PROGRESS® Corticon®

WHITE PAPER

www.progress.com

DEPLOYMENT ARCHITECTURE FOR JAVA ENVIRONMENTS

PROGRESS

TABLE OF CONTENTS

INTRODUCTION

The Progress® Corticon® Business Rules Management System ("Corticon BRMS") fits naturally in today's service-oriented architectures (SOA), deploying as a service, and leverages the enterprise-class performance, scalability, and high-availability features of leading application servers. It has been designed to support standards critical to enterprise customers building and integrating composite applications. The Progress® Corticon® Server is the patented "no-coding" rules engine for Corticon BRMS, with performance unmatched in the industry. This document describes Corticon Server's JAVA runtime architecture with particular emphasis on how Corticon Server fits within various enterprise environments.

PROGRESS® CORTICON® PRODUCT ARCHITECTURE

The Corticon BRMS consists of a set of rule modeling tools ("Corticon Studio") and deployment tools ("Corticon Deployment Console"), along with a highly scalable execution environment ("Corticon Server"). The business rules are modeled as rule sets and deployed as Corticon decision services.

Client applications call decision services to apply decision-making logic (i.e., business rules) as needed within the application flow. These fully encapsulated decision services can be invoked as web services or in-process, depending on technical requirements. A single instance of Corticon Server can supply decision services to diverse client applications. Figure 1 illustrates the Corticon BRMS architecture for Java environments.

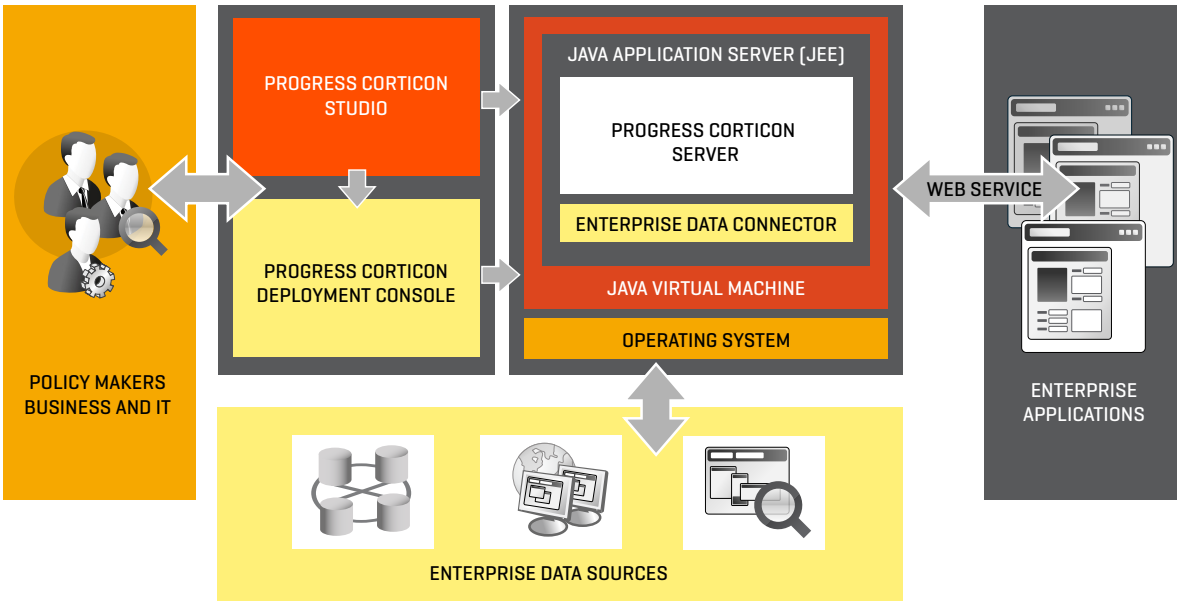

Figure 1
Corticon product architecture for Java environments

PROGRESS

Consider the following:

1. Each decision service encapsulates the logic of a single decision-making activity (e.g., calculate a rate, verify a claim, accept an applicant). Corticon Studio provides tools to analyze and test the logic, ensuring that it is complete and unambiguous; it always provides one, and only one, correct answer.
2. Corticon Server may be deployed in a Java EE web/application server or may be called in-process from a Java client application.
3. Decision services may be deployed into Corticon Server using a Corticon Deployment Descriptor ("CDD") or via Corticon Server APIs. All necessary deployment artifacts, including the executable services and WSDLs, can be generated by Corticon tools.
4. Client applications include any business program that uses decision services or business process management (BPM) systems that manage many activity steps in a business process flow.
5. Client applications invoke Corticon decision services by first constructing an XML message or by creating a collection of Java business objects that contain business facts. The client application communicates this information to Corticon Server via SOAP, REST/JSON, JMS, RMI or an in-process call.

