PALADIN: A Pattern-based Approach to Large-scale Dynamic Information Integration

Jürgen Göres

Heterogeneous Information Systems Group

University of Kaiserslautern

goeres@informatik.uni-kl.de

Abstract: The nascent data grid technology aims to make access to globally distributed structured and semi-structured databases possible. However, only integrated, transparent access to such data sources, abstracting not only from distribution and implementation details, but also from the diverse forms of heterogeneity, allows maximum leverage for the envisioned dynamic forms of inter-organisational cooperation. Traditional integration technologies rely on a human-driven development process and are therefore not applicable to the ad-hoc style of integration required in these scenarios. PALADIN aims to automate integration planning by using patterns that capture expert knowledge in a machine-processable way. A pattern describes a generic problem constellation encountered in information integration and provides a guideline to its solution, described as transformations on a graph-oriented representation of the data source schemas and their data.

1 Introduction

The IT infrastructure found in most organizations today often grew organically over decades. It is therefore characterised by several forms of heterogeneity, which can rougly be classified into three categories: *Technical heterogeneity* subsumes different hardware, operating systems, programming languages and APIs. *Logical heterogeneity* includes different data models (data model heterogeneity), the diverging use of identical datamodels (schematic heterogeneity) and different structuring (structural heterogeneity). *Semantic heterogeneity* originates from the often subtle discrepancies in the use and understanding of terms and concepts in schemas of different origin.

Existing information integration (II) technology is used successfully to consolidate such systems by providing a unified view that abstracts from these forms of heterogeneity. However, setting up an integration system is to date to a large part still a manual task that can be split into distinct phases: Initially, requirements for the desired integrated system, like the target data model and schema, are determined (*analysis phase*). Based on these results, suitable data sources that can contribute have to be found (*discovery phase*). In the *planning phase* the forms of heterogeneity are identified and resolved by defining a mapping between the source and the target schemas. This mapping or *integration plan* is then implemented by deploying it to a suitable runtime environment like one of the existing II products (*deployment phase*). The resulting system is then available for end users (*runtime phase*). This classical approach to II relies on human expertise in two areas: Integration experts have profound knowledge about how to remedy logical heterogeneity by defining mappings between the source and target schemas. To successfully define such mappings, an understanding of the domain of discourse is needed. Application domain experts are required to resolve the non-trivial semantic gaps between the different schemas, i.e., they deliver insight into the domain concepts and provide information about correspondencies between the different schemas. This elaborate process, while time-consuming and costly, is still adequate for scenarios where the requirements and the data sources deemed for integration are stable. However, it is difficult to apply to dynamic environments like the *virtual organisations* (VO) proposed by [3], a new form of short- to medium-term inter-organisational cooperations, that use grid technology to pool computing resources and data on a global scale.

PALADIN (PAttern-based LArge-scale Dynamic INformation integration) aims to reduce the dependency on human experts. Building upon existing and evolving middleware technology like web and grid services [5] and standard interfaces like those specified by the *Data Access and Integration Working Group* (DAIS) [1] that solve most aspects of the aforementioned technical heterogeneity, we

focus on resolving logical and semantic heterogeneity. Machine-processable *integration patterns* are used to capture II expertise. They enable the collection of both generic problem constellations encountered when integrating heterogeneous data sources as well as guidelines to their solution. If the problem described by a pattern is later discovered in the schemas of a set of data sources chosen for integration, the generic solution provided by the pattern is adapted to the concrete situation. By combining a set of patterns from a pattern library that cover data model, schematic and structural heterogeneity, a proper sequence of schema (and accompanying data) transformations can be discovered that describes a mapping from the chosen data sources to the desired target schema.

Like the integration experts they support or substitute, patterns cannot rely on schema information (i.e. the explicit metadata) alone, but also have to consider the semantics of the data source. Therefore, a medium to convey the semantics in a machine-processable way is needed. In practice, schema matching techniques are applied to identify those elements in the different schemas that correspond in some way. Many approaches to automatic schema matching have been explored (see [6] for an overview). These methods are usually based on a lexical and structural analysis of the schemas. While the results are often promising, they still lack the quality to solely rely on them for automated integration. PALADIN uses *domain schemas* to augment basic schema matching with domain knowledge. Being essentially multi-language ontologies, they describe the terms of discourse of a given domain, the relationships between these terms and their synonyms and optionally translations into different languages. However, even with domain schemas as support, the current state-of-the-art cannot yield schema matches with sufficient quality. Assuming that the user of a data grid is familiar with the application domain, we explicitly include him in the schema matching process by providing an interface that allows to check, correct and amend the correspondencies identified by automated matching.

