A PROTOTYPE FOR INTEGRATION OF WEB SERVICES INTO THE IRULES APPROACH TO COMPONENT INTEGRATION

Susan D. Urban, Vikram V. Kumar, and Suzanne W. Dietrich
Department of Computer Science and Engineering
Arizona State University
Tempe, Arizona USA

Keywords: web services, component integration, integration rules, event generation, component relationships

Abstract: The Integration Rules (IRules) environment provides a framework for using events and rules in the integration of EJB components. This research has investigated the extensions required to integrate Web Services into the IRules architecture and execution environment. The IRules language framework and metadata have been extended for Web Services, with enhancements to Web Service interfaces for describing services that represent object manipulation operations as well as component enhancements such as event generation, stored attributes, and externalized relationships between distributed components. Web service wrappers provide the additional IRules functionality for the enhanced Web Service interfaces, with a state management facility in the IRules environment providing persistent storage of stored attributes and externalized relationships. The IRules Web service wrappers are client-side, component-independent wrappers for Web Services, thus providing a more dynamic approach to the modification of service interfaces as well as the dynamic entry and exit of participants in the integration process.

1 INTRODUCTION

A Service-Oriented Architecture (SOA) (Brown et al., 2003) is a paradigm that has recently emerged, where software components are exposed as services that can be consumed by clients over the network. While an SOA represents an important concept for the interconnection of software components, Web Services (Web Services, 2000) provide a framework for a platform and programming language-independent implementation of SOAs.

The integration of Web Services is typically performed in a procedural manner. Languages such as BPEL4WS (Curbera et al., 2002) have been developed specifically to support the composition of Web Services into business processes. The Integration Rules Project (IRules) (Urban et al., 2001) has taken a different approach to business process specification, defining an event and rule-based environment for the integration of black-box components. The IRules approach provides an event-driven architecture, where an event, such as the execution of a method, triggers the execution of integration rules. An integration rule provides a declarative framework for the specification of conditions that are to be checked in response to events. If the condition is satisfied, additional methods or transactions can be invoked.

In the initial implementation of the IRules environment, integration rules were specifically used for the integration of Enterprise JavaBeans (EJBs) components (EJB, 2001). The Component Definition Language (CDL) of the IRules environment is used to enhance EJB components with IRules functionality, defining named extents, stored attributes, externalized relationships between components, and events for existing components. This functionality is provided through IRules wrappers for EJB components that are automatically generated from the compilation of the CDL (Patil, 2003). The wrappers, also implemented as EJB components, serve as proxies to components, generating events before and after the execution of component methods and providing persistent storage for extents, stored attributes, and externalized relationships.

One of the goals of the IRules project is to support the integration of different types of component models. Since many component models will likely expose their capabilities as Web Services, the primary objective of this research has been to develop a prototype for the use of Web Services within the IRules event and rule-based integration

D. Urban S., V. Kumar V. and W. Dietrich S. (2005).
A PROTOTYPE FOR INTEGRATION OF WEB SERVICES INTO THE IRULES APPROACH TO COMPONENT INTEGRATION.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 3-10
DOI: 10.5220/0002527900030010

framework (Kumar, 2004). To make use of the existing IRules execution environment, our investigation has experimented with the use of Web Services as an interface to EJB components.

The incorporation of Web Services into the IRules environment introduced several challenging design and architectural issues. Unlike EJB components, Web Services do not have any naming conventions for identifying create, remove, and findAll methods. Since these methods are needed for query processing within the condition evaluator of integration rules, this research has incorporated language features for identifying such methods in CDL for Web Services. Another difference between the original IRules environment and the use of Web Services is found in the state management capabilities. The IRules environment was originally developed with EJBs providing state management for the storage of stored attributes and externalized relationships defined in CDL. This research has defined a state management facility for Web Services within the IRules execution framework. The state management facility is a relational database that is constructed dynamically using the metadata generated by CDL together with Java Database Connectivity (JDBC) for incremental definition of relations.