DEPLOYMENT OPTIONS

Corticon decision services conform to SOA. Each decision service:

- Automates a business decision-making activity
- Is implemented by a set of business rules
- Is managed by software developers or business analysts

In each enterprise, the application architect needs to determine:

- How decision services become part of the enterprise architecture
- Which applications utilize decision services
- How client applications will invoke the decision services

These choices depend on the current and future technical requirements of the enterprise. The available options are summarized in the following table and addressed in detail below:

CORTICON DEPLOYMENT OPTION	APPROPRIATE FOR	MESSAGING OPTIONS
Option 1: Remote Server Application Server: Java EE, Spring Applications invoke decision services remotely as web services	▸ Service-oriented architectures ▸ Distributed architectures ▸ Rule modernizing legacy systems with ability to make calls to remote decision services	▸ XML ▸ SOAP ▸ REST/JSON ▸ JAX-RPC ▸ JAX-WS
Option 2: In-process Server: Applications invoke decision services directly via Corticon Server APIs	▸ Systems that require the fastest possible performance ▸ Systems that require tight coupling of client application with the Corticon Server	▸ XML ▸ Java objects

OPTION 1: REMOTE SERVER

This option requires the Corticon Server to be deployed in a multi-threaded Java EE or Spring server. Decision services can be invoked as web services using any of the protocols supported by the application server (e.g., SOAP, REST/JSON, JAX-RPC, JAX-WS). To prepare for web services deployment, the Corticon Deployment Console is used to generate CDDs (Corticon Deployment Descriptor files) and WSDLs/XSDs (service contracts).

A CDD is an optional deployment descriptor file which defines a set of decision services to be deployed in Corticon Server. The CDD specifies the locations of rule sets, performance-related parameters and other administrative details that the server needs to effectively deploy the decision services. A WSDL/XSD is effectively a service contract for a decision service. Each decision service has a WSDL/XSD that defines the decision service's expected inputs and outputs.

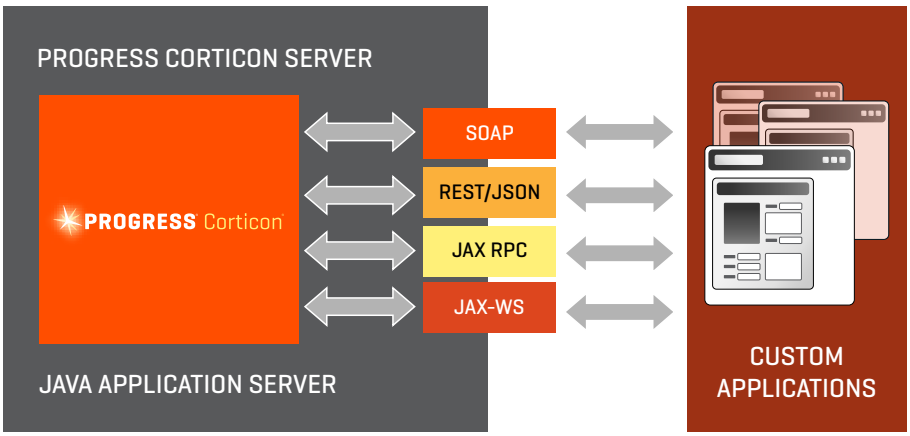

Figure 2
Corticon Server deployed in application server

OPTION 2: IN-PROCESS SERVER

Client applications can invoke decision services and perform administrative functions in-process via the Corticon Server API. This option can deliver high performance but lacks the flexibility and loose coupling that web services can provide.

Using Corticon Server as an in-process component, client applications have the option to communicate via XML or Java objects (POJOs). Using Java objects can yield superior performance by eliminating XML translation overhead.

