Deploying Rule Applications with ILOG JRules
Deploying Rule Applications with ILOG JRules

White Paper
# Table of Contents

Introduction .........................................................................................................................................................3

ILOG JRules Deployment Philosophy ....................................................................................................................3

Zero Development ......................................................................................................................................................4

Zero Deployment .....................................................................................................................................................4

Zero Administration ...................................................................................................................................................5

Rule Execution Server ...............................................................................................................................................6

1. IDE Integration ......................................................................................................................................................9
2. Java Enterprise Edition Execution Components .....................................................................................................10
3. Management Console Ant Tasks ...........................................................................................................................10
5. Web Management Console .....................................................................................................................................11
6. Ruleset Management Model (JMX) ..........................................................................................................................12
7. Execution Unit ........................................................................................................................................................14
8. Execution Unit Management Model (JMX) ................................................................................................................15
9. Persistence Ant Tasks .............................................................................................................................................15
10. File Persistence ....................................................................................................................................................15
11. JDBC Persistence ...............................................................................................................................................16
12. Cluster-aware Cache Invalidation (JMX) ................................................................................................................16
13. Rule Team Server Integration ............................................................................................................................16
14. Rule Scenario Manager Integration ....................................................................................................................17

Core Rule Engine APIs ..........................................................................................................................................18

Conclusions ..............................................................................................................................................................20
Introduction

Developing, deploying and administering any mission-critical application is difficult. It is one of the most challenging of the cross-cutting concerns, those requirements that slice through your IT infrastructure and require just the right interplay between hardware, software, people and their supporting processes. ILOG JRules alone cannot make your application scalable or robust; however, many of its features have been designed to fit within an overall strategy for deploying mission-critical applications. This white paper describes ILOG JRules’ deployment features, and is intended for software engineers, architects and their managers who make implementation and deployment decisions.

ILOG JRules Deployment Philosophy

Much of the design philosophy for deploying software with ILOG JRules is neatly summarized by Ted Neward's principle of Three Zeroes:

1. Zero development favors reusable software assets to maximize quality and return on engineering investment.
2. Zero deployment encourages architects to understand the cost of software changes and minimize the impact of these changes on administrators and end users.
3. Zero administration ensures applications are easy to administer, acknowledging that while an application may be developed in one year, its administration may span 10 or more years.

In addition to these three general principles, ILOG JRules provides software components and tools that reduce the total cost of ownership (TCO) of your deployed applications. TCO plays an important role in determining the long-term success of a project, and typically determines whether a developed application is perceived as “legacy” infrastructure or remains an active component of the overall IT system.

Of course, no application can be truly Three Zeroes, but by understanding the impact of these factors early in the application design and striving to minimize it, we can reduce the chances of a grave mistake in a mission-critical application. In this paper, we will use the Three Zeroes to guide our discussion on how ILOG JRules can impact the TCO of your applications.

In addition to these general principles, ILOG JRules imposes a “Chinese wall” between the tools and processes used for business rule management (Rule Team Server and Rule Studio) and the tools and processes used for production deployment (the core rule engine API and the execution components of Rule Execution Server). This is because business rule management is a “back-office” development activity (whether the “developers” are software engineers or policy managers), while executing sets of rules, or “rulesets,” is a front-office production activity. This allows applications running in production to be fully isolated (in terms of process, tools and network infrastructure) from servers being used for development, including the server that hosts the working business rule repository. A clear deployment artifact, the RuleApp, passes between the development and production environments, ensuring configuration auditability and enabling version management for server configurations.

To take this principle a step further, the Rule Execution Server imposes an additional

---

“Chinese wall” between ruleset execution and runtime administration. Rulesets can be executed by clients, irrespective of whether the Rule Execution Server management console is deployed, enabling very precise control of the production environment, tight security and increased robustness.

**Zero Development**

The principle of Zero Development is a caution to avoid reinventing the wheel. The more software components reused in the scope of a project, the more time software developers have for implementing end-user requirements. Enabling software developers to focus on implementing end-user requirements, rather than building infrastructure, also directly reduces the TCO of the final application.

The business rule approach in general is an important tool for moving toward Zero Development. A business rule management system (BRMS) empowers business users to own and manage their own business rules, allowing the IT group to spend more time on the architecture and infrastructure that will ensure a mission-critical application meets its service level agreements.

