

Decision Services: The Next SOA Challenge



Decision Services:

The Next SOA Challenge

White Paper

© ILOG, June 2006 – Do not duplicate without permission.

ILOG, CPLEX and their respective logotypes are registered trademarks.

All other company and product names are trademarks or registered trademarks of their respective holders.

The material presented in this document is summary in nature, subject to change,
not contractual and intended for general information only and does not constitute a representation.

Table of Contents

- Enhancing SOA Flexibility and Services Reuse2**
- Problem #1: Services as Black Boxes2**
- Problem #2: Inside the Black Box, What Is Happening?.....3**
- Problem #3: Mapping Policies to Reusable Decision Services4**
- A Better Approach: Transparent Decision Services™5**
- Transparent Decision Services and ILOG JRules6**
- Additional Resources.....8**

Enhancing SOA Flexibility and Services Reuse

Service-oriented architecture (SOA) is an enterprise integration strategy currently enjoying widespread adoption due to its promise of component reuse, flexibility and reduced inherent complexity, and its ability to be deployed alongside existing legacy infrastructure. SOA implements business processes by orchestrating business services that use standard data formats and communication protocols to perform business functions. These services typically fall into two broad categories:

- **Data services** – These services retrieve, modify and record data. They are the prototypical Web services that, for example, retrieve data from different sources for display on corporate dashboards, convert data formats from one application to another, and normalize data between corporate business partners.
- **Decision services** – These services perform decisioning that is unrelated to data transformation. Typically, they are passed data or documents and return a data item that reflects a decision or an action based on the execution of business rules. Examples include services that accept credit reports and return creditworthiness scores, or receive insurance applications and return a list of the available programs and rates for which applicants qualify. These services are central to the workings of the enterprise, and they often represent the core functions of a business process.

Data services, especially those implemented as Web services, are increasingly becoming well understood. Although collectively they can present important challenges (discussed later in this paper), the tools for managing them are available today, and more-advanced resources are coming to market.

Decision services, however, have numerous aspects that are still difficult for IT to manage effectively. This paper discusses the issues specific to decision services and presents solutions that facilitate their effective implementation in an SOA strategy.

Problem #1: Services as Black Boxes

SOA-based decision services in enterprises today are developed by a hierarchy of professionals. At the top, business managers are responsible for running business processes and coordinating activities with IT to develop the software needed to support the core business operations. In the IT department, an architect designs a sequence of Web and business services that implement the business processes. These services are implemented by a service developer. The actual decision-making logic is typically written by a software developer. In all, at least four different individuals are responsible for the service in some way. But if a service doesn't work, the business manager is held accountable, even

though the software developer is the only one who truly knows what decisions the code implements. The architect and service developer have some visibility, but the business manager, to whom the system is just a black box, has none. All the manager knows is that a given service determines the creditworthiness of an applicant.

This lack of visibility into the decision-making software becomes a serious problem when the service needs to be changed. Typically, the change is explained by the business manager in business terms and then implemented by the various architects and developers. However, deploying the code can be very difficult if the enterprise cannot tell who depends on the service and what exactly the service delivers. Often, this problem is handled by deploying the new version as a separate service and pointing the known users to it with the intention of eventually migrating any remaining users. This migration, however, is often delayed for fear that year-end closing or regulatory requirements might still depend on the old service. After several more changes to the service, the business manager usually discovers that the business depends on numerous slightly different services that are all variants of the original.

How does the manager know what each version does? And how can the manager's business effectively manage them all? Without visibility into the details of the operations and deliverables, nobody can manage the proliferation of services and functions properly. Moreover, the opaque nature of these services destroys their reuse. If a manager cannot tell for sure how a decision service makes decisions, that service cannot really be reused in a new context: You cannot – or should not – reuse what you don't understand.

Case History: Quantifying the Proliferation

“[The team] combed through every button and dropdown menu in every application it could find – roughly 2,900 in all – looking for shards of software functionality that might be incorporated into a service component. From the thousands of function points they found, the team isolated between 200 and 500 functions that are needed for the more than 90,000 business transactions (such as setting up a new landline account) that Verizon performs. Then the team looked across the infrastructures and found five to 25 redundant versions of each function.”

— *Verizon case study from “Integration’s New Strategy,” Christopher Koch, CIO Magazine, Sept. 15, 2005 (emphasis added)*

Problem #2: Inside the Black Box, What Is Happening?

Because decision services implement business policies and are often subject to government regulations, they must be auditable. In other words, it must be possible to reconstruct which steps the software took to come to the conclusion that, for example, a given candidate should be refused credit.

