An Architecture and Execution Environment for Component Integration Rules*

Ying Jin, Susan D. Urban, and Suzanne W. Dietrich
Arizona State University
Department of Computer Science
Tempe, AZ 85287-5406, USA
{yingjin | s.urban | dietrich}@asu.edu

Abstract

The Integration Rules (IRules) project at Arizona State University (http://www.eas.asu.edu/~irules) is developing a declarative event-based approach to component integration. Integration rules are based on the concept of active database rules, providing an active approach for specifying event-driven activity in a distributed environment. The IRules project consists of a knowledge model that specifies the IRules Definition Language and an execution model that supports integration rule execution. This research focuses on the execution model and the architectural design parts of the IRules project. The main objective of this research is to develop a distributed execution environment for using integration rules in the integration of black-box components. In particular, this research will investigate the design of an architecture that supports the IRules semantic framework, the development of an execution model for rule and transaction processing, and the design of a rule processing algorithm for coordinating the execution of integration rules. This research will combine the distributed computing framework of Jini, the asynchronous event notification mechanism of the Java Message Service (JMS), and the distributed blocking access functionality of JavaSpaces to support active rule processing in a distributed environment. The limitations of the underlying Enterprise JavaBeans (EJB) component model pose transaction processing challenges for the integration process. This research will develop a suitable transaction model and processing logic to overcome the limitations of the underlying EJB component model. Furthermore, the architectural design will allow an easy extension of the system to accommodate other component models. This research is expected to contribute to nested rule and transaction processing for active rules that have not been previously addressed in distributed rule processing environments. The development of the IRules execution environment will also contribute to the use of distributed rule-based techniques for event-driven component integration.

1 Introduction

The development of advanced distributed applications often requires the interconnection of many different data and software services. These services can be found among distributed sources within one company, as well as between companies for enterprise applications such as electronic commerce. To facilitate the integration process, software component technology is often used as a means for distributed sources to advertise services that can be accessed from remote locations. A software component is a unit of software composition with specified interfaces and explicit context dependencies [Szyperski96]. Components

* This research was supported by NSF Grant No. IIS-9978217

providing services typically publish their well-defined interfaces to allow access to relevant operations. Inherently, most components for commercial usage are black-box components that do not allow outside modification.

Standards organization and research groups have proposed component models to support distributed software integration. Examples include the Common Object Request Broker Architecture (CORBA) component model [OMG98], COM+ [Microsoft00], and Enterprise JavaBeans (EJB) [EJB01] component model of the J2EE development framework. Although the existing standards facilitate the integration process, their use still require an integrator's low-level knowledge of programming and transaction processing, where traditional integration solutions are hard-coded such as the work in [Xia98]. Hard-coded solutions can cause costly changes when applications evolve or when venders modify component interfaces. This difficulty with hard-coded integration solutions motivates the need for better approaches to build a more supportive environment to ease component integration.

The Integration Rules (IRules) project at Arizona State University is developing a declarative event-based approach to component integration. Integration rules are based on the concept of active database rules, providing an active approach for specifying event-driven activity in a distributed environment. An active rule typically consists of three parts: an event, a condition, and an action [Paton99]. An event causes a rule to be triggered. The condition is a query over a database that is checked when the rule is triggered. If the condition evaluation returns true, the action can be executed to modify data, retrieve data, or perform application procedures.

There are two major aspects to the development and use of integration rules within the IRules project. The first aspect involves the development of the semantic and language framework for the use of integration rules in the specification of event-based application. The second aspect involves the development of an execution environment for integration rules. In the context of the IRules project, this research is focused on the second aspect. In particular, the main objective of this research is to develop a distributed architecture and rule processor for using integration rules in the integration of black-box components.

