Champagne: Data Change Propagation for Heterogeneous Information Systems

Ralf Rantzau Carmen Constantinescu Uwe Heinkel Holger Meinecke

Computer Science Department
University of Stuttgart
Breitwiesenstr. 20–22, 70565 Stuttgart, Germany
{rantzau, constantinescu, heinkel}@informatik.uni-stuttgart.de

Abstract

Flexible methods supporting the data interchange between autonomous information systems are important for today's increasingly heterogeneous enterprise IT infrastructures. Updates, insertions, and deletions of data objects in autonomous information systems often have to trigger data changes in other autonomous systems, even if the distributed systems are not integrated into a global schema. We suggest a solution to this problem based on the propagation and transformation of data using several XML technologies. Our prototype manages dependencies between the schemas of distributed data sources and allows to define and process arbitrary actions on changed data by manipulating all dependent data sources. The prototype comprises a propagation engine that interprets scripts based on a workflow specification language, a data dependency specification tool, a system administration tool, and a repository that stores all relevant information for these tools.

1 Introduction

The IT infrastructures of many enterprises are highly diversified, and both applications and data management systems are constantly evolving. The integration of data as well as of functionality is generally termed *enterprise application integration* (EAI). In our approach, we focus on data integration. Instead of designing a single data model that fits the needs of all applications in an enterprise, we build "bridges" called *dependencies*. They connect only those data subsets managed by distributed information systems that actually share information. By defining such dependencies between data schemas, a user can specify the effects of a source system's data change on the data in all dependent systems. Such a loosely coupled approach is of great importance to many companies for several reasons: A full integration of the information models that are managed by an enterprise is often too costly. Rather, there is a trend for the past few years to keep information systems as autonomous as possible. Furthermore, new systems and legacy systems have to be added and managed without much effort because IT infrastructures are dynamically adapting to the needs of an enterprise.

Our focus is *not* on providing a global view for querying data sources but on providing a simple and flexible method for propagating data *updates* between autonomous information systems.

We developed our prototype as part of a larger research project on innovative concepts and techniques to enable highly flexible series production systems in the manufacturing industry (SFB 467, funded by the Deutsche Forschungsgemeinschaft) [2].

2 Terms and Definitions

In our approach, any software system providing access to data is called an *information system*.

A *data schema* is a specification of data structures for an information system. Every information system may have one or more schemas, and a single schema can be used in more than one information system. A 1-to-N relationship between a source system/schema combination and one or more destination system/schema combinations is called a *dependency*.

A *data change* in one system may require a change in all dependent systems, sometimes involving com-

Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

plex data transformations that may be different from the operation applied to the original data. For example, an insertion of an object in system A may lead to an update of an object in system B, if B stores aggregate values (sums, averages, etc.) of the object values stored in A.

Change propagation is the process of forwarding a data change from a source system to all dependent systems. The process of change propagation includes transformations and filtering of the changed data.

A *transformation* is an operation that maps given input data into output data according to a specification. The specification defines how the input data has to be adapted to represent valid destination data.

A *filter* is an operation that tests a boolean expression on input data. It returns the unchanged input data if the condition is true, otherwise no output data is returned.

3 Architecture

In this section, we present the building blocks of our prototype and show how they are used to define dependencies and to propagate data changes.

The combination of XML technologies and the explicit treatment of dependencies via a modular design guarantees the necessary functionality and flexibility of data propagation in EAI. We consider our prototype somewhere between Microsoft BizTalk Server, offering transformation facilities for business data, and IBM DB2 DataPropagator, which is primarily designed for database replication. Our system differs from these products by offering a more flexible specification of dependencies. Furthermore, unlike DataPropagator, we propagate more than merely database contents.

3.1 System Overview

Our prototype, called *Champagne* (**cha**nge pro**pag**atio**n** **m**anager), consists of two main components:

- the *dependency manager* that allows to define dependencies between the data schemas of several information systems, and
- the *propagation manager* that is responsible for propagating changed data of a source system to the respective destinations.

While the dependency manager is an interactive tool used for dependency specifications during design-time, the propagation manager is employed during runtime of the change propagation process.

Both the dependency and the propagation manager access the repository. It stores dependencies, schemas of each information system involved, propagation specifications, and further data to manage the processing of propagations.

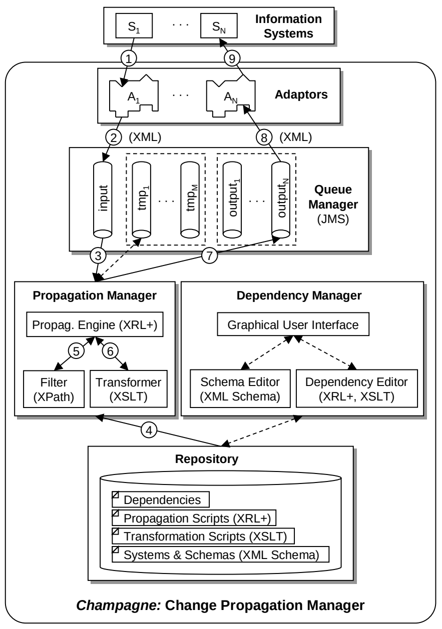

Figure 1: Architecture of *Champagne*. The technologies employed are indicated in brackets.

