TRANSFORMATION-ORIENTED MIDDLEWARE FOR LEGACY SYSTEM INTEGRATION

Guido Menkhaus
Software Research Lab, University of Salzburg
Austria

Urs Frei
University of Applied Science, St. Gallen
Switzerland

Keywords: Legacy System, Integration, Transformation System, Grammar, Middleware.

Abstract: Most established companies have acquired legacy systems through mergers and acquisitions. The systems were developed independently of each other and very often they do not align with the evolving IT infrastructure. Still, they drive day-to-day business processes. Replacing the legacy application with new solutions might not be feasible, practical or cost a considerable amount of time. However, immediate integration might be a requirement for a strategic project, such as supply chain management or e-business. This article presents a transformation system for legacy system integration that allows flexible and effective transformation of data between heterogeneous systems. Sequences of transformations are described using a grammar based approach.

1 INTRODUCTION

Supply chain management helps companies in controlling the flow of information and goods within their network of suppliers and customers by providing a full view on what happens in the network (Hieber, 2002; St"or et al., 2003). But before extending operation management beyond the company's wall and integrate companies' suppliers and customers into a single information network, the company's own operations must run smoothly towards cooperation and collaboration. This involves the integration and interoperability of different corporate databases, applications, and more and more often of legacy systems, acquired through mergers and acquisitions. These legacy systems produce structured or semi-structured data that add to the vast amounts of data that a company generates every day. This data needs to be communicated between heterogeneous systems within the same company and eventually beyond the company's walls. Transformations of communicated data are required to enable companies to tightly integrate their systems into a cohesive infrastructure without changing their applications and systems(DataMirror, 2001).

This article presents a legacy system data integration middleware that allows flexible and effective transformation of data between heterogeneous systems. Our data integration middleware provides a transformation system in which transformation sequences are described based on the grammar of the format of the source and the target data. It provides direct integration of applications and systems at the data level.

The remainder of the article is structured as follows: The motivation of this work is discussed in Section 2. Section 3 provides a brief overview about transformation systems. Section 4 describes strategies for legacy system integration and migration. Section 5 illustrates a use-case scenario for the legacy system data integration middleware. The architecture of the system is presented and discussed in Section 6. Section 7 concludes the article with a brief talk about our future research directions and work.

2 MOTIVATION

For companies to stay competitive, they must be able to interconnecting seamlessly their database, applications and legacy systems into an coherent IT infrastructure. However, heterogeneous systems including legacy systems, acquired through mergers and acquisitions, may not exchange data so easily. These systems produce data in different formats using different description languages, such as comma-separated-value lists or text-based proprietary formats. To communicate data from one system using a format A to a different system using format B, we need to transform

Menkhaus G. and Frei U. (2004).
TRANSFORMATION-ORIENTED MIDDLEWARE FOR LEGACY SYSTEM INTEGRATION.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 202-209
DOI: 10.5220/0002618702020209

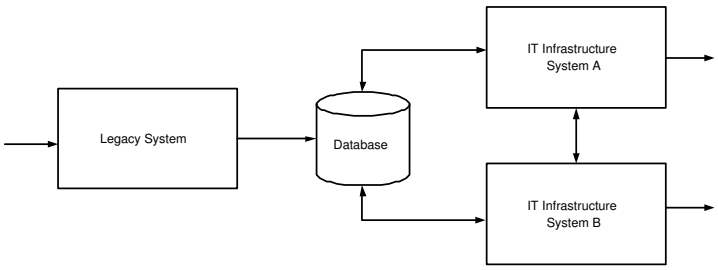

Figure 1: Application scenario: Integration of legacy systems.

the data, control that data and ensure that the transformation from format A to format B is correctly carried out.

Transforming data is usually done by writing custom programs (Fiore, 1998). However, if either the format of the source data or the target data changes, the custom programs need to be rewritten. Adapting to frequent changes results in high maintenance costs. There are systems that allow data integration. For example, relational databases allow the integration of comma-separated-value lists. However, this integration has limitations. The data is imported into a single database table, which need to be further processed internally, to integrate the data into different tables.

