# AN EVENT PROCESSING SYSTEM FOR RULE-BASED COMPONENT INTEGRATION

Susan D. Urban, Sunitha Kambhampati, Suzanne W. Dietrich, Ying Jin, Amy Sundermier
*Department of Computer Science and Engineering*
*Arizona State University*
*Tempe, Arizona USA*

Keywords: active rules, event specification, event processing, component integration

Abstract: The Integration Rules (IRules) project has developed an environment in which active rules, known as *integration rules*, are used together with transactions to provide an event-driven, rule-based approach to the integration of black-box components. This paper presents the event processing system that supports the use of integration rules over components. The event processing system is composed of the language framework for the specification of different types of events, an event generation system for generating event instances, and an event handler for communicating the occurrence of events to the integration rule processor. The language framework supports the enhancement of EJB components with events that are generated before and after the execution of methods on components. Since integration rules support an immediate coupling mode and execute in the context of nested transactions, a synchronization algorithm has been developed to coordinate the execution of immediate integration rules with the execution of methods on components. The synchronization algorithm makes it possible to suspend and resume distributed application transactions to accommodate the nested execution of integration rules with an immediate coupling mode.

## 1 INTRODUCTION

Distributed component technology standards address the need for well-known interfaces and middleware for discovering and integrating software components into applications (COM/DCOM, 2003; OMG, 1998; WSA, 2002; J2EE, 2003). Implementing distributed applications when components reside on multiple servers, however, can be a difficult task that often results in procedural, hard-coded solutions. The application integrator must not only understand the semantics for mediating the interactions between the components, but must also be skilled in the technical details associated with distributed event and transaction processing techniques.

The Integration Rules (IRules) project at Arizona State University has responded to the need for more flexible and declarative middleware technology by applying active rule processing technology to the area of software component integration (Urban et al., 2001). Active database technology enhances traditional database technology with the ability to monitor and react to circumstances that are relevant to an application (Widom and Ceri, 1996). The IRules project is extending the use of active rules to distributed components, providing a middle-tier, rule processing framework for the integration of distributed black-box software components. The current prototype implementation supports the Enterprise JavaBeans (EJB) (EJB, 2001) component model, with future plans to include support for other component models.

The IRules project provides an environment in which active rules, known as *integration rules*, are used together with transactions to provide an event-driven, rule-based approach to component integration. When an event is generated in the IRules environment, an integration rule is triggered to test a condition expressed as a query over distributed components. A successful condition evaluation may invoke additional methods on components in the action part of an integration rule. Integration rules therefore provide a declarative and distributed rule processing mechanism for responding to events in a distributed environment, thereby separating the logic associated with reactive behavior from the main



D. Urban S., Kambhampati S., W. Dietrich S., Jin Y. and Sundermier A. (2004).
AN EVENT PROCESSING SYSTEM FOR RULE-BASED COMPONENT INTEGRATION.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 312-319
DOI: 10.5220/0002619803120319

application logic expressed within distributed transactions.

One of the challenges of developing the IRules environment has been the design and implementation of the event processing system that supports the execution of integration rules (Kamphampati, 2003). The event processing system is composed of a language framework for the specification of several different types of events, an event generation system for generating instances of each event type, and an event handler for communicating events to the integration rule processor. The IRules language allows for the definition of events that are detected before and after the execution of methods on components and the execution of distributed application transactions. The language framework also supports the specification of events that are generated and published by the internal logic of black-box components as well as events that are generated by sources external to IRules.

To coordinate events with transactional behavior in the IRules environment, we have developed wrappers that serve as proxies for EJB components (Patil, 2003). All method calls are routed through the wrappers. The wrappers are responsible for generating events before and after method execution. The wrappers are also capable of suspending methods to support the nested execution of triggered rules with an immediate coupling mode. A synchronization algorithm uses semaphores implemented with JavaSpaces (Freeman et al., 1999) operations as a distributed communication mechanism. The synchronization algorithm coordinates the execution of application transactions with the nested execution of integration rules before and after the execution of methods on EJB components.

