FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Semantic Data Integration in (Software+) Engineering Projects

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering/Internet Computing

eingereicht von

Michael Handler
Matrikelnummer 0515280

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Ao. Univ.-Prof. Dipl.Ing. Dr. Stefan Biffl
Mitwirkung: Univ.-Ass. Dr. Thomas Moser

Wien, 17.11.2010

(Unterschrift Verfasser) (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Handler Michael
Wiener Straße 12/27
2700 Wiener Neustadt

"Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit — einschließlich Tabellen, Karten und Abbildungen —, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Ort	Datum	Unterschrift

Abstract

Experts act in different roles in a project and use different tools to fulfill their tasks and contribute to the overall project progress. The major problem that arises, when complex systems have to be engineered, is weak interoperability of tools in one engineering domain and especially between tools of different engineering domains. Incompatible syntactical representations of the same semantic concepts make efficient integration of engineering tools especially difficult. In addition high quality standards and rapidly changing requirements for large engineering projects create the need for advanced project management and quality assurance applications, like end-to-end tests across tool and domain boundaries.

The artificial intelligence and semantic web research community has been busy to develop languages and mechanisms to describe ontological knowledge about real life objects. Based upon these concepts ideas to improve the usability of all kinds of software are developed. Currently researchers try to facilitate semantic knowledge in other research areas, like enterprise application integration to ease the process of engineering. Unfortunately there is currently no integration platform for large software intensive engineering projects addressing both technical and semantic integration issues. To solve the integration problem often costly and hard to maintain point to point integration between the tools is done. But these systems are not capable of fulfilling the requirements of a modern engineering environment, like robustness, flexibility and usability.

The Engineering Knowledge Base (EKB) is a framework for engineering environment integration. In this thesis a semantic integration system based upon the EKB is developed as part of a technical integration system. The EKB concept is adapted for the integration into a technical integration system, which provides a common infrastructure for the EKB and other components. To make it possible for advanced applications to use data stored in different engineering tools, the EKB has to provide a virtual common data model, which contains schematic and semantic information about common engineering concepts and provides the infrastructure for semi-automatic concept to concept transformations.

Based upon the results of this research a prototype of the EKB is developed and integrated into the Open Engineering Service Bus (OpenEngSB). The prototype is evaluated against the current technical-only integration provided by the OpenEngSB with the help of two use cases: (1) *Definition Of Quality Criteria Across Tool Data Models in Electrical Engineering* and (2) *Change Impact Analysis for Requirement Changes*. The empirical evaluation shows that the EKB is an effective and efficient way to provide the necessary infrastructure for advanced applications like end-to-end tests across tool boundaries. Nevertheless additional configuration and maintenance effort is needed to manage the semantic knowledge. Yet if applications like those described in the real world use cases have to be performed, the additional features of the proposed semantic integration framework outbalance these disadvantages.

Keywords: Technical Integration, Semantic Integration, Software-Intensive Systems, Distributed Systems, OpenEngSB, EKB

Kurzfassung

Bei der Entwicklung von komplexen, softwareintensiven Systemen arbeiten Experten aus verschiedenen Gebieten in unterschiedlichen Rollen zusammen. Sie verwenden dabei Entwicklungswerkzeuge, die zwar optimal für das jeweilige Einsatzgebiet geeignet sind, aber nicht problemlos miteinander verbunden werden können. Besonders die verschieden Repräsentationen von denselben Konzepten in verschiedenen Entwicklungswerkzeugen führen zu Problemen bei der Toolintegration. Trotz dieser Probleme werden aber hohe Anforderungen an moderne Systeme gestellt, vor allem hinsichtlich Qualität und Sicherheit. Diese Anforderungen können nur durch toolübergreifende Qualitätstests erreicht werden.

Forscher auf dem Gebiet der künstlichen Intelligenz und des Semantic Web haben in den letzen Jahren viele verschiedene Ansätze und Sprachen zur Modellierung von semantischem Wissen über reale Objekte entwickelt. Basierend auf diesen wird versucht Softwareprodukte in anderen Gebieten, wie beispielsweise der Unternehmensanwendungsintegration, zu verbessern. Trotz dieser Bemühungen gibt es derzeit keine offene Plattform für die semantische Integration von Entwicklungsumgebungen für softwareintensive Systeme. Als Ersatz werden oft schwer zu erhaltende Integrationslösungen zwischen spezifischen Entwicklungstools manuell erstellt. Solche Punkt zu Punkt Integrationen sind aber nicht dazu geeignet, den Anforderungen eines modernen Entwicklungsumfeldes gerecht zu werden.