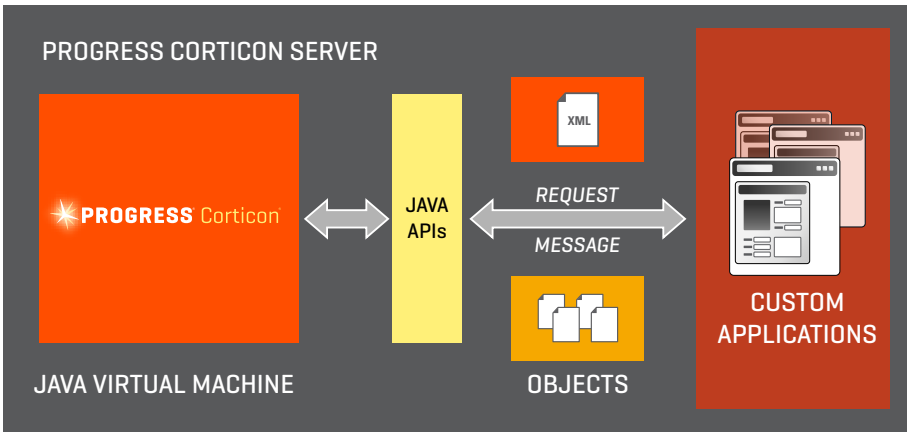

Figure 3
Corticon Server deployed in-process

Out-of-the-box, the Corticon Server is deployed in a Tomcat application server. The Corticon Server also ships with sample Java EE containers (e.g., servlet and session bean), which can be used as-is or customized and deployed in a Java EE or Spring application server. The Corticon Server JARS can also be added to any Java application classpath for in-process use.

INVOKING CORTICON DECISION SERVICES

Once Corticon decision services have been loaded into Corticon Server, they are ready to be invoked by client applications. The integration approach differs depending upon the selected deployment option. The service requests contain rich payloads that can be carried either via XML or Java objects. The contents of the payloads are analyzed in more depth below.

MESSAGING

Messaging between a client application and a decision service is conceptually identical across deployment approaches, regardless of the transport protocol used.

1. The client application sends a request message to the Corticon Server, containing a payload of data and targeted at a specific decision service.
2. The Corticon Server invokes the appropriate decision service, which processes its rules against the request payload.
3. When the rules processing is complete, the Corticon Server returns a response message.

REQUEST MESSAGE

The request message consists of data to be sent to the rules engine. The payload is a collection of application-specific entity instances that are defined in the user's vocabulary. Entity instances may have any number of attributes and may have associations with other entity instances. The collection of entity instances are the facts upon which the rules engine operates.

RESPONSE MESSAGE

The response message payload includes two discrete sets of data: data payload and posted messages. The data payload contains a copy of the incoming data that has been updated as a consequence of rules firing. Entity instances may be created, updated or removed. In addition, associations between entity instances may be created or removed. In other words, the response message data payload reflects the state of the decision service working memory after all rules have fired.

Posted messages contain an audit trail of rules fired by the decision service during the processing of the request, including the sequence of firing. The messages are posted using natural-language rule statements.

USING SERVICE CONTRACTS (WSDLS OR XML SCHEMAS)

The Corticon request and response messages must adhere to the message structure and data specification as defined in the service contracts. The service contracts can be generated for each Corticon decision service (one service contract per decision service) using the Corticon Deployment Console, as either WSDL or XML schema files (XSDs). Service contracts are helpful

integration specifications used by application developers to assemble services into larger applications.

WSDL is used when the customer has chosen web services architecture. The WSDL file defines a complete interface description, including:

- Decision service name
- Request message structure
- Response message structure
- SOAP envelope and binding information

XSDs can be used in conjunction with the in-process deployment option. The XSD file is identical to the WSDL file minus the SOAP envelope and binding information. When using XML payloads, the XSD file defines the necessary structure of the XML document (i.e., the XML document can be validated against the XSD using a validating parser).

You also have the option to use REST/JSON for executing decision services, in which case you do not have to adhere to a service contract.

CORTICON RULE ENGINE

Corticon Server employs an engine that has been built specifically to address the needs of business decision automation within the context of a business process. In decision automation, efficient processing of business rules and consistent results are essential to achieve fully automated straight-through-processing ("STP").

Most rule engines analyze rules during execution. This means significant processing is taking place when systems are looking for an answer from the rules system. The traditional pattern matching algorithm to efficiently process and execute rules is Rete. Rete-based inference algorithms are built to address the needs of expert systems in which rules are used for decision support, not decision automation. In those systems a person typically queries a large rule base with a question, and the system converges on zero or more (sometimes conflicting) answers. Because a human interprets the results, logical inconsistencies (such as no answers or conflicting answers) are tolerated and, in fact, architecturally required. In order to achieve STP, decision automation requires consistent answers (guaranteed, conflict-free results) each and every time because typically no human expert is available to be the final arbiter.

