The IRules Project

Using Active Rules for the Integration of Distributed Software Components

Susan D. Urban, Suzanne W. Dietrich, Yinghui Na, Ying Jin, Amy Sundermier, Akash Saxena
Arizona State University

Abstract: This paper provides an overview of the Integration Rules (IRules) approach to the interconnection of distributed software components. The intended use of this rule processing technology is for the specification of event-based processing logic in the development of component-based applications for distributed environments, such as E-commerce applications. The IRules approach builds upon the use of the Enterprise JavaBeans (EJB) software component model, providing an environment to facilitate the integration of black-box software components. An application integrator uses the IRules Component Definition Language to extend the definition of an EJB component in the IRules environment with derived and stored attributes, externalized relationships, and events. The application integrator then uses the IRules Integration Rule Language together with a scripting language to develop applications. Integration rules provide active rule capability to the environment so that as distributed components and external sources generate event notifications, integration rules are used to invoke methods on components or on higher-level application transactions. The IRules approach is unique in that it addresses the language and semantic framework needed in the context of software component models for the effective use of rules in distributed environments.

Key words: active rules, integration rules, distributed software component integration

1. INTRODUCTION

Applications such as electronic commerce are inherently distributed, requiring access to data and software from many different sources. Database

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35658-7_21](https://doi.org/10.1007/978-0-387-35658-7_21)

R. Meersman et al. (eds.), *Semantic Issues in E-Commerce Systems*

© IFIP International Federation for Information Processing 2003

technology has played an important role in supporting the needs of distributed applications, with past research focused on techniques for distributed query and transaction processing, as well as techniques for accessing multidatabase and federated database systems [Ozsu99]. The landscape of distributed computing has changed, however, due to the World Wide Web as well as recent developments in software component technology [EJB00] and middleware technology [OMG98, JMS98, JTS99, Edwards99]. These changes require a new look at database functionality and the manner in which such functionality is packaged to support distributed applications. As described in [Silberschatz97], certain forms of database functionality need to “break out of the box” to better serve the needs of applications that depend on distributed sources of information.

Active rule processing technology is a good example of a database component that can provide useful services to advanced applications if the appropriate technology exists for the use of rules in distributed environments. Traditionally, active rules have been used to transform passive, centralized database systems into reactive systems that respond to database and external events through the use of rule processing features. A fundamental component of an active database system is the notion of an *Event-Condition-Action (ECA)* rule. Using the ECA rule paradigm, when an event occurs, a rule associated with this event is triggered. If the condition of the rule evaluates to true, then the action of the rule is executed. ECA rules have been investigated in centralized database systems to maintain database constraints, to maintain materialized views, to notify users of specific conditions or events that occur, and to control workflow activities within a database application [Widom96].

The Integration Rules (IRules) project at Arizona State University (<http://www.eas.asu.edu/~irules>) is investigating the middle-tier, rule processing technology necessary for the use of active rules in the integration of distributed, black-box software components. The intended use of this rule processing technology is for the specification of event-based processing logic in the development of component-based applications for distributed environments, where the granularity of the components can range from low-level database objects to an entire software system.

The IRules approach builds upon the use of the Enterprise JavaBeans (EJBs) software component model specification from Sun Microsystems [EJB00]. The EJB component model promotes the vision of separating component services from the business logic of the components. Assuming that all databases and software sources of the application environment are encapsulated using EJBs, the IRules Component Definition Language is used to define relationships between components, to define derived and stored attributes for components, and to define the events that can be

generated by the IRules environment. Application integrators can then use object model specifications from CDL to create distributed applications using the Integration Rule Language together with application transactions. Integration rules provide a re-active capability so that as distributed components and external sources generate event notifications, integration rules are used to invoke methods on components or on higher-level application transactions. Conditions of integration rules can be expressed over the additional attributes and component relationships of the IRules environment, serving as a filter for the invocation of rule actions. Application integrators express transactions using a scripting language to define high-level processing logic that involves the invocation of methods on components defined within the IRules environment.

The purpose of this paper is to provide an overview of the IRules approach to the integration of distributed software components. We have already performed a study of the underlying distributed computing technology that is needed to support the IRules approach [Saxena00, Urban01]. The results of this study are being used to develop a distributed architecture for the execution of integration rules using Jini Connection Technology together with the Java Messaging Service (JMS) [Urban02]. The primary focus of this paper is on the semantic framework provided at the application development level of the IRules environment and the manner in which it can be used to support a rule-based approach to component integration. Unlike previous work that has focused on the mechanics of distributed rule processing, we are also addressing the necessary language issues needed in the context of software component models for the effective use of rules in distributed environments [Dietrich02, Urban02].