Our research provides an original approach to enhancing black-box components with event generation techniques, addressing issues for control of transactional scope, event handling, and suspension and resumption of activities in component-based integration architectures. These issues are described as future directions within the Business Process Execution Language specification (BPEL4WS, 2002), a standard choreography language for integrating Web Services. The techniques described within this paper, although currently implemented using Java technology, are applicable to other component models and integration languages as well.

The remainder of this paper is structured as follows. Section 2 provides an overview of the IRules environment. Section 3 presents the language framework for the specification of IRules events. The event handling component of the IRules environment is presented in Section 4. Section 5 provides an overview of the synchronization algorithm for coordinating method execution with rule execution. Our work is discussed in the context of related work in Section 6. The paper concludes with a summary in Section 7.

# 2 OVERVIEW OF IRULES

Figure 1 provides a high-level view of the functionality of the IRules environment. As shown in Figure 1, EJB components are organized into distributed containers, where each component was purchased or created independently of components in other containers. This specific example illustrates an Investment application from (Urban et al., 2001), consisting of four different containers with purchased software components: a Portfolio container, a Pending Order container, a Stock container, and an Account container.

Although the containers in Figure 1 are independent black-box containers that know nothing about the existence of the other containers, the IRules semantic framework supports the definition of *externalized relationships* (Rumbaugh, 1987) between such containers. These relationships are shown in Figure 1 by lines between components in different containers. For example, the externalized relationship between Stock and Pending Order represents the type of stock that is associated with a pending buy or sell order. Externalized relationships are managed by IRules wrappers around existing EJB components. In addition to externalized relationships, IRules wrappers add other functionality to EJB components, such as extents, derived attributes, stored attributes, and events (Dietrich et al. 2001).

Application logic is captured through the use of application transactions together with integration rules. Integration rules have been derived from active rules in active database technology (Widom and Ceri, 1996), providing an Event-Condition-Action (ECA) rule form as well as an Event-Action (EA) rule form for responding to events. As shown in Figure 1, when an application transaction executes a method such as Operation k on an EJB component, an event can be generated that triggers the execution of integration rules. These events are referred to as *method events* and can be generated before or after the execution of a method on a component. Integration rules may also be triggered by *internal events*, *application transaction events*, or *external events*. Internal events are events that were defined within a black-box component before the component joined the IRules environment.

Application transaction events are events that can be generated before or after the execution of an application transaction. External events are events that are generated from sources outside of the IRules environment.

The structure of an integration rule is also shown in Figure 1. When an integration rule is triggered by an event, the rule may execute a when condition that is expressed as a simple query over the event object and its parameters. If the condition evaluation is true, then an OQL-based (Cattell and Barry, 2000) define statement allows for the named specification of bindings using a query over the distributed components of the environment. The action of the rule may refer to these bindings when invoking a method of a component or an application transaction.

The rule-based approach to component integration within the IRules environment is expressed using the IRules Definition Language (IRDL) (Dietrich et al., 2001; Urban et al. 2001). IRDL consists of four sub-languages: the Component Definition Language (CDL) for describing components and defining the method and internal events associated with that component, the IRules Scripting Language (ISL) for describing application transactions, the Event Definition Language (EDL) for defining external and application transaction events, and the Integration Rule Language (IRL) for defining integration rules that respond to events. In the current version of IRDL, ISL is based on the JACL (DeJong and Laird, 2003) scripting language, which is a Java implementation of the Tool Command Language (TCL) (Ousterhout, 1994).

The IRules execution environment is a Jini-based architecture that supports the execution of events, rules, and application transactions over distributed EJB components (Jin et al., 2002). Architectural components include the rule manager for controlling the execution of rules, the transaction manager for the management of transactions, the object manager for providing abstract access to the underlying components of the system, the metadata manager for storing the integration semantics as extracted from the compilation of an IRDL schema, and the event handler for processing events. The Java Messaging Service (JMS) (JMS, 1999) provides asynchronous event notification.

