Change Management in Large-Scale Enterprise Information Systems

Boris Stumm

University of Kaiserslautern
stumm@informatik.uni-kl.de

Abstract. The information infrastructure in today's businesses consists of many interoperating autonomous systems. Changes to a single system can therefore have an unexpected impact on other, dependent systems. In our Caro approach we try to cope with this problem by observing each system participating in the infrastructure and analyzing the impact of any change that occurs. The analysis process is driven by declaratively defined rules and works with a generic and extensible graph model to represent the relevant metadata that is subject to changes. This makes Caro applicable to heterogeneous scenarios and customizable to special needs.

1 Introduction

In today's businesses, information infrastructures are getting more and more complex. There are many heterogeneous systems with a manifold of mutual dependencies leading to unmanageability of the overall infrastructure. New dependencies between existing systems evolve and new systems are added. Generally, there is no central management of all systems.

Small, local changes can have a major impact at company-wide scale due to the dependencies between systems. To keep everything running, it is therefore necessary to *preventively* analyze the impact of a change, to be able to make adjustments in case of conflicts without compromising the infrastructure. While the heterogeneity of systems and the problem of incomplete metadata make change impact analysis already a hard task, the situation becomes even more difficult as changes are not always planned globally and in advance. Thus, unexpected problems may occur after a change is carried out, making a *reactive* change impact analysis necessary. We present Caro, an approach for change impact analysis (CIA) that is able to operate even under these adverse conditions.

When speaking of changes, we refer to metadata changes. In our context, metadata includes not only data schemas, but also APIs, configuration files, assertions about data quality and performance, etc., in short, everything that other systems could rely on.

Problem Statement. The problems that we face in change management and which we address with our approach can be divided into three categories:

– Heterogeneity. The connected systems often have different data models (e.g. XML or SQL), different interfaces (e.g. query or function calls), etc.

- Incomplete metadata. In general, it is not possible or feasible to get all metadata for an exact CIA. There may be no easy way to query the metadata of a system, documentation is often outdated or non-existent, and dependencies between systems can be hidden in procedural code, which in the worst case would have to be decompiled to get the required information. While it is theoretically possible to get exact metadata, in practice, the costs may be too high.
- System autonomy and missing global management. In practice, many systems are black boxes that cannot be controlled from outside. This especially holds true if an integration environment spans over several departments or even several companies, and complicates access to such systems. Changes are applied without global analysis, and without notification to the affected systems. Thus, problems emerge unexpectedly, and it is hard to find the cause.

Contribution. Caro is a concept which includes three main components responsible for addressing the discussed problems:

- We propose an *architecture* which allows a central or distributed approach to change impact analysis. We have software components called *metadata agents*, which, amongst other things, monitor the systems participating in the integration infrastructure for changes. The *change manager* allows for preventive CIA as well as reactive CIA.
- We present a *metamodel* which allows us to handle and homogenize the heterogeneous metadata encountered. It is designed to be extensible to describe arbitrary metadata at arbitrary granularities.
- We use a robust and generic *analysis algorithm* which can handle incomplete metadata. It works on a best-effort basis based on the input metadata, and the quality of the analysis results will gracefully degradate as input metadata gets less complete or more coarse-grained.

With these concepts, Caro is applicable to a wide range of different systems, and thus a wide range of different changes can be detected and analyzed.

Related Work. In the context of information integration, much research has been done. Some approaches are complementary to ours, and others are similar to Caro in some aspects. The most important distinguishing facts of Caro are its genericity, robustness and scope. It makes no assumptions about the environment it operates in, and can be used for any scenario where change impact analysis is necessary.

Dorda et al. [8] present an approach which is quite similar to Caro with respect to the problems addressed. However, the solution they propose is different in two fundamental points: They require a central documentation (or metadata) repository and a strict process policy. This constrains their approach to scenarios where it is feasible to have a central repository and to enforce adherence to defined processes. While they want to avoid integration clusters[1], we think that such a clustering (and thus decentralization) in large EIS cannot be avoided.

[1] Integration clusters are called “integration islands” in [8].

Deruelle et al. [7] present another approach to change impact analysis. They use a multigraph and change propagation rules for analysis, which is very similar to Caro. Their approach has several limitations. The focus lies on preventive change impact analysis, thus they lack a framework to support reactive CIA. Apparently, they do not consider the problem of incomplete metadata. Also, their meta-model and rules are rather specialized, which makes the extension to support other data models and change types more difficult than with Caro.