Furthermore, Rete-based inference engines are designed to allow individual rules to be dynamically added or removed to/from the rule base, as this is a requirement for expert systems. Using a one-activated-rule-at-a-time process to support its dynamic conflict resolution architecture, the Rete algorithm inherently produces a processing bottleneck. While Rete scales well with an increasing number of rules, it degrades exponentially with the increasing complexity of data, resulting in a performance scalability problem well known as the "Rete wall." Corticon took a radically different approach for decision automation. Decision services, which are made up of sets of rules, are deployed/undeployed as a unit,
so the rule engine does not need to support dynamic tweaking of individual rules.

The Corticon engine was designed from the ground-up for decision automation, not decision support. Since Corticon Studio helps the rule author identify and resolve logical conflicts during the rule modeling phase, the Corticon engine's smart compilers automatically generate optimal processing sequences for any decision service. This results in a best-of-both-worlds solution as Corticon delivers blazing performance that scales linearly with both the number of rules and the complexity of the data.

Corticon decision service execution is stateless, where all state is maintained in the message payloads. A single incoming message payload seeds the engine working memory after which rules processing commences. Once rules processing ceases, the final state of the engine working memory is returned as an outgoing message payload along with an association rule audit log.

ENTERPRISE DATA CONNECTIVITY

Corticon supports two distinct modes of operation:

1. Requiring the client program to supply all input data as specified by the service contract
2. Allowing Corticon Server with EDC (enterprise data connectivity) to dynamically access data in an enterprise data source (e.g., relational database) as needed during execution

In the first mode of operation, the rule author can decide what information must be passed to the decision service. This is done through a combination of the vocabulary, the rules written using that vocabulary, and the service contract, which specifies the data elements required to process a particular decision service.

In the second mode of operation, the client application can supply primary key information in the request message while the rules engine dynamically retrieves (and optionally updates) additional information in an enterprise data source. For example, in a loan pricing application the client may supply the ID of a loan applicant, and, based on rules, the engine may automatically retrieve the applicant's payment history in order to infer new facts such as loan risk. Since the client application supplies only the minimum necessary information, this mode results in looser coupling between the client and the server; rules that refer to enterprise data can evolve freely without affecting the decision service contract or client program logic. Additionally, this mode is very useful when it is difficult to collect in advance all the data necessary to make a decision or if the amount of data required can be substantial in size and difficult to pass into the decision service.

In either mode of operation, rules are authored in precisely the same manner. Enterprise mapping specifications are declared in the vocabulary, but those mappings are transparent to the rule author. This key advantage can reduce costs associated with developing and maintaining enterprise rules that require data access. Traditional rule engines typically require technical skills to code database connectivity and the use of SQL to query the database.

MULTI-THREADING AND CONCURRENCY

Each incoming request message is processed in its own thread of execution: that is, each decision service instance runs in a separate thread. The Corticon Server does not spawn any threads and does not perform thread management; rather, the thread of execution is established and managed by the enclosing container that receives the request (i.e., the application server). In cases where an in-process call is made from the client application, the thread of Corticon execution is the same one in which the client runs.

When the Corticon Deployment Console is used, the person responsible for deployment decides how many instances of the same decision service may run concurrently. This is called the "decision service pool." Different decision services may have different pool sizes in the same Corticon Server depending on their individual load requirements.

CLUSTERING AND LOAD BALANCING

In large volume scenarios, enterprises will typically deploy multiple application servers as a cluster. Corticon Server, as a well-behaved, container-managed web service, can leverage this capacity to scale. In addition, a variety of means exist to spread the incoming workload across the multiple application servers.

Illustrations of the Corticon Server in active/passive (Figure 4) and active/active (Figure 5) clustering configurations are shown below.