In the following sections, we outline the details of the IRules approach to component integration. Section 2 first provides an overview of related work on the use of active rules in distributed environments. In Section 3, we provide an overview of the IRules approach, with a more detailed look at the EJB component model and the manner in which it contributes to the design of the IRules environment. We also introduce an investment example that will be used to illustrate IRules concepts throughout the rest of the paper. Section 4 introduces the IRules Component Definition Language, illustrating the definition of derived attributes, externalized relationships and component-generated events. Section 5 elaborates on our use of integration rules and application transactions. The paper concludes in Section 6 with a summary of our work and a discussion of future research directions for language evaluation issues as well as architecture and execution model issues.

2. RELATED WORK

The language aspects of the IRules environment build on several important areas of related work. The syntax of the IRules Component Definition Language is based on the Object Definition Language of the Object Data Management Group Standard (ODMG) [Cattell00]. The difference is that we are providing extensions that allow an application to encapsulate existing software components with IRules functionality for defining relationships between distributed components as well as events and derived attributes. The syntax of the IRules Integration Rule Language is based on research with active rules in centralized active database systems [Widom96]. Our work, however, is defining a more explicit approach to the specification of bindings that are passed between the condition and action parts of a rule. We are also using the Object Query Language (OQL) of the ODMG standard as the basis for the expression of rule conditions and bindings and, in addition, are investigating new forms of couplings modes that may be more appropriate for distributed environments.

Recent work on distributed rule processing has focused on the use of ECA rules for the interconnection of distributed sources within the framework of the Common Object Request Broker Architecture (CORBA) [OMG98]. In [Le98, Chakravarthy98], a CORBA-based framework is defined for composite events and rules in a distributed, heterogeneous environment. Pissinou and Vanapipat [Pissinou96] describe a reactive, rule-based multidatabase architecture that separates user/client services from the provider/server services through the use of intelligent agents. In a continuation of [Pissinou96], Pissinou et al. [Pissinou97] give a detailed description of a rule service to support active distributed objects. The C²offein distributed information system described in [Bultzingsloewen96, Koschel98a, Koschel98b] is a CORBA-based system composed of an event history mechanism, an event detection component, a rule system, and an action execution component. The FRAMBOISE (FRAMework using oBject OrIented technology for Supplying active mEchanisms) system [Fritsch98] is used to provide a sophisticated active database layer on top of existing database systems using a toolbox approach that realizes tasks such as event definition, event detection, and rule execution.

In the area of event notification, the research in [Ma98] uses a *publish-register-notify* model called COBEA (CORba Based Event Architecture) that extends the CORBA event service with features such as event filtering, composite events and quality of service. The work in [Barrett96] describes a formal framework for event-based software integration so that there can exist a uniform way to discuss various integration approaches and specify new ones. The research in [Barrett96] is based on previous event-notification

systems such as FIELD [Reiss90], PolyLith [Purtilo94], and Yeast [Krishnamurthy95]. In [Bacon95], the authors have extended the Interface Definition Language of CORBA to include the definition of the events that a component generates. Such extensions enhance programming, reusability, and portability.

Our own past work in the area of active database systems has influenced the research presented in this paper. In particular, our work with the ADOOD RANCH (Active, Deductive, Object-Oriented Database - Relating Action, Negation, Constraints, and Horn Clauses) project resulted in a declarative language for the integration of active, deductive, and object-oriented concepts [Urban97], together with a framework for capturing the metadata of such an environment [Abdellatif99b] and an execution model that supports the incremental examination of the database state during rule processing [Abdellatif99a]. Our approach to the use of derived attributes in the IRules Component Definition Language, as well as the structure of integration rules, extends our results from the ADOOD RANCH project to distributed domains. The work in [Ayyaswamy99] represents our initial investigation of a CORBA architecture for distributed ECA rule processing for the purpose of maintaining constraints in a loosely-coupled, federated database environment. More recently, we have performed a comparison of CORBA [OMG98], Java [JMS98, JTS99], and Jini [Edwards99] technologies for evaluating different architectures for the execution of integration rules [Saxena00, Urban01].

The IRules project described in this paper is different from the above research projects in several ways. First of all, this work assumes that all distributed sources have well-defined interfaces expressed using the EJB component model. Previous work on distributed active rules has not been addressed in the context of component technology. The most important difference with respect to this paper is that we are also addressing the semantic framework required to interconnect EJB components using an event-based approach. Previous work has primarily focused on rule execution issues and has not thoroughly addressed the environment needed for the expression of active rules in a distributed environment.