The remainder of this paper is structured as follows: Section 2 briefly describes the PALADIN metamodel that is capable of handling arbitrary data and metadata. Section 3 describes the machine-processable integration patterns in more detail. Domain schemas are discussed briefly in section 4. Section 5 concludes with a summary and an outlook on future work.

2 Metamodel

To seamlessly handle data and metadata from data sources with different data models, the PALADIN infrastructure has to provide a uniform internal representation for the diverse data-model- or even implementation-specific schema description languages like DTDs, XML Schema or SQL DDL in its many proprietary variants. An existing approach is OMG's *Common Warehouse Metamodel* [2]: It uses a stack of four meta-layers, numbered from M0 to M3, where every element on one layer is an instance of an element on the layer above, and itself forms a class for instances on the layer below: On the M3 layer, the *Meta Object Facility* (MOF) serves as meta-metamodel to describe arbitrary metamodels on the M2 layer. Here, CWM and its extensions already provide metamodels for many standard data models like SQL and XML. One the M1 or model layer, actual schemas are described by instantiating the elements of these metamodels. The data layer M0 finally represents the data stored in these schemas.

While at first glance the CWM appears to provide the ideal infrastructure for PALADIN, certain deficits restrict its use for our purposes: Besides a high degree of complexity, which makes implementation and use cumbersome, the current CWM specification is somewhat outdated. For example, the predefined metamodel for XML is limited to the expressiveness of DTDs, thus making an extension to include XML Schema necessary. This can easily be accomplished using the MOF, but still does not address the fundamental flaw of CWM: the lack of a proper M0 or data layer. CWM provides an instance *meta*model, which places the actual data on the M1 layer, i.e., on the same conceptual layer as their metadata. This not only violates the strict concept of distinct meta layers, but also results in an overboarding complexity of the instance metamodel, making it unsuitable to describe more than sample data. However PALADIN requires the M0 layer to describe an essential part of integration pat-

terns. Therefore, we provide the PALADIN Metamodel (PMM), which borrows CWM's general concept of meta layers and many of its established terms, but remedies its deficits, like providing a conceptionally sound data layer. Using the meta-metamodel, we define those metamodels that are the most relevant, namely XML, object-oriented and object-relational. Further metamodels can easily be added if the need arises. In addition to the metamodels that are used to describe actual source or target schemas, a Match metamodel allows the description of correspondencies within and between arbitrary schemas and across different metamodels which are identified during schema matching. Unlike the generic correspondencies supported by existing schema matchers, which usually only allow to describe the fact that there is some kind of relationship between the elements with a given confidence, we provide an extensible set of correspondencies of a finer granularity that carry more precise semantics, like part-component-relationships, sub- or supersets and many more.

3 Integration Patterns

Integration patterns are PALADIN's fundamental concept to describe human II expertise in a machine-understandable way. Each pattern describes a certain elementary or complex integration problem, i.e., a constellation of schema elements and the correspondencies between them, and supplies a guideline to solve this problem by applying a suitable transformation on both the schemas involved and on the data held in these schemas. Instead of using imperative algorithms to capture this kind of II experience, patterns use a declarative notation which allows the easy extension of the library of available patterns without programming effort. A language is required that offers sufficient expressiveness to describe all kinds of structural, schematic and data model transformations. Many existing languages come to mind: SQL views can transform relational schemas and data, recent extensions of the standard like SQL/XML even allow bridging from SQL to XML. However, SQL is unable to bridge the gap between data and metadata (i.e. solve schematic heterogeneity). Approaches like SchemaSQL [4] try to remedy this situation, but have not gained widespread support. XQuery offers sufficient expressiveness to turn data to metadata and vice versa, but is limited to the XML realm alone. Therefore, neither language is suitable to express the full range of patterns. Defining our own textual language to cope with this problem would not only add to the existing plethora of proprietary data modelling languages that usually more or less imitate and build on SQL, but would also make the extension of PALADIN with additional metamodels difficult, as it would likely require new language constructs. We therefore suggest an approach that understands any schema (and its data) described in the PALADIN metamodel as a typed, attributed multigraph and every operation on the schema and data as a graph transformation. Graph transformations are a well explored concept (see [7]). To represent integration patterns, we chose an approach that is based on the semi-graphical language defined for the PROGRES graph replacement system [8]. While the graphical elements provide an easily readable, yet semantically precise description of the most relevant aspects of a pattern, the textual notation adds expressiveness that would be hard to capture graphically. A graph transformation is described using production rules. A production rule has a left-hand side (LHS) that describes the situation it can be applied to as an abstract subgraph, and a right-hand side (RHS) that describes the result of its application.