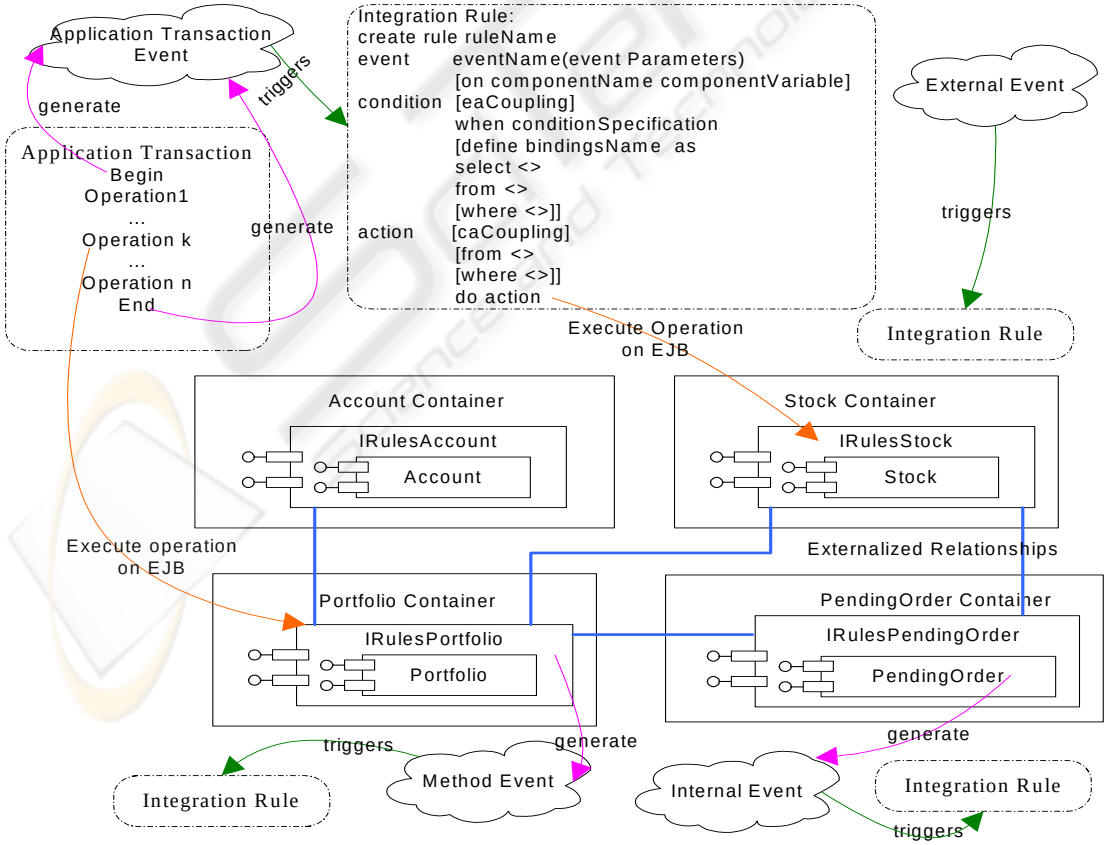


Figure 1: Purchased Components

# 3 IRULES EVENT DEFINITION

The subsections that follow illustrate the specification of each type of event supported in the IRules environment.