Das Konzept der Engineering Knowledge Base (EKB) versucht eine Lösung für die Integration von Entwicklungsumgebung zu liefern. In dieser Arbeit wird eine semantische Integrationslösung entwickelt und implementiert, die auf einem technischen Integrationssystem basiert und versucht das EKB Konzept zu verwirklichen. Die EKB muss dabei einerseits ein virtuelles gemeinsames Datenmodell zur Verfügung stellen, das schematische und semantische Information enthält und als Basis für die Entwicklung von komplexen Applikationen dient. Andererseits muss die EKB die Infrastruktur für die Transformation zwischen unterschiedlichen Konzepten und für die Integration der Daten aus verschiedenen Werkzeugen bereitstellen.

Basierend auf dem OpenEngSB Projekt wird ein Prototyp einer EKB basierenden semantischen Integrationslösung entwickelt und implementiert und anschließend mit dem aktuellen OpenEngSB System, das nur technische Integration ermöglicht, verglichen. Dazu werden zwei reale Anwendungsfälle untersucht: 1.) *Definition von Qualitätskriterien über verschiedene Tools hinweg im Elektroingenieurwesen* und 2.) *Change-Impact Analyse nach einer Anforderungsänderung*. Die empirische Untersuchung dieser Anwendungsfälle zeigt, dass die entwickelte Lösung für komplexe Anwendungen über Toolgrenzen hinweg verwendet werden kann. Der Hauptnachteil dabei ist, dass zusätzlicher Konfigurations- und Entwicklungsaufwand bei der Integration von Tools in das System anfällt. Sind aber komplexe Anwendungen für den Erfolg des Projektes notwendig, dann lohnt sich der investierte Aufwand, da diese einfacher umgesetzt und gewartet werden können.

Schlagwörter: Technische Integration, Semantische Integration, Software-Intensive Systeme, Verteilte Systeme, OpenEngSB, EKB

Contents

1 Introduction

Modern automation systems are developed to supply our society with goods and services like steel or electrical energy. These systems have to fulfill high safety standards, because a malfunction could cause severe financial loss and could even endanger the operators of the system or other humans. These high standards have to be enforced in all phases of the system life-cycle including development. Yet during development experts from different domains, like software engineering, mechanical engineering, electrical engineering or process engineering have to cooperate and to integrate their work (Biffl et al., 2009). Because software engineering plays an important part in the development of these systems, but other engineering disciplines are also involved in the engineering process and place requirements upon the resulting software, it is called (software+) engineering (Pieber, 2010). The domain experts typically use domain specific tools and data formats, which are not designed for interoperability. These tools have been designed as large closed systems in the past, which had either no need for interaction with other tools, or didn't have the possibility for interaction with other systems, because no suitable infrastructure was available when they were created. Additionally the development of these systems often takes place in physically separated groups, distributed all over the world. This results in weak integration of the different engineering tools. Many integration tasks have to be done manually causing defects and a higher risk of cost and time overrun. System integration is becoming increasingly complex as the overall system as well as its parts become more complex and powerful. If this process is done manually the quality of the resulting systems depends heavily on the abilities and the performance of the system integrators.

To solve the integration problem often costly and hard to maintain point to point integration between the tools is done. This makes tool exchange very difficult, because in addition to the typical migration costs, like data migration and user education also the integration with all other tools has to be updated for the new tool. Therefore more advanced forms of technical integration are necessary. Different integration architectures for technical integration have been proposed to reduce the complexity for tool integration and exchange. These advanced integration methodologies have in common that they enforce a standardized way of tool integration and communication between tools (Kaushal & Saravanan, 2004). Technical integration provides a logical connection between the different tools, a messaging infrastructure and a common message format. It thereby makes communication between the different tools possible.