ILOG JRules includes a number of reusable software components, provided as application programming interfaces (APIs), that are used when building and deploying mission-critical applications:

- **Java Standard Edition-compatible APIs for the core rule engine and associated classes** (IlrContext, IlrRuleset). These APIs allow a ruleset to be parsed and executed with maximum flexibility.
- **Java Standard Edition-compatible APIs for the Rule Execution Server**, including both runtime management APIs (IlrJmxMutableRuleApp) and execution APIs (IlrSimpleRuleSessionProvider). These APIs implement the typical use cases for managed execution, including rule engine pooling, hot-deployment and long-lived persistence.
- **Java Enterprise Edition APIs for the Rule Execution Server**, allowing invocation of the rule engine from transactional, secure, remote, local, synchronous and asynchronous clients (IlrManagedRuleSessionProvider and IlrRemoteRuleSessionProvider, as well as JMS clients that conform to the message signature specified for the Rule Execution Server Message Driven Bean).

We will return to the capabilities of the Rule Execution Server in a subsequent section.

**Zero Deployment**

Applications you develop must be installed and made available to your end users. The architectural deployment choices you make have a major impact on TCO. One of the objectives behind server-based development in general has been to minimize the time and cost of performing client desktop deployments across large enterprises. The good news is that ILOG JRules can bring major TCO benefits when evaluated purely in terms of deployment processes. By choosing to externalize your business policies and data from your application code, your deployments can be made faster, more fail-safe and cheaper. For example, a typical ILOG JRules rating application might support the following deployment channels:

1. Live update of the rating tables stored in a central relational database
2. Live update of the business rules that determine the business policies for rating decisions
3. Rolling update of the rating application code running on an application server

Changing central rating tables can be made extremely efficient. A simple Web GUI may enable an administrator to edit and commit the rating tables while the rating application is still running. Alternatively, a batch edit-backup-update process may be put in place to ensure audit and traceability requirements are met.

Changing the business rules that govern rating can be done through a “hot deploy” process directly to the production application server, with zero application downtime. Typically, the new business rules will have gone through an extensive testing cycle on a staging server prior to hot deployment to the production server. The support for hot-deployment of rulesets is one of the key features of the Rule Execution Server, and discussed in detail in a subsequent section.

Changing application code to fix bugs or introduce new capabilities will still be necessary; however, these infrastructure changes can now be largely decoupled from changes driven by business policy. A manual or automated process of rolling updates is typically put in place, perhaps updating one data-center before another and switching loads between them, or using the application server’s high availability (HA) features to perform rolling application upgrades.

Zero Administration

If your application is successful, it will spend more time running in production than it took to develop. System administrators are often the first line of support for your application, and it is time and cost effective to provide them with diagnostic tools to accurately predict, troubleshoot and resolve issues as autonomously as possible. The ability to hot-deploy rulesets brings tremendous agility and power to your applications. However, with great power comes great responsibility! Within such a dynamic and agile environment, system administrators require more than cryptic log files, stack traces and error messages to do their job. At a minimum, the system administrator needs to be able to determine which rulesets are deployed to a server.

Providing administrators with more information, such as runtime performance statistics, comprehensive logs, remote management, warning and error alerts, automated regression testing, installation diagnostics and ruleset version management, helps create a relationship between developers and administrators based on mutual trust and cooperation. Each side of the relationship can focus on their core skills and bring value to the organization, rather than apportioning blame or wasting valuable time tracking down issues. The net impact on the organization is reduced TCO for the deployed application, as administrators and developers can work more productively over the deployment lifetime of the application.

The Rule Execution Server Web console provides a comprehensive runtime view into the deployed rulesets and their execution performance. It empowers both system administrators and developers by allowing them to work independently or collaboratively to ensure applications behave as predicted within the development, staging and production environments.
Rule Execution Server

The most critical ILOG JRules reuse decision is whether to base the application architecture on the core engine API (IlrContext et al, see below) or the Rule Execution Server. Unless you have very unusual performance or runtime ruleset management requirements, the recommended approach is to deploy the Rule Execution Server, and it should always be evaluated during application design. The Rule Execution Server provides many of the execution services required for successful mission-critical deployments:

- Efficient vertical and horizontal scalability
- Tight integration with host application servers
- High-performance execution of rulesets
- High-availability, with the ability to hot-deploy changed rulesets to a running server
- Detailed technical logging
- Audit traces from the rule engine
- Remote ruleset debugging from the Rule Studio IDE
- Support for clustered execution

The Rule Execution Service is composed of several components that can be deployed alone or embedded within your application. Some of the components make use of features in the Java Enterprise Edition, while others are based on the Java Standard Edition and are usable outside an application server or within a micro-container. A typical Java Enterprise Edition logical deployment is shown in Figure 1.
The Java Enterprise Edition application is hosted on an application server. In addition to the deployment of the end-user application, the Execution Unit for the rule engine is deployed along with the Management & Monitoring console for the Rule Execution Server. Both the Execution Unit and the Management & Monitoring console share persistence services that can connect to a file system or relational database.
A typical stateless execution sequence using the Rule Execution Server components is shown in Figure 2. The client application creates a stateless Rule Session using the Rule Session Provider and then populates a Ruleset Execution Request, which is passed to the Rule Session to be executed.