The common way to address this need is to have the service generate a log of the elements of the logical decisions made by the software. This log then serves as a record of sorts, should questions arise about a specific decision.

Logs, however, have many significant limitations:

- **Difficult to reconstruct decision path** – Working backward from a log entry to establish a specific sequence of events can be a challenging experience, as anyone using system logs knows. Logs frequently contain extraneous data (decision paths of other data items) and require a developer to spend hours poring over printouts to determine what happened when.
- **Incorrect level of data detail or IT orientation** – Unless a logging framework is designed for a specific task, it will contain too many details (the “capture everything in case you need it later” approach), too few or information that is too technical. In the former case, auditing is cumbersome, while in the latter, it is impossible.
- **Rarely properly maintained** – For a log to be an accurate reflection of the decision process, every modification to the decision logic requires the data output to the log to be modified as well. In the daily rush that comes with business, developers are unlikely to consistently update logging systems as they change program code. Over time, logs fall out of sync with the service they monitor, and it becomes impossible to reconstruct a trail of decision logic. This problem is particularly acute for services that are updated frequently.

What is needed is a decision service that can automatically log each rule as it is fired and generate audit-quality business reports from this data.

Problem #3: Mapping Policies to Reusable Decision Services

Migrations to SOA are often depicted as two-stage tasks that consist of wrapping existing programs with Web-service interfaces and then wiring the services together to implement SOA.

This simple view, however, fails to take into account an important constraint: existing software modules seldom work well as SOA components – without extensive modification. One of the defining benefits of SOA is reusability. However, legacy applications are rarely written with service-style reusability in mind. Rather than coarse-grained services being woven together, many SOA implementations cobble together oversized and unwieldy components into a matrix that looks like SOA only in that Web service protocols are used for communication.

This situation is worse for decision services. Often, business rules are expressed by source code that is hard-wired deep within an application. Other programs that need to apply the same business rules often use their own coded versions of

the rules – generating slight discrepancies, some intentional and others accidental. To create a successful SOA, the business rules need to be extracted from this embedded logic and placed inside a single decision service that can be accessed by the legacy applications.

The difficulty in attempting this is that these rules are often black boxes and difficult to boil down into a uniformly articulated set of constraints and decisions. To get around this problem, IT sites today just wrap the entire application – thereby robbing the SOA of two of its principal benefits: reuse and agility.

For SOA to work properly, business processes must be mapped to properly designed sequences of decision services that are transparent, easily customized, reusable and auditable.

A Better Approach: Transparent Decision Services™

To meet the goals of a successful SOA implementation, enterprises need to reconsider how they provision decision services. If they continue with the unmanageable, black-box approach of wrapping legacy code, they will not attain the rewards of SOA conversion. Instead of flexibility and reuse, they will experience only an additional layer of complexity. Enterprises need to move decision services to a new form of transparency – one in which the rules are designed and managed by business analysts. Table 1 compares this kind of transparent decision service – based on the business rule management system (BRMS) – with a traditional decision service.

Traditional Decision Service	Transparent Decision Service
- Hard-coded decision logic	- Externalized decision logic
- Developed by IT	- Modeled by business analysts
- Maintained by IT	- Maintained by policy managers
- Managed by IT	- Managed by IT
- Dependent upon custom logs	- Automatic auditable capture with extensive reporting
- Hard to modify and reuse	- Easy to modify and reuse

Table 1. Comparison of traditional decision service and Transparent Decision Services

The key distinction between these two approaches is that Transparent Decision Services put the design and maintenance of business rules into the hands of the business users – thereby providing them with greater insight into the decisioning. Once these users can view, modify, test and simulate business rules, the problems associated with black-box decisioning disappear.

Like much of SOA, implementing Transparent Decision Services is based on separating the key activity– decisioning – and making it a standalone service that is accessed by other services. Using this approach, business rules are formulated (by business analysts) and placed in an enterprise repository. When another service or application needs decisioning, it simply accesses the service

and specifies the ruleset to be executed. A business rules engine then applies the ruleset and returns the appropriate decision or value.

