THE SWS MEDIATOR WITH WEBML/WEBRATIO AND JABC/JETI: A COMPARISON

Tiziana Margaria*, Christian Winkler*, Christian Kubczak†, Bernhard Steffen†

*Institute for Informatics, University of Potsdam, 14482 Potsdam, Germany,
{margaria,winkler}@cs.uni-potsdam.de
†Department of Computer Science, University of Dortmund, 44227 Dortmund, Germany,
{christian.kubczak,steffen}@cs.uni-dortmund.de

Marco Brambilla‡, Stefano Ceri‡, Dario Cerizza§, Emanuele Della Valle§, Federico M. Facca‡, Christina Tziviskou‡

‡Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy
{mbrambil, ceri, facca, tzivisko}@elet.polimi.it
§CEFRIEL, Milano, Italy
{cerizza, dellava}@cefriel.it

Keywords: Semantic Web, Web services, jABC, WebML

Abstract: In this paper we compare the SWS Challenge Mediator solutions provided using the WebML/Webratio and the jABC/jETI approaches.

1 Introduction

In this paper we compare two solutions to the mediation scenario of the SWS challenge that are based on the use of WebML (Ceri et al., 2002) and of the jABC (Steffen et al., 2006; jABC Website, 2007) as modelling and execution platforms.

We compare only the solutions to the first version of the SWS Challenge scenario, since the jABC solution was discussed at the SWS review workshop only for this first Phase. Nevertheless, the technical comparison of the approaches is of general interest and valid also for the second phase of the scenario.

Both groups adopt a model based approach, supported by model driven design tools and environments. This allows modelling the mediator in a graphical high level modelling language and supports the derivation of an executable mediator from these models. The solutions are thus similar in their spirit, and we provide here a first description and comparison of the similarities and differences, at the modelling, language, tool, and change management levels.

In the following, we briefly describe the two concrete solutions (Sect. 2 and 3) from the point of view of the used technologies, then we sketch a comparison (Sect. 4), we present a reduction of the two solutions to their mere, common essence (Sect. 5), and finally we conclude in Sect. 6.

2 WebML

The specification of a WebML application (Ceri et al., 2002) consists of a set of models: the application data model (an extended Entity-Relationship model), one or more hypertext models (i.e., different site views or different service views), expressing the navigation paths and the page composition of the Web application or the chain of operations needed to describe a Web service; the presentation model, describing the visual aspects of the pages for user views.

WebML covers also the development of B2B Web applications implementing business processes, thereby supporting full-fledged collaborative workflow-based applications, spanning multiple individuals, services, and organizations. In such case the service view or site view is partially generated by a BPMN model representing the workflows involved in the application. The core elements of a WebML diagram are units. Each WebML unit has its own well defined semantic and its execution complies with its semantic. The composition of different units leads to the description of the semantic of hypertext or Web services. WebML provides standard units for querying data (e.g. Index unit, Selector unit), modifying data (e.g. Modifying unit). The WebML conceptual model has been also extended with a service model that includes a set of Web service units (Manolescu et al., 2005), corresponding to the WSDL classes of Web service operations, and components for workflow management and tracking. The model supports both the grounding of Web services to the XML
format of Web service messages, and datamediation capabilities. In particular the Request-Response and One-way operations are used to consume external Web services, while Solicit and Response units are used to publish Web services; finally the Error Response unit takes care of error handling and returning messages to the invoker in case of errors. The WebML methodology is supported by WebRatio (WebModels s.r.l., 2007), a commercial CASE tool that covers all design and development phases from the BPMN modeling to the deployment of Web applications and Web services (see Figure 1).

2.1 The WebML Mediator

The solution for the mediation problem starts by designing the data model underlying the RosettaNet messages with an extended E-R model. We identified three main entities: the Pip3APurchaseOrder, the Partner and the ProductLineItem. Each Pip3APurchaseOrder instance is related with one or more ProductLineItem instances, one Partner representing the Buyer, one Partner representing the Seller and one Partner representing the Receiver. Every ProductLineItem instance may have one Partner representing a Receiver for the single line.

Then the WebML solution models the high-level scenario of the challenge using BPMN (see Figure 3). This model includes all parts of the scenario on the whole – Blue’s side as well as the mediator and Moon’s side – and describes the scenario workflow.

