A Model and Rule Driven Approach to Service Integration with Eclipse Modeling Framework

Isaac Cheng, Neil Boyette, Joel Bethea, and Vikas Krishna

IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, U.S.A
{isaacc, nboyette, bethea, vikas}@us.ibm.com

Abstract. BPEL is fast becoming the most widely-adopted standard for business processes involving web services; however BPEL is geared mainly at the higher level processes and is not well suited for the lightweight, short-lived "micro-processes" that share the same service space. Such processes require the advantages of interoperability and asynchronicity offered by an SOA approach but at a more granular logical level. This paper details a way to use a declarative approach to define the micro-processes that occur in the services called by an SOA based application. Using the context of a global call center workflow application framework named CCF, for Custom Call Flows, this paper describes how micro-processes (call flows) can be defined, and how declaratively defined rules can be used to integrate these micro-processes with other services to build a flexible service system.

Keywords: architecture, call center, call flow, script, CRM, IT, Web, labor, asset, business transformation, customer, enterprise, global, infrastructure, inference, integrate, internet, leverage, logic, management, offshore, outsource, reasoning, rich client, rule, thin client, workflow, worldwide, support, EMF, XML, UML, BPEL, SOA.

1 Introduction

In an engagement with the call management team of a global enterprise, the authors introduced the Custom Call Flow (CCF) framework that enabled the enterprise to compose services from legacy mainframe-based back-ends to newly introduced third-party systems in a service-oriented architecture (SOA). Using this framework, business analysts can visually design models and declarative rules without requiring any programming skills. At the time application programmers can develop applications independent of business processes and workflows. CCF enables businesses and technical people to work independently and productively. It also enables enterprises to strategically outsource and offshore efficiently and effectively.

The first real-world deployment of CCF was for improving a call center, where customers call for services. A call flow, a special type of workflow, describes the steps that systematically guide a call-taker to solve a customer problem in multiple scenarios. Call flows are essential to enable customer service representatives to

B. Krämer, K.-J. Lin, and P. Narasimhan (Eds.): ICSOC 2007, LNCS 4749, pp. 474–484, 2007.

support products, and as the products evolve, the associated call flows need to be updated with an authoring tool (see Fig. 1). In the existing Call Management System, the authoring tool was tightly coupled with the rest of the system, which posed significant challenges both in evolving the tool and updating the call flows themselves. In particular, enabling the authoring tool to keep up with the current user-interface (UI) technologies proved to be extremely difficult due to the interconnected nature of the UI with the rest if the system. This lack of a modern UI in turn made it hard for a business architect to get a holistic view of a call flow to understand the design. Since the information that can be displayed on the UI is extremely limited (see Fig. 1), the business architect is likely to make local changes that may have unexpected side effects globally. Therefore, updating call flows using the existing tool was often error-prone as each change required both a business analyst to define the change, and a programmer to implement the change in code. Finally, there were also limitations in the proprietary protocol used between the authoring tool and the call flow repository, which made supporting foreign languages and cultural information impossible.

The runtime components also suffered from problems stemming from the tightly coupled nature of the architecture. Similar to the authoring tool, the runtime UI could not keep up with modern technologies which resulted in usability problems., In addition, changing or upgrading algorithms used in the call flows such as those used for business-rule inference, was quite difficult, if not impossible.