For the sake of platform independence, a vital requirement in heterogeneous environments, we built all components in Java.

Figure 1 illustrates the architecture of Champagne as well as the control flow during the processing of a data change in source system S_1 and its effects on the dependent system S_N. This flow is described in Section 4.

3.2 Dependency Manager

The dependency manager allows specifying all the information needed by the propagation manager to propagate data. With the help of the dependency editor, a user can create, update, and delete dependencies in the repository. To create a dependency, one can browse the names of the systems registered in Champagne, as well as their associated schemas. These are specified in the schema editor using XML Schema. Furthermore, editors can be used to modify propagation scripts that refer to one or more transformation scripts. Both propagation and transformation scripts are explained in detail in Section 5.

3.3 Propagation Manager

The propagation manager is the runtime component of our architecture. It processes changed source data according to the propagation script that is associated with a dependency. The resulting data is then delivered to another component of Champagne, called adaptor. Adaptors map data between the local schemas and the corresponding XML schemas.

A propagation script is interpreted by the propagation engine, which calls the components *transformer* and *filter*. The transformer processes a transformation script, while the filter checks a condition on the values of a data object.

3.4 Adaptors

An *adaptor* provides a bi-directional translation between a local data representation and a representation that conforms to an XML schema. The designer of the adaptor has to define an appropriate schema in the repository of Champagne using the schema editor.

Adaptors are responsible for putting a message describing changed data objects of their associated information system into the input queue of Champagne and for fetching a message for their system from the respective output queue.

Suppose, for example, that the information system is a relational database system. If a table of a database is the *source* of a dependency, then triggers can be employed to deliver the changed data objects (set of records) to the adaptor. If the database system is the *destination* of a dependency, then the adaptor may use SQL DML statements to update/insert/delete the affected records in the database.

We distinguish two types of adaptors, depending on the requirements of the dependency and the characteristics of the data source: active adaptors that are able to detect a change and passive adaptors that are notified by the data source.

3.5 Repository

The components of Champagne use a repository based on an object-relational database system for all objects that need to be stored persistently: (1) dependencies, consisting of the IDs of the source and destination systems and schemas, as well as the ID of the propagation script, (2) propagation scripts involved in any dependency, (3) transformation scripts referenced in any propagation script, (4) information system IDs and names and their associated XML schemas, and (5) authentication information needed when systems connect to Champagne.

For the remainder of the project, we plan to store even more data in the repository. Some examples are the time when a system has connected to or disconnected from Champagne, and a log of (a subset of) the messages exchanged. Such a data collection will then be subject to analysis with business intelligence tools and may reveal a potential for communication optimizations. For example, it may show that currently distributed data should better be integrated into a single schema because of a considerable communication overhead observed for propagations.

3.6 Messages and Queues

To enable both synchronous and asynchronous communication between the adaptors and the propagation manager, we use *Java Message Service* (JMS). The in- and output of an adaptor is a JMS message consisting of three parts: header, properties, and body. The action that has occurred in the source system (update, insertion, or deletion) is specified in the message properties. The message body contains the XML representation of one or more changed data objects and it conforms to a single (source or destination) XML schema. The XML document reflects either the values of a newly inserted object, the new values of an updated object, or the values of a deleted object.

The queue manager provides an output queue for each adaptor and a single input queue for all adaptors. In addition, the propagation manager may use temporary queues to store intermediate XML documents after each transformation.

4 Processing Model

We briefly describe the control flow that is initiated by a data change in an information system, as illustrated in Figure 1. First, the adaptor A_1 of system S_1 either detects a changed data object or is notified by S_1 (1). Then, A_1 maps the changed data object into an XML representation conforming to an XML schema, which has been stored in the repository using the dependency manager. The adaptor puts a message containing the XML representation of the changed object into the input queue of the queue manager (2). Then, the propagation manager fetches the new message from the input queue (3) and retrieves all dependencies from the repository where the source matches the system/schema combination of the input data object (4). A propagation script, described in the next section, is associated with each dependency. The script is interpreted by the propagation engine, which interacts with the filter and the transformer components to process and filter the given source data object according to specifications in the propagation script (5, 6). During the processing, intermediate transformation results are stored in temporary queues. The final data objects that result from the transformations defined in the propagation script are put into the respective output queues of each dependent destination system (7). Then, the adaptor of each destination system fetches the message from its output queue (8), maps the XML representation of the data object in the message body into its local representation. Finally, the adaptor performs or triggers the operations (insertion/update/deletion), which are also specified in the properties of the output message (9).

