reported. The whole application thus must be aborted. Otherwise, the process will start a timer with a value `T`, in sync with the execution of the `proceed` action. The `proceed` action and the timer are surrounded by an interruptible activity region (denoted as a dashed rectangle with rounded corners), representing that whenever the flow leaves the region via interrupting edges, all of the activities in the region will be terminated. Specifically, if the `proceed` action completes execution successfully before it runs out of time, the control of the flow will return to the base process (via a bull's eye symbol) and continue with the next activity that follows the `proceed` action. Otherwise, the `proceed` process will be shut down properly and a timeout failure will be captured and passed to the failure analyzer and mitigator, which is responsible for determining the failure risk and calculating the corresponding mitigation strategies for reconfiguring algorithm parameters. The control loop for handling the timeout failure is thus realized by re-running the algorithm with the new parameter values.

C. *Deploying the Timeout Handler Aspect on Base Model*

The timout handler aspect is deployed to the base models of the INFM alarm correlation activity. In this particular case study, the timeout handler aspect is applied to the `ExecuteAlgo` action. The base models and the aspect models are then integrated through an underlying aspect-activity weaver. The woven model resulting from the inlining mode weaving is shown in Figure 8. The `proceed` action in the aspect specification is replaced by the `ExecuteAlgo` action. The initial and return symbol of the aspect models are connected to the pre- and post- flow of the `ExecuteAlgo` action, respectively. By adopting the aspect-oriented approach to specifying activities, the aspect models (usually non-functional activities) are defined independently from the base functionality. Such kind of separation of concerns greatly improves the reusability, changeability, and maintainability of the system.

V. Related Work

Although the aspect-orientated application originated at the programming language level [13], it now extends to other software lifecycle stages and is applied to different levels of software abstractions. For example, there is a growing community investigating Aspect-Oriented Modeling (AOM) [1] techniques, providing various concepts, notations and mechanisms to handle crosscutting concerns at the modeling level. This section summarizes some of the existing research that relates to aspect and activity models.

Barros et al. [7] propose a graphical composition operation supporting the addition of crosscutting requirements in activity models through node fusion, addition, and subtraction. In contrast to our current implementation of the aspect-activity extension, their approach considers all types of activity nodes as potential join points (i.e., not only action nodes, but also object and control nodes). However, as their approach is based on pure graph composition theory, it lacks semantic support for non-graphical activity node

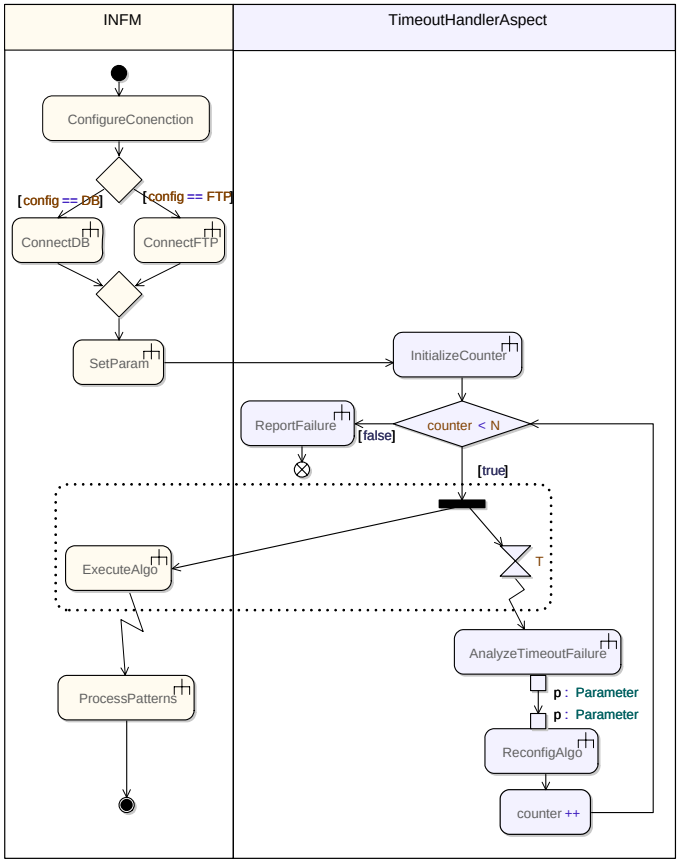

Figure 8. Integrating the Timeout Handler Aspect with the INFM Alarm Correlation Base Model

specification, such as reflective APIs and regular expression-based operation pattern matching.

Charfi et al. [9] introduce an aspect-oriented extension to Business Process Execution Language for Web Services (BPEL4WS) – a variation and application of activity modeling language. With web service composition captured in aspects, dynamic adaptation of composition logic can be supported. In their model, each BPEL activity is a possible join point during the execution of processes. The aspect language proposed by them is similar to ours except that they use XPath (a query language for XML documents) as the pointcut designator language, whereas our aspect-activity models are based on MOF/UML specification, which is more generic and can be applied to any activity modeling based on UML.