The placement and nature of the IRules wrapper for Web Services has also been redefined as a result of this research. Web Services represent true black-box components, where the integrator does not have control over how the services are exposed. The containers where services exist are also not accessible to the integrator. For this reason, this research has defined the IRules Web Service wrappers to exist in the integrators environment rather than within the component's container. In addition, the Web Service wrapper design is dynamic in nature, where services can be added and removed from the overall integration process without the need to regenerate the Web Service wrapper as with the EJB design.

The remainder of this paper is structured as follows. Section 2 provides an overview of the IRules environment. Section 3 presents the Web Service wrapper architecture and design. Section 4 discusses the IRules property repository for providing state management. Related work is presented in Section 5. The paper concludes with a summary in Section 6.

2 OVERVIEW OF IRULES

The IRules environment enhances distributed EJB components with IRules specific semantics that enable event and rule-based integration. IRules follows a declarative approach where the integration logic and component enhancements are explicitly specified by the integrator using various IRules languages.

CDL is used as a semantic layer over components and provides a way of defining externalized relationships between components, event definitions, extents, and stored attributes. The syntax of CDL is presented in Figure 1 from (Patil, 2003). Extents provide a mechanism for iterating through the objects of a specific component type. Stored attributes are a persistent capability provided by the IRules wrapper. Externalized relationships are also persistent and are used to establish associations between component instances. IRules also provides the capability of defining before and after events on methods as part of CDL. Before events are generated before the invocation of a component method, while after events are invoked after method invocation. Events in IRules are used to trigger integration rules expressed using the Integration Rule Language (IRL) (Dietrich et al., 2001).

```
Component ComponentName implements ComponentType
(extent ExtentName)
{
  attribute AttributeType AttributeName
  [{SessionBeanName.MethodName (SessionParameters) } ];

  relationship TargetOfPath RelationshipName inverse
  InverseRelationshipTarget::InverseRelationshipName;

  event IRulesEventName (EventParameters)
  {method Modifier MethodName (MethodParameters) };};
```

Figure: 1: Syntax of CDL (Patil, 2003)

The IRules Scripting Language (ISL) is used to create application transactions that invoke methods on components (Kambhampati, 2003). The Event Definition Language (EDL) is used to specify additional events associated with application transactions and other external events (Kambhampati, 2003; Urban et al., 2004). IRL is used to define integration rules and is based on Event-Condition-Action (ECA) rules from active database systems. Each integration rule consists of an event, a condition, and an action. An event triggers a rule associated with the event. A condition is evaluated over distributed components. The action is a call to a method of a component or an application transaction if the condition evaluates to true. Several coupling modes have been defined to handle the transactional relationships that exist between the event, condition, and action parts of an integration rule (Jin et al., 2002).

3 WEB SERVICE WRAPPER ARCHITECTURE AND DESIGN

To generalize the use of the IRules approach, this research has investigated the use of Web Services as an interface to EJB components. Web Services, however, must still be wrapped to provide the IRules functionality described in Section 2.

3.1 Web Service Wrapper Location

Two approaches have been investigated for the location of the IRules Web Service wrapper. The wrapper can exist in the integrator's environment, or the wrapper can exist with the service provider as in the original EJB wrapper design of the IRules environment. In the later case, the provider wraps the component with the IRules wrapper and exposes the wrapped service as a Web Service, as shown in Figure 2. One of the restrictions of this approach is that all clients of the Web Service must have access to the service through the IRules wrapper. This approach is unnecessary for applications that are not a part of the IRules environment. Moreover this approach assumes an environment where the integrator has access to the implementation of each of the Web Services involved in the integration. In a service-oriented environment, this approach for the use of wrappers may not be possible for access to services provided by business partners. In addition, different wrappers would have to be developed for different component models, defeating the purpose of achieving component independence.