The IRules approach is similar to the concept of mediator objects. Use of a mediator object to encapsulate the interaction of a set of objects is the basis of the behavioral design pattern *Mediator* [Gamma95]. A mediating component contains logic describing the interconnections between other components. From a software engineering perspective, this allows looser coupling between components. A mediator may also encapsulate the process of reacting to events in an event-based system and then route the events to the appropriate components via method calls using a mechanism described as *implicit invocation* [Sullivan92]. The IRules environment promotes the

concept of a mediator pattern for loosely-coupled components and event-based implicit invocation by allowing integration rules to describe the business logic normally encapsulated within a mediating component. The IRules environment, in effect, acts as a mediator.

3. AN OVERVIEW OF THE IRULES PROJECT

This section provides a high-level view of the IRules approach to software component integration. Since we assume the use of EJBs as a component model, Section 3.1 first provides an overview of EJB technology. Section 3.2 provides an introduction to the IRules environment. Section 3.3 then introduces an investment example to illustrate the integration functionality provided by the IRules environment.

3.1 Enterprise JavaBeans

Enterprise JavaBeans (EJB) is a specification from Sun Microsystems that provides a server-side component model for the Java language [EJB99, EJB00]. The goals of the EJB component model are similar to the goals of the CORBA standard since both standards are concerned with simplifying the task of programming in a distributed object environment. However, the EJB specification addresses the division of labor between distributed computing specialists and business logic specialists in a more comprehensive manner than the CORBA standard, placing more stringent requirements upon the EJB environment and more restrictions upon the EJB programmer.

Enterprise JavaBeans separates component services and infrastructure from the business logic of the components. In particular, the EJB component model facilitates an efficient division of responsibilities between different types of programmers by specifying three different entities that participate in the execution of EJBs: *Objects*, *Containers*, and *Servers*. EJB objects are software components with application-specific programming logic, while the container and server provide the run-time environment for the components. Application server vendors implement EJB containers and servers to provide robust infrastructure and services for EJB software components, such as transaction management and security. Business logic specialists focus on developing EJB objects that implement enterprise business processes. Application assemblers may construct business systems by using purchased EJB objects for the majority of the system, therefore leveraging reuse of commercially available components, while still having the flexibility of building custom business components that reflect their company's unique competitive strategy. By separating the responsibilities of the server

provider, component developer, and application assembler, each group can develop expertise independently, thereby segregating application development into areas of developer specialization.

Enterprise JavaBeans objects are developed by application software vendors. Version 2.0 of the EJB Specification supports the use of *Entity* and *Session* Enterprise JavaBeans. An Entity Bean represents a database or persistent business object in the distributed environment, while a Session Bean provides transient, per client, session-based processing. EJB applications typically require both Entity and Session Beans because they represent different aspects of the application.

Figure 1 illustrates the EJB Component Model and the conceptual view of client/EJB interaction. Each EJB component provides two types of interfaces: the *EJBHome* interface and the *EJBObject* interface. The *EJBHome* interface represents the life-cycle methods of the component. A client calls the operations in the *EJBHome* interface to create or delete instances of components or to use the find operation to lookup a specific object instance in an Entity Bean. The *EJBObject* interface, also known as the remote interface, defines the signature of business methods for changing attribute values and carrying out business logic functions that are specific to the EJB component. The *EJBHome* and *EJBObject* interfaces are implemented by container-generated classes that are specific to the EJB component. As illustrated by Figure 1, all instances of an EJB component share the same *EJBHome*, but each instance has a corresponding *EJBObject* remote interface.

An EJB container runs within a server and hosts one or more different types of EJB objects, as illustrated by Figure 1. Containers provide development tools to generate component-specific implementations for the *EJBHome* and *EJBObject* interfaces, handling remote object access details and providing a level of indirection between client requests and business methods necessary to support transactions, security, and persistence.

3.2 Component Integration Within IRules

Advanced distributed applications are typically constructed from components in many containers provided by multiple companies in distributed locations. The goal of the IRules project is to provide an abstract working environment in which relationships can be established between EJB components in multiple containers and integration rules can be used together with application transactions to create an event-driven approach to interactions between components spanning multiple containers. The IRules project assumes the EJB components are purchased black-box software

entities, therefore the environment must provide services without requiring modification to the source code of the EJB components in the containers.

