

# The RComponent Re-configurable Enterprise Model – Technical Background

## *Abstract*

This paper presents a technical background to a generic model for building enterprise class applications using adaptable and re-configurable components, known as RComponents. RComponents provide Enterprise Application Integration (EAI) through middleware to integrate application programs, databases, and legacy systems involved in an organisation's critical business processes. Typical applications include banking, payroll, and e-commerce systems. We present a model that is based on a N-tier architecture where the business logic is dynamically adaptable to cope with evolving organisational requirements. An interactive share-dealing scenario is outlined using the adaptable enterprise model and this is coupled to Enterprise Java to provide data tier and distribution services through a servlet tier.

## *Introduction*

The aim of this short paper is provide a scenario where we demonstrate REFLEX RComponents' integration with the Enterprise Java model to provide re-configurability in enterprise class applications. This is achieved by RComponents being used to provide processing logic to stateless session beans running inside an enterprise Java container. The EJBs therefore delegate to the RComponent and the RComponent can allow its evolvable services to be exploited within the enterprise application server. To illustrate this we provide a simple scenario that implements a web-based share dealing service, where the service can be evolved to support new technologies such as WAP, or adding additional behaviour such as a logging service without taking the application off-line.

Structural changes to the code are possible with RComponent delegation within the application server. For example, the addition of share price analysis code, whereby potential customers can more easily track the current share price in respect to previous pricing trends. Such changes can be made to the application dynamically through a service administration interface without taking the application off-line.

The REFLEX RComponent model presents a component based design pattern implementing an Object Request Broker (ORB) that facilitates the deployment of adaptive, dynamically evolvable, distributed applications. Internally, the RComponent uses delegation patterns based on a dynamic proxy [Pemberton, 2001] to achieve its evolvable nature. RComponents consist of three core entities,

1. The metalevel (or configuration interface) and dispatcher, through which all adaptive change to the component is made.
2. External call interfaces, providing a typed invocation interface to the component for its clients, in this case the EJBs.
3. Processing behaviour, i.e. the business logic to be executed by the component, in this case the share pricing code.

It is the decoupling between these three entities that allows flexibility and adaptation within components in terms of interface, and behavioural component evolution. Separation of concerns is therefore the key to evolution in RComponents. Reconfiguration is performed at the RComponent's metalevel through the internal method dispatcher based around the dynamic proxy pattern. The use of introspection in RComponents allows application programmers to determine a program's characteristics at the application's metalevel and react according to the current configuration. RComponents support both **structural reflection** (i.e. the introspection of the component's structure though its attributes and method signatures) & **behavioural reflection**, whereby the behaviour of the

component can be altered at a number of levels such as through behavioural replacement, or method interception. Both these types of reflection are illustrated in the share dealing scenario.

Each RComponent presents a set of interfaces that are used by component clients to invoke the component's provided services through the ORB making invocations location transparent. An XML ADL verifies component deployment with the RComponent also including support for conditional execution and notification.

The following sections of this paper explore the use of RComponents in enterprise class applications to provide configure-ability to business logic tiers in N-tier based applications. This allows RComponents to make use of the services provided by an EJB application server whilst offering the application server dynamic configurability of its components.

## ***Share Dealing Example Scenario***

Our example application to illustrate the RComponent re-configurable enterprise model is from the banking domain providing a basic interactive share dealing service. For simplicity in this case we do not look at non-functional requirements of the service such as security, although of course this is a very important concern for web-base transactions.

The scenario allows us to illustrate the architecture's adaptability in terms of,

- ❑ The addition of new data sources in the data tier
- ❑ The addition of new clients for example, a WAP client, HTTP based interface, or custom Java2 user application
- ❑ The addition of new processing functionality for price trends analysis, or data logging
- ❑ The addition of service monitoring.
- ❑ A web-based configuration meta interface that allows real-time reconfiguration of the service by the administrator where services may be added or withdrawn without taking the software off-line or restarting the application servers. Role-based security is associated with the administrative reconfiguration service to ensure application integrity.