The other approach to the design of the IRules Web Service wrapper is to wrap the Web Service proxy as shown in Figure 3. This approach provides a more flexible environment since every client who uses the Web Service need not go through the IRules environment. Furthermore, the service wrapper is common irrespective of the component model employed by the Web Service provider, thus helping to achieve component independence within the IRules environment. The Web Service provider is therefore not concerned about the implementation and deployment of the IRules wrapper since the burden of integration is placed on the integrator at the client side.

In a Web Service environment, Web Services can also be dynamically discovered. The service provider approach shown in Figure 2 does not support these dynamic requirements. The service provider approach also suffers from the disadvantage of having to generate and re-compile the wrappers for each of the services involved in the integration. This is a limiting factor that hampers seamless and flexible integration. The wrapped proxy approach is more generic and does not require modifications to be done by the service. As a result, the wrapped proxy approach of Figure 3 was chosen for this research.

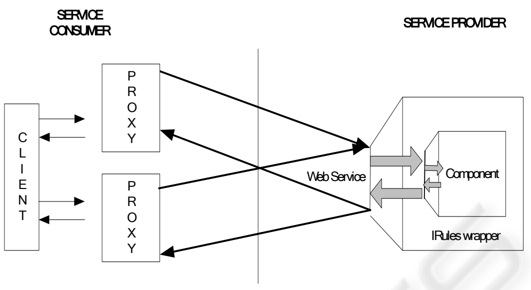

Figure 2: IRules Wrapper with the Service Provider

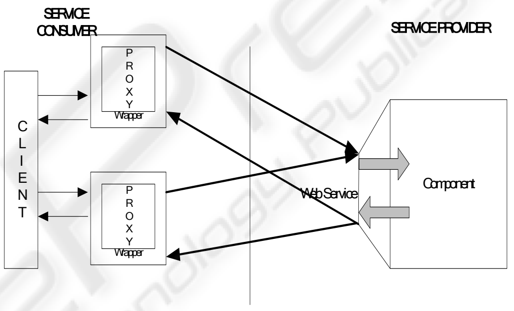

Figure 3: IRules Wrapper on the Web Service Proxy

Figure 4 illustrates the architecture of the IRules Web Service integration prototype. The core of the architecture is composed of the Web Service proxies and the IRules Web Service wrapper that encapsulates the proxies. The IRules Web Service wrapper is responsible for providing an enhanced interface to the Web Service. The service wrapper implements externalized relationships and stored attributes (i.e., IRules properties), which are made persistent in the *IRules Property Repository*. The repository is a centralized database storing all of the tables that are created to maintain IRules property values. These tables are created dynamically according to integration needs. An IRules Web Service wrapper is responsible for retrieving the required contents from these tables when the IRules Property accessor methods are called. The IRules Web Service Wrapper is also responsible for generating method events as shown in Figure 4.

The execution environment shown in Figure 4 makes extensive use of metadata generated from CDL, EDL, ISL and the IRL. This metadata is made