The IRules project uses the IRules Definition Language (IRDL) to define components in the IRules environment and to declare event-based integration rules between EJB components in multiple containers. The IRDL sublanguage that supports the definition of a global object model for distributed components is known as the IRules Component Definition Language (CDL). The sublanguage that defines the active integration rules is called the IRules Integration Rule Language (IRL).

Figure 1: The EJB Component Model

CDL allows the application integrator to extend the definition of a component by adding *derived attributes*, *stored attributes*, *externalized relationships*, and *events*. A derived attribute is a value that can be accessed as if it were a property of the component, but the value is calculated as a result of a method call on a session bean. In comparison, a stored attribute

provides a way to enhance a component with an additional value that is physically stored in the IRules environment. An externalized relationship is a reference to another component that is maintained by the IRules environment, essentially allowing the application integrator to extend the object model of an application by declaring new relationships between objects. Externalized relationships may be declared for components that reside within different containers, where the use of CDL makes the inter-container complexities transparent to the application integrator. Events in the IRules environment can either be external events from outside of the IRules environment or events that are raised by software components within the environment, signifying changes in database state or the initiation or completion of method execution.

To illustrate the IRules approach, consider an investment application with components to represent portfolios of stock holdings. Figure 2 illustrates the intra-container relationships between four different containers with purchased software components. The portfolio container has data about stocks owned in the form of current and past holdings. The portfolio may also have pending orders that are waiting for execution when a particular market condition is met. The stock component represent stocks that exist independently of the portfolios that own them. The stock component generates events to signal changes in value depending upon buy/sell transactions in the stock market.

Figure 2: Purchased Software Components

Figure 3 represents the object model desired for the application, with relationships between components on different servers. For example, we wish to represent the fact that a Portfolio may have orders waiting for execution in the PendingOrder component. The application integrator uses the IRules CDL to describe such relationships. CDL is also used to describe derived attributes as well as events that can be generated by accessing the methods of a component. The IRules environment generates wrappers for each purchased component to represent the derived attributes and externalized relationships declared by CDL. An IRules wrapper enhances the interface of an EJB component, creating the meta-level interface required for manipulation by the IRules environment and also providing application-specific extensions. Using the object model description found in a CDL specification, the IRules IRL is used together with transactions to describe the business logic of the application.

4. THE IRULES COMPONENT DEFINITION LANGUAGE (CDL)

The purpose of the IRules CDL is to define EJB black-box components and associated IRules extensions within the IRules environment. Figure 4 provides the CDL for the investment example shown in Figure 3. For each component in the environment, the CDL language allows for the declaration of a named extent, derived attributes, and externalized relationships. In addition, CDL provides support to define component-generated events based on method invocations. The compilation of a CDL schema results in the generation of a corresponding IRules wrapper for each component that provides the extensions required by the IRules environment. The rule processor uses the wrappers to access components in the IRules environment so that all method calls to components may be intercepted by the wrapper to generate events defined by the components.

In CDL, all declared components implement the EJB component model. The `implements` clause is used to declare whether the component is an entity bean or a session bean. The use of the `implements` clause also allows for future expansion to support other component models. Optionally, an IRules component can define a named extent, which provides a name that the application integrator can use in the specification of integration rules to refer to the set of references of all instances of that EJB component.

Additional IRules extensions to a component include the definition of derived attributes and externalized relationships. A derived attribute is a value that can be accessed as if it were a property of the component, but the value of the attribute is based on a specification that can be derived from

existing information. A derived or virtual value is a common abstraction provided by database applications so that different views of the underlying data can be provided to various users of the database. In Figure 4, the Stock component has a derived attribute named `rateChange` of type `float`, to derive the percentage rate change of the current price of the stock to its closing price. The `rateChange` value is calculated by executing the `calcRateChange` method of the `stockAI` session bean. The suffix `AI` on the name of the session bean indicates that an application integrator developed the session bean for the purpose of calculating the `rateChange` value.

The definition of externalized relationships is a powerful abstraction provided by the IRules environment. The externalized relationships provide the methodology to establish references between black-box components deployed in various containers in a distributed environment. For example, Figure 4 defines a `pendingTrades` relationship for `Stock` as a set of type `PendingOrder`, having an inverse relationship named `actUpon` in `PendingOrder`. The IRules wrapper maintains the references to the related components, and the IRules environment provides default behavior for accessing the externalized relationships for use by the application integrator.