Figure 1 shows a simplified sample pattern that describes the necessary steps when source and target schema feature a different degree of normalisation. It also introduces the most important elements of the graphical part of the notation. Every pattern consists of two production rules or *facets*, which are situated on the M1 and on the M0 layer, respectively. The M1 facet describes the effects on the schema: The left-hand side describes a situation, where two tables are connected via a foreign key between a subset of their columns.

To connect left- and right-hand side, every element that should be preserved by the pattern is bound to a variable. An element that is not repeated on the RHS indicates a deletion, an element that is new on the RHS indicates the creation of an element. In the example, the two tables bound to *t1* and

t2 are replaced by a new single table *t3* which receives all columns that were part of either *t1* or *t2*, except those that were part of the foreign key, which are equivalent and are therefore added to *t3* only once. All inbound context edges to a node in a pattern are preserved if the node itself is preserved, or can be rerouted to another node using the diamond symbol. The M0 facet describes the conversion of the data stored in the schema elements of the M1 facet. It essentially describes an equi-join between the two tables and uses a simple expression of the textual notation for the join condition.
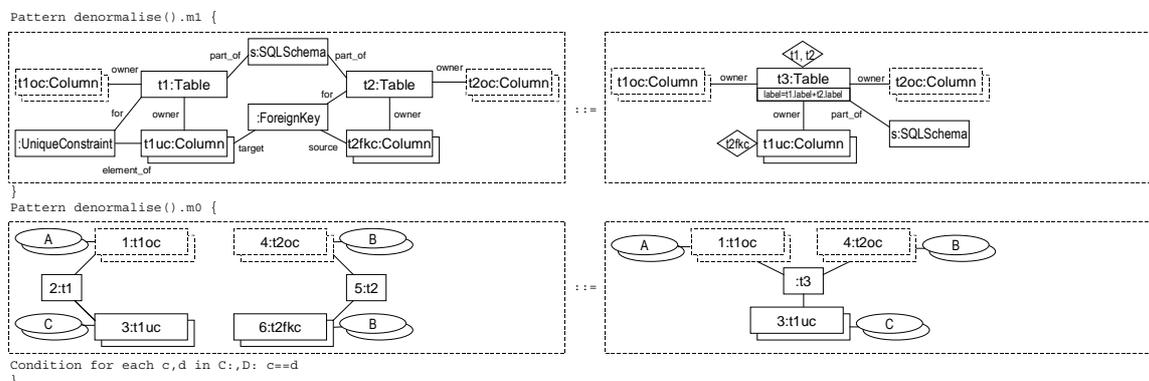

Fig. 1. An integration pattern expressed as a graph transformation

Given a sufficiently large library of patterns, the task of integration planning can now be understood as proving a hypothesis (the target schema) from a set of axoims (the source schemas) using a set of rules (i.e. the patterns). Any successful deduction, i.e., a sequence of rule or pattern applications that can be found, describes a logical query graph that transforms the source data into the desired target schema, with each pattern's M0 facet representing an operator in this graph. To use this integration plan, it has to be translated into a physical plan using the operators provided by the selected runtime environment (RE). While a RE could use the M0 facet directly to execute the query by working on the M0 graph representation of the data, many patterns will not need the additional expressiveness and only use operations that are already part of an algebra for the respective data model. In this cases, it is often more efficient to use a native implementation of an operator if it is provided by the runtime environment, using the graph-oriented operator description only as a fallback, e.g., to configure a generic graph-transformation operator. The same holds true for the specification of the patterns: while the use of graph transformations allows the expression of arbitrary operations on schemas and their data in a declarative way, existing query languages are often sufficient and we therefore support the translation of patterns in a data model specific language into our internal format.