2 Research Questions

This research is specifically working on component integration with well-defined interfaces based on the EJB component model. The integration of black-box components introduces several challenges to the development of a rule and transaction processing framework for integration rules. First of all, black-box components can't be modified and they are not aware of their participation in the integration framework. As a result, black-box components alone do not provide the necessary behavior for participating in more global rule and transaction processing activities. Furthermore, the EJB component model has its own notion

of transactional behavior, which is beyond the control of external environments such as IRules. Therefore, suitable transaction control logic needs to be designed at the global level to overcome the restrictions of the underlying EJB component model. Active rules can also trigger other active rules, thus forming a nested structure. Since the nesting of rules and their transaction control in a distributed environment may span across different locations, distributed rule processing is more difficult than that of centralized active rule environments. Other distributed projects have been limited to the flat form of transaction control. In contrast, this research will investigate designs to support nested active rule processing in a distributed environment. Given the challenges outlined above, the development of the integration rule processor for the IRules environment will involve 1) designing the architectural components to support the IRules execution framework, 2) designing the execution model including coupling models for integration rules, and 3) developing the rule processing algorithms for coordinating the execution of integration rules with distributed events and transactions.

3 Related Work

This section reviews different fields of work related to the proposed research.

- Active Database System

The IRules project is based on the concept of active database systems. Active database systems extend traditional databases by supporting mechanisms to automatically monitor and react to events that are taking place either inside or outside of the database system itself [Paton99]. An active database includes a knowledge model and an execution model. A common approach for knowledge model is by using Event-Condition-Action (ECA) rules. The execution model of an active rule determines how a set of rules behaves at runtime. The rule execution model defines issues for the execution of rules such as coupling modes, transition granularity, net-effect policy, scheduling, priorities, and error handling [Paton99].

- Event-based Architecture for Component Integration

The event-based approach to component integration is a technique that uses event notification for component interoperation. The Event-Based Integration (EBI) framework [Barrett96] is a high-level reference model that outlines architectural concepts for interconnection through events. The CORBA-based Event Architecture (COBEA) [Ma98] is another general event-driven architecture. Although the event-based approach provides reactive functionality to integration, it does not address how to specify events and how to respond to events declaratively as in the active rule approach discussed in the next subsection.

- Distributed Active Systems

Most of the existing integration projects using ECA rules are based on the CORBA specification. In [Chakravarthy98], ECA rules are used to solve distributed communication for the components that describe interfaces in OMG

IDL. The specification, detection, and management of composite events are the focus. The C^2offein project [Koschel98a] is a CORBA-based distributed information system. The underlying data sources, such as relational databases, are wrapped to enable read access in CORBA environment. The FRAMBOISE (FRAMework using oBject OrIented technology for Supplying active mEchanisms) project [Fritschi98] is a construction system that provides services for definition and execution of active database mechanisms on top of a passive DBMS in heterogeneous, distributed environments. In [Cilia01], active rules are used to glue existing applications in a distributed environment by using the publish/subscribe mechanism of X^2TS [Liebig00] that is based on CORBA Notification Service.

Most of the above distributed rule projects have all been based on the use of the CORBA standard. The proposed research is different from these projects in several ways. First of all, the IRules project is based on the EJB component model that provides a more enterprise-level approach to integration rather than the use of basic CORBA objects. Second of all, this research is using Jini connection technology [Edwards99] as the primary mean for distributed object computing. Since no existing research project has addressed a rule processing solution based on Jini, this research requires a new look at rule processing architectures and algorithms developed around the use of Jini services.

4 Overview of The IRules Project

The IRules Definition Language (IRDL) provides the semantic base for component integration within the IRules Project. The IRDL consists of four sub-languages: the Component Definition Language (CDL) for defining IRules components, the IRules Scripting Language (ISL) for describing application transactions, the Event Definition Language (EDL) for defining events, and the Integration Rule Language (IRL) for defining active rules.

The IRules project is based on the EJB component model, assuming the purchased black-box components are EJBs. An investment application is used as the motivating example, which consists of four different containers with purchased software components: a Portfolio container, a Pending Order container, a Stock container, and a User container. IRules adds a semantic layer, known as the IRules wrapper, on top of purchased EJB components. IRules wrappers provide additional functionality to black-box components, such as defining external relationships, specifying extents, derived attributes, and stored attributes for each component, as well as describing the events generated before and after method calls of components. The specification of this functionality is expressed using the CDL sub-language of IRDL together with the EDL sub-language for specifying events. Wrappers are generated by the compilation of CDL.