## 3.1 Event Specification in CDL

Method and internal events are expressed using CDL since they are associated with component definitions. The syntax of CDL is based on the syntax of the ODMG Object Definition Language (ODL) (Cattell and Barry, 2000). Using the Investment example introduced in Section 2, Figure 2 shows an example of describing a portion of the Stock component to the IRules environment, which includes the definition of a method event. The event specification is composed of an event name and a list of parameters, followed by curly braces that enclose the definition of the details of the event. The method keyword defines the type of the event, while the before keyword is an event modifier, indicating that the event is raised before the execution of the method. The keyword after can be used to define an event raised after the execution of a method. The name and type of the method parameters are also specified. The names of the method parameters in CDL allow for naming the attributes that are to be propagated as parameters to the IRules event.

```
component Stock implements EntityBean
(extent    stocks)
{ ……..
  event     beforeSetPrice(newPrice)
            {method before  setPrice(double newPrice);}}
```

Figure 2. Example of a Method Event

Figure 3 presents the stockBuy rule as an example of an integration rule that is triggered in response to the beforeSetPrice method event. The condition part of the rule includes an OQL-like query expressed over the distributed components of the environment. The when condition checks whether the new price of the stock is less than the current stock price. The action part of the rule does a buy transaction for the stocks that are currently pending.

When an EJB component joins the IRules environment, it is possible that the black-box component is capable of generating and publishing events as part of its internal logic. The IRules environment refers to these events as internal events and allows the use of internal events in the integration process. Since internal events are associated with components, these events are defined as part of CDL.

```
create rule stockBuy
event       beforeSetPrice(NewPrice)
            on stock S
condition   immediate
            when NewPrice > S.price
action      immediate
            from Pn in S.pendingTrades
            where S.price<=Pn.desiredPrice and
                                  Pn.pnaction="buy"
            do buyStock(S,Pn)
```

Figure 3. Example of an Integration Rule Triggered by a Method Event

As an example of an internal event, Figure 4 presents the specification of the afterUpdateCash event within the StockBroker_PastHolding component of the Portfolio container. This is an event that is generated as a result of the execution of an update operation on the cash value of a portfolio. Internal events are always after events, since the IRules environment has no control over the execution of the internal logic of a black-box component. As a result, an event modifier is never specified for internal events. The event handler currently supports internal events published by EJB components using the JMS system. Internal events must specify the proprietary name and message type of the event that accompanies the JMS message. In Figure 4, the updateCashEvent token is the name of the event that is sent to the appropriate JMS topic (i.e., a data structure for collecting JMS messages) by the black-box component. The map token represents the JMS map message type with its corresponding parameters.

```
component     StockBroker_PastHolding     implements
EntityBean
(extent       pastHoldings)
{relationship Stock pastStock
              inverse  Stock::soldBy;
 event        afterUpdateCash (portfolioID,  cashValue)
              {internal  map(int portfolioID, float
                        cashValue) updatecashEvent};};
```

Figure 4. Example of an Internal Event

## 3.2 Event Specification in EDL

Application transaction events and external events are defined using EDL. Since application transactions in IRules are defined using ISL, Figure 5 provides an example of the sellStockOnNewPO ISL transaction for selling stock. The code in the body of the transaction uses JACL syntax. The newinstance

command is a JACL extension command used to create a new instance of a PortfolioSessionBean. The sellStock method is then executed on PortfolioSessionBean and the PendingOrderComponent status is set to 'executed'. The last step in the body does the task of printing the status of the pending order by executing printSellInfo extension command.

```
application transaction
sellStockOnNewPO(string stockId, float price, string
          portfolioId, int numOfShares, PendingOrder pn)
tcl newInstance,printSellInfo
{ set session [newInstance PortfolioSessionBean]
  $session sellStock $stockId $price $portfolioId
                        $numOfShares
  $pn setStatus "executed"
  printSellInfo $pn}
```

Figure 5. Example of an Application Transaction in ISL

Figure 6 illustrates the specification of an application transaction event, named afterSellStockOnNewPO, that is generated after the execution of the sellStockOnNewPO application transaction specified in Figure 5.

```
event afterSellStockOnNewPO(stockId, price, portfolioId,
                                        numOfShares, pn)
{ appTrans after sellStockOnNewPO(String stockId, float
        price, String portfolioId, int numOfShares,
        PendingOrder pn);}
```

Figure 6. Example of an Application Transaction Event

Since it is likely that the application integration process may need to monitor events from external sources, the IRules environment supports the definition and use of external events. We assume that external events are generated using JMS. External events are defined in the same manner as internal events and must therefore specify the proprietary name and message type of the event that accompanies the JMS message. There is no modifier explicitly stated with external events since they are inherently after events.