To integrate legacy system data we need middleware that provides the following features:

1. *Adaptation:* The way data is processed and stored is diverse and might be subject to changes. If the format of the source data or the target data in a transformation sequence changes, quick adaptation to the transformation sequence is essential to sustain system interconnection.

2. *Control:* When data is transformed while communicated between two systems, data might need not only change the format but the target system might require the data to change, to be enriched, filtered, and modified.

3. *Format Guarantee:* The transformation sequence guarantees that the data results in a specified format. The specified target structure of the data is produced, because the transformation is generated based on the structure of the target format described by a grammar.

We present a grammar-based transformation system, in which the transformation sequence is generated originating from a set of grammars describing the target formats of each transformation step. Semantic controls need to be programmed manually. The system provides means to integrate them into the transformation sequence. Adaptation is accomplished by respecifying the grammars of the data formats.

3 SHORT OVERVIEW OF TRANSFORMATION SYSTEMS

Transformation systems transform elements of a source language into elements of a target language. The source and the target language can be very different from one another (Winter, 1999). Partsch and Steinbruggen classify transformation systems into manual, semi-automatic, and automatic transformations (Partsch and Steinbruggen, 1983).

- *Manual:* In manual transformation systems, the user chooses from a predefined set of transformations those, which the user wants to apply to the source language. Manual transformation systems provide an environment that puts the user in the position to use transformations more effectively than the current programming paradigm that requires s programmer to manually code a transformation.
- *Semi-automatic:* The objective of semi-automatic transformation systems is to automate the process of transforming and to minimize the intervention of the user. Although the major decisions will still be made by the user.
- *Automatic:* The intent of automatic transformation system is to fully automate the transformation process.

The class of problems that can be solved using manual transformation systems is the largest, since most transformation solutions require insight in the problem domain and decision taking that is beyond what automation techniques can do. Semi-automatic systems need a restricted problem domain where difficult decisions about transformation configuration do not occur and transformations can be generated automatically. Most limitations are in the automatic transformation system class, where the system selects on the basis of a knowledge base the transformation sequence. However, the system can only be as good as the programmer has designed the knowledge base.

Restated from a different viewpoint, Partsch and Steinbruggen divide transformations in (Partsch and Steinbruggen, 1983) into two types of processes: *procedural* and *schematic*.

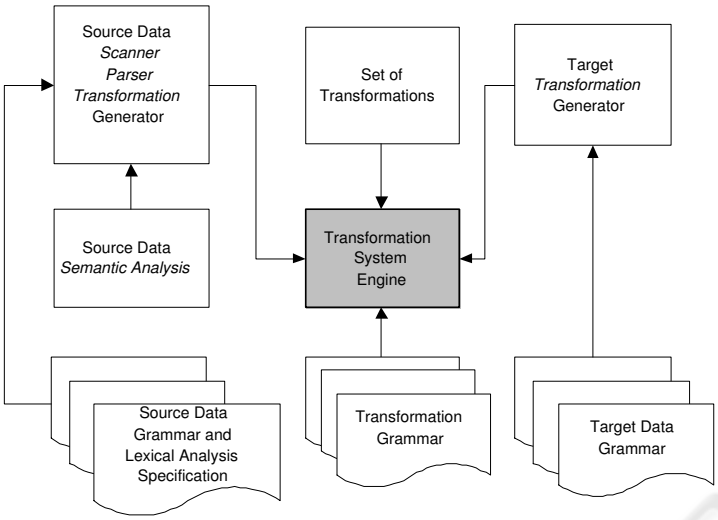

Figure 2: Transformation System Architecture.

- *Procedural:* Procedural transformations specify semantic rules that can be applied globally to the entire source data. They include consistency checks and analysis tasks.
- *Schematic:* Schematic transformations are syntax-oriented and make local changes to the source data.