Another IRules extension to a component is the definition of an event. An event definition consists of the keyword `event`, the name of the event in the IRules environment with its associated event parameters, and the name of the method call. For example, consider the CDL for the `Stock` component, which defines a named event `beforeSetPrice`. This event is raised before the `setPrice` method is called on the `Stock` component to update its price in our database in response to a change in the stock market:

```
event    beforeSetPrice(NewPrice)    {method before setPrice(NewPrice) }
```

Figure 3: The Investment Application

```
component Stock implements EntityBean
(extent  stocks)
{attribute float rateChange {stockAI.calcRateChange(Stock self)};
relationship set <CurrentHolding> heldBy inverse CurrentHolding::stockHeld;
relationship set <PendingOrder> pendingTrades inverse PendingOrder::actUpon;
relationship set <PastHolding> soldBy inverse PastHolding::pastStock;
event beforeSetPrice(NewPrice) {method before setPrice(NewPrice)};}
```

```
component User implements EJB
(extent  users)
{relationship set<Portfolio> owns inverse Portfolio::ownedBy;}
```

```
component Portfolio implements EntityBean
(extent  portfolios)
{attribute float portfolioValue;
relationship set<User> ownedBy inverse User::owns;}
```

```
component PendingOrder implements EntityBean
(extent  pendingOrders)
{relationship Stock actUpon inverse Stock::pendingTrades;}
```

```
component CurrentHolding implements EntityBean
(extent  currentHoldings)
{relationship Stock stockHeld inverse Stock::heldBy;}
```

```
component PastHolding implements EntityBean
(extent  pastHoldings)
{relationship Stock pastStock inverse Stock::soldBy;}
```

```
component OrderExecution implements EntityBean
(extent  orderExecutions) {}
```

```
component CloseOutOrder implements EntityBean
(extent  closeOutOrders) {}
```

```
component StockAI implements SessionBean{}
```

Figure 4: Component Definition for the Investment Application

The modifier *before* indicates that the event is raised before the call to the method on the component. A corresponding *after* modifier is available in CDL to raise the event after the execution of a method. We are using a similar approach to define additional types of IRules events. For example,

the price of a Stock in the database is updated based on price fluctuations in the stock market, which is an external event. Therefore, the IRules definition language must also provide the ability to define external events, such as an `afterStockMarketQuote` event in the investment application.

5. INTEGRATION RULES AND APPLICATION TRANSACTIONS

The IRules CDL establishes the framework for the expression of integration rules and application transactions for the interconnection of software components. This section elaborates on the use of rules and transactions in the IRules environment. Section 5.1 presents the integration rule language using a scenario within the investment application. Section 5.2 then discusses our investigation of a scripting language for the development of application transactions in the context of actions that are triggered by integration rules.

5.1 The IRules Integration Rule Language (IRL)

Integration rules follow the form of active rules as defined in active database systems, supporting event-condition-action (ECA) rules as well as event-action (EA) rules. Integration rules have been specifically designed for expressing conditions over the model as defined by a CDL schema. Integration rules can therefore refer to derived attributes, externalized relationships, and events in addition to the objects, attributes, and methods of a black-box component. Consistent with our use of ODL as the basis for describing relationships between components, OQL is used for the basis for condition expression in IRL. A unique aspect of the IRL is the manner in which OQL is used for the explicit definition of rule bindings within the rule condition and the way in which such bindings are subsequently referenced within the rule action.

Figure 5 provides an abstract specification of the IRL syntax, where square brackets are used to indicate optional items and italics indicate a named item. A rule is given a name and the event upon which the rule is triggered. If an event is component-generated, then the `on` clause provides a mechanism to name the component and associate a component variable to reference the event object in the condition and action specification. As in traditional active rules, we assume the use of event-condition and condition-action coupling modes, such as immediate and deferred. We are also

investigating new coupling modes that will support synchronized versus asynchronous execution of the condition and/or action.

```
create rule ruleName
event    eventName(event Parameters)
         [on componentName componentVariable]
condition [ecCoupling]
         when conditionSpecification
           [define bindingsName as
            select <>
            from <>
            [where <>]]
action   [caCoupling]
         [from <>
         [where <>]]
         do action
```

Figure 5: Integration Rule Language Syntax

The condition specification consists of a when clause and an optional define clause. The when clause establishes a condition that must hold true on the event object and parameters. The optional define clause explicitly defines bindings that may be referenced in the action, using an OQL-like syntax. The action clause includes a from-where clause to allow for accessing the named bindings from the condition or to further express conditions over the objects specified in the action by the do clause, which calls a method or an application transaction defined by the application integrator.