This design provides the promised deliverables of SOA:

- **Reuse** – The same rules are used by all the applications that need a specific decision or evaluation of data. Gone is the traditional method of hard-coding rules into multiple applications and then trying to keep them in sync. All the applications that need the same decisioning can use the same rules.
- **Agility** – Business rules can be hot-deployed so that changes in policy or regulations can be implemented quickly and universally through a central rule repository. Because policy changes occur frequently – as new policies are introduced and older ones revised – this agility is a key benefit.
- **Flexibility** – Because text-based rules are much easier than code to write and deploy, they endow an enterprise with much greater IT flexibility. Business rules give companies the ability to create one-off programs, test new policies, implement new controls, and design unique offerings for subsegments of their market – all without having to program. The decision logic is exposed as a reusable service and can be deployed efficiently throughout the enterprise.
- **Easy implementation** – Business rule technology does not require a rip-and-replace implementation. It can be installed incrementally and run side by side with existing applications. These can be migrated as needed and as events permit. As such, BRMSs are good IT citizens and very much fulfill this important promise of SOA.

The key aspect to these deliverables is that personnel on the business side of the enterprise can formulate and implement rules. This aspect underlies the agility and flexibility of business rules. In addition, the greater involvement of business analysts and policy managers naturally provides the transparency and manageability of decisioning discussed previously.

Nonetheless, enterprises face two challenges:

- How is rule management moved from IT to business policy managers?
- How can business users actually write, test and deploy rules that are used in IT processes?

Transparent Decision Services and ILOG JRules

ILOG's JRules business rule management system is uniquely suited for the formulation and implementation of Transparent Decision Services:

- JRules empowers business teams to manage the business rules that automate their policies.

- JRules helps development teams to deploy these business rules as fully formed decision services and weave them into SOA platforms from IBM, BEA, and Oracle among others.

JRules brings several business-side roles together to collaborate on Transparent Decision Services:

- **Policy manager** – The policy manager is the “service owner” specifying what the decisioning mechanics should be. For example, a policy manager decides which factors need to be weighed to establish eligibility for a new type of life insurance.
- **Business analyst** – The business analyst takes this information from the policy manager and creates a domain-specific lexicon for writing the business rules. For example, the analyst may create business names for other services (such as “life expectancy” or “nonsmoker discount”). The analyst also sets up the basic rule environment and specifies the signature for the new Web service (often in collaboration with an SOA architect). The policy manager then implements the rules using the lexicon and a variety of drag-and-drop tools.
- **Integration developer** – The integration developer writes any necessary integration logic, such as WSDL files or UDDI entries.

Business analysts and policy managers can validate and simulate rules prior to deployment. And once the rules are fully tested, the administrator can hot-deploy them in minutes.

Rules themselves are stored in JRules’ rule repository; the common repository greatly enhances rule reusability. For example, any decision activity that needs the same set of rules that the underwriting department uses to accept a policy can simply access those rules (either directly or through the Web service) without having to duplicate them or write variations. If another department needs to slightly modify the rules, it simply creates a ruleset that includes the underwriting group’s rules plus its own requirements. Now, if the underwriting group’s rules change, all the rulesets are updated automatically.

Change versioning is built-in – and this feature is crucial to avoiding duplication and proliferation. If the rules for eligibility, for example, should change, the old rules are not discarded. Instead they are simply replaced in that service’s ruleset by updated versions. An organization can easily go back and retrieve the rules that were used at a prior time, should that need ever arise. And if businesses need decisioning services to rely on versions of retired rules, they can do so without spawning new services. This aspect is particularly useful if two sets of rules are in effect during a transitional period.

This easy configurability also enables enterprises to determine which applications are currently in use. By applying BRMS to decisioning, code that is

frozen in applications can be supplanted by a decision service that integrates properly with an SOA. No longer must the entire application be converted into a Web service, but rather only the decisioning part. Then other applications that duplicated some or all of that logic can access the same rules and consolidate the many variants into a single implementation.

JRules directly addresses the problem of auditability. It can capture various levels of operational detail, down to recording every rule that fired in a service and what data item was the trigger. This data, which can be copious in the case of complex decisioning, is supported by a highly configurable reporting system. Report formats that fit the specific needs of a site can be designed and implemented by the policy manager or business analyst. And for high-visibility decisioning that must demonstrate compliance with regulations, it is possible to automate report generation with every call to a decisioning service.

JRules includes tight integration with the top Java™-based SOA platforms, lowering barriers to its adoption for companies already investing in SOA technology:

- IBM WebSphere Process Server
- BEA AquaLogic
- Oracle Fusion

In short, JRules BRMS enables truly effective development, deployment, maintenance and governance of decision services by making them more visible, auditable, reusable, and agile. JRules delivers Transparent Decision Services.

Additional Resources

To find out more about ILOG JRules and Transparent Decision Services point your browser to <http://brms.ilog.com>.