4 Domain Schemas

While the pattern shown in figure 1 relies solely on schema information, many patterns require information about the semantics of a data source. This is especially relevant when a given schema constellation allows the application of several schematically equivalent patterns. Semantic information discovered or defined during schema matching is stored using the Match metamodel and can be used in the definition of the preconditions for patterns. To improve the degree of automation achievable by schema matching techniques, thereby reducing the amount of user interaction to rectify and complete these matches, we introduce the concept of domain schemas as our "Swiss army knife to semantic integration". A domain schema can in principle be modeled using an arbitrary data model and represents a base of reference for a given domain. We provide a simplified variant of the CWM object model, that allows each class to have multiple labels in different languages, which enables their use as a dictionary and acronym list for lexical schema matching techniques. The backbone of each domain schema is the directed acyclic graph (DAG) created by the (multiple-)inheritance relationships between its classes, providing a natural broader/narrower-term relationship which can be used both as a thesaurus and as a means to address subdomains with path expressions. Additional relationships and

special attributes can be used to add further information, like hints to potential references (e.g. foreign keys that were not explicitly defined in a schema) and strong or weak candidate keys. While it is possible to provide monolithic schema matchers that make use of domain schemas, they integrate smoothly with the concept of integation patterns: Figure 2 shows how graph transformations can be used to transfer information from a domain schema to a concrete schema. The *is_a* relationship between two classes *c1* and *c2* in the domain schema *ds* is annotated using an *is_a* match between two model elements *me1* and *me2* that carry the same label as *c1* and *c2*, respectively. Note that since a domain schema class can have multiple labels, the equals-operator used in the condition is interpreted as an existential quantifier.
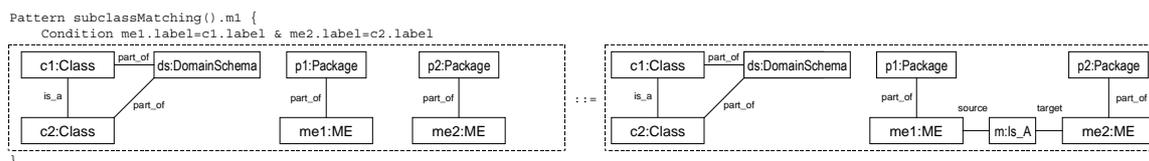

Fig. 2. Using patterns and domain schemas for schema matching

Since this kind of match is independent of the concrete data model of the matched schema elements, the pattern refers to the abstract *Package* and *ME* (ModelElement) classes of the core metamodel of PMM, which are superclasses for every schema or schema element of a concrete metamodel.

5 Conclusion and Future Work

We have introduced two fundamental concepts, integration patterns and domain schemas, to provide an automated approach to information integration with the two essential types of machine-processable knowledge and experience. We currently evaluate existing technologies like the PROGRES graph transformation engine introduced in [8] for their suitability and also analyse different implementation alternatives for creating a new system from scratch. While focussing on the planning phase of the II process, we can transfer both the concept of patterns and domain schemas to other phases. Domain schemas can be used in the discovery process to narrow down the potentially very large set of candidate data sources to the most promising ones. Patterns can also guide deployment by describing the mapping from logical integration plans to the physical plans, i.e., the deployment info for different runtime environments.

References

[1] Mario Antonioletti, Malcolm Atkinson, Susan Malaika, Simon Laws, Norman W. Paton, Dave Pearson, Greg Riccardi: *The Grid Data Service Specification*. September 19, 2003.

[2] Object Management Group: *Common Warehouse Metamodel (CWM) Specification Version 1.1*. March 2003.

[3] Ian Foster, Carl Kesselman, Steven Tuecke: *The Anatomy of the Grid - Enabling Scalable Virtual Organizations*. In Proceedings of the First IEEE International Symposium on Cluster Computing and the Grid, May 15-18, 2001.

[4] Laks V. S. Lakshmanan, Fereidoon Sadri, Iyer N. Subramanian: *SchemaSQL – A Language for Interoperability in Relational Multi-Database Systems*. In Proceedings of the 22nd VLDB Conference, pp. 239-250, 1996.

[5] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, D. Snelling, P. Vanderbilt: *Open Grid Services Infrastructure*. June 27, 2003.

[6] Erhard Rahm, Philip A. Bernstein: *A survey of approaches to automatic schema matching*. In The VLDB Journal 10, pp. 334-350, 2001.

[7] Andy Schürr: *Programmed Graph Replacement Systems*. 1997.

[8] A. Schürr, A. W. Winter, A. Zündorf: *The PROGRES Approach: Language and Environment*. In Handbook of Graph Grammars and Computing by Graph Transformation 1, pp. 487-550, 1997.