Semantic Web Services Challenge 2006

The jABC Approach to Mediation and Choreography

Christian Kubczak[1] Ralf Nagel[2] Tiziana Margaria[1] Bernhard Steffen[2]

[1] Service Engineering for Distributed Systems, Institute for Informatics, University of Göttingen, 37083 Göttingen, Germany, {kubczak,margaria}@cs.uni-goettingen.de
[2] Chair of Programming Systems, University of Dortmund, 44227 Dortmund, Germany, {ralf.nagel,steffen}@cs.uni-dortmund.de

Abstract. Our approach to the SWS-Challenge 2006 Phase I uses the JavaABC [1] for mediation and choreography. jABC is a flexible and powerful framework for service development based on Lightweight Process Coordination. Users easily develop services and applications by composing reusable building-blocks into (flow-)graph structures that can be animated, analyzed, simulated, verified, executed, and compiled. We show here briefly how to handle the mediator design and the remote integration of web services.

1 Basic Concepts of the jABC Modelling Framework

We present how we are addressing the SWS-Challenge 2006 Phase I using jABC[1], a flexible and powerful framework for service development based on Lightweight Process Coordination [5]. Predecessors of jABC have been used since 1995 to design among others industrial telecommunication services [6], and Web-based distributed decision support systems [2].

jABC allows users to easily develop services and applications by composing reusable building-blocks into (flow-)graph structures. This development process is supported by an extensible set of plugins that provide additional functionality. This way, we are able to adequately support all the activities needed along the development lifecycle like e.g. animation, rapid prototyping, formal verification, debugging, and code generation. It does not substitute but rather enhance other modelling practices like the UML-based RUP (Rational Unified Process, [7]), which are in fact used in our process to design the single components.

Lightweight Process Coordination offers a number of advantages that play a particular role when integrating off-the-shelf, possibly remote functionalities, as in this Challenge.

- **Simplicity**. jABC focuses on application experts, who are typically non-programmers. The basic ideas of our modelling process have been explained in past projects to new participants in less than one hour.

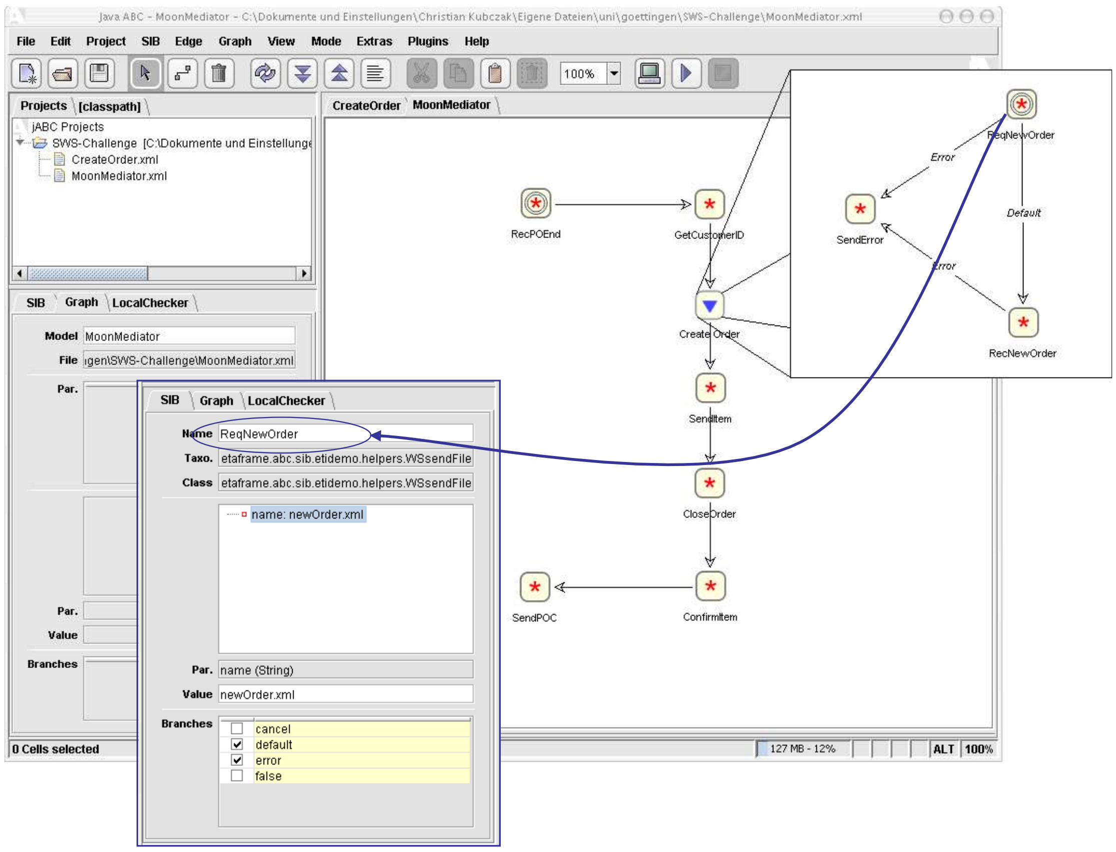
– **Agility**. We expect requirements, models, and artefacts to change over time, therefore the process supports evolution as a normal process phase.
– **Customizability**. The building blocks which form the model can be freely renamed or restructured to fit the habits of the application experts.
– **Consistency**. The same modelling paradigm underlies the whole process, from the very first steps of prototyping up to the final execution, guaranteeing consistence of the semantic paradigm and traceability.
– **Verification**. With techniques like model checking and local checks we support the user to consistently modify his model. The basic idea is to define local or global properties that the model must satisfy and to provide automatic check mechanisms.
– **Service oriented**. Existing or external features, applications, or services can be easily integrated into a model by wrapping the existing functionality into building blocks that can be used inside the models. This is a central issue for the Challenge.
– **Executability**. The model can have different kinds of execution code. This can be as abstract as textual descriptions (e.g. in the first animations during requirement capture), and as concrete as the final runtime implementation.
– **Universality**. Thanks to Java as platform-independent, object-oriented implementation language, jABC can be easily adopted in a large variety of technical contexts and of application domains.