Figure 7 gives an example of an external event in the context of the Investment application that monitors changes in stock prices from the stock market. The afterStockMarketQuote event is defined as a JMS map message type with an originating event name of TickerEvent. Within the Investment application, the occurrence of this event can be used to update local information about stock prices.

```
event afterStockMarketQuote(stockId, newPrice)
{external  map(String stockId, double newPrice)
            TickerEvent;}
```

Figure 7. Example of an External Event

# 4 THE EVENT HANDLER

The IRules framework uses the publish-subscribe approach of JMS to support asynchronous message passing during rule execution. Method and application transaction events are sent to a pre-defined IRules Event Topic. Internal and external events are sent to a proprietary JMS topic that must be made known to the IRules environment. The IRules Event Handler listens to the relevant topics and communicates event information to the rule processor using a push strategy.

As shown in the Figure 8, method events are generated by IRules wrappers. The execution of an application transaction or the action part of a rule can invoke method calls to the black-box components. The wrapper intercepts any call to the EJB instance of the black-box component. The wrapper is responsible for generating method events and for invoking the method on the component instance. Event parameters for a given method event are packaged into a generic IRules data structure called EventMsg. The EventMsg object is then published to the IRules Event Topic. The transaction manager is responsible for generating application transaction events. Similar to method events, application transaction events are also packaged into the IRules EventMsg data structure and published to the IRules Event Topic.

For external and internal events, since the messages are in their proprietary JMS format, the message has to be extracted, interpreted, and translated into the IRules EventMsg format. The event handler initiates the necessary listeners or subscribers for all of the topics to which internal and external events are being published. When an internal or an external event is received, the JMS message is extracted and the necessary parameters are interpreted from the information available in the IRules metadata repository. The interpretation and extraction process depends on the type of the message that is received at the event handler.

After the interpretation and extraction process, the EventMsg instance is propagated to the rule manager. As a result, all of the different types of events ultimately share the same event structure. The rule processor then executes the triggered rules for a newly detected event depending on the semantics of the coupling modes associated with the rules.

# 5 SYNCHRONIZATION ISSUES

Integration rules are processed based on coupling modes as defined in traditional active database systems. Coupling modes define transactional

behavior for rule execution. The IRules system supports four coupling modes (Jin et al., 2002): *immediate synchronous* (the condition of a rule must be executed as a nested transaction immediately after an event is raised), *immediate asynchronous* (same as immediate synchronous except that the rule and triggering transaction are processed concurrently), *deferred* (condition evaluation occurs at the end of the outermost transaction in which a rule is triggered), and *decoupled* (a new top-level transaction is created for processing a rule). The coupling mode of interest in this paper is the immediate synchronous coupling mode, which results in a nested execution structure where the triggering transaction suspends while the triggered rule executes as a sub-transaction. IRules must suspend the transaction that executes the triggering event and release the suspended transaction when rule execution completes.

Consider the case of method events. An IRules wrapper wraps each black-box instance and acts as a proxy to the black-box. Since the EJB component model does not allow threading and does not support primitives for a blocking mechanism, there is no built-in mechanism for suspending method execution to allow for rule execution. A synchronization process has been implemented as part of this research to coordinate the execution of methods on components with the execution of rules by the rule processor. Since any method of a component executes in the context of a transaction within IRules, the transaction identifier that represents the IRules transactional context is used to help generate a semaphore for the synchronization process. The transaction identifier alone, however, is not sufficient for the creation of a semaphore since the JACL interpreter for executing ISL transactions allows multi-threading. As a result, two or more methods on components within one transaction can be called concurrently by the JACL interpreter.