To illustrate the rule language design, consider the following rule triggering scenario. The rules referenced in this scenario are illustrated in Figure 6. When a source such as the New York Stock Exchange generates information indicating the change in a stock price, this information raises the external event `afterStockMarketQuote` in the IRules environment. This external event triggers the `stockQuote` rule that causes a subsequent change to the stock price information maintained within the Stock component of the IRules environment. Before this change is made, the `beforeSetPrice` event is raised as defined in the IRules Stock component, triggering the `stockSell` rule and the `stockBuy` rule. These two rules examine the information in the `PendingOrder` component to initiate `sellStock` and `buyStock` transactions for pending orders that have been waiting for price changes on the stock in question. After the completion of the `sellStock` and `buyStock` transactions, the

`afterSellStock` and `afterBuyStock` events are raised to trigger the `billingToUserOnSell` and `billingToUserOnBuy` rules, respectively.

The first rule triggered in this scenario is the `stockQuote` rule. This rule provides an example of an EA rule and the manner in which bindings can be generated within the action part of the rule. In particular, when a `NewPrice` is generated for a `StockTicker` in the external event, the `setPrice` method is invoked on the appropriate stock within the `Stock` component. The `from-where` clause in the action part of the rule identifies the stock object on which the `setPrice` method is executed.

The `stockSell` and `stockBuy` rules provide examples of ECA integration rules that make use of externalized relationships in the expression of the rule. These rules also illustrate several important points with respect to the timing of the event that triggers each rule and the use of bindings in the rule action for deferred rule actions. Since both of these rules have similar structures, we will only address these issues in the context of the `stockSell` rule.

The purpose of the `stockSell` rule is to initiate `sellStock` transactions when a price increase occurs. We only want to initiate such transactions, however, for pending orders where the `NewPrice` exceeds the desired selling price in the pending order. This situation implies that we need to compare the new price of the stock with the old price of the stock to determine if there was a price increase. This can only be done by examining the old and new values before the execution of the price change in the `Stock` component, thus illustrating the need for the `beforeSetPrice` event as defined in Figure 4. The coupling mode on the rule condition is `immediate`, indicating that the check for a price increase should be performed as soon as the rule is triggered. The `sellStock` transactions on the appropriate pending orders, however, should only be executed after the completion of the `setPrice` method. As a result, the action part of the rule is deferred, meaning that the action will not be performed until the end of the outer-most transaction in which the rule was triggered (i.e., the transaction in which the action of the `stockQuote` rule is executed). Rather than binding to the pending orders in the condition part of the rule, the bindings are established in the action part of the rule using the `from-where` clause. This approach avoids binding to pending order objects at condition evaluation time that may no longer be valid by the time the action is executed at transaction commit time. Note that the use of externalized relationships for rule expression is also illustrated in the `stockSell` rule. The `from-where` clause in the action part of the rule accesses objects in the `PendingOrder` component by using the `pendingTrades` externalized relationship defined in the `Stock` component.

5.2 Application Transactions

Integration rules are used together with application transactions in the IRules environment to specify the processing logic for the interconnection of software components. Whereas integration rules add an event-driven aspect to component integration, transactions are used to specify well-defined sequences of activities as high-level, global operations in the environment. Integration rules can be used to trigger the execution of application transactions. Operations that occur inside of transactions can also raise events that trigger rules for invoking other application transactions.

In Figure 6, `sellStock`, `buyStock`, `setUserBillingOnSell`, and `setUserBillingOnBuy` are application transactions in the IRules environment. The `sellStock` transaction, for example, is responsible for invoking several operations on the different components within the Portfolio container to update information about current holdings, past holdings, and cash on hand when a sale of stock transpires. These activities represent well-defined procedures for updating the data within each component and maintaining relationships between components within the container. As another example, the `setUserBillingOnSell` transaction is invoked after a sell activity is complete to identify the billing user of a portfolio and to invoke a billing operation on the appropriate instance of the User component in the User container.

