Babel: Representing Business Rules in XML for Application Integration

Huaxin Zhang, Eleni Stroulia
Computing Science Department, University of Alberta
Edmonton, AB, T6G 2H1, Canada
{hxzhang,stroulia}@cs.ualberta.ca

Abstract[1]

In this paper, we discuss Babel, a prototype tool for integrating multiple heterogeneous applications, by wrapping them and by specifying the logic of their interoperation in XML.

1. Introduction and Background

Application interoperability is a challenging problem because different applications assume different data types and control-of-processing models. Even when related applications are aggregated in so-called portals, it is up to the user of the portal to access the different applications, collect information, and then combine this information to access yet other applications.

There has already been a lot of work on integrating existing data resources, such as MIX [2] and CQ [3]. Both these projects focus on query planning over multiple sources of data to answer complex user's questions (MIX) or to continuously monitor these resources in order to generate alarms of interest to the user (CQ).

The Babel project takes this work one step further: from query planning and monitoring to application monitoring and integration. Babel supports the specification of related applications in terms of the functionalities they deliver and the data they expect as input and produce as output. Furthermore, it enables the specification of "business rules" for how these functionalities should be integrated. Based on these specifications, Babel produces a run-time mediator that receives records of the behavior of the underlying applications, evaluates the defined rules on the global state of the integrated system, and, in response, generates triggers for more behaviors to be accomplished by other applications in the interoperating consortium.

2. The Babel System Architecture

The Babel mediator is a component in a three-tier environment. At the lowest tier exist the integrated applications and in the middle tier exist the wrappers of these applications. The Babel mediator is situated at the top tier, interacting with the applications through the wrappers.

Three important differences distinguish Babel from related systems. First, Babel has been designed and developed based on XML and its related languages and APIs. All information exchanged between the mediator and the wrappers are XML documents. The mediator itself uses XSLT—an XML transformation standard—to apply the predefined rules on the data exchanged. Second, Babel integrates applications, not databases. These applications have their own control of processing and each one makes different assumptions regarding its interaction with users and other applications in its environment. The Babel design-time environment enables the specification of explicit rules for how these independent applications may interact with each other, i.e., what data they exchange and under what conditions. These "business rules" define the data and control inter-dependencies among the pre-existing applications and therefore, the overall behaviors of the integrated system. Finally, the wrappers of the underlying resources, with which the Babel mediator interacts, are of a novel type. They consist of (a) a canonical multi-platform user interface and (b) a planner that, given sufficient data, can navigate through (and interact with) the original user interface of the underlying application to accomplish a task. When accessed through its user interface, a wrapper records the task data exchanged between the user and its underlying application and feeds these records to the Babel mediator. The mediator, in turn, processes the records, evaluates whether they trigger some pre-defined business rule, applies the rule, and if the rule succeeds, generates new task data. This data is then forwarded to some other wrapper, to execute another task in another application in the interoperating consortium.

2.1 The Babel Mediator

The Babel mediator receives as inputs *tasks* and *rules*, specified in XML. Task XML documents contain the actual data that flow between the underlying applications. A task is the record of some specific behavior instance of an underlying application. Rule XML documents define the required data manipulation that the Babel mediator is intended to accomplish at run time. Each rule is the XML specification of an Event-Condition-Action rule,

[1] This work was funded through a CRD grant by NSERC-CRD 215451-98 and Celcorp.

implemented by a XSLT program. The output of the Babel mediator is also task XML documents, containing only the data inputs for the tasks to be executed by an application wrapper. All Babel data, tasks and rules, are built according to predefined DTDs.