Solberg et al. [17] present a Model-Driven Development (MDD) framework that uses aspect-orientation to facilitate separation of concerns. The primary and aspect models defined in a platform-independent manner are transformed to platform-specific models through separate mappings. The resulting models are in turn composed to obtain an integrated design view. Unlike our approach, they don't provide an explicit support for modeling aspect constructs. Instead, the weaving mechanism is controlled by using extra composition directives that instantiate aspect models and bind them to the primary models.

Grassi et al. [11] propose a UML-based graphical notation for specifying aspects for static and dynamic structure of the system model. Similarly, Cazzola et al. [8] present a high-level join point selection mechanism, which decouples the aspect definition from the base program structure and syntax. Both approaches focus on using activity models to represent pointcut patterns, rather than provide aspect support to activity models.

VI. Conclusions

One of the ultimate goals of software engineering is to construct software that is easily modified and extended. A desired result is to achieve modularization such that a change in a design decision is centralized to one location [15]. Our experience has led us to believe that aspect-oriented software development is a promising approach to support such modularization. Furthermore, aspect-orientation can be beneficial at various levels of abstraction and at different stages of the software lifecycle.

This paper applies an aspect-oriented approach to supporting separation of crosscutting concerns in activity modeling. Aspect-specific constructs have been introduced as an extension to the activity models. The current implementation of the pointcut specification only allows join point to be referred to action nodes. The future work will cover other kinds of activity nodes and investigate more advanced pointcut selection patterns.

References

[1] AOM Website: http://www.aspect-modeling.org/
[2] AOSD Website: http://www.aosd.net/
[3] AspectJ Website: http://www.eclipse.org/aspectj/
[4] MOF Core Specification, v2.0, *Object Management Group* (http://www.omg.org/cgi-bin/doc?formal/2006-01-01).
[5] UML Specification v1.5, *Object Management Group* (http://www.omg.org/cgi-bin/doc?formal/03-03-01).
[6] UML 2.1.1 Superstructure Specification, *Object Management Group* (http://www.omg.org/cgi-bin/doc?formal/07-02-03).
[7] Joao Paulo Barros and Luis Gomes, "Towards the Support for Crosscutting Concerns in Activity Diagrams: a Graphical Approach," In the *Fourth Workshop on Aspect-Oriented Modeling in the Sixth International Conference on the Unified Modeling Language (UML'03)*, San Francisco, CA, October 2003.
[8] Walter Cazzola and Sonia Pini, "Join Point Patterns: a High Level Join Point Selection Mechanism," *9th International Workshop on Aspect-Oriented Modeling*, Genoa, Italy, October 2006.
[9] Anis Charfi and Mira Mezini, "Aspect-Oriented Web Service Composition with AO4BPEL," In *Proceedings of the European Conference on Web Services (ECOWS'04)*, LNCS Volume 3250, Erfurt, Germany, September 2004, pp. 168-182.
[10] Thomas Cottenier, Aswin van den Berg, and Tzilla Elrad, "Joinpoint Inference from Behavioral Specification to Implementation," In *Proceedings of the 21st European Conerence on Object-Oriented Programming (ECOOP'07)*, LNCS Volume 4609, Berlin, Germany, July 2007, pp. 476-500.
[11] Vincenzo Grassi and Andrea Sindico, "UML Modeling of Static and Dynamic Aspects," *9th International Workshop on Aspect-Oriented Modeling*, Genoa, Italy, October 2006.
[12] Jeff Gray, Yuehua Lin, and Jing Zhang, "Automating Change Evolution in Model-Driven Engineering," *IEEE Computer*, Vol. 39, No. 2, February 2006, pp. 51-58.
[13] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin, "Aspect-Oriented Programming," In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'97)*, Volume 1241, Springer-Verlag, Jyväskylä, Finland, June 1997, pp. 220-242.
[14] Yan Liu, Jing Zhang, Michael Jiang, David Raymer and John Strassner, "A Model-based Approach to Adding Autonomic Capabilities to Network Fault Management System," In *Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS'08)*, Salvador, Brazil, April 2008.
[15] David Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules," *Communications of the ACM*, December 1972, pp. 1053-1058.
[16] James L. Peterson, "Petri Nets," *ACM Computing Surveys*, September 1977, pp. 223-252.
[17] Arnor Solberg, Devon Simmonds, Raghu Reddy, Sudipto Ghosh, and Robert France, "Using Aspect Oriented Techniques to Support Separation of Concerns in Model Driven Development," In *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, Edinburgh, Scotland, UK, July 2005, pp. 121-126.
[18] Jing Zhang, Thomas Cottenier, Aswin van den Berg, and Jeff Gray, "Aspect Composition in the Motorola Aspect-Oriented Modeling Weaver," in *Journal of Object Technology - Special Issue Aspect-Oriented Modeling*, Volume 6, Number 7, August 2007, pp. 89-108.