We are currently using JACL [DeJong97] for the specification of application transactions. JACL is a Java-based scripting language that extends the Tool Command Language (TCL) [Ousterhout94]. JACL allows allocating Java objects and invoking Java methods inside of TCL. JACL also supports user-defined commands as extensions to the pre-defined set of TCL commands. This capability is especially useful in the IRules environment since we need to interact with EJBs. In particular, our initial prototyping of the investment application using JACL indicates that a request to create a new instance of an EJB involves a sequence of several commands since the language was designed for access to Java objects rather than EJBs. Instead of requiring the application integrator to remember such a sequence, JACL can be used to create an abstract command, such as *newEJBInstance*, with appropriate parameters for creating new instances of a component. Such extensions can be used to ease application integration and provide a more abstract way to access the services that are provided by the components in the IRules environment.

```

create rule stockQuote
  event    afterStockMarketQuote(StockTicker, NewPrice)
  action   from S in stocks
           where S.StockTicker()=StockTicker
           do S.setPrice(NewPrice);

create rule stockSell
  event    beforeSetPrice(NewPrice)
           on stock S
  condition immediate
           when NewPrice>S.price
  action   deferred
           from Pn in S.pendingTrades
           where S.price>=Pn.desiredPrice AND Pn.action="sell"
           do sellStock(S,Pn);

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

create rule billingToUserOnSell
  event    afterSellStock(S,Pn)
  action   setUserBillingOnSell(Pn.orderedBy)

create rule billingToUserOnBuy
  event    afterBuyStock(S,Pn)
  action   setUserBillingOnBuy(Pn.orderedBy)

```

Figure 6: Integration Rules for the Investment Enterprise

6. SUMMARY AND FUTURE RESEARCH

This paper has presented an overview of the language framework that is provided in the IRules project for developing an event-based approach to the interconnection of distributed software components. Our future work is focused in two primary directions: one involving metadata representation

and language evaluation issues and the other involving architectural design and execution model issues.

We have designed the metadata to store the results of compiling CDL [Dietrich02] and IRL specifications. We have also prototyped the use of the IRules wrappers to store the externalized relationships and to act as a proxy to its associated black-box EJB [Urban02]. With respect to the execution of integration rules, we have already developed several different prototypes for experimenting with recent tools for distributed object computing [Saxena00, Urban01]. As a result of our investigations, we are refining the execution model of integration rules and developing a general-purpose rule processing architecture based on Jini and JMS. For the execution model, we are examining the applicability of several advanced transactions models and investigating the need for new forms of coupling modes for distributed rule processing.

ACKNOWLEDGMENTS

This research has been supported by NSF Grant No. IIS-9978217.

REFERENCES

- [Abdellatif99a] T. Abdellatif. An Architecture for Active Database Systems Supporting Static and Dynamic Analysis of Active Rules Through Evolving Database States, Ph.D. Dissertation, Arizona State University, Department of Computer Science and Engineering, Fall 1999, 375 p.
- [Abdellatif99b] T. Abdellatif, R. Chan, S. W. Dietrich, B. Siddabathuni, A. Sundermier, and S. D. Urban, "Meta-Data Components in Support of an Active Deductive Object-Oriented Database System," Proceedings of the 3rd IEEE Meta-Data Conference, Bethesda, MD, On-line publication: <http://computer.org/conferen/proceed/meta/1999/>, paper #16, 1999.
- [Ayyaswamy99] K. Ayyaswamy, "The Design and Implementation of a CORBA based environment for Distributed Constraint Maintenance," M.S. Thesis, Department of Computer Science and Engineering, Arizona State University, 1999.
- [Bacon95] J. Bacon, J. Bates, R. Hayton, and K. Moody, "Using Events to Build Distributed Applications," Proceedings of SDNE, 1995.
- [Barrett96] D. Barrett, L. Clarke, P. Tarr, and A. Wise, "An Event-Based Software Integration Framework," ACM Transactions on Software Engineering and Methodology, vol. 5, no. 4, October 1996.
- [Bultzingsloewen96] G. Bultzingsloewen, A. Koschel, and R. Kramer, "Active Information Delivery in a CORBA-based Distributed Information System," Proceedings of the First International Conference on Cooperative Information Systems (CoopIS'96), Brussels, Belgium, June 1996.
- [Cattell00] R. G. G. Cattell, The Object Database Standard, ODMG 3.0, Morgan Kaufmann, 2000.