The run-time Babel mediator interacts with the wrappers through an RMI interface and consists of a *Data Validation* component, a *Rule Manager*, and a *Database Manager*. The Data Validation Component evaluates whether the incoming tasks or rules are well formed and valid. The Database Manager controls the mediator's task repository, recording new tasks to its history tasks depository and providing the history tasks DOM to the rule-manager. To overcome memory limitations, Babel uses RDBMS for its task repository. Babel saves the parsed task DOMs as byte streams in an RDBMS together with the task's meta-data, i.e., type, and time-stamp. The Rule Manager initiates a fixed number of "working threads" when the Babel mediator starts. As tasks are received, the rule-manager identifies all the rules applicable to the incoming tasks, and adds each of these task-rule pairs to the "bag of jobs" to be processed by the "working threads". Each working thread applies the rule on the task in the context of all the history tasks. To do that, the rule manager gets the DOM tree of the history from the database. The relevant rules are applied and the task instances implied by the rules are sent to the destination application wrappers.

2.2 The Babel Design-time Environment

The interoperation rules of the Babel mediator are defined with the Visual X-Rule Generator (VXG) of Babel. The tool's graphical user interface provides a tree-like document structure for the incoming task (event), the history of tasks (part of the condition), and the output task (action) according to the task DTD.

The VXG user defines the rule condition by applying simple conditions on the document elements of the incoming task and the tasks in the history and by combining these conditions with compound logic operators. Furthermore, the VXG user specifies the data elements of the new task by dragging and dropping constrained elements from the incoming task and the task history to the output task document. VXG automatically constructs XSLT programs to implement the operations defined by the user, thus defining rules using VXG does not require the user to possess much knowledge of XSLT programming.

The design-time environment serves two purposes. It is a planning tool for the application integration process. The environment enables modeling the integration to ensure that all the underlying applications are capable of providing the required information and that each component is receiving the information it requires. It also constitutes the configuration tool for the run-time mediator. It creates the rules according to which the mediator handles the data-exchange operations for the modeled integration.

3. An Example

To illustrate the functionality of the Babel mediator, let us consider two systems: a text-based legacy library that allows users to search for books using their title, and an email system. Suppose that we construct an email-reader and an email-sender wrapper and a library-search wrapper using the CelLEST environment [1]. We can then define two rules for Babel in VXG: "*When a user emails a book title as subject, a search task is forwarded to the library wrapper*", and "*When a search task is executed, a send-mail task is forwarded to the email-sender wrapper to inform the user, who have requested searches with the same title, of the search results*".

At run-time, when a user sends an email with a search request the email-reader wrapper records the relevant data of the executed task and forwards it to the Babel mediator. In response, the Babel mediator invokes the search query and when the wrapper has executed it, it sends a record with the task data back to the mediator. At this point, the mediator creates a new email-send task and forwards it to the email-send wrapper that send the relevant email. Finally, the user may access the message through the email-read wrapper.

4. Reflections

This example illustrates how Babel integrates multiple heterogeneous, independent applications, and increases the usefulness of legacy systems, such as the library system. Users do not need to use the library text interface and do not have to wait to receive the results of their search. We are currently working on (a) extending VXG to construct more complex XSLT programs, for more complex rules (such as, "*if the user has borrowed a book more than 3 times, send a message to urge him to buy it from an on-line book retailer*"), (b) enabling the mediator to execute them efficiently at run time, and (c) exploring other models for complex, web-based mediators [4].

References

[1] E. Stroulia, M. El-Ramly, L. Kong, P. Sorenson, B. Matichuk: Reverse Engineering Legacy Interfaces: An Interaction-Driven Approach. In Proc. of the 6th Working Conf. on Reverse Engineering, 292-302, IEEE.

[2] C. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, V. Chu: XML-Based Information Mediation with MIX. In exhibition program, ACM Conf. on Management of Data, 597-599, ACM.

[3] L. Liu, C. Pu, W. Han: XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources. In Proc. of the 16th Int. Conf. on Data Engineering, 611-621, IEEE.

[4] E. Stroulia, J. Thomson, Q. Situ: Constructing XML-speaking wrappers for web-applications: Towards an interoperating WEB. In Proc. of the 7th Working Conf. on Reverse Engineering, 59-69, IEEE.