## RComponent Re-configurable Enterprise Model

The RComponent re-configurable enterprise model consists of four tiers to allow for middle-tier component evolution. Each tier has associated test drivers to diagnose errors and localise architectural problems within a specific tier. The tiers are arranged as in the following diagram,

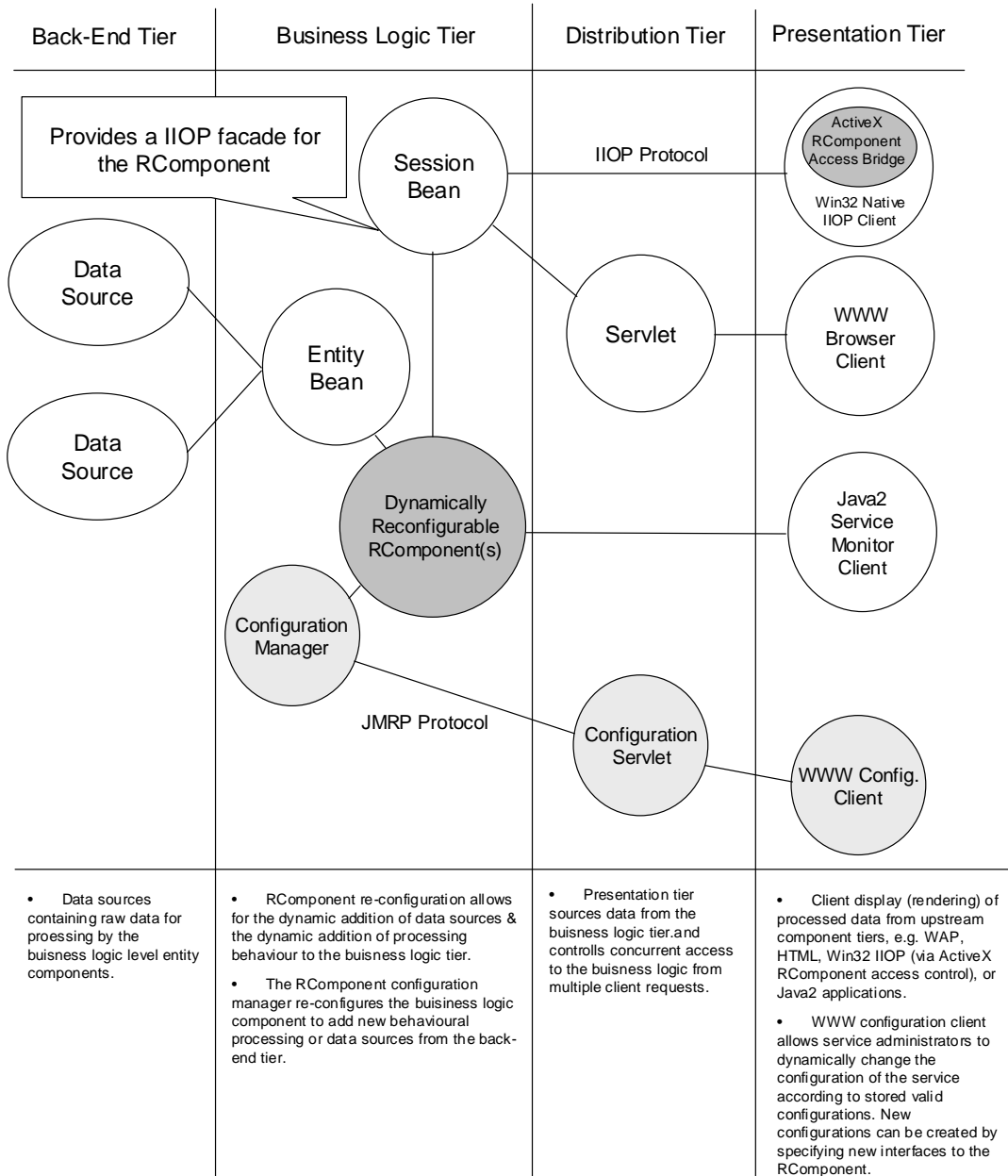


Figure 1 - The RComponent-EJB Enterprise Model

- Back-end tier**, contains the data sources where information is stored using either relational data models, or an OODB.
- Business logic tier**, contains the DAOs/Entity beans (to abstract over the actual data sources), and the RComponents that manage the evolution of the business logic tier. The tier can be

evolved through the interface presented by the RComponent's metalevel, called the configuration manager. This allows services to be replaced and new data sources to be added to the application. The session bean provides an IIOP (CORBA) façade [Gamma, 1994] to the RComponent.