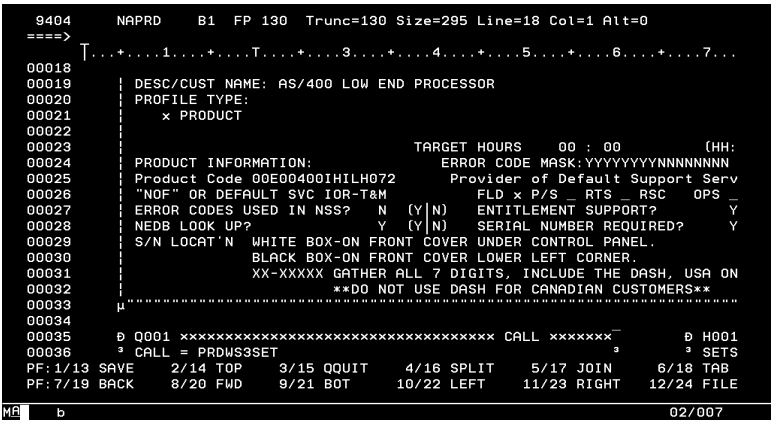

Fig. 1. The Authoring Tool of the Existing Call Management System

2 Related Work

In the service-oriented computing (SOC) area, frameworks and adaptation technologies have been developed for composing and integrating heterogeneous services and processes [2], [7], [11]. A model-driven development approach and its technology elements for SOA were described in [6]. A state-of-the-art model-based framework for developing and deploying data aggregation services was described in [10]. The design of this framework hides the complexity of web-service development, but nevertheless

requires programming skills to use. Many people we worked with who compose services and design business processes do not have programming skills. To address their business needs, we introduced the CCF framework [3], which is based on the Eclipse Modeling Framework [5]. The CCF architecture features an authoring environment based on Eclipse Rich Client Platform which allows business users to design, test, and deploy workflows visually using a subset of Unified Modeling Language without requiring any traditional programming. The runtime component provides agility by making it possible for business processes to change independent of application changes. The work described in this paper focuses on how a variety of BPEL and non-BPEL processes can integrate seamlessly using the CCF framework.

In the service-rule processing area, a rule driven approach for service development for collaboration exists [9]. Facilitated by the Business Collab175oration Development Framework, our authoring tool allows business users to specify rules in the service design perspective and the task design facet. An intelligent runtime for rule processing using Agent Building and Learning Environment is described in [4]. A declarative pattern-based approach is introduced in [8] that supports the specification and use of service interaction properties in the service description and composition process. We also found that a declarative pattern-based approach is more natural to business users than a procedural program-based approach. However, unlike in [8], the pattern part of our service rules is data-oriented rather than operation-oriented.

3 System Architecture

The software system that implements the CCF framework consists of four major components: an authoring tool, an administration tool, a repository, and a runtime environment.

The authoring tool allows a business analyst to create, modify and test call flows. Call flows are presented in a graphical workflow editor which allows non-programmers to easily work with the call flow objects. The (integrated) administration tooling lets administrators manage call flows. Both tools communicate with a call flow repository via a web service. The repository stores the call flows and provides an interface to search, publish and retrieve the call flows. The runtime consists of an ABLE-based execution engine which provides an API for clients to retrieve and execute call flows from the repository.

Fig. 2 describes the CCF system architecture. A typical scenario would begin with a call flow author using the authoring tool to create or update a call flow. The administrator would then publish this call flow in the repository, making it available to the runtime. An end user can contact the service center in a number of ways including voice (telephone), email, web, etc. The end-user can interacts with the runtime engine through either a self-help application or a customer service representative who interacts with a call management application instead. The runtime environment interacts with the repository to retrieve, display, and execute the appropriate call flows to the end users. As shown in Fig. 2, the ABLE [1] based runtime component enables the runtime to interact with other services by dynamically generating the required web service client interfaces. These interfaces are then used to communicate over the Enterprise Service Bus. This loose coupling allows the business processes and rules to evolve over time without requiring code changes.