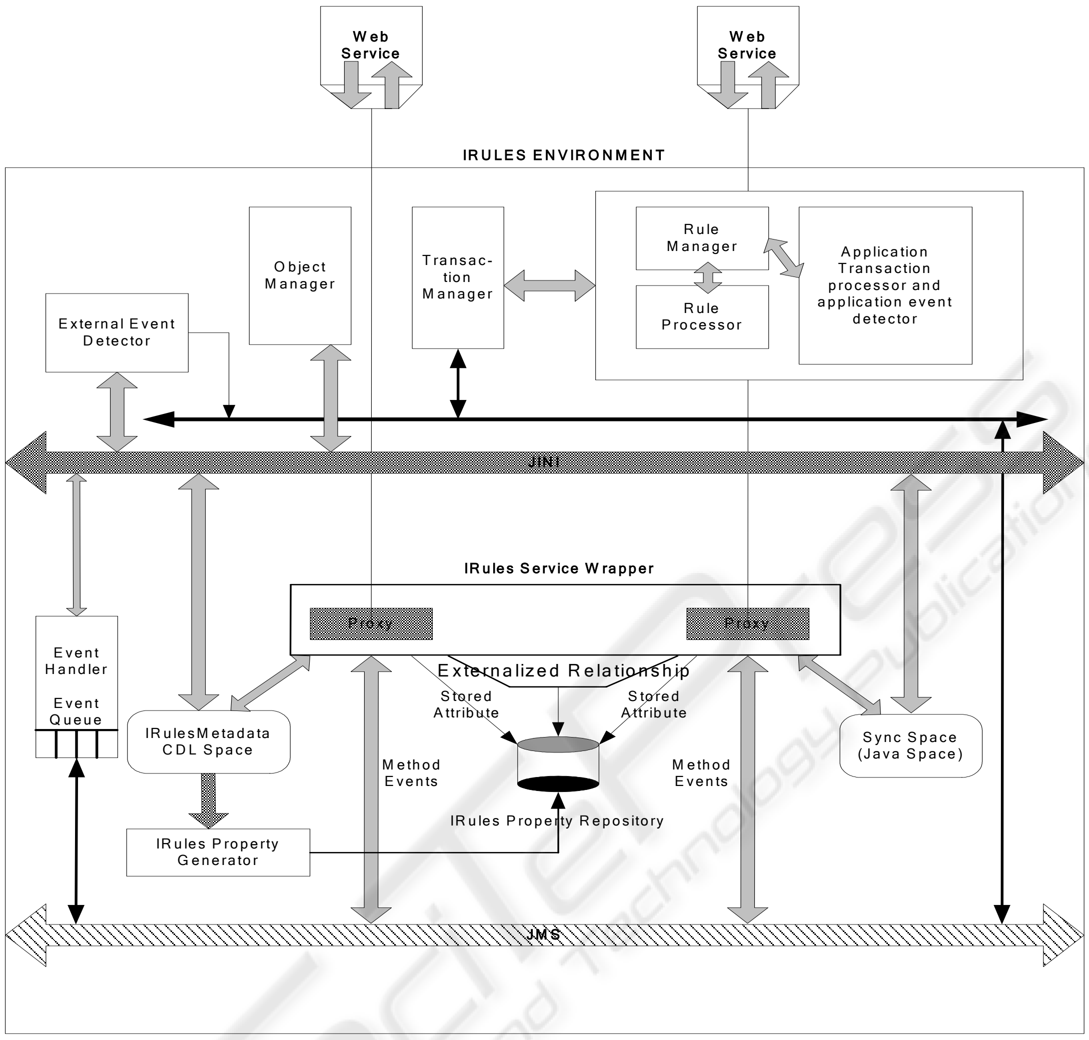

Figure 4: IRules Service Integration Environment.

persistent using the storage capabilities of JavaSpaces. In addition, JavaSpaces provides a synchronization mechanism that is used to coordinate the execution of rules with the invocation of Web Services. The coordination algorithm appears in (Kambhampati, 2003; Urban et al., 2004).

3.2 IRules Web Service Specification

The syntax of the extended CDL for Web Services is illustrated in Figure 5. The ComponentName is the name given to the component being exposed. The syntax implements WebService identifies that the component is exposed as a Web Service. Every WebService has a proxy class that is used to access the service. The syntax proxy ProxyClassName identifies the proxy class associated with the component exposed as a Web Service. The name of the proxy class is used to instantiate wrapper objects. The syntax (extent extentName) is used to specify the name of the extent for the component.