IRL is the rule sub-language of IRDL. An example of IRL is presented in Figure 1, where an integration rule is expressed based on component definitions and

relationships in CDL. IRL is based on the traditional ECA rule format in active database systems. But integration rules are different from active database rules in that the condition is in an OQL-like query expressed over distributed components. The action part can be presented either as a method or as an ISL transaction. ISL is based on the JACL scripting language, providing integrators the functionality to describe well-defined sequences of processing logic of an application transaction. In this rule example, the event is signaled after creating a new instance of a pendingOrder EJB. The condition part will check whether certain market conditions are met, using component states and the externalized relationships. The action part executes the sellStockOnNewPO application transaction to perform the functionality of selling stocks.

```
create rule newStockSellPendingOrder
event
afterCreatePendingOrder(pnId,portfolioId,stockId,numOfShares,desiredPrice,action,actUpon,orderedBy)
condition    immediate
             when action = "sell"
             define stockAndPendingOrder as
                  select struct ( stk: s, newPo: pn )
                  from s in stocks, pn in pendingOrders
                  where pn.id=pnId and pn.actUpon=s and desiredPrice<=s.price and pn.status="waiting"
action       immediate
             from sp in stockAndPendingOrder
             do sellStockOnNewPO(stockId, sp.stk.price, portfolioId, numOfShares, sp.newPo)
```

Figure 1 An IRL Example

The IRules language framework for CDL and EDL are being developed by the work of [Kambhampati01, Patil01] within the IRules group. The research presented in this paper is developing an environment for executing IRL rules and ISL application transactions over distributed EJB components within the context of the IRules language framework.

5 Proposed Approach and Methodology

This section discusses the approach and methodology for the proposed research objectives. The global objective of this research is to develop a generalized processing environment for the execution of integration rules and application transactions within the semantic framework of the IRules component integration domain. To achieve this global objective, the following specific objectives will be investigated: 1) Design a distributed architecture for the execution of IRL rules and ISL transactions, 2) Develop an execution model for IRL, 3) Design and implement execution algorithms for the execution of IRL in the distributed environments, and 4) Evaluate the functionality of the IRules system.

5.1 A Distributed Architecture for the IRules Environment

The IRules research group has selected Jini Connection technology as the primary distributed computing framework for the IRules environment. According to the

results of an initial IRules investigation of the major distributed computing techniques by [Saxena00], Jini provides more flexibility than CORBA for custom development of transaction control.

Figure 2 presents the proposed Jini architecture design with a high level representation of the interfaces among the different components forming the IRules environment. In the proposed environment, the object manager, the rule manager, and the event handler will be modeled as Jini Services. This research will combine the distributed computing framework of Jini, the asynchronous event notification mechanism of JMS [JMS01], and the distributed blocking access functionality of JavaSpaces together to support active rule processing in a distributed environment. The research issues for architecture design involve:

- Metadata Management
  The metadata stores the semantic descriptions of the supported environment captured as an IRDL Schema, which is populated by the compilation of IRDL [Kambhampati01, Patil01]. This aspect of the research involves the design of the metadata interface based on the needs of the architectural components involved in the execution of rules and transactions.