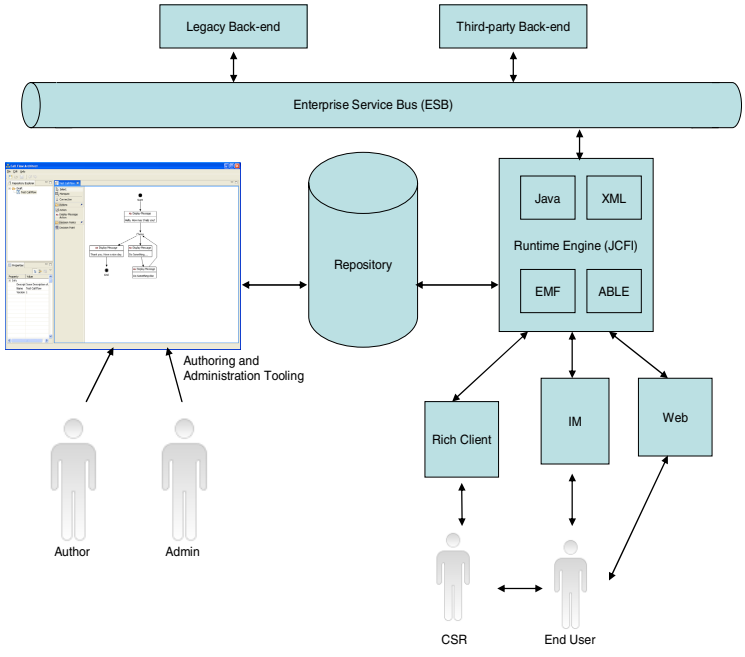

Fig. 2. CCF Architecture

4 Call Flow Authoring Tool

The Call Flow Architect (CFA) tool enables the authoring and administration of call flows. The tool allows business analysts to work with the call flows without requiring any programming skills. The main interface used by a call flow author is the call flow editor which visually displays a given call flow. CFA also provides an interface for searching and browsing the repository, which enables existing call flows (including all versions and locales) to be checked out for editing.

To integrate call flows with external services the CFA application provides a UI to define service rules. A service rule describes a service to be called, as well as the conditions that should be satisfied before calling the service. Each time a field in a call flow's dataset is changed, the CCF runtime examines the available service rules, and if one's conditions are satisfied it automatically calls the specified service.

Once the user has submitted the call flow to the repository, it enters a draft state. An administrator verifies the call flow and then publishes it. Published call flows can now be activated and thus become available to the runtime. Before a call flow can transition to a new state (except rejected or deleted) it will first undergo syntactic and semantic validation by the system. This ensures that the collection of call flows remains valid.

5 Call Flow Repository

The call flow repository provides storage, search, and retrieval of call flows from a distributed set of authoring tools and runtimes. Since the call flows are persisted in XML, the repository implementation is based on a database that natively supports XML. This has three major advantages:

- Direct storage of XML documents without the overhead of shredding them into data elements
- In-place update of elements and attributes of the XML document i.e. call flow
- Search by a constituting element or attribute

All of these features are heavily leveraged by our framework as call flows are often updated on a frequent basis due to additions or changes in locale or service level agreement, or the evolution of the supported products.

In order to support access to the repository from a distributed set of clients, a web service is used to provide an abstraction layer for hiding database level details and database specific client-side libraries. This makes the repository truly interoperable with both the current and future sets of clients in an SOA manner along with providing all the other benefits that SOA facilitates

6 Call Flow Runtime Engine

The runtime engine executes call flows. A major advantage of the runtime engine is agility. As shown in Fig. 2, the runtime engine leverages the fact that the call flows do not contain any UI information (or assume any UI knowledge for that matter) by using the Java Call Flow Interface (JCFI) API. In applying the classic Model-View-Controller design pattern to this system, the call flow is the Model; the client application is the View; and the CCF runtime engine is the Controller. JCFI is the programming interface between the View and the Controller. This UI agnostic design enables a wide variety of applications to invoke the runtime engine. These applications can in turn implement various user interfaces. The advantage is similar to that of decoupling application development from data management. CCF decouples process management from application development resulting in process independence. This greatly enhances the agility of the service system by enabling business analysts to work productively independent of application developers.

7 Call Flow Runtime Sample Applications

A few client applications have been developed to explore the opportunities presented by this design. Each application is independent of the business process under which it is used and demonstrates the agility that results from process independence.

7.1 Web Client

Web clients demonstrate the traditional thin-client model by providing access to the powerful features in the runtime engine via a Web browser.