Various other approaches to CIA in information systems exist that are limited with respect to the supported data models [10] or scope and support of exact analysis [12]. The concepts of change impact analysis in software systems [6,3,16] are similar to the ones we use. However, the models and analysis procedures focus on the elements that are found in software: methods, signatures, classes, attributes and so on. In addition, CIA for software systems is usually done preventively. Aspects of heterogeneity, metadata incompleteness and distribution are not that relevant as they are in information systems.

Research done in the field of schema evolution [15,4,17], schema matching [14,13,11] or model management [5] are complementary to our approach. Especially the latter approaches are used to plan and realize integration, generally between only two or a small group of systems, as well as adapt systems to changing requirements. Caro is not designed for use in the initial stages of an integration project. It will take the results of such a project, namely the dependencies between the systems that were created based on schema matches or mapping definitions, and monitor them for changes. When a change occurs, Caro will analyze the impact of it and notify the responsible person. If problems are encountered, the output of Caro can be the input for the information integration tools that are used to repair the impacted systems. Caro focuses on the monitoring of systems participating in the overall information infrastructure and the detection of the global impact of changes. As such, it "fills the gap" to an overall management of a heterogeneous integrated environment.

Structure of the Paper. In the following sections, we will first give an overview over the architecture of our CIA approach (Sect. 2). We discuss the conceptual meta-model on which our approach is based on in Sect. 3. In Sect. 4 our approach to conduct the analysis is presented. In Sect. 5 we will discuss some of the issues that arise during the preceeding sections. Finally we finish with conlusions and outlook in Sect. 6.

2 Overview

Central architectural components of Caro are the metadata repository (MDR) and the change manager (CM) (see Fig. 1). The MDR is a passive component that holds the metadata of the different information systems in a common representation. It provides an interface to query and update the stored metadata. All metadata is versioned, to be able to keep track of any changes that happened in the past. The CM is a reactive component responsible for the analysis of changes. It can analyze change proposals issued via the user interface, or react to changes that have happened in an observed system. The third component in our architecture is constituted by metadata agents (MDA). Every system participating in CIA is monitored by an MDA responsible for mediating

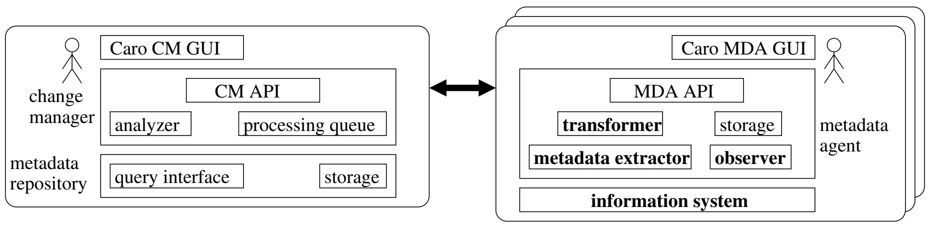

Fig. 1. Caro architecture

between the CM, the observed information system and the human responsible for it. An MDA consists of various subcomponents. The metadata extractor is needed to initially extract all metadata from the underlying system and to later pick up changes. A transformer component maps the extracted metadata to the Caro format. The observer component serves as a guard and watches for changes in the information system. For caching purposes there is a storage component. The MDA communicates with the CM via asynchronously to not block either component. The MDA parts written in bold face are those that need to be customized for each information system. Caro provides generic functionality, and specific functionality can be added via a plugin mechanism. To configure the components, GUIs for the CM and the MDAs will be provided. Furthermore, the GUIs give a global (CM GUI) or local (MDA GUI) view of metadata and dependencies and are used as interface for preventive CIA. The MDA GUI is constrained to a local analysis, which is also useful (e.g., to analyze how views are affected if a base table changes). In the following sections, the main focus lies on the metadata model and the analysis approach. The issues that arise in the functionality of the MDAs, such as detecting changes in system metadata, conversion from a source's native metadata representation or modeling dependencies, are discussed in Sect. 5.

Caro can also be used in a distributed way. Several change managers, each responsible for a part of the overall system, can communicate with each other and pass on their analysis results. This enables the use of Caro in cases where a centralized solution is not feasible. An example scenario for this is shown in Fig. 2. This way it is possible to restrict the data passed on to the other servers, which can be important for security reasons.