It should be noted that global, procedural transformations can be accomplished by schematic processes, but that the required transformation might become arbitrary complex. Complex rules are better expressed applying procedural than schematic transformations (Winter, 1994).

In this paper, we present a transformation system that is semi-automatic. The automatic part of the system is schematic-based and syntax-oriented. The procedural part of the transformation consists of semantic analysis and actions, which are applied to the entire source data.

4 LEGACY SYSTEM INTEGRATION AND MIGRATION

Legacy systems are generally defined as ”any information systems that significantly resist modification and evolution” (Brodie and Stonebraker, 1995). Legacy systems still drive day-to-day business processes (IBM, 2003). Migrating the application, i.e. replacing the legacy systems with new solutions, might not be feasible, practical or costs a considerable amount of time. The legacy systems may operate in business critical processes and immediate integration might be a requirement for a strategic project, such as supply chain management or e-business. Legacy system integration deals with accessibility and availability of data, in a way that legacy systems align with the new IT infrastructure.

Bateman and Murphy propose the forward and reverse migration methods for legacy system integration and eventually migration (Richardson et al., 1997). We follow their line of argument:

- *Forward Migration:* Forward migration integrates the legacy system into the new IT infrastructure before it attacks its migration. It integrates the legacy system by transforming and continually importing legacy data to a relational database. It then incrementally migrates the legacy system’s interfaces and business processes. While the application is being redeveloped, the legacy system interoperates with the new IT infrastructure using transformation oriented middleware that operates as a gateway (Wu et al., 1997). The middleware translates and transforms the legacy system’s data and imports them into the database.
- *Reverse Migration:* Reverse migration gradually redevelops the legacy system and integrates the new applications as soon as they are capable of partly or completely replacing the legacy system. During the redevelopment phase, the legacy system remains operable on the original platform.

Forward migration might results in a longer transition phase, because migration consists in a migration and an additional integration step. Reverse migration, however, blocks further progress in other areas while the legacy system is being redeveloped. Is progress a mission critical issue, reverse migration is not an op-

tion.

The transformation-oriented middleware proposed in this article is a part of a forward migration approach for legacy system integration.

5 APPLICATION SCENARIO

The transformation system was designed as middleware for the integration of legacy systems. This section outlines in brief a pratice area, which demonstrates the value of our transformation system.

A company requires integrating a legacy system into its new IT infrastructure (Figure 1). The legacy system's integration is accomplished via a central database, which is used by various systems. The legacy system's output data is in a proprietary format, and the data needs to be imported into a variety of database tables. The database configuration was designed with respect to the new IT infrastructure and the legacy system must adapt and integrate to the new structure.

In our application scenario, the legacy system produces data as comma-separated-value lists. This data is checked and verified, transformed into an internal and intermediate XML format, and finally imported into the database.

6 ARCHITECTURE OF TRANSFORMATION SYSTEM

The architecture of the transformation system is illustrated in Figure 2. The transformation system runs a sequence of transformations, in which source data complying with a source data grammar is transformed into target data described by the target grammar. The schematic part of each transformation sequence is generated using parser and transformation generating systems. The procedural aspect is manually programmed and integrated in the schematic part.

Each transformation in a sequence consists of three intermediate subtransformations:

1. Source Grammar Driven
2. Configuration Driven
3. Target Grammar Driven

Source Grammar Driven Subtransformation The source grammar driven (SGD) transformation consists of the following four processes:

1. *Lexical Analysis:* The lexical analysis is done using a scanner component. The scanner is generated on the basis of a lexical analysis specification of the source data and produces a sequence of tokens. A token is a syntactically structures symbol, whose structure is described in the lexical analysis specification.
2. *Syntactic Analysis:* The sequence of tokens produced by the scanner is forwarded to the parser, which verifies the structure of the source data against the source data grammar. We use an attributed grammar, which can be seen as dynamic description of a transformation process, i.e. a syntax-driven algorithm.
3. *Semantic Analysis:* The semantic analysis checks local and global context conditions, during the syntactic analysis phase.
4. *Transformation:* The transformation converts the data that has passed the syntactic and semantic analysis into an internal, intermediate format.