2 Designing the Moon Mediator

In jABC, every functionality used within an application or service is encapsulated within a Service-Independent Building Block (SIB). In fact, we use SIBs to form the workflow of the Moon Mediator within a Service Logic Graph (SLG), as shown in Fig. 1. A SIB could contain a single functionality, or also contain a subgraph (another SLG), thus serving as a macro that hides more detailed and basic steps. Using graph SIBs we are able to model the workflow exactly as described for the Challenge task.

The Moon Mediator Workflow
In Fig. 1 we see the top-level flow of the mediator application (corresponding to the Challenge Specification). The CreateOrder SIB contains the subgraph for creating a new order, including error handling. The more detailed and basic steps (ReqNewOrder and RecNewOrder) are hidden within the subgraph, as shown in Fig. 1 on the right, and can be expanded at need to the required level of detail. In our case, they just contain the calls to the external web service provided by the Moon Order Management System (createNewOrder operation).

Integrating External Web Services
These two SIBs in fact are designed as generic WebService wrappers: they can communicate with any web service. The ReqNewOrder SIB is able to send an XML message to an external service, RecNewOrder is responsible for receiving an answer. Which XML file should be passed is specified as a parameter of these


Java ABC - MoonMediator - C:\Dokumente und Einstellungen\Christian Kubczak\Eigene Dateien\uni\goettingen\SWS-Challenge\MoonMediator.xml

jABC Projects
SWS-Challenge [C:\Dokumente und Einstellunge
CreateOrder.xml
MoonMediator.xml

Model: MoonMediator
File: ıgen\SWS-Challenge\MoonMediator.xml

Name: ReqNewOrder
Taxo.: etaframe.abc.sib.etidemo.helpers.WSsendFile
Class: etaframe.abc.sib.etidemo.helpers.WSsendFile
name: newOrder.xml
Par.: name (String)
Value: newOrder.xml
Branches: [ ] cancel [✓ default] [✓ error] [ ] false

0 Cells selected
127 MB - 12%

Fig. 1. The Moon Mediator workflow designed within the jABC, and configuration of the ReqNewOrder SIB

SIBs as we can see in Fig. 1 in the separate SIB Inspector window. This way the communication with any kind of web service is easy and intuitive.