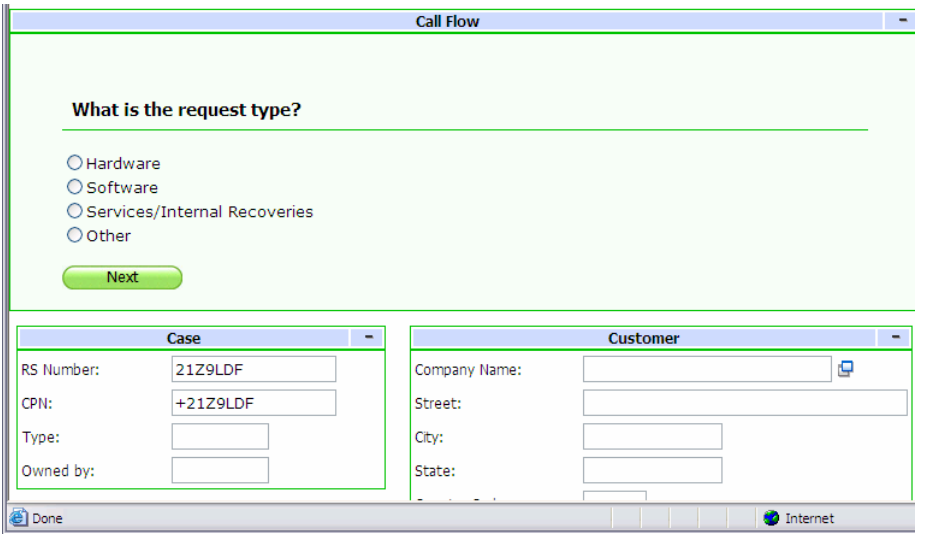

Fig. 3. A Web-based User Interface

7.2 Instant Messaging Client

Instant messaging clients (also known as chat programs) can be used to have a text-based conversation in real-time. Usually the conversation is between two humans, but there are also applications which can provide automated information. These are known as chat bots. In this case, a chat bot is used as the interface to the runtime engine, translating call flow prompts into chat responses in real-time.

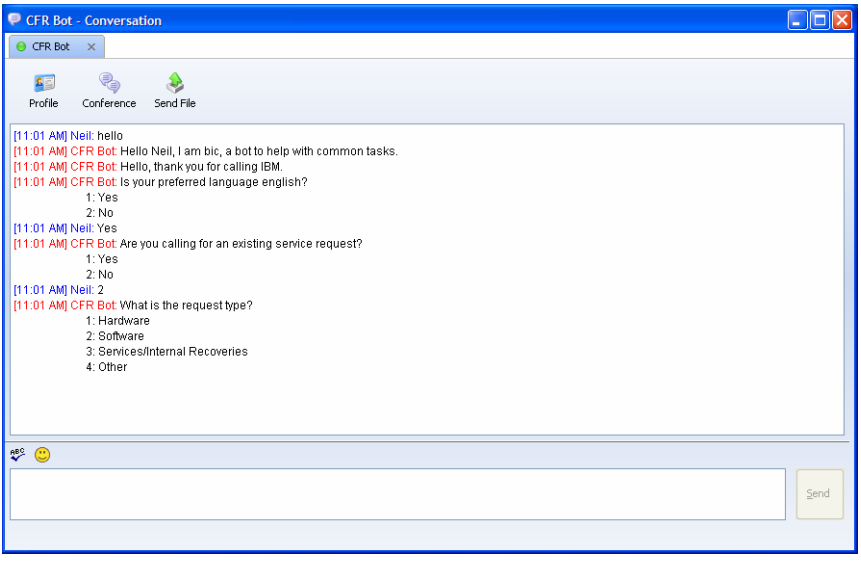

Fig. 4. An Instance Messaging (IM) User Interface

8 CCF, BPEL, and Web Services

Process independence is also a feature of Business Process Execution Language (BPEL), which is becoming the most widely-adopted standard for business processes involving Web services. Like CCF, BPEL provides a means to formally specify business processes and interaction protocols. The main difference between BPEL and CCF is that BPEL is used to specify macro processes, where as CCF is used to specify micro processes. In addition, CCF provides complete end to end tool support for process developers. This is not present in the state of the art on the BPEL front at the time of writing of this paper.

While BPEL can be used to specify any type of process, it is especially good at supporting long-running conversations with business partners. These high-level interactions, or macro processes, make use of Web services to implement the logic of business processes. CCF on the other hand, is geared towards shorter-running processes. In these micro-processes, business functions are specified as a series of actions where some interact with other systems, some interact with end users and some are self contained. CCF thus fills a niche by supporting fully-featured business processes at a smaller granularity of logic.

With this relationship CCF and BPEL complementing each other, they can be used together when building larger frameworks. BPEL is used to define the overall architecture and interaction between different functions/services. CCF is used to define how a given business function accomplishes its task. CCF in turn can interact with the BPEL defined process by exposing itself as a service and call other services in the BPEL process when needed.