5 Propagation and Transformation Scripts

The processing of an input XML document is specified as a *propagation script* that conforms to an XML language that we call XRL+, which is based on the *eXchangeable Routing Language* (XRL). Originally, XRL has been proposed for workflow specifications [1]. Among other things, XRL+ provides constructs for parallel and sequential execution, propagations, transformations, filtering, as well as event wait conditions.

We have implemented many elements of XRL and extended the language with new elements that are tailored for defining a propagation process. The key benefit of propagation scripts is their flexibility for defining the operations to be performed on the data. The main new elements and their attributes are:

- *TRANSFORM* (*xml_in*, *xml_out*, *xslt*),
- *FILTER* (*xml_in*, *xml_out*, *xpath*),
- *MESSAGE_EVENT* (*system*, *schema*, *xml_out*), and
- *PROPAGATE* (*system*, *schema*, *xml*).

The attributes *xml*, *xml_in*, and *xml_out* are IDs of XML documents, *xslt* is the ID of a transformation script, *xpath* is an XPath expression, and *system* and *schema* are the IDs of the source or destination system and schema used.

Our XRL+ engine interprets a propagation script. It delivers a transformation script to the transformer, an *XSLT* (eXtensible Stylesheet Language: Transformations) processor, whenever it encounters an XRL+ *TRANSFORM* element. If the engine processes a *FILTER* element, the filter's XPath expression is evaluated with an XML document as input. The *MESSAGE_EVENT* element waits for the arrival of a specific message, belonging to a given system/schema combination, and fetches it from the input queue. Finally, the *PROPAGATE* element sends a message containing transformed data objects to a destination system's output queue.

The example in Figure 2 illustrates the flexibility of this approach, using a concise graphical notation instead of the actual XRL+ code. Here, the person data of an information system is transformed into the data structures and contents expected by two other systems. Both destination systems store a person's age instead of the birth date. All operations that follow the first transformation are processed in two parallel threads. While the marketing database is interested in adults, the mailing system only stores data on female customers.

A transformation is specified in an XSLT file that we call *transformation script*. Such a script consumes and produces an XML document, offering powerful mapping operations. The input/output document conforms to an input/output XML schema that is defined in the repository.

6 Demonstration Outline

The first part of the demonstration shows how a new dependency is defined using the graphical user interface of the dependency manager. After connecting to the repository, a user can browse the XML schemas of each information system that has been registered in Champagne. Then, the user selects a system and one of its schemas as the source of a new dependency. The user edits a propagation script by selecting elements from the GUI and supplying them with appropriate parameter values. The dependency's destination systems and schemas can be selected from a list. Then, the user can choose reusable transformation scripts from the repository and add them to the propagation script or create new ones. Finally, the user registers the dependency in the repository, which automatically stores all newly defined transformation scripts together with the propagation script.

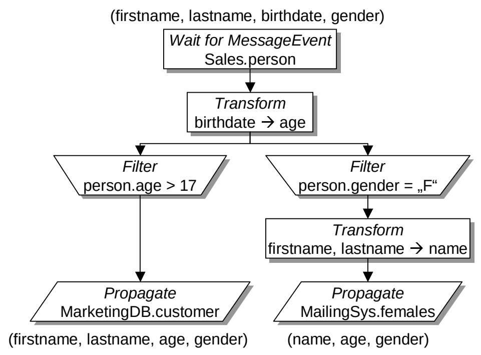

Figure 2: Sample propagation script with two output schemas.

The second part of the demonstration illustrates how data changes are processed and propagated by Champagne. First, the user can browse the list of example information systems. We show several scenarios involving relational database systems, like IBM DB2 UDB and Microsoft SQL Server, as well as a raw file system. We illustrate the effects of arbitrary manipulations of tables and files that are subject of one or more dependencies.

Finally, we show an administration tool of Champagne that monitors the current propagation activities. In addition, the tool displays statistical information, such as the number of messages that are waiting in any queue, or the average message processing rates. Furthermore, a user can specify various system parameters for tuning the performance of Champagne.

References

[1] W. v. d. Aalst and A. Kumar. XML Based Schema Definition for Support of Inter-Organizational Workflow. In *Meeting on XML/SGML based Interchange Formats, Conf. on Application and Theory of Petri Nets, Aarhus, Denmark*, June 2000.

[2] C. Constantinescu, U. Heinkel, R. Rantzau, and B. Mitschang. A System for Data Change Propagation in Heterogeneous Information Systems. In *Proc. ICEIS, Cuidad Real, Spain*, April 2002.