We associate a counter, initialized to zero, with the transaction identifier to create a ResourceSemaphore object as a pair ($t_i$, $c_{ij}$), where $t_i$ is the transaction identifier and $c_{ij}$ is a counter for the execution of methods within $t_i$. The ResourceSemaphore ($t_i$, $c_{ij}$) is used to provide a unique event identifier, known as the EventId, across all of the distributed components that execute within the IRules environment. The EventId is used to coordinate the suspension of a wrapper for a specific method execution with the execution of immediate rules in response to before and after events.

JavaSpaces has been used to implement the synchronization process. A **take** operation provides a way to remove an object from the JavaSpaces persistent store. The **take** operation is a blocking operation that will wait until an object matching a template is found. A ResourceSemaphore with a

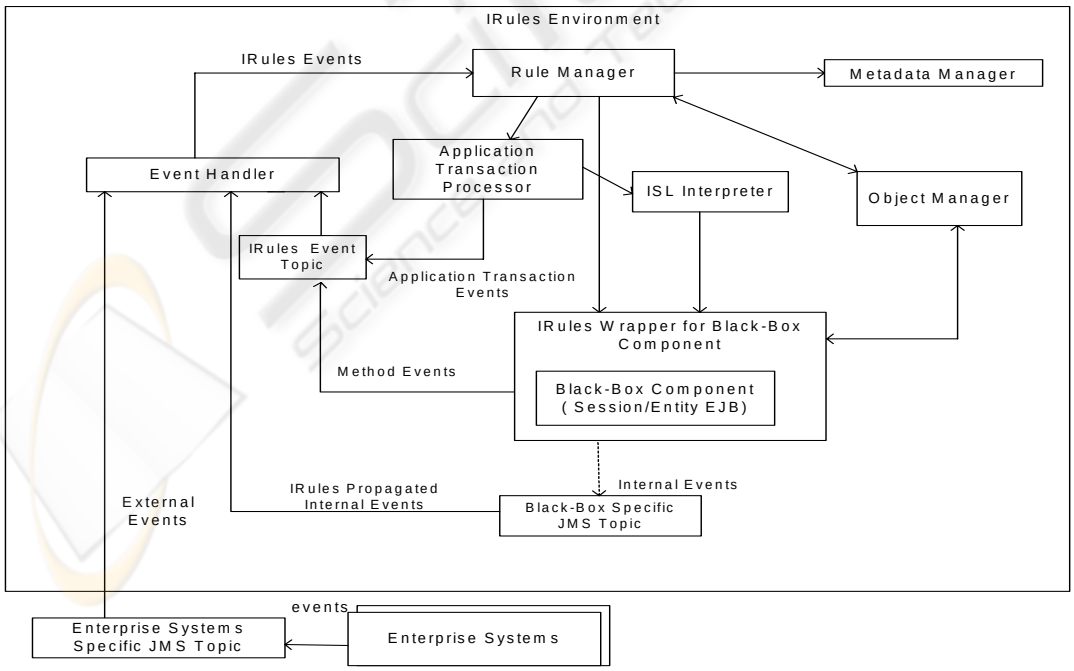


Figure 8: Events in the IRules Environment

transaction identifier and a counter initialized to zero is placed into JavaSpaces at the start of a transaction. The wrapper generating the before event must use a **take** operation to access the ResourceSemaphore and to create a unique EventId object. When the ResourceSemaphore object is retrieved, the counter of the object is incremented and the modified ResourceSemaphore is written back to the JavaSpace to prevent unnecessary waiting for the concurrent execution of multiple methods within the same transaction. Since the ResourceSemaphore object is unique for each transaction and method within a transaction, deadlock cannot occur.

The EventId object is used as the synchronization object for the suspension and release of the wrapper execution. The EventId is packaged along with an EventMsg and sent to the IRules Event Topic. The event handler detects the event and propagates the information to the rule processor. The rule processor executes the immediate rules for the particular event. When the rule execution is completed, the rule processor notifies the wrapper that it can continue by writing the EventId back into the JavaSpace. The wrapper, which has been waiting during the rule execution by executing a **take** operation on the EventId, succeeds in retrieving the EventId and returns from the blocking operation. The synchronization process for after events is similar to before event processing, with the full details of the algorithm in (Kambhampati, 2003).