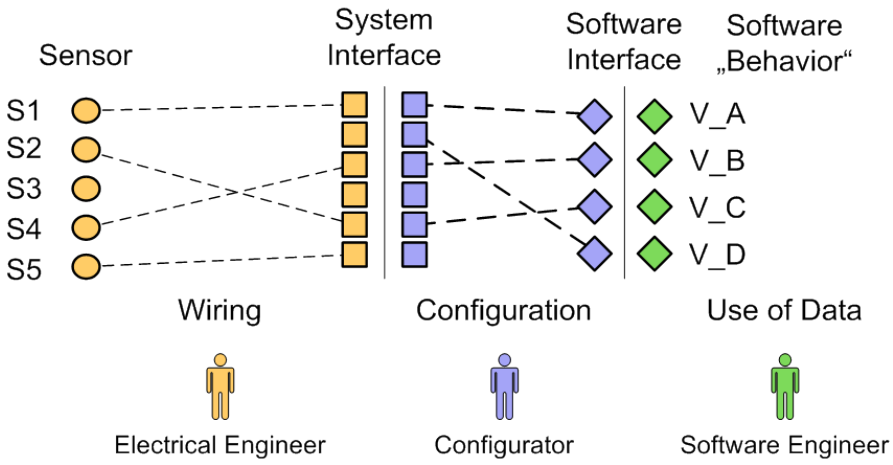

Figure 1.1: Sensor to variable mapping according to Biffl (2010).

The Open Engineering Service Bus (OpenEngSB) framework (OpenEngSB-Project, 2010) is a possible solution for the technical integration in the (software+) engineering domain. It extends the Enterprise Service Bus architecture by core components that provide basic functionality that is needed to cope with the heterogeneity of the tools and to enable process automation and advanced quality management across multiple engineering disciplines (Pieber, 2010).

Each tool used for (software+) engineering has its own scope and defines its own concepts. Figure 1.1 shows how these concepts can be related in the respective domains. The example is from the electrical engineering domain and taken from a real world use case. The sensors are mapped via intermediate steps to software variables. This mapping has to fulfill constraints. Without working integration of the tools involved in this example, these constraints have to be verified manually. To automate these tasks besides technical integration also semantic integration of the involved concepts is necessary.

Semantic integration defines mappings between logically equivalent or related concepts in a machine understandable way (Gruninger & Kopena, 2005). In the (software+) engineering domain this typically means mapping the common concepts used in different engineering disciplines and their tools. Figure 1.2 illustrates the identification of these common concepts.

Besides system integration the establishment of an engineering framework that provides technical and semantic integration is also useful for quality assurance and project management. Semantic integration provides a virtual common data model, which can be queried for information from different data sources, different tools and different engineering disciplines. Besides the data, semantic integration also describes the meaning of the data and the dependencies between the different data elements. Thus it is the basis for advanced quality assurance methods, like end-to-end tests across different engineering disciplines.

The main difficulty for achieving semantic integration is the semantic heterogeneity of the do-

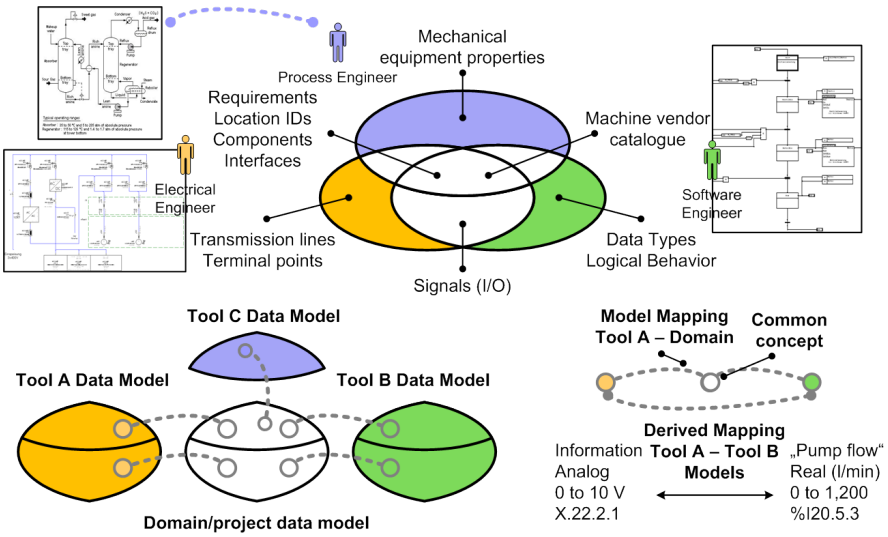

Figure 1.2: Identification of common concepts across engineering disciplines according to Biffl (2010).

mains and tools that have to be integrated. They often define the same or similar concepts in different ways. This is the result of the narrow scope of the domains and tools and the requirement to use the optimal data model for the tasks they have to fulfill. The tool data models were designed for different business needs and with different goals in mind. To resolve these difference both technical and domain expertise is necessary. The definition of the mappings between the different concepts for example depends on a deep understanding of all involved domain concepts in addition to technical knowledge (A. Y. Halevy, 2005). But because an integration platform for (software+) engineering has to be flexible and to provide the possibility to integrate new tools easily, the semantic integration cannot be done up front. It rather has to be part of the tool integration effort. Therefore these steps are performed by software integration and domain experts, which are not necessarily experts for semantic integration.