Our base assumption is that every single system in an integrated information infrastructure provides various kinds of services to other systems. We refer to this set of services as the system's *provision*. For each accessing client system, there may exist a different provision set, depending on the authorizations of it. Complementary to this, the part of the provision that is used by the client system is called *usage*. A client system as a usage for each system it depends on. Note that the usage of the client needs not to be identical to the provision of the server system. In general, the usage will be a subset of the provision, or may even contain elements not present in the provision. If that happens, there exists a problem which will be recognized by our approach.

We do not use the more common terms import or export schema, since provisions and usages can contain more than only schema data, and may, for example, include configuration data, technical metadata, quality assertions ("The data provided is less than

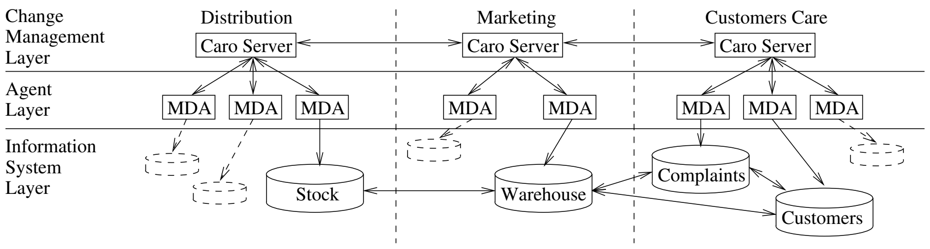

Fig. 2. Distributed Caro architecture

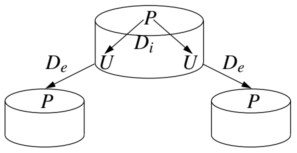

Fig. 3. Provision and usage specifications, internal and external dependencies

a day old.") or activity information ("The ETL process runs every Saturday at 0:00."). Figure 3 illustrates provisions P, usages U and the dependencies D between them. *External dependencies* (D_e) exist between a provision and a usage of different systems. The usage depends on the corresponding provision to be made. *Internal dependencies* (D_i) exist between the provision and usages *within* a system. Services provided (in the provision) may be dependent on the use (in the usage) of other system's services. A simple example is a federated DBMS, whose provision is basically a view on the provisions of the base systems. In this case, the internal dependencies are represented by the view definitions in the federated DBMS.

Change impact analysis is an integral part of a larger *system evolution process*, which is happening in every information infrastructure. System evolution includes all changes that occur to systems that are part of the infrastructure. In ideal scenarios, before any change is applied, its impact will be analyzed. We call this *preventive CIA*. Depending on the analysis result, some adjustments may be made to minimize the impact or to adapt the impacted systems. Caro supports this process by providing tools and interfaces to do preventive CIA before changes are made. In practice, such ideal scenarios do not exist, mostly due to the autonomy of systems involved. The larger the number of integrated systems, the more probable it is that changes are made without prior analysis or coordination. Caro monitors every system and detects changes shortly after they occur. *Reactive CIA* is then initiated automatically, and administrators of impacted systems are notified. The analysis process itself is identical for both cases. The difference lies only in the type of input data (proposed changes vs. already applied changes) and in the actions taken after analysis. With preventive CIA, results will have no effect on running systems, whereas with reactive CIA, affected systems may be disabled, or other measures may be taken, to prevent data corruption or incorrect query results.