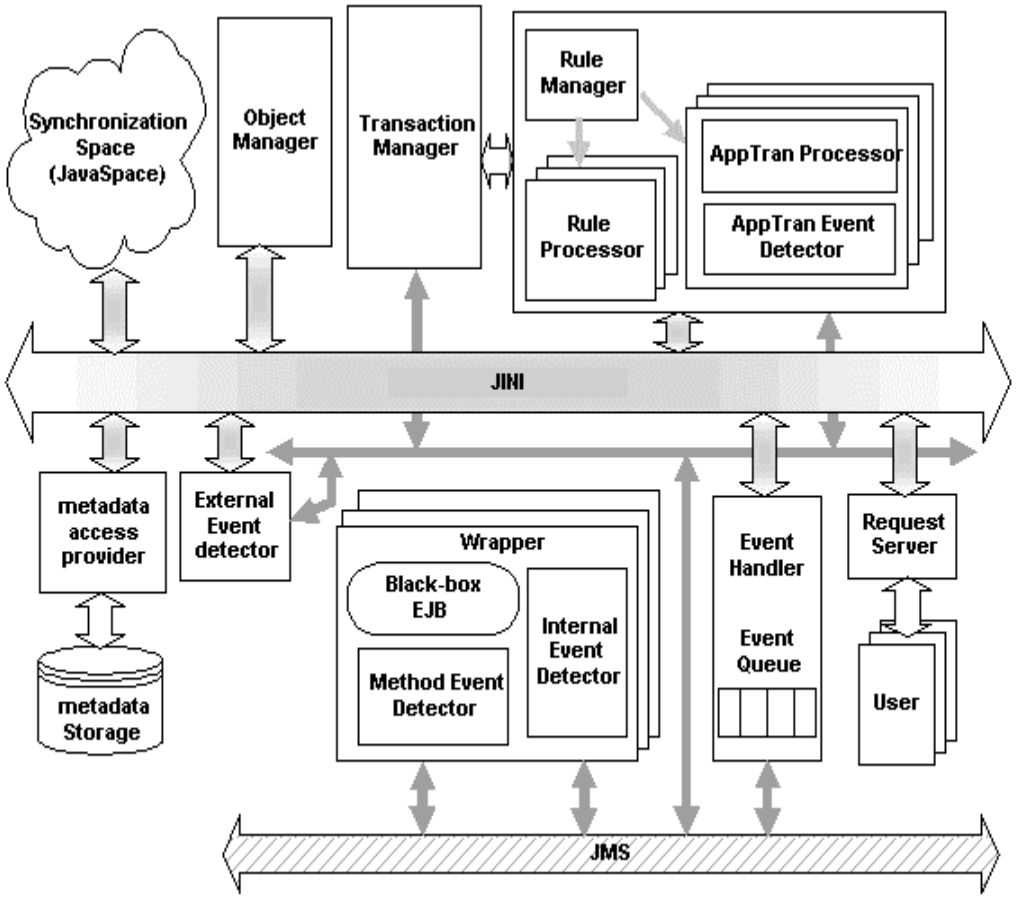

Figure 2 IRules Architecture

- Event Detector and Event Handler
  Allocated among several architectural components, event detectors can signal events through JMS to the event handler [Kambhampati01]. There are two possible types of the interaction between the event handler and the rule manger: push and pull. This aspect of the research is focused on designing the interaction that must occur between the event handler and the rule manager.
- Rule Manager
  The rule manager is responsible for processing rules upon the occurrence of events and for the processing application transactions upon user requests from the request server. The rule manager will contact the metadata, the transaction manager, and the object manager at run time to process rules according the IRules active rule processing algorithm.
- Transaction Manager
  The transaction manager is responsible for transaction control during active rule processing. The transaction manager must ensure that the processing of application transactions and active rules within IRules is within the appropriate transaction context.
- Object Manager
  Research issues are involved with developing the abstraction capability of the object manager and the interface of the object manager with the rest of the system. The abstraction provided by the object manager allows application transactions and rules to be described without details of remote invocation on components. By isolating component access details from other architectural components, the IRules system will more easily accommodate additional component models in the future, such as COM+, where a separate object manager can be constructed for access to different component models.

5.2 Execution Model for Integration Rules

The development of the execution model will focus on the design of coupling modes, the semantics of rule processing with coupling modes, and the transaction model for the support of rule execution.

- Coupling Modes
  The coupling modes of active rules allow rule definers to specify how to execute active rules at run time. This research will investigate four types of coupling modes for integration rules: immediate synchronous, immediate asynchronous, deferred, and decoupled.

  The execution logic of the immediate synchronous mode and the deferred mode is the same as that typically found in traditional active databases. The immediate asynchronous mode is a new coupling mode that is being defined as part of this research. An immediate asynchronous rule, for example, will be evaluated immediately, but the transaction that raised the event will not be suspended. The evaluation of rules and the execution of the current transaction are therefore concurrent, thus achieving better performance over the synchronous mode for distributed execution.

The challenge of this part of the proposed research is in defining the operational semantics of the asynchronous mode, as well as in defining the execution of the decoupled mode. Furthermore, an integration rule can be triggered before or after an event happened. The development of all coupling modes together with *before* and *after* modifiers needs to be coordinated with the development of the transaction processing model.