Workflow Granularity

The top-level worklow designed within the jABC shown in Fig. 1 is rather simple: it is for instance cycle free. The loops described in the challenge task can be modelled in different ways, mostly depending on the desired abstraction of the workflow:

– they can be modelled within the implementation code of the specific SIBs, as iterations over variables. This is desirable, if there is no need to reason (or prove anything) about that behaviour, which is considered an implementation issue.
– if we are interested in analyzing the loop behaviour, we can refine the SLG of the workflow and model the (relevant) loops at the workflow level, either

of the whole Moon Mediator, or inside specific graph SIBs if that portion of the workflow needs specific attention.

– In principle, workflows can be refined up to the detail of single statements, if necessary.

Successive analysis of the code can help also in cases where the workflow has not been refined to the very end.

Workflow Execution
After designing the workflow, by means of the tracer plugin we are able to animate, simulate or interprete it (depending on the kind of executable code associated with the SIBs: mock code, simulation code, or real implementation).

We can also generate source code of the SLG by invoking one of the jABC code generators. They differ in the structure and efficiency of the generated code, but all of them allow getting a running application that is independent of the jABC.

Workflow Evolution
We think that the whole process of designing the solution to the Challenge can be solved with little coding effort: by instantiating existing SIBs and graphically designing and configuring the workflows at the SLGs level. In this case we are confident to be able to react flexibly to changes, as required in Phase II.

3 Ongoing Work

At the moment we use a wrapper SIB to encapsulate the communication with external web services. This SIB is manually implemented, and, although it is rather generic, it may need changes in special cases.

To overcome this limitation we have developed jETI [4] (Java Electronic Tool Integration), a technique for including remotely provided third party tools as (remote) SIBs within the jABC, and to communicate with them seamlessly from within the jABC. As for all jABC extensions, jETI is available as a plugin. The advantage of jETI is that a tool provider is able to supply functionality in a very easy way, independently of Web Service technology (which is still often the case in more technical application domains, like algorithms for verification and analysis of technical systems).

Currently we are extending jETI to support also more mature external services. It should be soon able to accept WSDL descriptions as input and generate the corresponding SIBs, serving as a communication layer between the jABC and the provided web services, improving over the previous implementation ETI [3]. As a main effect, an application expert is able to use a remote tool without caring about detailed knowledge of web services.

We are also planning to link our approach with BPEL, in particular the SLG style of graphic workflow modelling and the underlying semantics.

References

1. jABC Website: www.jabc.de
2. M. Karusseit, T. Margaria: Feature-based Modelling of a Complex, Online-Reconfigurable Decision Support Service, WWV'05. 1st Int'l Workshop on Automated Specification and Verification of Web Sites, Valencia, Spain, March 14-15, 2005, – Post Workshop Proc. appear in ENTCS.
3. T. Margaria: Web Services-Based Tool-Integration in the ETI Platform. SoSyM, Int. Journal on Software and System Modelling, Vol. 4, N. 2, May 2005, pp. 141 - 156, Springer Verlag
4. T. Margaria, R. Nagel, B. Steffen: Remote Integration and Coordination of Verification Tools in jETI. Proc. ECBS 2005, 12th IEEE Int. Conf. on the Engineering of Computer Based Systems, April 2005, Greenbelt (USA), IEEE Com-puter Soc. Press, pp. 431-436.
5. Margaria, T., Steffen, B.: Lightweight coarse-grained coordination: a scalable system-level approach. STTT **5** 2-3 (2004) 107–123.
6. T. Margaria, B. Steffen, M. Reitenspieß: Service-Oriented Design: The Roots, IC-SOC 2005: 3rd ACM SIG-SOFT/SIGWEB Intern. Conf. on Service-Oriented Computing, Amsterdam (NL), Dec. 2005, LNCS N. 3826, Springer Verlag, pp.450-464.
7. Rational Unified Process. http://www-306.ibm.com/software/awdtools/rup/
8. Semantic Web Services Challenge 2006: Challenge on Automating Web Services Mediation, Choreography and Discovery - organized by DERI, Stanford (USA). http://www.sws-challenge.org/