The Rule Session delegates to the Execution Unit, which works with the Persistence layer to retrieve the ruleset from the runtime store (if required) and then updates an in-memory cache. Once execution is complete, the client retrieves the calculated output parameters from the Rule Session.

The complete suite of Rule Execution Server software components is shown in Figure 3. The section that follows discusses the capabilities and TCO impact of each component.
1. IDE Integration

Rule Studio includes a number of capabilities specific to the Rule Execution Server. These include:
- Hot deployment of RuleApps to a local or remote Rule Execution Server
- Management of multiple Rule Execution Server configurations within the workspace
- Wizards to generate code to invoke rulesets deployed to the Rule Execution Server
- Remote debugging of rulesets deployed to the Rule Execution Server

Tight integration into the Rule Studio IDE provides an efficient development experience for the technical users of your business rule application.

Integrating the Rule Execution Server with the Rule Studio IDE lowers the TCO of applications through increased developer productivity during the development and maintenance phases.

- Java developers can quickly and easily generate the code to invoke a ruleset deployed to the Rule Execution Server.
- Quickly find errors in business rules or integration code by setting breakpoints within business rules and executing locally or remotely on a Rule Execution Server.
• Changed business rules can be deployed to multiple servers in just a few clicks, ensuring efficient integration testing.

2. Java Enterprise Edition Execution Components
The Rule Execution Server includes reusable software components to ease integration of the rule engine into your Java Enterprise Edition applications. They include:
  • Stateless Session Enterprise JavaBean
  • Stateful Session Enterprise JavaBean
  • Message-Driven Enterprise JavaBean

The Rule Execution Server provides a set of out-of-the-box execution components that can be integrated into most common Java Enterprise Edition deployment architectures. These components allow your application to easily invoke the rule engine while benefiting from the security, transaction, remote invocation and pooling services provided by the application server.

Reusing the software components provided with ILOG JRules allows you to concentrate on the specifics of your application infrastructure and end-user requirements, and build higher-quality applications faster.

3. Management Console Ant Tasks
The Rule Execution Server includes automation tasks, or Ant Tasks, for common project actions. Based on the Apache Ant project automation tool, the Ant Tasks include:
  • Creating RuleApp archives for deployment
  • Deploying RuleApp archives to local or remote servers
  • Undeploying RuleApp archives from local or remote servers
  • Downloading an entire server configuration to disk

The Ant Tasks allow you to put in place a fully automated continuous integration build system for your application. Build and deployment automation, along with server configuration management, are key to building and deploying mission-critical applications. By automating repetitive build and deployment tasks, you can perform these tasks at lower cost and with fewer possibilities for human error.

Project automation serves as an important factor for reducing TCO for developed applications by decreasing the time required to make changes and increasing the quality of your change management processes.

The Rule Execution Server includes reusable software components to ease integration of the rule engine into your Java Standard Edition applications. They include:
  • Stateless JavaBean (Plain Old Java Object – POJO)
  • Stateful JavaBean (Plain Old Java Object – POJO)

These components allow you to benefit from many of the services provided by the Rule Execution Server outside an application server. They can be used to facilitate out-of-container testing and in command-line batch applications, as well as run in a micro-container or Servlet container.
ILOG JRules ensures you get to make the right integration decisions based on your internal enterprise architecture standards or specific project requirements. This flexibility drives TCO lower by allowing software developers to efficiently integrate calls to the Rule Execution Server from any modern Java Runtime Environment.

5. Web Management Console
The Rule Execution Server Management Console is a graphical Web user interface for server management. It has many powerful features, including:

- View and edit deployed RuleApps
- Upload and deploy RuleApps
- Browse deployment metadata for all resources
- View Execution Units (rule engines) deployed and available within a cluster
- View runtime execution statistics, including the performance of individual rulesets on specific machines in a cluster
- Browse consolidated error, warning and information logs
- Run detailed installation diagnostics
- Test ruleset invocation using a script interface

![Rule Execution Server Console](image)

*Figure 4: Rule Execution Server Console, showing the execution statistics for a deployed ruleset*

The Web management console provides a very rich graphical environment for system
administrators responsible for managing and monitoring production applications. It can also be used by developers to troubleshoot development issues.