For integration of components in the IRules environment, it is necessary to identify the create, remove, and the findAll methods of a component. Create methods are responsible for creating an instance of a bean, while the remove method deletes an instance. The findAll method is responsible for retrieving the extent of a particular EJB. Extents are used in IRL to query instances of objects representing rows in a database. Unlike EJBs, which have naming conventions for creating, removing, and finding instances of EJBs, Web Services do not have any naming conventions. These methods are explicitly identified by the integrator as shown in the extended CDL for Web Services in Figure 5.

```
Component ComponentName implements WebService proxy
ProxyClassName (extent extentName)
{return_value create (method parameters)
 {method Name (method parameters)};

 void remove (method parameters)
 {method Name (method parameters};

 return_value  findAll ()
 {method Name (method parameters)};

 relationship TargetOfPath RelationshipName inverse
 InverseRelationshipTarget::InverseRelationshipName;

 event IRulesEventName (EventParameters)
 {method Modifier MethodName  (MethodParameters) };}
```

Figure 5: Extended CDL for Web Services

3.3 IRules Web Service Wrapper Design

Every method of a Web Service in the IRules environment is accessed through the IRulesWrapper class. A single IRulesWrapper class exists for all of the services in the IRules environment. The constructor of the class takes the name of the service that needs to be invoked as a parameter. The name of the service is the same as the name of the Web Service proxy class. Thus, depending on the value of the parameter passed to the constructor of the wrapper class, an instance of the wrapper is configured for that particular Web Service. As a result, many instances of IRulesWrapper are created - one for each service. Once a particular instance of the wrapper is created, the integrator cannot change the Web Service for which that particular instance of the wrapper is configured.

The methods of a Web Service are called through the callWebService generic method of the IRulesWrapper class. The parameters to the callWebService method, indicated in Figure 6, include the name of the method to be called, the parameters of the method to be called, and an additional transaction identifier to be passed with each method. Therefore a call to callWebService is actually a call on a proxy of a particular Web Service. This approach provides a dynamic nature to method invocation in IRules Web Service composition.

The implementation of callWebService takes care of invoking the appropriate method on the Web service using reflection as illustrated in Figure 6. The value of the wsProxy variable is the name of the proxy class of a particular Web Service, the value of which is passed as a parameter during the creation of the wrapper instance.

There are several advantages to the IRules Web Service wrapper design. One advantage is that there is no need for code generation for each of the components involved in the integration as in the original EJB wrapper design. An instance of each generic wrapper is simply created using the IRulesWrapper class. If a separate wrapper existed for each Web Service, a new wrapper would have to be generated for each new service added to the system. The IRules Web Service wrapper design makes it possible to dynamically change the underlying Web Service.

The wrapper instance also reads the appropriate metadata corresponding to the particular Web Service invoked and generates the method events required by the IRules environment. In addition, the wrapper instance maintains the externalized relationships through calls to create and remove methods on CDL relationships. Since all methods on the Web Service, including the methods of the IRules enhanced interface, are accessed through the generic callWebService method, there is no regeneration of the wrapper needed when new externalized relationships are created for a component exposed in a service. This enables the dynamic creation of relationships between components, where externalized relationships can be added at any point in the integration process without regenerating the wrappers. Similarly, the integrator can define new method events at run time.

```
public class IRulesWrapper {
private string wsProxy;
public Object callWebService (String   methodName, Object []
methodParams, int TransactionID)

{     Class cls = Class.forName(wsProxy);
      Object obj = cls.newInstance();
       Method  method =   obj.getMethod(methodName,null);
       Object returnValue =
       meth.invoke(obj.methodParams);}}
```

Figure 6: Invoking the Service Method through Reflection

3.4 A Motivating Example

This section presents an ONLINE SHOPPING APPLICATION to illustrate the integration of Web Services using the IRules service-oriented architecture. The application consists of six Web Services as shown in Figure 7. These Web Services access EJB components, which