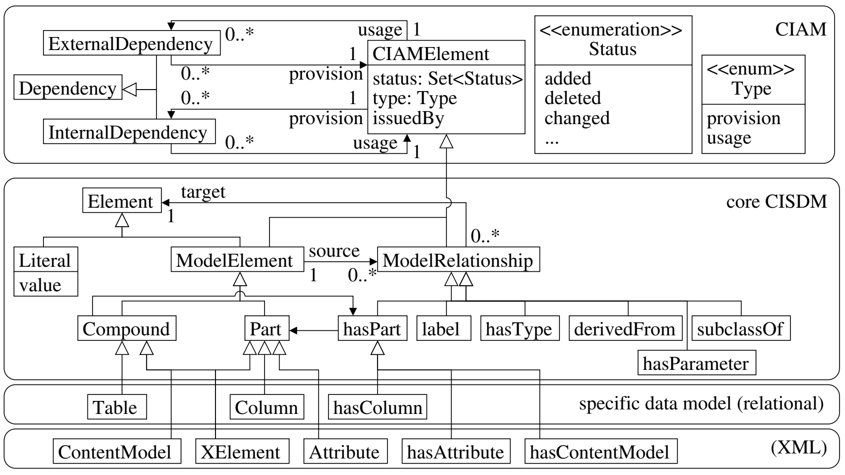

Fig. 4. Caro meta-models

3 Conceptual Model

An important consideration was the choice of the meta-model to use in our approach. It must be possible to represent arbitrary metadata and dependencies, without assuming any data model (like SQL or XML) or types of dependencies. There has to be support for a declarative specification of change impact, and the possibility to describe metadata at different granularities. These requirements are met by our conceptual model.

The base assumption we build our model on is the following: A metadata description consists of *elements* and the *relationships* between them. Elements are *atomic information units*. In the relational world, a table definition consists of many elements, namely the element representing the table itself, the name of the table, elements for every column, column name and column type, and so on. A metadata description can then be expressed as a bipartite digraph with node types E and R representing elements and relationships, similar to the ER-model. Relationship nodes represent a binary relation between element nodes and thus always have one incoming and one outgoing edge. Expressing relationships as nodes and not as edges has its reason in that there can be dependencies between relationships.

The elements in the metadata graph are instances of elements defined in the Caro meta-model. This meta-model has two parts, the *change-impact system description model* (CISDM) and the *change-impact analysis meta-model* (CIAM). Both of them are depicted in Fig. 4. We aim to provide more complex class-building constructs, like it is possible in OWL [1], but for readability we used an UML-like syntax in the figure.

The CISDM defines classes that capture the semantics that are relevant to CIA. The figure shows a selection of these. The top level classes are Element, which all element nodes are instances of, and ModelRelationship for the relationship nodes. ModelRelationships connect two Elements, as we have stated before. Literals have no outgoing edges, since they only function as containers for values of other Nodes. For each CIA-relevant "role" that a node may have, the top level classes are subclassed. In the figure, two roles for element nodes (Compound and Part), and several roles for relationship nodes are shown. The CISDM itself is not intended to model metadata graphs directly. It is an abstract meta-model from which concrete meta-models can inherit, assigning CIA semantics to their elements. In the lower part of the figure, this is shown for some elements of the relational and XML data model. The change impact analysis is done only with the information that the generic part of the CISDM provides, whereas the metadata is described in terms of the corresponding data model. Change impact properties are assigned to meta-model elements by inheritance, which makes it easy to adapt existing meta-models for use in Caro by simply adding the CISDM classes as superclasses to the model.

While the CISDM is used to model the change impact properties of metadata descriptions, the CIAM provides means to connect different graphs via dependencies and enables setting the status of nodes (e.g., to added or deleted). The upper part of Fig. 4 shows the CIAM. Main classes are Dependency and CIAMElement. Each dependency connects two CIAMElements, which are either ModelElements or ModelRelationships. The connected elements have one of two roles: provision or usage. CIAMElement has two other properties. The status property holds the current analysis result for this element. For simplicity, only the three status values added, deleted and changed are shown in the figure. The issuedBy property denotes the observed system which the graph belongs to. With this model, not only dependencies between elements, but also between relationships can be expressed.

There is no requirement for metadata graphs to be complete, or every dependency to be modeled. If there is a dependency between a table in a source system and a federated DBMS, the individual column elements need not be connected via dependencies. The most coarse-grained metadata graph would consist only in one element node per system, and dependency nodes showing how systems are related to each other. This does not allow a very precise analysis, but in this way no system will be "forgotten" if a change occurs somewhere. Since fine-grained metadata can be very expensive to get, it can be decided on a case-by-case basis if an exact analysis is required or if more false alarms are acceptable. There are no constraints on the types of metadata changes that can be captured. If the corresponding elements and their dependencies are modeled, changes will be detected. Although in our prototype we focus on schema changes, Caro is not limited to that. Some examples that come to mind are function signatures, classes, methods, directory layouts, application configuration files, installed software. Even more dynamic metadata such as network capacity, free disk space, or CPU performance can be modeled and analyzed. Of course, the CISDM will probably have to be extended, and some more analysis rules may be required. We will discuss this in Sect. 5.