Semantic integration for (software+) engineering has to deal with different, tool specific data models of existing tools and frameworks. It has to be powerful enough to map these concepts, but nevertheless has to be efficient enough to produce fewer costs than benefit. It has to be easy enough to be done by domain experts with little experience with semantic integration. The following key points are main requirements a successful semantic integration solution for (software+) engineering has to fulfill:

Open platform: In (software+) engineering many different tools, both open and closed source, free and commercial are used. To provide a platform that will be accepted by the majority of actors in this heterogeneous environment a major requirement is that it is open and readily available. This gives everyone the opportunity to cooperate during the development of the platform and to customize it to fit their special requirements. Tool vendors can

provide integration solutions for their tools themselves using their unmatched knowledge and experience with their own tool.

Effective, efficient and robust integration: If technical and semantic integration is successfully implemented it provides a platform for quality assurance across tool boundaries and automation of error prone manual work. These advantages must exceed the effort for integration. The platform has to be flexible enough to integrate all the different tools, but complexity should be kept low to make sure the system is usable and robust enough to be accepted by domain experts.

Easy tool integration and exchange: Tool integration has to be possible with reasonable amount of work for the system integration experts and the domain experts and tools exchange should be even easier. This gives the domain experts the possibility to use the best suited tool for every individual project, regardless if there is already an existing integration solution for this tool or not.

Interface for advanced semantic applications: The semantic integration layer of the platform should provide an interface that can be used to develop advanced applications based on the semantic knowledge and data in the system. It should be possible to provide end-to-end tests across engineering domains and other quality assurance and project management applications, like change impact analysis or conflict detection across tools.

Support for technical integration: The semantic and technical integration have to be well coordinated, thus reducing the effort for both steps.

To fulfill these requirements a semantic integration layer is developed for the Open Engineering Service Bus (OpenEngSB) framework (OpenEngSB-Project, 2010). The first step is to develop a mechanism for modeling of the tool, tool-domain and engineering domain semantics. Besides ontologies also other modeling languages commonly used to capture semantic information are evaluated. An Engineering Knowledge Base (EKB) based solution is used to provide the semantic integration. The EKB has three main features (Moser, 2010):

1. data integration using mappings between different engineering concepts
2. transformations between different engineering concepts utilizing these mappings
3. advanced applications building upon these foundations

In this thesis the EKB concept is applied to the (software+) engineering domain to provide semantic integration for engineering tools. This means the EKB will make it possible to capture and manage the tool and domain semantics and to transform messages sent from one tool to another based on this semantic knowledge. The effective and robust management of the semantic information will be provided by a life-cycle model that defines all possible states a tool can

reach. To reduce the effort necessary for integration semi-automatic derivation of transformation instructions for the messages sent and received by the different tools is developed, providing the platform with the necessary scalability and usability. The EKB will also be the interface for queries against the virtual common data model. These queries are essential for project management and quality assurance tasks as they provide the possibility to retrieve and combine the data from all tools integrated in the OpenEngSB.

To make this platform acceptable for the engineering experts it is necessary to ensure the feasibility and quality of this solution. Furthermore the additional effort for tool integration added by a semantic integration layer has to be evaluated. Therefore an empirical study will be performed based upon two important use cases for (software+) engineering:

Definition of Quality Criteria Across Tool Data Models in Electrical Engineering: This use case is an example of the integration of tools from different engineering domains including software development and electrical engineering. It shows the importance of semantic integration to provide a common data model across tool and engineering domain boundaries. The seamless integration of the different tools involved in this use case is the basis to provide advanced functionality like end-to-end tests. The power of the possibility to query the virtual common data model with the help of the EKB is shown by formulating a sample query against this data model that validates quality criteria for sensors across tool boundaries.