This model is used as a guidance in the production of two WebML models that implement the workflow’s functionality. The BPMN workflow is split by a corresponding annotation of the BPMN model into two separate WebML models that represent two independent parts of the process: sending a purchase order and receiving acknowledgments for the ordered items. The generation of WebML diagrams is based on an algorithm that populates the WebML diagram with the standard general purpose units mentioned before, e.g. for receiving Web service calls and calling Web services and sending Web service responses, according to the BPMN model. The design of the mediator is refined manually by configuring existing units and adding new ones from the WebML unit library (no new unit had to be developed to cope with the mediation scenario). It could be possible also to model the mediator with out storing data but working only in memory. The data storage was preferred to allow a better monitoring of the mediation process. The conversion from RosettaNet messages is handled by Adapter units that are configured by a proper XSLT stylesheet that transforms messages in an XML format compatible with WebML's internal data format. In the same way conversion to and from
Moon legacy messages are handled by proper XSLT stylesheets that act as templates for SOAP messages and that are then populated by runtime queries during the workflow execution.

3 jABC/jETI

The jABC solution is realized within the jABC framework (Steffen et al., 2006; jABC Website, 2007), an environment for model-driven service orchestration based on lightweight process coordination. It has been used over the past 12 years for business process and service logic modelling in several application domains, including telecommunications, bioinformatics, supply chain management, e-commerce, collaborative decision support systems, as well as for software and system development. In this paper, we restrict us to the jABC facilities relevant to producing and consuming Web services.

Semantically, jABC models are control flow graphs with fork/join parallelism, internally interpreted as Kripke Transition Systems (Müller-Olm et al., 1999). This provides a kernel for a sound semantical basis for description formalisms like BPNM, BPEL, UML activity diagrams, and dataflow graphs, and constitutes a lingua franca adequate for the analysis and verification of properties, e.g. by model checking (Müller-Olm et al., 1999). BPNM and BPEL are considered different syntactic (visual) means for representing jABC models tailored for specific communities of users. In this Challenge, we chose to privilege the abstract semantic view of the executable models over ‘syntactic’ sugar, and therefore use only the jABC notation.

Concerning the data semantics, the

Using external Web services. As in WebML, the jABC mediator is largely generated automatically. The jETI framework (Java Electronic Tool Integration) (Kubczak et al., 2006; Margaria et al., 2005; Arenas et al., 2006) that enhances the jABC to support seamless integration of remote services (both REST and Web) can generate basic service types (called SIBs, Service-Independent Building Blocks) from the WSDL file of a third party service, and export the orchestrated/choreographed services inside the jABC (called SLGs, Service Logic Graphs) as Web services.

Fig. 4 shows the distributed architecture of this infrastructure. SIBs are analogous to the WebML units: both concepts represent the atomic functionality of an involved service. In the jABC, domain-specific SIB palettes are shareable among projects, and organized in a project-specific structure and with project-specific terminology. This is a simple way for adopting or adapting to different ontologies within the same application domain. Domain-specific SIB palettes are complemented by a library of SIBs that offer basic functionalities (e.g. SIBs for I/O or memory handling), or control structures (as used here), or handling of data structures like matrices (e.g. in our bioinformatics applications (Margaria et al., 2006)).

Using Web service components inside the jABC requires a valid WSDL file, or alternatively the URL with a signature. As shown in Fig. 5(top), jETI’s SIB generator extracts the information about the functionality defined in the WSDL file and creates a SIB for each function. Input parameters are handled as hierarchical SIB parameters: they enable the user to freely define input values for the web service, using the pre-existing graphical user interface of the jABC.

This is useful to face the dynamic scenarios of the Mediation problem without need of programming: if a web service changes its interface, as in the next level of the Mediator scenario, we only need to reimport its WSDL description into a (new) SIB.
By generating Web service SIBs, the execution of the service remains on the server. The SIB simply serves as a communication component with the Web service, in this example the Apache AXIS framework\(^2\) to call the specific web service. In this solution, the scenario's data and status information are maintained in the session memory. We do not need to load or store data persistently during the mediation.

### Producing Web services

To export a composite jABC service as a Web service we use our technology to generate stand-alone web services from fully implemented composite services available as jABC SLGs.\(^3\) As shown in Fig. 5(bottom)

1. Since it is required to provide the mediator as a service, we first transform the composite, hierarchical SLG of the mediator into a single SIB, using the subgraph feature of the jABC - as we usually do to provide hierarchical models as a single functionality. This creates a GraphSIB representing the corresponding SLG. Its implementation is the argument SLG, executable within the jABC Tracer, the interpreter (or a virtual machine) for SLGs.

2. To provide a Web service mediator completely independent of the jABC, we use the code generator plugin (Steffen et al., 2006) to obtain executable source code from the GraphSIB. This code is then deployed on a server using the AXIS framework, this way making the functionality accessible to other users. We then generate a WSDL description that contains all the necessary information to access the deployed service as a web service.

This way, users can call the newly added web service the way they are used to, independently from the jABC.