- [Chakravarthy98] S. Chakravarthy and R. Le, "ECA Rule Support for Distributed Heterogeneous Environments," Proceedings of the International Conference in Data Engineering, Orlando, 1998.
- [Dietrich02] S. W. Dietrich, S. D. Urban, A. Sundermier, Y. Na, Y. Jin, and S. Kambhampati, "A Language and Framework for Supporting an Active Approach to Component-Based Software Integration," to appear in Informatica, 2002.
- [DeJong97] M. DeJong and C. Laird, "TCL+Java = A Match Made for Scripting," <http://www.sunworld.com/sunworldonline/swol-11-jac1.html>.
- [Edwards99] W. Edwards, Core Jini, Prentice Hall, 1999.
- [EJB00] Enterprise Java Beans Specification 2.0, May 2000.
- [Fritschi98] H. Fritschi, S. Gatzu and K. Dittrich, "FRAMBOISE - an Approach to Framework-Based Active Database Management System Construction," Proceedings of 7th International Conference on Information and Knowledge Management (CIKM'98), Washington DC, November 1998.
- [Gamma95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Design, Addison-Wesley, Reading, MA, 1995.
- [JMS98] Java Message Service 1.01, October 5, 1998, <http://www.javasoft.com/products/jms>.
- [JTS99] Java Transaction Service, Dec 01, 1999, <http://www.javasoft.com/products/jts>.
- [Koschel98a] A. Koschel, R. Kramer, "Configurable Event Triggered Services for CORBA-based Systems," Proceedings of 2nd International Enterprise Distributed Computing Workshop (EDOC'98), San Diego, California, November 1998.
- [Koschel98b] A. Koschel, P. Lockemann, "Distributed Events in Active Database Systems: Letting the Genie Out of the Bottle," Journal of Data and Knowledge Engineering (DKE), vol. 25, pp. 11-28, 1998.
- [Krishnamurthy95] B. Krishnamurthy and D. Rosenblum, "Yeast: A General Purpose Event-Action System," *IEEE Transactions on Software Engineering*, vol. 21, no. 10, October 1995.
- [Le98] R. Le and S. Chakravarthy, "Support for Composite Events and Rules in Distributed Heterogeneous Environments," Technical Report, Computer and Information Science and Engineering Department, University of Florida, January 1998.
- [Ma98] C. Ma and J. Bacon, "COBEA: A CORBA Based Event Architecture," 4th Conference on Object Oriented Technologies and Systems (COOTS), New Mexico, April 1998.
- [OMG98] Object Management Group: The Common Object Request Broker, Architecture and Specification, Revision 2.3, December 1998.
- [Ousterhout94] J. Ousterhout, TCL and the TK Toolkit, Addison-Wesley Publishing, 1994.
- [Ozsu99] T. Ozsu and P. Valduriez, Principles of Distributed Database Systems, Second Edition, Prentice Hall, Upper Saddle River, NJ, 1999.
- [Pissinou96] N. Pissinou and K. Vanapipat, "Active Database Rules in Distributed Database Systems: A Dynamic Approach to Solving Structural and Semantic Conflicts in Distributed Database Systems," Computer Systems Science and Engineering, vol. 1, pp. 35-44, 1996.
- [Pissinou97] N. Pissinou, K. Makki, and R. Krishnamurthy, "An ECA Object Service to Support Active Distributed Objects," Informatics and Computer Science, pp. 63-104, 1997.
- [Purtilo94] J. Purtilo, "The POLYLITH Software Bus," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 1, January 1994.

- [Reiss90] S. Reiss, "Connecting Tools Using Message Passage in the FIELD Environment," *IEEE Software*, vol. 7, no. 4, July 1990.
- [Saxena00] A. Saxena, An Evaluation of Distributed Architectures for the Integration of Black-Box Software Components, M.S. Thesis, Arizona State University, Department of Computer Science and Engineering, Tempe, AZ, Fall 2000.
- [Silberschatz97] A. Silberschatz and S. Zdonik, "Database Systems – Breaking Out of the Box," *ACM SIGMOD Record*, vol. 26, no. 3, September 1997.
- [Sullivan92] K. J. Sullivan and D. Notkin, "Reconciling Environment Integration and Software Evolution" *ACM Transactions on Software Engineering and Methodology*, vol 1, no. 3, July 1992, pages 229-268.
- [Urban97] S. D. Urban, A. P. Karadimce, S. W. Dietrich, T. Ben Abdellatif, and R. Chan, "CDOL: A Comprehensive Declarative Object Language," *Data and Knowledge Engineering*, vol. 22, 1997, pp. 67-111.
- [Urban01] S. Urban, S. Dietrich, A. Saxena, A. Sundermier, "Interconnection of Distributed Components: An Overview of Current Middleware Solutions," *Journal of Computer and Information Science in Engineering*, vol. 1, no. 1, March 2001, pp. 23-31.
- [Urban02] S. D. Urban, S. W. Dietrich, A. Sundermier, Y. Jin, Y. Na, and S. Kambhampati, "Distributed Software Component Integration: A Framework for a Rule-Based Approach," to appear in *Handbook of Electronic Commerce in Business and Society*, P. Lowry, J. Cherrington, and R. Watson (editors), CRC Press, 2002.
- [Widom96] J. Widom and S. Ceri, *Active Database Systems: Triggers and Rules for Advanced Database Processing*, Morgan Kaufmann Publishers, San Francisco, 1996.