We use CoCo/R (M"ossenbeck, 1990) as transformation tool. The lexical analysis specification is described by regular expressions. The attributed grammar of the source data is defined in EBNF.

Attributed grammars were introduced by Knuth in (Knuth, 1968) to formalize the semantics of context-free languages. They describe in their original form dependencies between attributes of symbols, originating from the lexical analyzer. However, attributed grammars can be seen as a dynamic description of a process, i.e. as a syntax directed algorithm. The structure of the source data determines the order of the global semantic analysis and the local transformations.

Configuration Driven Subtransformation A crucial part of the transformation system is the configuration driven subtransformation (CD). The transformation system contains a set of CD types with associated configurations for different target data formats. The CD transformation is an intermediate transformation that functions as a bridge. It decouples the source data grammar from the target data grammar so that the two can vary independently. This avoids a binding between the associated transformations and allows flexible adaptation in case of a modification or extension of the source or the target data grammar.

Target Grammar Driven Subtransformation The target grammar driven (TGD) transformation is generated from the target data grammar. It takes the data from the CD transformation and generates data in the target grammar format.

Since the transformation is produced from the target grammar, the transformation system guarantees that the data results in the specified format.

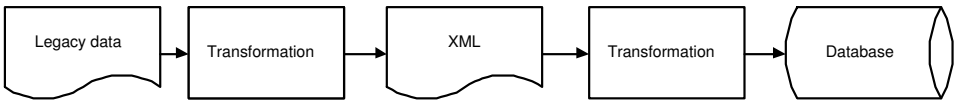

Figure 3: Transformation Sequence for Legacy System Integration.

6.1 Legacy System Integration

In the application scenario we take the data from the legacy system and import the data into a central database.

The transformation system applies a sequence consisting of two transformations. The first transformation converts the legacy data into XML format while verifying the data during the SGD subtransformation. The second transformation parses and processes the data and imports them into the database.

6.1.1 Token-XPath Matrix

The first transformation converts the legacy system's proprietary data format into an intermediate format.

In the SGD subtransformation, the scanner and parser are generated and perform a syntax check on the source data. The semantic verification (in our application scenario suppression of duplicate data entries in the source data) is manually programmed and integrated into the generated parser.

The CD subtransformation determines where data, originating from a token produced by the scanner component and semantically checked and converted during the semantical analysis, is inserted into the resulting XML document, serving as an intermediate data format in the transformation sequence. This is performed applying a Token-XPath-Assigment matrix (TXPA matrix) $M_{TX} = T \times X$, which consists of the tokens symbols T of the source data grammar and the target data grammar XML elements, expressed as XPath elements X.

The target grammar is presented as a XML Schema. The target grammar driven subtransformation is generated using JAXB (SUN Microsystems, 2003), which generates a suite of hierarchical classes that produces an XML document complying with the XML Schema. This suite of classes is subsequently used by the CD and the TGD transformation. They represent an interface that both transformations apply in cooperation using introspection.

The intermediate (CD) subtransformation decouples the source and the target grammar driven subtransformation (Figure 4). If the source or the target grammar is modified or the semantic analysis changes, only the TXPA matrix needs to be adapted. This makes the transformation system flexible and robust in the case of changes.

6.1.2 XPath-Database Configuration

The second transformation imports the data from the XML document into a database. Most databases allow importing XML data, or comma-separated vaue lists. However, data can only be inserted into a single table, and most often this data requires further processing such as splitting the data and distributing the data among several database tables.

The SGD transformation is accomplished employing an XML parser. The CD and the TGD transformations use OJB (The Apache DB Project, 2003). OJB generates a set of classes on the basis of a database design allowing transparent persistent mapping of Objects against relational databases. It allows storing objects, or part of an object in relational databases, and reading data from a relational database into the generated object structure.

The grammar oriented transformation needs to rework the data from an XML into a OJB object representation. The OJB object structure is then imported into the database (Figure 5).