### Choreography

jABC originated in the context of the verification of distributed systems (Müller-Olm et al., 1999), therefore SLGs are inherently adequate as choreography models. The SIBs can physically run in a distributed architecture. They communicate directly or with a shared space (called the context). The SLGs are fully hierarchical: SIBs can themselves be implemented via SLGs. The macro mechanism described in (Steffen et al., 1997) allows defining what communication actions of an SLG are visible to the environment (for choreography). Hierarchy is however not needed for the mediator solution. Orchestration is as far as the jABC is concerned just a degenerate case of choreography.

### Data Semantics

The static data semantics is captured automatically during the WSDL-to-SIB import as the SIB parameters.

These parameters, additional semantic properties attached to the SIBs, possibly imported from an ontology, and the SIB branch labels are visible to the model checker (Steffen et al., 2006), which allows automatically proving global compliance constraints on the business logic of an SLG. These constraints are expressible in mu-calculus and its derivatives, a family of modal (temporal) logics.

Additionally, arbitrary relations between data elements can be provided as local checking expressions, with the expressiveness of Java. This facility allows expressing and checking pre and post conditions.

Neither the local checker nor the model checker were used for the mediation solution.
### Table 1: Comparison of the presented technologies

<table>
<thead>
<tr>
<th>Function</th>
<th>WebML</th>
<th>jABC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPMN model</td>
<td>Manually modelled from the SWSC task description. Manually annotated to steer the WebML generation to meet the challenge’s needs.</td>
<td>Not a distinct model, just an abstract jABC graph.</td>
</tr>
<tr>
<td>Mediator control flow</td>
<td>WebML model structure with standard units generated from BPM model. Units are then configured and other units are added from the library manually (no need for any implementation, no code generation, just component configuration).</td>
<td>Manually created SLG along the SWSC task description (by refining the abstract model equivalent to the BPMN model), using automatically generated- and standard SIBs.</td>
</tr>
<tr>
<td>Data Management</td>
<td>ER-model manually created from analyzing the RosettaNet messages and adding status information. Used to keep data persistent.</td>
<td>ER-model possible, manually created from analyzing the RosettaNet messages. Not necessary due to WSDL import. The data for the mediator are kept in the session memory.</td>
</tr>
<tr>
<td>Web Service invocation</td>
<td>Generic standard units for calls to WSs</td>
<td>Automatically generated SIBs representing WS functions.</td>
</tr>
<tr>
<td>Web Service publishing</td>
<td>Generic standard units for receiving SOAP messages.</td>
<td>WSs automatically generated and published from jABC SLGs.</td>
</tr>
<tr>
<td>Passing data to a Web Service</td>
<td>Data must be included in the SOAP messages. SOAP (XML) message templates have to be created in advance.</td>
<td>Data is passed to the WSs via the generated SIB parameters.</td>
</tr>
<tr>
<td>Receiving data from a Web Service</td>
<td>Data are extracted from the raw SOAP messages.</td>
<td>Data is received from the WSs via the generated SIB parameters that are correct by construction.</td>
</tr>
<tr>
<td>Handling XML messages</td>
<td>Standard units for handling XML messages exist, performing XSL transformations on XML messages.</td>
<td>No need to handle raw XML messages.</td>
</tr>
<tr>
<td>Monitoring User Interface</td>
<td>Standard units to generate web pages, displaying database data.</td>
<td>Monitoring of flow graphs and state information within the SLG Tracer (interpreter).</td>
</tr>
</tbody>
</table>

### 3.1 The jABC Mediator

In the Mediator scenario, we have a rather flat domain structure for the SWS specific services (see Fig. 7 left): we only distinguish SWS from common services, and we import the entities in the domain of discourse (part of an underlying ontology) through the WSDL import.

As shown in Fig. 6, we obtain directly the full structure of the XML messages. Here, we see for example the rich hierarchical parameter structure of the OMSERVICECREATENEWORDER SIB.

The workflow is created manually, by drag and drop from the palette of automatically produced SIBs

4. We are working on the automatic generation of the workflow from declarative specifications.
Figure 8: The compared Mediators: Functional correspondence of the WebML (left) and jABC (right) solutions

Figure 7: The SWS Mediator SIBs and the abstract process model

4 Comparison

Table 1 summarizes the profiles of the two solutions, which we describe in more detail below:

Workflow:

WebML covers the high level specification of the business flow by means of BPMN models. A coarse WebML skeleton is automatically generated from the BPMN model. This model contains standard units for the web service calls. Other functions that are necessary to complete these calls have to be configured to meet the actual requirements.