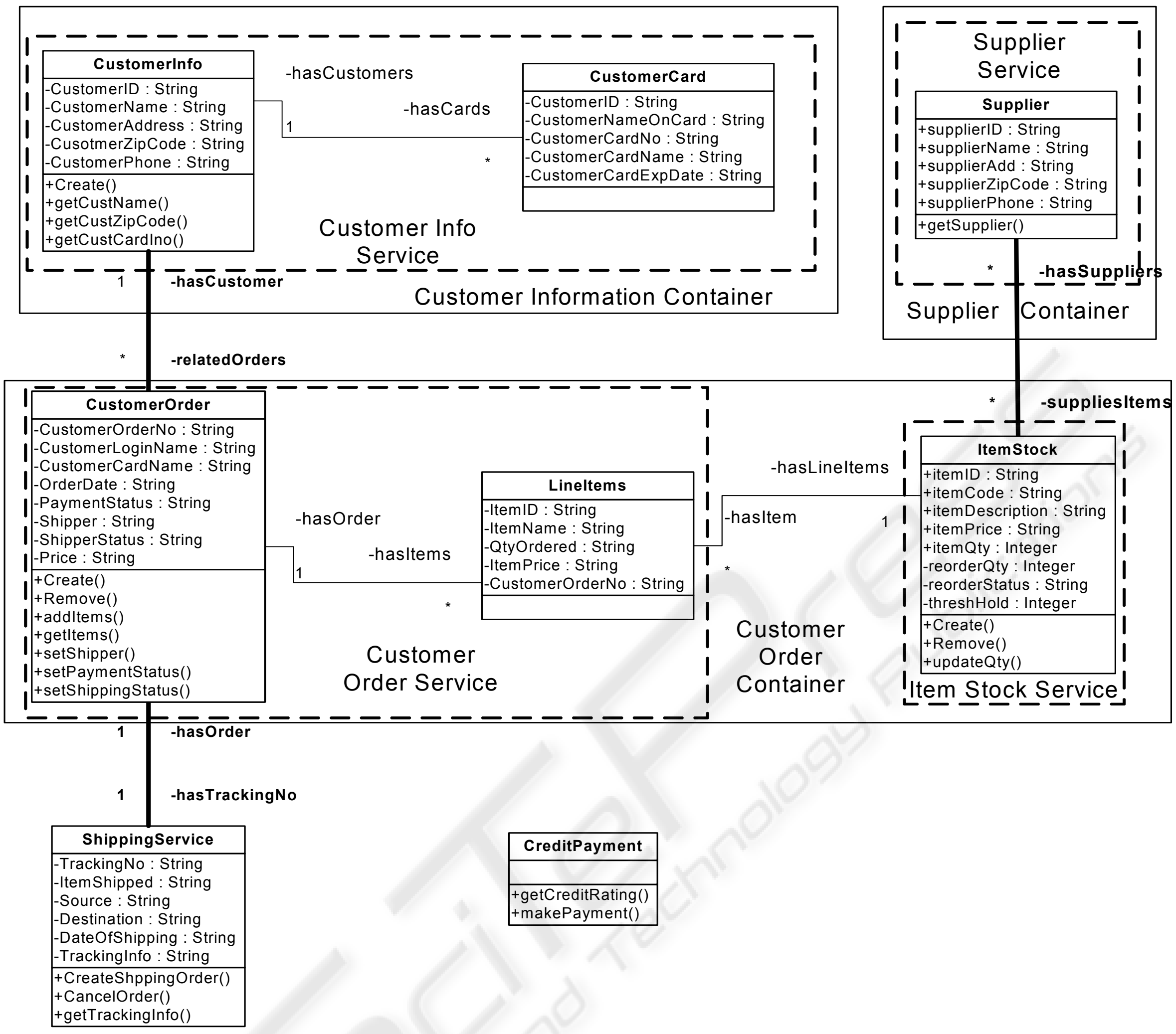

Figure 7: Online Shopping Application Components with Externalized Relationships

have been developed using the Web Logic application server. The Web Services were developed using BEA Web Logic Workshop, accessing the underlying EJBs.

Figure 7 depicts the various containers and components for the ONLINE SHOPPING APPLICATION. The CustomerInfo and CustomerCard components are located in one container. The CustomerOrder component, the LineItems component, and the ItemStock component are located in a separate container. A third container is used to store the Supplier component. These containers provide Web Services that are internal to the organization.

The application also makes use of two external Web Services. A CreditPayment service is used to approve credit card payments from customers. ShippingService is used to handle the shipping of the purchased items to the customer destination.