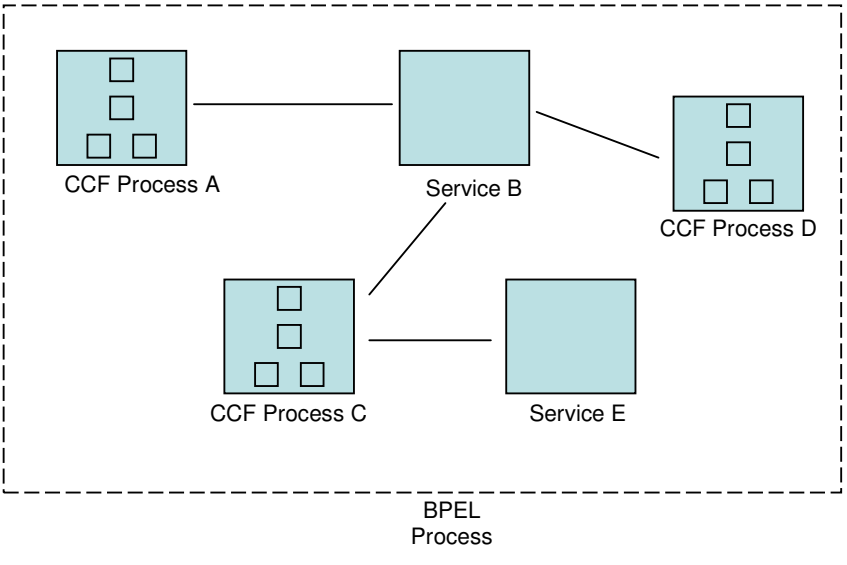

Fig. 5. Interactions between CCF and BPEL

The relationship between CCF and BPEL is analogous to other technologies that partially overlap but still complement each other. An example would be Bluetooth and Wifi wireless technologies. Both can be used to wirelessly connect two devices, but each has its own unique strengths. Wifi is faster but requires more electrical and processing power on the device. Bluetooth is more efficient in terms of electrical and processing power but has a slower speed. The two technologies co-exist because they complement each other. For instance a remote control can connect to a receiver using Bluetooth to request a song. The receiver then connects using Wifi to a home media server to retrieve the song and play it. The remote could have connected using Wifi to the receiver and the receiver could have connected using Bluetooth to the home media server, but that would not have been the most efficient use of technologies.

In a call center scenario, BPEL can be used to specify the interaction between all the services. When for example it requires a customer's information it can contact a CCF service. The CCF service would then gather the information in different ways for different customers. For instance, for preferred customers it may only ask for a customer number; then contact another service (using the BPEL framework) to retrieve the customer data from a database. For new customers it may ask a series of questions to gather the name, address, etc. Using a BPEL process to ask a series of questions is not very efficient so delegating to a CCF service is preferred. Conversely, using a CCF process to orchestrate connections between a set of services is not very efficient either, so having it call a service in the BPEL process is preferred.

In the call center engagement the difference between the CCF micro-processes and BPEL's macro processes was especially evident. This call center had in excess of 80.000 customizations to the standard process in just the United States. In addition hundreds of these customizations are changing every day; new customizations are added, others are removed and others again are changed. The system has to be flexible enough to handle both this level of customizations and this level of daily changes, without effecting system performance and without requiring a large workforce to manage the changes. The current state of BPEL application servers simply can not handle this. Updating the servers with the daily changes would be a full-time job for several people. Alternatively, the CCF framework loosely couples the call flows together to make up the larger process. As it is geared for micro-processes it is designed from the start to support large quantities of smaller processes, while still allowing integration into the larger BPEL processes. The CCF framework also allows changes to be made and call flows to be activated or deactivated by the business analysts themselves. As there is no complex deployment, supporting hundreds, even thousands of changes daily poses no significant workload.

9 Composing Heterogeneous Services with Declarative Rules

One of the challenges in our deployment was that there are many database back-ends. Many of them come from the legacy system, and some of them are newly acquired from third-party vendors. In the original system, supporting such a broad and ever changing collection of data sources proved to be extremely difficult, as call flows essentially were hard-coded at design time with decision-making for invoking every possible operation. This difficulty was then compounded further by the

tightly-coupled nature of the call flow and UI displayed on the runtime client, which entangled call flow logic with UI logic, and frustrated efforts to update the existing UI or support additional types of client application.