The jABC abstract model is essentially equivalent to the BPMN model. It can be refined manually into the mediator graph. As done here, the main domain specific (peculiar to the SWS Challenge) components (SIBs) used in this model are automatically generated from the Web Service’s WSDL descriptions. The SLG also contains standard control SIBs (provided with the jABC as a library) to realize the specific control flow for the SWSC scenario descriptions.

Data Model:

The WebML model comprises a data description model consisting of an E-R model that is derived from analyzing the data structures in the RosettaNet messages. This E-R model is used to store the BP’s data as well as status information regarding the state of the process execution. It is also possible to use in memory data storage as configuration option.

The jABC mediator does not use persistent data storage since it keeps the information in the session memory. The same ER-model could however also be realized persistently via the DB-Schema plugin, if necessary.

Dealing with WS:

The WebML solution offers four generic WS-related functional units to use or realize a Web Service’s functionality: two units to issue calls to a WS, one for sending a request and one for also waiting for a response, and two units to provide a WS functionality, one to wait for a request and one for also sending a reply. These units are parameterized (configured) with the WSDL description and with SOAP message templates that realize the particular WS functionality and return the results.
of the WS call as SOAP message. These units can also be configured dynamically at runtime passing as parameter the dynamic end point, as shown in the discovery scenario. Such feature is particular useful if combined with dynamic Adapter units since it allows to interact with arbitrary Web services and to store the results in the internal data model regardless of the invoked Web services.

The jABC solution realizes a dedicated access to external Web services: for each WS functionality, a separate SIB is automatically generated from the corresponding WSDL description. These SIBs are already fully instantiated: they provide access to the data exchanged with the WS through automatically generated SIB parameters, that can be accessed in each jABC model. Similarly, a standalone WS (including the corresponding WSDL) can be generated from each jABC SLG and automatically provided on a web server.

Dealing with XML messages: To deal with Web Services WebML has to prepare the corresponding SOAP (XML) messages, that are passed to the units that execute the call to a WS. If a WS returns a result, this value has to be extracted from the returned SOAP message as well. To do so, WebML offers standard units that perform (lifting and lowering) XSL transformations on XML messages. Eventually this operations can be performed directly in units that perform the actual Web service calls. The use of lifting and lowering adapters grants a more generic approach since they can be configured dynamically.

As in the jABC there is no need to deal with raw XML messages, no such special functions are required. The messages are created within the SIBs according to the structure prescribed by the original WSDL, which is reflected later in the complex parameters of the SIBs and thus known to them.

State Monitoring:
The WebML language offers standard abilities to display information from a relational database on a web page. This functionality can be efficiently used to monitor the state of the model workflow, as this information is stored in a relational database as well.

The jABC offers white box monitoring via its SLG interpreter, the Tracer plugin, which allows monitoring variables and communication activity of the whole hierarchical SLG, at wish separately for each hierarchy level and each thread, in case of a distributed execution. Additionally the LearnLib plugin (Raffelt et al., 2005) provides black box monitoring and automatic (re-)construction of a user-level model, based on the observation of the conversation with the environment.

5 Boiling Down to the Essence

Figure 8 compares the WebML and the jABC workflows, with a layout that respects the functionalities within the mediator solutions. Taking into account the discussed differences, it is easy to reduce the WebML solution to the jABC solution in a systematic way.

- As mentioned before, the WebML model needs a
pair of lifting and lowering actions for each web service call to create and decode the needed XML messages. These transformations are not necessary in the jABC solution, as it does not reduce communication to the level of exchanging raw XML messages. So all the service units dealing with the transformation of XML messages in the WebML model are crossed out in black in Fig. 10.

- All units dealing with database access in the WebML model are additionally identified and crossed out in red in Fig. 10. These components do not arise in the jABC modelling, which allows to also virtualize these access functions.
- The jABC models error handling explicitly while the WebML solution does not (even if the modelling language offers support for that). Therefore we remove the error handling SIBs from the jABC solution.

The remaining workflow, shown in Fig. 9, represents the essence of the desired solution, abstracted from approach-specific details of the communication, storage, and error handling choices.

6 Conclusion

In this paper we presented two solutions to the mediation in a Semantic Web Service scenario described in the SWS challenge 2006\(^5\). The two approaches, WebML and jABC, offered two different views on the mediation problem, both in terms of the design-time modeling of the solution and of the runtime execution platform. Both approaches presented advantages and drawbacks. jABC offered a more abstract and synthetic view of the solution, e.g., disregarding some grounding details of the communication; on the other hand, WebML offered a wider coverage of the technical details and of the efficient runtime execution. The WebML approach is based on software engineering and web engineering practices, while jABC takes more advantage of the SOA and web service design fields. Both the methods are not natively meant to face Semantic Web applications, but both proved to adapt rather well to this new class of problems.

REFERENCES


\(^5\)http://sws-challenge.org