- Transaction Model
  Both Jini and EJB have their own transaction model. The transaction model of EJBs can be controlled by JTA (Java Transaction API) [JTA01]. The standard of J2EE specifies that the J2EE transaction manager that control JTA transactions does not support nested transactions. Although the Jini specification supports nested transactions, there is no existing implementation for nested transactions. On the other hand, active rule processing is inherently nested - the action of a rule may act as an event to trigger new rules, forming a nested structure. Furthermore, Jini transactions and JTA are incompatible. Rather than re-implementing the existing Jini transaction manager and controlling the context switch between JTA and the Jini Transaction, this research will develop its own transaction infrastructure for flexible control of active rule processing.

  Another research issue with respect to transaction processing is to design a transaction model that is appropriate for the nested execution of rules over EJB components. In the transaction control of traditional databases, when to release locks and when to update permanent storage can be fully controlled by the transaction manager of the database. But it is not possible to control black-box EJBs in such a manner. Entity beans must be accessed with a *container managed transaction*, where the transaction for method invocation of an entity bean is totally controlled by the EJB container. With no notion of the parent-child hierarchy of the outside transaction semantic, the container is independently determining when to retrieve and update the database. Among the transaction models for transaction nesting, the flexible transaction model is more suitable for integration rules since it allows unilateral commit of subtransactions. This research will develop techniques for the use of the flexible transaction model to support the nested execution of integration rules. In the scope of this research, we assume a failure semantics where individual rules might abort without affecting the triggering transaction, which is one of the typical failure semantics in active systems [Paton99].

5.3 Active Rule Execution Algorithm

The active rule execution algorithm is a suit of methods that together form the circuit through which active rules are processed. The proposed active rule execution algorithm is based on the algorithm of the ADOOD RANCH project [Abdellatif99], using *cycles* to control the nested execution of active rules. This research will re-design the rule execution algorithm for a distributed environment, fully supporting the proposed coupling modes and transaction processing model.

Another aspect of rule execution is rule scheduling. When a set of rules is triggered, they can execute concurrently to achieve better performance in a distributed environment. However, the concurrent execution of two conflicting rules can cause non-deterministic results. A scheduling algorithm will be designed in this research to allow concurrent execution deterministically. The algorithm will use the triggering and conflict relationships between active rules to coordinate the order of rule execution.

5.4 Evaluation of the Work

The evaluation will focus on the practicality and functionality of the proposed methodologies and techniques, compared to other similar research projects in four aspects: data and software sources of the distributed environments, semantic support, rule processing, and transaction processing. The evaluation of the work will also involve an assessment of the Jini architecture, addressing the advantages and disadvantages of the architecture, and identifying improvements that should be addressed in future versions of the IRules environment. The evaluation will experiment with different loads for the arrivals of events for observation of rule processing throughput to identify potential bottlenecks. This experiment will be used to propose future enhancements to the architectural design of the IRules environment.

6 Preliminary Ideas and Current Results

To date, this research has identified the necessary architectural components and specified the interfaces for interaction among the architectural components. The basic functionality of the architectural components has been designed and demonstrated by a centralized prototype implementation. Current activities are focused on implementing the architectural components using Jini Connection techniques.

The research has so far specified the definition of four types of coupling modes and the constraints of using different coupling modes between different parts of an active rule. The active rule execution algorithm has also been designed to regulate the rule execution to support the specified coupling modes. Current activities are focused on designing the rule scheduling algorithm and interfacing the rule manager with the IRules event handler for several different types of events that are supported by the IRules environment.

7 Summary of Expected Contribution

This research will contribute to the current efforts towards integrating distributed black-box components. The expected contributions of this research include the following:

- The design and implementation of a new distributed architecture for component integration by using active rules based on the Jini computing framework over the EJB component model.
- The definition of the operational semantics of the immediate asynchronous coupling mode for distributed integration rules. This new coupling mode is especially designed for a distributed environment to enhance performance by the concurrent execution of a transaction and its triggered rules.
- The development of the active rule execution algorithm, supporting rule nesting in a distributed environment that has not been addressed by in other distributed active rule system.
- The design of a new active rule scheduling algorithm allowing concurrent rule execution deterministically in a distributed environment.
- Demonstration of the use of distributed rule-based techniques for event-driven component integration as compared to traditional hard-coded solutions.