The RES Console lowers the TCO for applications by supporting the common needs of system administrators. Efficient change management requires administrators to have accurate and up-to-date knowledge of the running application:

- **What** rulesets and resources are deployed?
- **When** were resources deployed or modified?
- **Where** is the console running, as well as where are the Execution Units deployed within a cluster of machines?
- **Who** has access to the console (checked through application server security configuration)?
- **Why** has performance been impacted (checked by viewing real-time execution statistics for rulesets and machines)?

6. Ruleset Management Model (JMX)

The Web Management Console displays and manipulates information in the Ruleset Management Model. This model is presented as a set of named Java Management Extensions (JMX) Managed Beans. You can interact with the Management Model directly from Java programs, scripting languages or third-party JMX-aware tools such as the Open Source MC4J, jManage and jConsole, or many commercial management tools, including HP OpenView and IBM Tivoli. By manipulating the Management Model, you can automate many management tasks:

- Receive notifications when new management entities are created
- Write programs or scripts to automatically deploy, undeploy or update resources
- Track runtime statistics for reporting purposes or to generate real-time alerts

![Figure 5: Using jManage to browse the Rule Execution Server management model](image)
Figure 6: Using jManage to create an e-mail alert for service level agreement violations for a ruleset

Figure 7: Using jManage real-time charts to assess performance and throughput
7. Execution Unit
The Execution Unit is the packaging of the rule engine within the Rule Execution Server architecture. It is deployed as a Java Connector Architecture (J2C) Resource Adapter. The J2C specification defines a contract between the J2C Container (typically an application server) and the ILOG JRules Resource Adapter. Through this contract, the application server provides deployment, caching and management facilities for the rule engine. In addition, the application server has management control over the resource adapter and can dynamically allocate additional engines or make other resource balancing decisions to optimize the overall throughput of the application server.

Note that calls to a J2C Resource Adapter are always local calls, hence the performance overhead is relatively low.

The Execution Unit Resource Adapter can be deployed in three modes:
- **JEE Shared**: the Execution Unit is deployed to the application server and shared (and possibly accessed) by all the applications deployed to the server. This is analogous to installing a device driver into an operating system, in that the application server has become globally enhanced with ruleset execution capabilities. From an administrative perspective, one version of the Execution Unit is deployed and can be easily upgraded, started, stopped and monitored using the application server management console or other tools.
- **JEE Isolated**: the Execution Unit is deployed within a single application (packaged
inside a JEE Enterprise Archive). This is a more advanced deployment option that enables multiple applications to be deployed to the same server, and for each application to use completely separate versions of ILOG JRules classes.

- **JSE Embedded**: the Execution Unit is deployed as a simple Java Standard Edition JAR and ILOG JRules implements a lightweight J2C container, as well as the Resource Adapter. This mode allows the Execution Unit (and hence, the Rule Execution Server) to be used outside a JEE application server. This mode can also be used to implement class isolation scenarios on an application server. Note that the application server no longer manages the Resource Adapter in this mode and that the ILOG JRules pooling infrastructure is used.

8. **Execution Unit Management Model (JMX)**

The Execution Unit also publishes information and actions through JMX Managed Beans. In particular, JMX is used by the Management Console to discover running Execution Units to provide the following capabilities:

- Retrieval of runtime execution statistics for each Execution Unit deployed
- Hot-deployment notification to invalidate runtime caches

Clusterwide JMX Servers (as provided by most application servers) allow the RES Management Console to discover and interact with multiple Execution Units no matter where or how they are deployed across the cluster. Providing a single administration facility for all rule engines deployed across a cluster of machines allows system administrators to very quickly gain an understanding of the deployment topology and diagnose runtime or configuration errors.

9. **Persistence Ant Tasks**

Ant automation tasks are provided for the persistence layer of the Rule Execution Server. These tasks allow deployments to be performed directly to the persistent store, without having to have an application server running or the Management Console deployed. Note that using these tasks while Execution Units are running will not invalidate runtime caches. The Execution Units should either be restarted or the IlrManagementSession API should be used to explicitly invalidate local caches.

The Ant tasks provide the following capabilities:

- `bres-write-file` deploys a RuleApp to the file system
- `bres-write-db` deploys a RuleApp to a database
- `bres-delete-file` deletes a RuleApp from the file system
- `bres-delete-db` deletes a RuleApp from the database

The persistence Ant tasks are very useful for project automation, as they allow RuleApps to be deployed from scripts without having to deploy the Rule Execution Server management Web console. They also provide automation capabilities for out-of-container applications, and often batch processing applications or unit or regression tests.