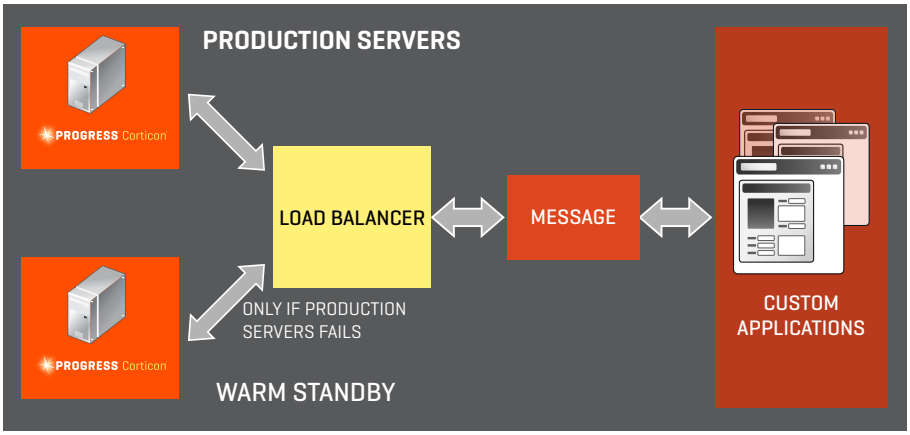

Figure 4
Corticon server with active/ passive clustering

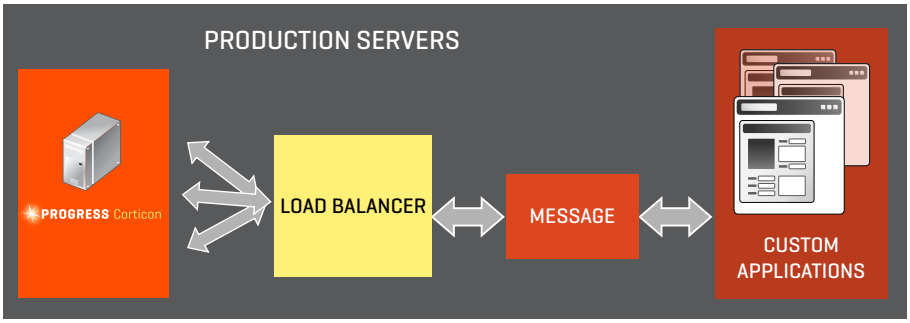

Figure 5
Corticon Server with active/ active clustering

MAINFRAME INTEROPERABILITY

One of the primary reasons for modernizing legacy applications running on the mainframe is the need to provide transparency to the business policies embedded in them. Often, the life of such legacy applications can be extended by externalizing their business logic and automating them as decision services running in Corticon. There are a number of alternatives for invoking an external decision service from a legacy mainframe application including:

OPTION 1 (REMOTE):

- Linux for z/OS
- Java application server (e.g., Tomcat) running on Linux
- Corticon Server deployed in a Java application server
- From a COBOL application invoke decision services using SOAP or REST

OPTION 2 (REMOTE):

- WebSphere for z/OS
- Corticon Server deployed in WebSphere
- From COBOL application invoke decision services using SOAP or REST

OPTION 3 (IN-PROCESS):

- Corticon Server running in a Java Virtual Machine for z/OS
- From an enterprise COBOL application invoke decision services using Corticon Server API

Figure 6 shows a schematic of option 2 above:

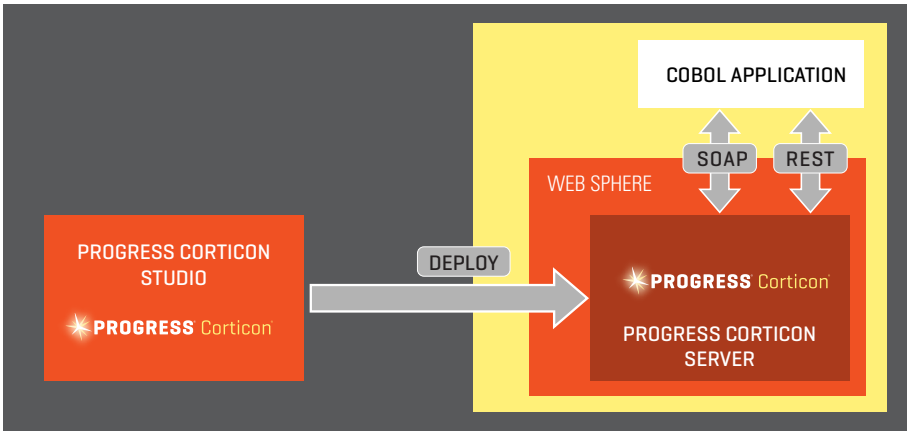

Figure 6
Legacy mainframe interoperability

SUMMARY

The Corticon Server supports a variety of enterprise deployment options. Corticon Server's architecture is well-suited for enterprise environments, leveraging the inherent scalability, availability, and manageability of enterprise-grade application servers.

Corticon-based applications can easily adjust to changes in the business landscape and can help organizations achieve the benefits of a service-oriented architecture.