References

[Abdellatif99] 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.

[Barrett96] D. J. Barrett, L. A. Clarke, and P. L. Tarr, "A Framework for Event-Based Software Integration," *ACM Transactions on Software Engineering and Methodology*, vol. 5, no. 4, October 1996, pp. 378-421.

[Cilia01] M. Cilia, C. Bornhovd, and A. Buchmann, "Moving Active Functionality from Centralized to Open Distributed Heterogeneous Environments," *Proceedings of 9th International Conference on Cooperative Information Systems* (*CoopIS'01*), Trento, Italy, September 2001, pp. 195-210.

[Chakravarthy98] S. Chakravarthy, and R. Le, "ECA Rule Support for Distributed Heterogeneous Environments," *International Conference on Data Engineering* 1998, pp. 601.

[Edwards99] W. K. Edwards, *Core Jini*, Prentice-Hall PTR, Second Edition, 2000

[EJB01] Enterprise JavaBeans Specification 2.0, Proposed Final Draft 2, 19 April 2001.

[Fritschi98] H. Fritschi, S. Gatziu, K. and R. Dittrich, "FRAMBOISE – an Approach to Framework-Based Active Database Management System Construction," *Proceedings of the 7th ACM International Conference on Information and Knowledge management,* Nov. 1998, pp. 364-370.

[JMS01] Java Message Service Tutorial 1.3 Beta Release, January 2001, Http://java.sun.com/products/jms/tutorial/html/jmsTOC.fm.html.

[JTA01] Java Transaction API (JTA) 1.0.1 Specification, http://java.sun.com/products/jta/.

[Kambhampati01] S. Kambhampati, *An Event Service for a Rule-Based Approach to Component Integration,* M.S. Thesis Proposal, Arizona State University, Department of Computer Science and Engineering, 2001.

[Koschel98] A. Koschel, and R. Kramer, "Configurable Event Triggered Services for CORBA-based Systems," *Proceedings of 2nd International Enterprise Distributed Object Computing Workshop (EDOC'98)*, San Diego, California, November 1998, pp. 306-318.

[Liebig00] C. Liebig, M. Malva, A. Buchmann, "Integrating Notfications and Transactions: Concepts and X2TS Prototype," *Proceedings of the 2nd International Workshop on Engineering Distributed Objects*, University of California, Davis, USA, Nov. 2-3, 2000, pp.194-214.

[Ma98] C. Ma, and J. Bacon, "COBEA: A CORBA-Based Event Architecture," *Proceedings of USENIX COOTS'98*, Santa Fe, New Mexico, USA, April 1998, pp. 117-131.

[Microsoft00] Microsoft Corporation, COM+, http://www.microsoft.com/com/tech/complus.asp, December 2001.

[OMG98] *Object Management Group: The Common Object Request Broker, Architecture and Specification*, Revision 2.3, December 1998.

[Patil01] R. H. Patil, *The Development of a Framework Supporting an Active Approach to Component Based Software Integration*, M.S. Thesis Proposal, Arizona State University, Department of Computer Science and Engineering, 2001.

[Paton99] N. W. Paton, O. Diaz, "Active Database Systems," *ACM Computing Surveys*, Vol. 31, No. 1, March 1999, pp. 3-27.

[Saxena00] A. Saxena, *An Evaluation of Distributed Architectures for the Integration of Black Box Software Component*, M.S. Thesis, Arizona State University, Department of Computer Science and Engineering, Fall 2000

[Szyperski96] C. Szyperski, and C. Pfister, Workshop on Component-Oriented Programming, Summery, *In M. Muhlhauser (ed.) Special Issues in Object-Oriented Programming – ECOOP96 Workshop Reader*, Springer-Verlag, 1996.

[Xia98] B. Xia, *Object-Oriented Distributed Software Component Development and Integration in Common Object Environments*, Ph.D. Thesis, Department of Computer Science and Engineering, Arizona State University, 1998.