When a CustomerOrder is created, it needs to be associated with the customer information for that particular order. Since CustomerInfo is stored in a separate container, an externalized relationship is needed to associate a particular CustomerOrder with the CustomerInfo of the customer who created the order. Figure 8 illustrates the CDL definition for the Web Service associated with CustomerOrder. The definition enhances the CustomerOrder component with an explicit relationship to CustomerInfo. The Web Services wrapper for the CustomerOrder will provide the appropriate accessor methods to establish such relationships, with the details of the relationship stored in the IRules Property Repository (described in Section 4). A similar relationship exists between ItemStock and Supplier to indicate the supplier of a specific inventory item.

```
Component CustomerOrder implements WebService proxy
CustomerOrder (extent orders)
{CustomerOrder create (String orderId, String custId, String
orderDate, String creditCardName, String paymentStatus, String
shipper, String shippingStatus, String price)
{CustomerOrder createOrder (String orderId, String custId, String
orderDate, String creditCardName, String paymentStatus, String
shipper, String shippingStatus, String price);}

Void remove (String orderId)
{Void removeCustomerOrder (String orderId);}

Collection  findAll ()
{Collection findAllCustomerOrder ();}
relationship CustomerInfo hasCustomer inverse
CustomerInfo::relatedOrders;

relationship Shipper hasTrackingNo inverse Shipper::hasOrder;}
```

Figure 8: CDL of CustomerOrder

Externalized relationships can also exist with services outside of the enterprise. In Figure 7, ShippingService is an external service used by the application. The ShippingService provides a unique tracking number for items being shipped. An externalized relationship is used to relate a CustomerOrder with its ShippingService tracking number.

4 THE IRULES PROPERTY REPOSITORY

The IRules Property Repository shown in Figure 4 is a centralized state management facility for storing the stored attributes and externalized relationships of distributed components. The repository has been designed to take care of the dynamic aspects of Web Service composition by dynamically generating the relational tables for storing these IRules properties. The necessary SQL statements for creation of the appropriate tables are constructed using the CDL metadata at run time, using JDBC to send create table statements to the property repository for each stored attribute and externalized relationship to be defined. Therefore, any addition of services can be dynamically incorporated into the overall integration process.

For example, to relate an instance of CustomerOrder with the CustomerInfo component across Web Services, the primary keys that are used as references to instances of EJBs across the two Web Services are stored in an externalized relationship table, named CustomerOrderCustomerInfo within the IRules Property Repository. The table is dynamically constructed through an algorithm that retrieves the IRules Web Service metadata to issue the following create table statement:

```
create table CustomerOrderCustomerInfo (
    customerOrderNo varchar(5) primary key
    customerId varchar(30));.
```

Since the relationship is an N:1 relationship (i.e., many orders related to one customer), the customerOrderNo is determined to be the key of the table. To set the value of an externalized relationship, the integrator calls the setCustomerOrderCustomerInfo (keyCustomerOrder, keyCustomerInfo) accessor method for that particular relation. Accessor methods are provided by default by the Web Service wrapper for each relationship defined. On calling the relationship method, the keys passed to the method are stored into the externalized relationship table. Additional accessor methods can be called to retrieve the details of relationships between component instances. As with the invocation of methods on components, the relationship accessor methods are invoked by calling the generic callWebServiceMethod for the appropriate instance of the IRulesWrapperClass.