3. **Distribution tier**, contains a servlet that distributes the presentation of application data to the clients.
4. **Presentation tier**, clients of the service may be WAP browsers, WWW browsers, IIOP or Java2 application clients.

The architecture of the example scenario is presented in figure 1, above.

## ***Integration With Enterprise Java (EJB)***

Enterprise Java is a server-side component architecture for writing reusable business logic and portable enterprise applications. EJB is the basis of Sun's Java 2 Platform, the Enterprise Edition (J2EE). Enterprise Java was first introduced in 1998 by Sun Microsystems as an architectural platform for the development of enterprise class applications. J2EE is an evolution of several Java technologies, such as Java Database Connectivity (JDBC), and Java Remote Method Invocation (RMI), Java Message Service (JMS), and Java Naming And Directory Interface (JNDI). The cornerstone of J2EE is the Enterprise JavaBeans (EJB) technology, a standard for building server-side components. The Java Enterprise framework provides a very feature rich platform for the implementation of evolvable data driven distributed applications.

Enterprise JavaBean (EJB) components are written entirely in Java and run on any EJB compliant server. EJBs are operating system, platform, and middleware independent, preventing vendor lock-in. EJB application servers provide system-level services such as transaction support, security, threading, and persistence services within the container environment. The EJB architecture is therefore inherently transactional, distributed, multi-tier, scalable, secure, and wire protocol neutral. Any protocol can be used: IIOP, JRMP, HTTP, DCOM etc. EJB applications can serve assorted clients: browsers, Java, ActiveX, CORBA, etc. EJB can also be used to wrap legacy systems. In our case we use EJB components to delegate to RComponents to provide support for evolution within Enterprise applications we can therefore take advantage of the services provided by EJB application servers, whilst using the evolution services of RComponents.

J2EE's underlying component model supports limited customization without requiring access to component source code. Application behaviours and runtime settings are defined at deployment time through a set of attributes (component properties) that can be changed at deployment time via XML descriptor files associated with each component. The application server's vendor often supports application creation, deployment, and bean configuration by providing a number of configuration (sometimes known as deployment) tools.

XML files are used by the EJB management container to define the Enterprise JavaBean's component activation, transaction, and persistence model. All middle tier logic components are fully managed by the EJB container for the application clients. Therefore with Enterprise Java as an architectural framework we have a readymade flexible, adaptable structure providing service support for evolvable software in distributed information management systems. The problem with this support in EJB is the evolution is static rather than dynamic in its nature, meaning that often applications have to re-deployed in order for re-configuration to be possible, therefore the application may be off-line while this takes place. By using RComponent delegation with EJBs we can take advantage of the DAO facilities, transactions, and security built into the EJB application server whilst also supporting dynamic runtime application evolution. For example, with RComponent reflection we can discover the internal workings of the business logic tier and inject new behaviors into the application or change the application's existing behavioral nature in order to cope with changing organizational pressures.

## **Summary**

The paper has outlined the need for evolution in existing enterprise application servers, and web services. A technical description outlining a model for re-configurable and hence dynamically evolvable enterprise class applications was presented.

The paper argues that having EJB components that delegate behaviour to RComponents will allow for more flexibility within deployed EJB applications in terms of evolving the application after deployment dynamically. Using EJB to RComponent delegation allows changes to both the structure and behaviour of the application at runtime. For example, new data sources can be added, or processing logic inserted to augment the existing application through the RComponents. This is behaviour is typically not supported without EJB delegation to RComponents described in this paper. We describe a web-based share pricing service as an example, as demonstration of which is available at,

<http://egb006000007.lancs.ac.uk/REFLEX/demo/>

Currently, the interactive web based share dealing service is being evaluated to evaluate and refine the proposed architectural model.

## **References**

- [Monson-Haefel, 2000] Monson-Haefel, R.: “Enterprise JavaBeans”, (O’Reilly, 2000, 2<sup>nd</sup> Edition).
- [Pemberton, 2001] Pemberton, D.: “RComponents - Reflexive & Adaptable N-tier Components”, [publication details to follow].
- [Gamma, 1994] Gamma E., and Helm R.: “Design Patterns Elements of Reusable Object-Oriented Software”, (Addison Wesley Professional Computing Series, 1994).