The objective of the CD transformation is to remain independent from the grammar of the source XML document and the target configuration of the database. We need to take into account the following requirements:

1. Specification of a mapping between XML elements and OJB objects.
2. Instantiation of OJB objects creating a new dataset.
3. Relations between the OJB objects.
4. Processing of duplicate datasets. Duplicates are already filtered out in the first transformation. However, at this stage we cannot detect duplicates, which might occur during the reordering of the data in the second transformation, nor can we detect duplicates that are already in the database.
5. Declaration of an import sequence to prevent primary key violation.

We have developed XML2OJB, a mapping from XML documents to OJB object structure (Appendix A). It allows flexible, adaptable, and independent import of arbitrary structured XML data into arbitrary database table configuration.

Appendix A shows part of an example where an XML address list is inserted into a database. The XML2OJB configuration is divided into five parts.

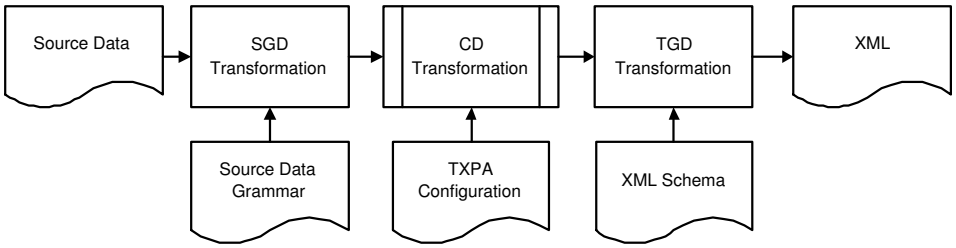

Figure 4: Transformation from legacy data to XML.

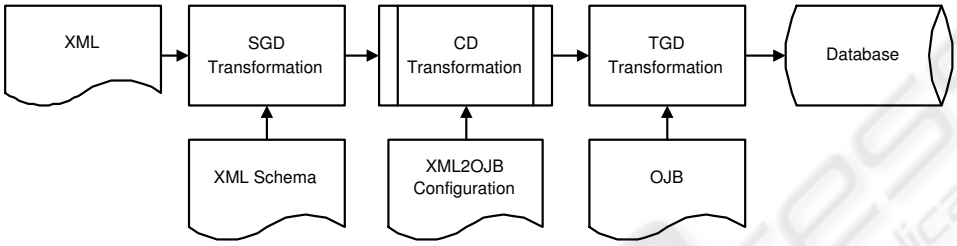

Figure 5: Import from XML into a database.

- *ClassDefinition:* The ClassDefintion section defines the objects that are imported into the database. These objects are instantiated and the attributes are set using the methods specified in the SetMethod element. Aliases are declared, which are later used in the SourceDocument section.

- *Assembly:* The Assembly section defines the behavior when importing a new data record. The **Repeat** element specifies the start of a new data record in the XML document. The **Insert** element specifies where the data is set in the OJB objects, and the **CreateObject** element defines the objects that are required to be instantiated.

- *DuplicateRecord:* The Duplicate Record section specifies the element that functions as autokey. The specification of an autokey is necessary to avoid duplicate entries in database tables.

- *ImportSequence:* The ImportSequence section determines the sequence in which the objects import their data into the database.

- *SourceDocument:* The SourceDocument element declares where to find the necessary information in the XML source document.

The TGD transformation consists of importing the set of OJB classes into the database. The process is configured using a specific OJB configuration file.

7 CONCLUSION

We have presented a transformation system that manages sequences of transformation. Each transformation of a sequence consists of three subtransformations and is grammar driven. The source grammar driven subtransformation converts data into an intermediate format. The inner subtransformation is a bridge between the data represented in source and target format. The target grammar driven transformation converts the data from the intermediate format into the resulting target format. The introduction of an intermediate transformation allows the source and the target grammar to vary independently.