4 Analysis

The analysis of a change is done by applying impact rules to the metadata graph until no more rules can fire. Conceptually the rules and the graphs they operate on have the following characteristics:

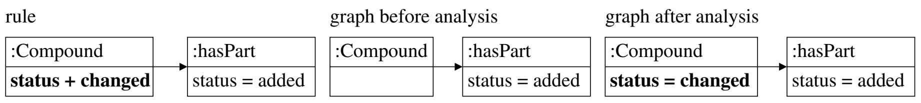

Fig. 5. Example rule

- Each rule has a *premise*, which is a graph pattern specifying nodes and their properties. If the premise matches a subgraph, the *conclusion* of the rule is applied. The conclusion is always a list of property values that will be added to a specific node.
- Each node has a finite set of properties that are identified by property names.
- If a part of a conclusion already exists in the graph, only the missing part is added.

These characteristics make the appliance of rules monotonic. Besides that, order of rule appliance does not matter. This ensures that analysis will always produce the same output if given the same input and that the calculation will always terminate.

Figure 5 shows a simple rule in a graphical notation on the left. Text written in normal font constitutes the premise. The conclusion is written in boldface. The analysis rule shown in the figure adds the changed-status to a compound if a part was added. Although the rule is quite simple, we argue that in the majority of cases, such simple rules suffice, making the analysis procedure similar transitive closure algorithm. In some cases, more complex rules which contain more nodes and edges may be needed, therefore the reasoner used must not rely on having only simple rules. Our current implementation uses RDF [2] and the Jena framework [9] with its generic rule reasoner for analysis. We mapped our conceptual model to RDF triples. The implementation details cannot be discussed here due to space restrictions. Rules always specify the most general class to which they apply, but also match subclasses. For the example analysis rule this means that a hasColumn relationship, which is a specialization of hasPart, between a table and its columns will also be matched. If a meta-model needs to be analyzed in a way not covered by the standard ruleset, special rules can easily be added by using the corresponding subclasses in the rule definitions.

5 Deploying Caro

In the previous two sections we introduced the conceptual meta-model and the analysis rules that work on it. We showed that we can handle arbitrary metadata models, and even cope with incomplete data. For this to work, we make several basic assumptions: for all metadata there is a specific meta-model extending the CISDM, and all metadata to analyze will be transformed automatically into our common format. Further, the metadata agents detect all changes and notify the change manager component. In this section we will discuss the manual effort that is needed to fulfill these assumptions.

Extending the CISDM. All metadata needs to be described by a meta-model which is an extension of the CISDM. Although we aim to provide meta-models for SQL and XML directly, there will in general be the need to define custom meta-models. We believe that for most cases the effort will be rather small, and depending on the resources available, one can decide to have a not-so-detailed model at the cost of a more coarse-grained analysis. Tightly coupled with the extension of the CISDM is the addition of new analysis rules. As we mentioned, rules will generally have a very simple structure, so this is also an unproblematic task.

Transforming the Metadata. Since Caro needs all metadata in form of a graph, the system metadata will need to be transformed to the graph format. This basically amounts to writing custom transformer components for the corresponding MDA. This is not a scientific effort, only a technical one, since there is already a meta-model for the system. While this is a manual task, it can be accomplished in a straightforward way.

Monitoring and Extracting the Metadata. Perhaps the biggest problems that Caro and all similar approaches are facing is how to monitor systems for changes, and how to extract the metadata in an automated way. All relational DBMS have an information schema[2], which makes it very easy create a custom MDA-component to extract the schema and other metadata. Listening for changes gets more difficult, since triggers on system tables are usually not allowed. A solution here can be a periodic poll and use of a "diff" tool to find out what changed, or inspecting logfiles. In this and similar cases, monitoring and metadata extraction poses no problems. But there are other scenarios, e.g., systems only allowing function calls with no simple query mechanism to inspect metadata, or where access rights prevent the MDA from inspection. There is no general solution for these scenarios. An implementation of a custom MDA-component might, e.g., analyze the source code, do probing or check the timestamps of files. Even if the information that is gathered this way is incomplete, Caro is still able to do analysis on a more coarse-grained level.

While the manual effort to make the assumptions work may seem high, it is far less than the manual effort needed when integrating information systems. In information integration, specific data schemas (models) have to be integrated, matched and mapped to each other. In Caro, we work with meta-models. Most of the work has to be done only once for all instances of a proprietary system type, or could be provided by third parties.