Change Impact Analysis for Requirement Changes: Requirement traceability is a well known goal of software development processes, which makes the implicit interdependencies between requirements and other artifacts explicit. These interdependencies are semantic information, so the EKB based semantic integration layer can be used to implement requirement tracing. In this use case a change impact analysis is done for a changing requirement to find out which issues and developers are affected by the change request. This information can be used to mark all dependent artifacts for review and to contact all involved developers automatically. Furthermore it allows better estimates for the costs of the changes.

Section 2 describes technical integration, which is the basis for semantic integration. An overview about the different types of technical integration and various integration architectures is given. This section is interesting for system integration experts and software engineers, who want to study the technical basis for semantic integration.

Section 3 gives a definition for semantic integration and summarizes the problems solved and the issues raised by semantic integration. The different types of semantic integration are described and currently available semantic integration platforms are discussed. Finally the EKB concept is explained in detail. This section is interesting for integration experts and software engineers that want to understand the basic concepts of semantic integration and an EKB based integration solution.

In section 4 the theoretical background of data integration is discussed. Different integration mechanisms as well as the typical problems are covered. Finally the connection to the area of data virtualization is explained. The target group for this part is everyone interested in the basic principles and technologies used in data integration, like integration experts or domain experts. It is especially interesting for everyone who wants to capture the semantics of the data of a tool that has to be integrated.

In the following section the research issues are derived from the requirements for semantic integration in the (software+) engineering context (see section 5). The methodology that is used to resolve these issues is described and the use cases are explained in detail. Experts for semantic integration will be interested in the key research issues in this domain.

In section 6 the requirements of an EKB based semantic integration solution for the (software+) engineering domain are derived from two real world use cases. Domain experts as well as system designers and software engineers, who plan semantic integration systems will be interested in this section.

In section 7 the architecture of the EKB based semantic integration layer for the OpenEngSB is discussed. This part is interesting for system designers and software engineers who plan to implement a semantic integration system.

The solution is evaluated in section 8 with respect to feasibility robustness and efficiency. This part is especially interesting for system analysts and project managers, because the results for the two engineering use cases are presented.

Section 9 gives a detailed discussion about the results. It shows the advantages and problems of semantic integration in this domain. In addition the architecture of the semantic integration layer and the integration into the OpenEngSB will be discussed. This section is focused on analysts that plan to use the OpenEngSB platform with semantic integration.

Finally this work is concluded with section 10, which contains a summary of the thesis and discusses future work.

2 Technical Tool Integration for (Software+) Engineering

In this thesis a complete integration solution for (software+) engineering, supporting both technical and semantic integration will be developed based upon an existing technical integration solution . Therefore this section provides a definition of technical integration, describes different types of technical integration and architectures used to achieve integration. Finally, technical integration in the (software+) engineering domain is discussed.

2.1 Introduction

In modern enterprises a large number of different tools, each with a different scope and different properties are used. The reason for this is on the one hand that building a single application that serves all business needs is next to impossible and on the other hand that functional sectors of a company want to use specialized software, which supports their work in a locally optimal way (Yan Du, 2008). Furthermore after company mergers heterogeneous IT systems are created through the combination of the different IT infrastructures. In modern businesses the integration of these systems is important due to many different factors including fast changing economic conditions, high quality standards, increasing complexity or high cost pressure in today's globalized economy (Chappel, 2004). For technical integration the term Enterprise Application Integration has been coined by Linthicum (2000), who defines it as the sharing of business processes and information between all connected systems of an enterprise. Yan Du (2008) takes a slightly different viewpoint and defines EAI in the following way:

> *"The integration of applications that enables information sharing and business processes, both of which result in efficient operations and flexible delivery of business services to the customer."*

Both definitions have in common that they show two elements of EAI, namely data and functionality sharing. The second definition puts the focus on the expected result of EAI, which is an improvement of business processes.

Figure 2.1: Information Portal (Hohpe & Woolf, 2004).

Unfortunately many systems have not been built to integrate well with each other, which makes the task of enterprise application integration difficult. In addition, enterprise integration leads to the need to change corporate politics as the different groups have to give up some control over their toolset to make integration possible (Hohpe & Woolf, 2004). Despite these difficulties companies have tried to move towards better integrated systems. The scientific community has undertaken thorough research of these topics and identified different types of technical integration and different architectural patterns for integration solutions.

2.2 Types of Technical Integration

Hohpe and Woolf (2004) identify six different types of integration that show the different ways software architects have chosen to provide integration solutions. A typical integration solution often cannot be easily classified into one of these types, but contains elements from multiple integration types.