We have tested the event service and rule execution modules using the synchronization process for the investment example. We have also evaluated the performance of event detection for the four different types of IRules events. Detection time for application transaction and method events is measured as the duration from event occurrence to the time that the rule manager receives the event. Method event detection is more time consuming than application transaction event detection. Both types of event detection must query the metadata to determine the need to generate before and after events. Metadata access from component wrappers, which are also implemented as EJB components, is more time consuming than Java method calls to the metadata in the detection of application transaction events. The detection of internal and external events is measured differently from that of method and transaction events. The detection time of internal and external events is the duration required to wrap an event and push the event to the rule manager once the event handler gets events from JMS. The detection of internal and external events varies according to the data type of the message. Detection for the map type, for example, is slower than the stream data type.

# 6 RELATED WORK

Several event processing/distributed rule systems are related to our research. In (Carzaniga, 2001), the authors describe a general-purpose, Internet-scale notification service called SIENA. Unlike SIENA, the IRules project concentrates on integration of transactional context along with events to develop distributed applications. JEDI (Cugola et al., 2001) supports an object-oriented infrastructure for the development and operation of event-based systems, also based on active rules. IRules differs from JEDI by coordinating the generated events with application transactions, thus allowing for seamless integration instead of intermixing reactions to events with application logic.

In (Collett et al., 1998), the authors introduce dimensions to characterize event definition, detection, production, and notification for event services in active database systems, where the producers and consumers communicate via the CORBA push and pull protocols. There is no mention of a rule execution service for active rules as in the work presented in this paper.

In (Liebzig et al., 2000), the authors present a CORBA-based transaction and notification service $X^2TS$ as part of a larger project for unbundling the concepts of active object systems. In (Cilia et al., 2001), the authors propose an architecture to move active functionality from centralized to open distributed heterogeneous environments using the results of $X^2TS$ to allow for different coupling modes. Unlike IRules, $X^2TS$ requires event-consuming active objects to pre-register as a group with the triggering transaction so that a call-back mechanism can be invoked by consumers to release the event producer in support of an immediate coupling mode. The authors state the desire to explore a declarative means to introduce the concept of active objects into the CORBA object model.

The research in (Benatallah et al., 2002) has developed an approach known as SELF-SERV (composing wEb accessible information & business sERVices) for declarative composition of dynamic web services in a peer-to-peer architecture. ECA rules are used in SELF-SERV to control state transitions. In IRules, integration rules play a more direct role in the design of the application integration logic.

# 7 SUMMARY

This paper has presented an original approach to enhancing black-box components with the specification and generation of events, including a

synchronization algorithm for coordinating method execution with the nested execution of integration rules. In addition to the implementation of the event processing system, we have implemented a query processor for IRules based on the monoid algebra of (Fegaras and Maier, 1995), and a rule processor that uses the flexible transaction model for the nested execution of rules (Jin et al., 2002). Full development of compensating transactions for dealing with failure in the rule execution process is an area for future research. Other directions for future research include extending the environment to include different component models, different messaging services, and composite events. We have already developed a prototype implementation of the IRules environment using Web Services (WSA, 2002) together with the synchronization algorithm described in this paper. We are also redesigning the environment for use with Grid services (Foster, 2002).

## REFERENCES


Benatallah, B, Dumas, M., Sheng, Q., and Ngu, A., 2002. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services, *Proc. of the 18th Inl. Conf. on Data Eng.*, San Jose, CA, 297-308.

BPEL4WS, 2002. *Business Process Execution Language for Web Services*. Version 1.0. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

Carzaniga, A., Rosenblum, D S., and Wolf, A L., 2001. Design and Evaluation of Wide-Area Event Notification Service. *ACM Trans. on Computer Sys.* 19(3), 332 -383.