10. **File Persistence**

Rule Execution Server maintains a long-lived store of deployed RuleApps. This store can be configured to be on the file system as a simple set of files under a directory structure. This mode is particularly useful for lightweight Java Standard Edition deployments (or for unit tests) where the overhead for configuring and using a database is considered inappropriate.
File persistence is not intended for use in clustered configurations, where JDBC persistence is recommended.

11. JDBC Persistence
The Rule Execution Server maintains a long-lived store of deployed RuleApps. This store can be configured to use a database accessed through the JDBC API. The Rule Execution Server supports all commonly used development and production databases, including:

- Oracle
- IBM DB2
- IBM Cloudscape
- Sybase
- MySQL
- Apache Derby
- Pointbase
- MS SQL Server

JDBC Persistence can be used in any environment that provides an implementation of the JDBC DataSource interface (required by JEE), including Apache Tomcat.

12. Cluster-aware Cache Invalidation (JMX)
When a RuleApp is hot-deployed through the Rule Execution Server Web console, the caches on all running Execution Units are notified so that they can reload the new ruleset contents. This cache invalidation is based on JMX technology and is cluster-aware. For example, if eight servers are running within a cluster, when a RuleApp is deployed, all instances of the Execution Unit will be notified on all eight machines.

This cluster-aware capability ensures that redundant servers can be used for mission-critical applications. Through this capability, Execution Units are not a single point of failure within your application architecture, ensuring that if a server goes down, a sibling server within a cluster has access to the same deployed RuleApps.

13. Rule Team Server Integration
Rule Team Server includes a policy-manager deployment facility to generate and then deploy a RuleApp to Rule Execution Server. This deployment is performed programmatically through the Rule Execution Server Web console, ensuring that all the running Execution Units are automatically notified of rule updates.
Rule Team Server deployment support allows nondevelopers to deploy RuleApps directly to a running Rule Execution Server. This feature, often used in concert with Rule Scenario Manager, allows changes to rulesets to be quickly verified within a staging or test environment.

14. Rule Scenario Manager Integration

Rule Scenario Manager is an add-on module that manages and executes test and simulation scenarios. Rule Scenario Manager connects to running Rule Execution Servers to execute tests and simulations. The Rule Execution Servers pass runtime information back to Rule Scenario Manager, including details on fired rules and executed tasks.
Figure 10: Use Rule Scenario Manager to define and execute tests and simulations against Rule Execution Server

By integrating Rule Scenario Manager with the Rule Execution Server, you can put in place automated regression tests that run against staging or development servers. Integrating Rule Scenario Manager within your quality assurance processes ensures that applications deployed to production are high in quality and have passed a rigorous group of regression tests.

Core Rule Engine APIs

The core rule engine APIs provide the most flexibility in your application’s interaction with ILOG JRules. A typical client application can load a ruleset from a hard disk or other storage location, parse the ruleset into its runtime representation, create a rule engine bound to the ruleset and then execute the rule engine instance with incoming application data.

Using these low-level APIs, your application can (and typically must) provide its own pooling infrastructure for rulesets, as ruleset parsing time take several seconds or more. All runtime management facilities, including hot deployment and cluster-aware ruleset cache invalidation, must also be implemented through application code. Applications based on the core rule engine APIs do not integrate out of the box with other ILOG JRules modules, such as Rule Studio for ruleset hot deployment, Rule Scenario Manager for testing and simulation, and Rule Team Server for ruleset hot deployment. These integration scenarios will be discussed in detail in the section on Rule Execution Server.
The core rule engine APIs are Java Standard Edition compatible, single threaded (i.e., do not create background threads) and multithread safe. They can, therefore, be integrated into a very wide variety of application host environments, including application servers, servlet containers, message-oriented middleware and command-line batch applications.

The core rule engine API is typically recommended for advanced users with unusual requirements or independent software vendors implementing ruleset execution within a proprietary runtime environment.

Figure 11 shows the typical interactions between a Java client and the ruleset and rule engine. Once prepared, the ruleset is associated with an engine context and can be executed as many times as necessary.
Conclusions

ILOG JRules includes a wide array of product capabilities that provides a great deal of flexibility in how you choose to deploy your business rule applications. These capabilities range from the most lightweight JAR-based deployments based on the core rule engine API to a fully JEE-hosted Rule Execution Server with a powerful integrated management console.

In addition to runtime software components, ILOG JRules includes tools that can be tightly integrated into your processes for software development, quality assurance and production. These tools work in an integrated fashion to allow you to easily automate key project tasks.

Throughout the project development life cycle, ILOG JRules facilitates your development, deployment and administrative tasks, lowering the total cost of ownership for your developed applications, and helps you meet the challenges of building and supporting mission-critical applications.