Currently, there are two transformation configurations. The TXPA matrix maps a sequence of tokens onto XML elements. The XML2OJB configuration maps XML elements to OJB objects, which can be imported into a relational database. The XML2OJB transformation proved to be successful due to its flexibility. The architecture of the transformation system represents a viable solution to systems that require frequent reconfiguration and maintenance.

Future work will focus on extending the set of predefined transformations. We will continue working on fault tolerance and error recovery within a single transformation.

REFERENCES

Brodie, M. and Stonebraker, M. (1995). *Migrating Legacy Systems Gateways, Interfaces and the Incremental Ap-*

proach. Morgan Kaufman.

DataMirror (2001). Managing your data the XML way: Data transformation, exchange and integration.

Fiore, P. (1998). Data Warehousing. *Evolving Enterprise*, 1(1).

Hieber, R. (2002). *Supply Chain Management. A Collaborative Performance Measurement Approach*. vdf Hochschulverlag, Zürich, Switzerland.

IBM (2003). IBM Legacy Transformation Services. Technical report, IBM.

Knuth, D. (1968). *Mathematical System Theory 2*, chapter Semantics of Context-Free Languages, pages 127 – 145. D.E. Knuth.

M"ossenbeck, H. (1990). A Generator for Fast Compiler Front-Ends. Technical Report Report 127, Institut für Computersysteme, ETH Zürich.

Partsch, H. and Steinbruggen, R. (1983). Program Transformation Systems. *ACM Computing Surveys*, 15(3).

Richardson, R., Lawless, D., Bisbal, J., Wu, B., Grimnson, J., and Wade, V. (1997). A Survey of Research into Legacy System Migration. Technical Report TCD-CS-1997-01, Computer Science Department, Trinity College Dublin.

St"or, M., Birkeland, N., Nienhaus, J., and Menkhaus, G. (2003). IT Infrastructure for Supply Chain Management in Company Networks with Small and Medium-sized Enterprises. In *Proceedings of the 5th International Conference of Enterprise Information Systems*, volume 4, pages 280 – 287, Angers, France.

SUN Microsystems (2003). Java Architecture for XML Binding (JAXB).

The Apache DB Project (2003). Object/Relational Bridge (OJB).

Winter, V. L. (1994). *Proving the Correctness of Program Transformations*. PhD thesis, University of New Mexico.

Winter, V. L. (1999). Program Transformations in HATS. In *Proceedings of the Software Transformation Systems Workshop*, California, USA.

Wu, B., Lawless, D., Bisbal, J., Grimson, J., Wade, V., O'Sullivan, D., and Richardson, R. (1997). Legacy System Migration: A Legacy Data Migration Engine. In Experts, C. C., editor, *17th International Database Conference*, pages 129 – 138, Brno, Czech Republic.

A XML2OJB Mapping

```
<XML2OJB>
      <ClassDefinition>
              <Class Name="addressDB.Adress">
                      <SetMethod Alias="lastName">setLastName</SetMethod>
                      <SetMethod Alias="addressPlace">setAdressPlace</SetMethod>
              </Class>
              <Class Name="addressDB.Place">
                      <SetMethod Alias="place">setPlace</SetMethod>
                      <SetMethod Alias="postalCode">setPostalCode</SetMethod>
              </Class>
      </ClassDefinition>

      <DuplicateRecord>
           <AutoKey>email</AutoKey>
      </DuplicateRecord>

      <Assemblies>
              <Repeat Name="/organisation/person">
                      <Insert In="addressPlace" ObjectToAdd="addressDB.Place"/>
                      <CreateObject>addressDB.Adress</CreateObject>
                      <CreateObject>addressDB.place</CreateObject>
              </Repeat>
      </Assemblies>

      <ImportSequence>
              <Class>addressDB.Adress</Class>
              <Class>addressDB.Place</Class>
      </ImportSequence>

      <SourceDocument>
             <element xpath="/organisation/person/LastName"/>
             <element xpath="/organisation/person/Place"/>
      </SourceDocument>
</XML2OJB>
```

Figure 6: Configuration for mapping a XML document onto an OJB object structure.