6 Conclusions and Outlook

We presented a generic approach to change impact analysis which uses inference rules for processing. The approach can be applied to a wide range of scenarios. There are no constraints on which systems can be monitored and analyzed for changes. If a system with a proprietary metadata format is to be analyzed, only some custom MDA-components need to be developed. Since Caro can also function with incomplete and coarse-grained metadata, the initial development time and cost of these components can be kept low, at the cost of a less precise analysis leading to more false alarms.

An important question that arises is how Caro handles metadata other than SQL and XML schemas. It is neither possible nor desirable to include elements for all possible

[2] Not all DBMS may have a information schema conforming to the newer SQL standards, but all have a proprietary variant of it.

metadata descriptions in the CISDM or CIAM. Instead, these meta-models themselves can be extended by adding more possible values to the status property or subclassing ModelRelationship and ModelElement. In addition to the model extensions, new analysis rules need to be defined, too. This imposes no problem, since the rules are generally very simple. The main goal of our work is to analyze the impact of changes in an integrated environment of heterogeneous information systems. It would be interesting to know to which extent our approach could be used in other areas where the analysis of change impact is important, like CIA in software development.

One of the next steps is to extend Caro to not only be able to automatically analyze changes but also to handle problems that are detected, and help the developers by correlating the "old" and the "new" elements (i.e., to better recognize renaming or moving of elements). Furthermore, it is necessary to provide possibilities to give behavioral advice to systems affected by a change, to enable automatic reaction to problems. By using additional properties for element and relationship nodes, this can happen without interference with the current system. While the existing system was developed with this in mind, the details are subject to future work.

References

1. OWL Web Ontology Language Guide, 2004. http://www.w3.org/TR/2004/REC-owl-guide-20040210/.
2. RDF/XML Syntax Specification (Revised), 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.
3. S. Ajila. Software Maintenance: An Approach to Impact Analysis of Objects Change. *Software – Practice and Experience*, 25(10):1155–1181, October 1995.
4. P. Andritsos, A. Fuxman, A. Kementsietsidis, R. J. Miller, and Y. Velegrakis. Kanata: Adaptation and Evolution in Data Sharing Systems. *SIGMOD Record*, 33(4):32–37, December 2004.
5. P. A. Bernstein. Applying Model Management to Classical Meta Data Problems. In *Proc. of the 1st Conference on Innovative Data Systems Research (CIDR)*, 2003.
6. S. A. Bohner and R. S. Arnold, editors. *Software Change Impact Analysis*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.
7. L. Deruelle, M. Bouneffa, G. Goncalves, and J.-C. Nicolas. Local and Federated Database Schemas Evolution: An Impact Propagation Model. In *Proc. of the 10th International Conference on Database and Expert Systems Applications (DEXA)*, pages 902–911, 1999.
8. C. Dorda, H.-P. Steiert, and J. Sellentin. Modellbasierter Ansatz zur Anwendungsintegration. *it – Information Technology*, 46(4):200–210, 2004.
9. Hewlett-Packard. Jena – A Semantic Web Framework for Java, 2005. http://jena.sourceforge.net/.
10. A. Keller and C. Ensel. An Approach for Managing Service Dependencies with XML and the Resource Description Framework. Technical report, IBM, 2002.
11. P. McBrien and A. Poulovassilis. Automatic migration and wrapping of database applications – a schema transformation approach. In *Int. Conf. on Conceptual Modeling/the Entity Relationship Approach*, 1999.
12. R. McCann, B. AlShebli, Q. Le, H. Nguyen, L. Vu, and A. Doan. Mapping Maintenance for Data Integration Systems. In *Proceedings of the 31st VLDB Conference*, 2005.
13. S. Melnik, E. Rahm, and P. A. Bernstein. Developing Metadata-Intensive Applications with Rondo. *Journal of Web Semantics*, 1(1), 2004.

14. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10:334–350, 2001.
15. J. F. Roddick. Schema Evolution in Database Systems – An Annotated Bibliography. *SIGMOD Record*, 21(4):35–40, 1992.
16. B. G. Ryder and F. Tip. Change Impact Analysis for Object-Oriented Programs. In *Proceedings of PASTE*, 2001.
17. X. Zhang and E. A. Rundensteiner. Data Warehouse Maintenance Under Concurrent Schema and Data Updates. Technical report, Worcester Polytechnic Institute, 1998.