Cattell, R. and Barry D., 2000. *The Object Database Standard: ODMG 3.0*, Morgan Kaufmann.

Cilia, M., Bornhovd, C., and Buchmann, A.. 2001. Moving Active Functionality from Centralized to Open Distributed Heterogeneous Environments. *Proc. of the 6th Int. Conf. on Cooperative Information Sys.*, 2172 LNCS. Springer.

Collet, C., Vargas-Solar, G., and Grazziotin-Ribeiro, H., 1998. Towards a Semantic Event Service for Distributed Active Database Applications. *DEXA*, LNCS 1460.

COM/DCOM, 2003. *COM/DCOM Specification.* http://www.microsoft.com/com/resources/comdocs.asp

Cugola, G., Nitto, E., and Fuggetta, A., 2001. The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS. *IEEE Trans. on Software Eng.*, 27, 827-850.

Dietrich, S W., Urban, S D., Sundermier, A., Na, Y., Jin, Y., and Kambhampati, S., 2001. A Language and Framework for Supporting an Active Approach to Component-Based Software Integration. *Informatica* 25,4 443-454.

DeJong, M., and Laird, C., 2003. TCL+Java = A match made for scripting. http://www.tcl.tk/software/java/java.html

EJB, 2001. *Enterprise JavaBeans Specification 2.0*. Proposed Final Draft 2.

Fegaras, L., and Maier, D., 1995. Towards an Effective Calculus for Object Query Languages. *ACM SIGMOD Int. Conf. on Mgmt. of Data*, San Jose CA, 47-58.

Foster, I, Kesselman, C., Nick, J., and Tuecke, S. "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", http://www.globus.org/research/papers/ogsa.pdf, 2002.

Freeman, E., Hupfer, S., and Arnold, K., 1999. *JavaSpace: Principles, Patterns, and Practice*. Addison-Wesley.

JMS, 1999. Java Messaging Service 1.0.2. http://java.sun.com/products/jms/docs.html

J2EE, 2003. *Java TM 2 Platform, Enterprise Edition (J2EE)*. http://java.sun.com/j2ee/

Jin, Y., Urban, S D., Sundermier, A., and Dietrich, S W., 2002. An Execution and Transaction Model for Active, Rule-Based Component Integration Middleware. *Proc. of the Eng. and Deployment of Cooperative Info. Sys.*

Kambhampati, S., 2003. *An Event Service for a Rule-Based Approach to Component Integration*, M.S, Thesis, Dept. of Computer Sci. and Eng., Arizona State University.

Liebig, C., Malva, M., and Buchmann, A., 2000. Integrating Notifications and Transactions: Concepts and X2TS Prototype. Proc. of 2nd Int. Workshop on Eng. Distributed Objects.

OMG, 1998. *Object Management Group: The common Object Request Broker, Architecture and Specification*, (December 1998).

Ousterhout, J., 1994. TCL *and the TK Toolkit*. Addison-Wesley Publishing.

Patil, R., 2003. *The Development of a Framework Supporting an Active Approach to Component-Based Software Integration*, Arizona State University, Dept. of Comp. Sci. and Eng., Spring 2003.

Rumbaugh, J., 1987. Relations as Semantic Constructs in an Object-Oriented Language. *Proc. of OOPSLA*, 446-481.

Urban, S D., Dietrich, S W., Na, Y., Jin, Y., Sundermier, A., and Saxena, A., 2001. The IRules Project: Using Active Rules for the Integration of Distributed Software Components. *Proc. of the 9th IFIP 2.6 Working Conf. on Database Semantics: Semantic Issues in E-commerce Sys*. Hong Kong, 265-286.

WSA 2002. *Web Services Activity*, http://www.w3.org/2002/ws

Widom, J., and Ceri, S., 1996. *Active Database Systems*. Morgan Kaufmann publishers, San Francisco CA.