With CCF, one possible solution to this problem involved granting call flow authors the ability to minimize decision-making complexity by encapsulating the invocation of operations as script nodes. This would then allow client UI applications to avoid the headache of hard-coding service invocation by instead designing the call flow to instead link to pre-defined script nodes. This approach still has the drawback however, that the call flow author is forced to predict in advance exactly when each particular service operation needs to be invoked within a call flow. Since there can often be many places within a call flow where it might be appropriate to invoke a given operation (or combination of operations), this approach makes the call flows complex, inflexible, and hard to maintain.

To address this issue, CFA provides a better alternative in the form of service rules. These service rules allow specific conditions to be linked to an operation, such that when the conditions are satisfied the operation is invoked. At design time, call flow authors specify conditions declaratively as if-then rules in a view separate from the call flow. The exact point of invocation within a call flow is then determined at runtime as guided by the design rules. This method is further enhanced by the use of a reasoning engine. When executing a call flow, the CCF runtime engine infers knowledge behind the scenes by sending the rules and its currently known facts to a forward-chaining reasoning engine, which has been implemented efficiently by scaleable Rete-style pattern matching in ABLE [1]. Although a Prolog-style backward-chaining reasoning engine may be more efficient, it would force the author to come up with a search goal per rule set at design time. This is often a difficult task because it is unlikely that the author can predict what knowledge will be useful to be inferred until runtime. In contrast, a forward-chaining engine can infer new facts based on the currently known facts and a set of rules. This desirable characteristic allows CCF to optimize the execution paths of call flows intelligently at runtime, removing the burden from the call flow authors and client application developers, and making it possible to keep the call flows and the client applications as reusable and maintainable as possible. Another advantage is that since the CCF runtime is determining when service rules are called, it can optimize this behavior and the system would not experience slow downs because of the business analysts adding too many rule calls at once, or in the non-optimal location. The call center owners can be assured that business analysts are shielded to an extent from specifying low-level execution details that do not carry out their intent. This is especially important as the call flows are componentized and thus a business analyst may not be aware of all the contents in which a particular call flow may execute, but the call flow runtime engine will be.

Although getting business owners to accept runtime choices may appear to be challenging when one looks at it in the abstract, it can be quite straightforward in many cases in practice. For instance, consider the service rule in Figure 6. It is straightforward for a business owner to specify that if any of the attributes, such as a customer's phone number and email is set at runtime, invoke the web service SearchForContact. It would be more difficult for the business owner to specify particular points in a call flow at which the web service needs to be invoked. With CCF, business owners still have the control of making important business decisions, such as the conditions by which certain services should be invoked. The runtime

system executes the business decisions by employing an efficient pattern-matching algorithm to perform logical inference on the rules.

Service rules consist of the description of the service to be called, the mapping of the service's parameters to data elements in the call flow, and the specification of the condition that governs the rule's execution. As the authoring tool is geared towards business analysts (i.e., non-programmers), service rules are configured in an intuitive and easy to understand UI as shown below in Figure 6.