5 RELATED WORK

There are several recent projects related to the use of Web Services in the IRules event and rule-based integration environment. The research in (Benatallah et al., 2002) presents a novel declarative language for composition of Web Services using state charts. A state chart is composed of states and transitions, where transitions are represented as active rules. The *eFlow* system (Casati et al., 2000) is a service process engine that supports specification, enactment, and management of composite e-services. A composite service in *eFlow* is described as a process schema that may include services, decisions or rules, and event nodes. Rules control the business flow while event nodes enable sending and receiving different types of events. The research in (Zeng et al., 2002) presents a rule-based approach for composition of Web Services, enabling dynamic composition of Web Services with run-time activation of rules. The research in (Zeng et al., 2002) provides a declarative language for the composition of services. The service composition models are generated on demand and can be re-configured at runtime.
In comparison to the related work described above, the IRules project follows a database centric approach. The concept of externalized relationships and stored attributes are absent in the Web Services

project of (Benatallah et al., 2002). The research presented in this paper demonstrates the use of IRules functionality within Web Services. The *eFlow* system (Casati et al., 2000) provides features for service processes to be adaptive and change according to business requirements and environment conditions. Similar functionality can be achieved in the IRules system by defining integration rules, which may change according to business requirements to enable adaptive integration of components.

6 SUMMARY

This research has developed a prototype for extending the IRules environment for the use of Web Services as a service interface to EJB components. The IRules Web Service Wrappers have been designed to give a dynamic nature to the IRules integration environment, in addition to supporting the functionality of the IRules environment. This research has also developed a state management facility for the dynamic creation and storage of IRules properties without recompilation of the wrappers, thus enabling dynamic business processes.

There are several directions for future research. One direction involves the incorporation of other component models into the IRules framework using the Web Services interface to demonstrate the component independence of the environment. Furthermore, a Grid Service provide a means for exposing a Web Service as one that provides a set of well-defined interfaces that follow specific conventions. Future research should extend the IRules approach for the integration of Grid Services. The ISL can also be enhanced to provide an XML-based process specification language, such as that of BPEL4WS, together with a mechanism for handling events and integration rules. The processing language can be further extended to support Grid Services.

REFERENCES

Benatallah, B., Dumas, M., Sheng, Q., and Ngu, A. 2002. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. *Proc. of 18th Int. Conf. on Data Eng.* (San Jose, USA), pp. 297–308.

Brown, A., Johnston, S., and Kelly K., 2003. Using Service Oriented Architecture and Component Based Development to Build Web Services Application. Rational-IBM White paper.

Casati, F., Ilnicki, S., Jin, L., and Krishnamoorthy, V., Shan, M. 2000. eFlow: a Platform for Developing and Managing Composite e-Services, Software Technology, Hewlett-Packard, HPL-2000-39 (March).

Curbera, F., Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, S. and Weerawarana 2002. Business Process Execution Language for Web Services, Version1.0. http://www.ibm.com/developerworks/ li-brary/ws-bpel.

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

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

Jin, Y., Urban, S., Sundermier, A., Dietrich, S. 2002. An Execution and Transaction Model for Active, Rule-Based Component Integration Middleware, *Proc. of the Engineering and Deployment of Cooperative Information Systems,* (Sept) (Beijing, China), pp. 403-417.

Kambhampati S, 2003. An Event Service For A Rule-Based Approach To Component Integration, M.S. Thesis, Arizona State University, Dept. of Computer Sci. and Eng.

Kumar, V. 2004. A Prototype for Integration Of Web Services Into The IRules Approach To Component Integration, M.S. Thesis, Arizona State University, Dept. of Computer Sci. and Eng.

Patil, R. 2003. A Framework for supporting an active approach to component based software integration, M.S. Thesis, Arizona State University, Dept. of Computer Sci. and Eng.

Urban, S., Dietrich, S., Na, Y., Jin, Y., Sundermier, A., 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 Systems*, (Hong Kong), pp. 265-286.

Urban, S., Kambhampati, S., Dietrich. S., Jin, Y., and Sundermier, A., 2004. An Event Processing System for Rule-Based Component Integration *Int. Conf. on Enterprise Information Systems*, (Porto, Portugal), pp. 312-319

Web Services, 2000. Web services architecture overview. The next stage of evolution for e-business, IBM Technical Document, Web Architecture Library.

Zeng, L., Benatallah, B., Lei, H., Ngu, A., Flaxer, D., and Chang, H., 2002. Flexible Composition of Enterprise Web Services, IBM T.J. Watson Research Center.