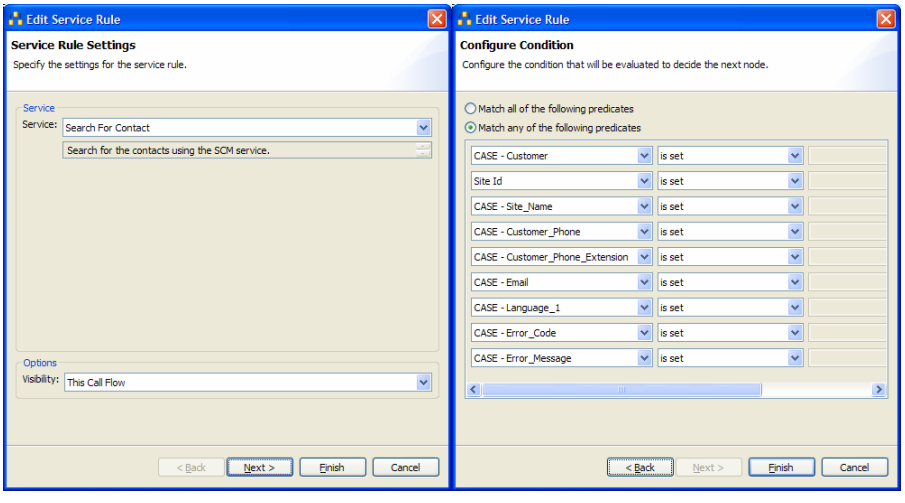

Fig. 6. Editing a Service Rule with CFA

Since CCF allows for the componentization of call flows, this concept was also extended to the service rules. Each service rule has an associated visibility which acts as the scope the service rule is valid for. Visibility can be set only for the call flow which specifies it, for the call flow and any call flows that it references, or for the entire session. The runtime engine takes the visibility into account when it generates the code used by the forward-chaining reasoning engine. The advantage of using componentization in this manner is that different rules can be applied at different times in the problem resolution session. Some rules may only apply during a given call flow, whereas others may apply during the whole session. For instance, the call flow may have a rule which looks up the address information based on the customer's last name. If the name is changed anytime during the session, the address information should be updated.

10 Conclusion

As an initial technology deployment, 40.000 users will use the suggested framework to process 10.000.000 service requests annually. The current systems for defining and handling call flows and supporting calls are mainframe based. This requires mainframe programmers to maintain the calls flows and a certain degree of mainframe expertise on the part of customer service representatives to handle the calls. Both of these factors result in huge costs to the corporation to maintain skills in

these areas. The new CCF framework moves the definition and handling of the call flows to the easier to use graphically driven Eclipse platform. With the CCF framework, a system has been designed to handle more than 80.000 customizations to each standard process per country when hundreds of these customizations are changing every day. This is beyond the limit that the current state of BPEL application servers can practically handle. The transformation will result in significant cost savings to the enterprise in managing and running its call centers.

Acknowledgments. The authors would like to thank Dawn Fritz for communicating the business value of CCF to many key initiatives in IBM and thank Priyanka Jain for developing a thin-client application as an important part of the proof of concept for CCF.*

References

1. ABLE Rule Language: User's Guide and Reference, Version 2.3.0. ABLE Project Team, IBM T. J. Watson Research Center (2006)
2. Brogi, A., Popescu, R.: Automated Generation of BPEL Adapters. Service-Oriented Computing - ICSOC, pp. 27–39 (2006), http://dx.doi.org/10.1007/11948148_3
3. Cheng, I., Boyette, N., Krishna, V.: Towards a Low-Cost High-Quality Service Call Architecture. In: IEEE International Conference on Services Computing – SCC, pp. 261–264. IEEE Computer Society Press, Los Alamitos (2006), http://doi.ieeecomputersociety.org/10.1109/SCC.2006.106
4. Cheng, I., Srinivasan, S., Boyette, N.: Exploiting XML technologies for intelligent document routing. In: Proceedings of the 2005 ACM Symposium on Document Engineering, Bristol, United Kingdom, November 02 - 04, 2005, pp. 26–28. ACM Press, New York, NY (2005), http://doi.acm.org/10.1145/1096601.1096609
5. EMF: Eclipse Modeling Framework (2006), http://www.eclipse.org/emf/
6. Johnson, S., Brown, A.: A Model-Driven Development Approach to Creating Service-Oriented Solutions. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 624–636. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11948148_60
7. Kongdenfha, W., Saint-Paul, R., Benatallah, B., Casati, F.: An Aspect-Oriented Framework for Service Adaptation. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 15–26. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11948148_2
8. Li, Z., Han, J., Jin, Y.: Pattern-Based Specification and Validation of Web Services Interaction Properties. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 73–86. Springer, Heidelberg (2005), http://dx.doi.org/10.1007/11596141_7
9. Orriens, B., Yang, J., Papazoglou, M.: A Rule Driven Approach for Developing Adaptive Service Oriented Business Collaboration. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 61–72. Springer, Heidelberg (2005), http://dx.doi.org/10.1007/11596141_6
10. Soma, R., Bakshi, A.K.V., Da, W.: A Model-Based Framework for Developing and Deploying Data Aggregation Services. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 227–239. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11948148_19
11. Zhao, H., Doshi, P.: A Hierarchical Framework for Composing Nested Web Processes. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 116–128. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11948148_10