Information Portal (see figure 2.1) Information portals are used if users have to access more than one system to perform a typical operation. Simple solutions just provide the information from different systems in one interface, whereas more sophisticated portals also enable some interaction between the different background systems. The latter can also be seen as a simple form of integrated application, which shows that it is hard to make a strict distinction between the different integration types.

Data Replication (see figure 2.2) Replication provides different software systems with the same data. Often each system has its own way to store and manage data and cannot use a central storage solution. By replicating the common data each system can be provided with the data and still use its own persistence mechanism. Some databases support replication out of the box, but replication can also be achieved through import-export architectures or by using message oriented middleware. Data replication decouples the integration system from the operative system, because the data is stored in physically and logically separated locations. The main disadvantages of replicating data are inconsistencies between the operative and the integration data store and the overhead for keeping

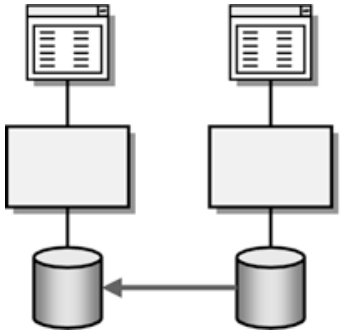

Figure 2.2: Data Replication (Hohpe & Woolf, 2004).

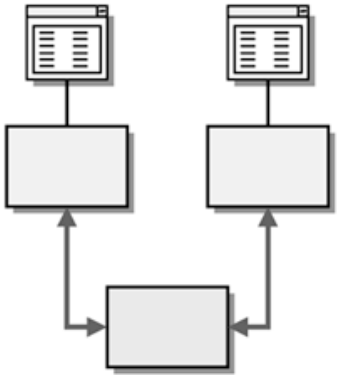

Figure 2.3: Shared Business Function (Hohpe & Woolf, 2004).

the data twice and for synchronization between the different storage systems. In dynamic systems, where data changes rapidly and where these changes have to be consistent other integration types are used. Data replication is typically used in situations, where data is retrieved much more often than it is changed.

Shared Business Function (see figure 2.3) Different software systems in one company have usually some functionality overlap. The common functionality can either be implemented separately in each system or provided as a single service available to all systems through a common interface. Many problems can either be solved by data replication or by shared business functions. The choice depends upon different criteria including the frequency of retrieval and change of the data. If the data is retrieved far more often than it is changed data replication is the better solution as it provides better retrieval performance. If the data is likely to change often the advantages of a shared business function, like a single point of data management become more important. If the functionality is complex, multiple implementations of the same functionality are always critical as they are likely to diverge and are hard to maintain. A shared business function is therefore a better solutions for such situations.

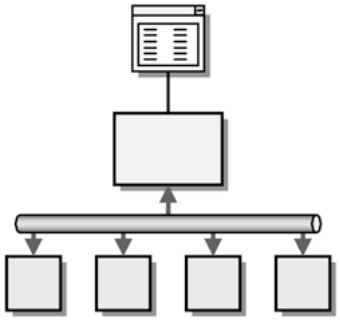

Figure 2.4: Service-Oriented Architecture (Hohpe & Woolf, 2004).

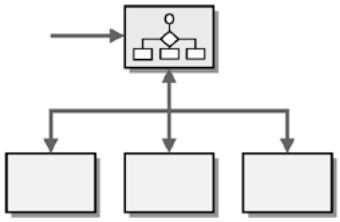

Figure 2.5: Distributed Business Process (Hohpe & Woolf, 2004).

Service-Oriented Architecture (see figure 2.4) According to Davis (2009) a service oriented architecture consists of reusable business services, which are combined to build new applications and to implement business processes. The two main features of a SOA are a.) the possibility to find services through a service registry and b.) that each service has an interface that describes how and under which conditions a service call may be performed. As the services are typically provided by different applications the process of calling services in a SOA can be seen as application integration. Because SOA is not only a specific type of integration but also a architectural form suited to build any of the listed types of integration solutions it will be covered in more detail in section 2.4.1.

Distributed Business Process (see figure 2.5) Business processes usually use functions of many different applications. While each of these functions is already implemented and provided by the different applications a central management component is needed to realize the business processes. The way the functions are accessed by the management component can vary and range from simple RPC calls to service calls in a SOA including service discovery and service call contract negotiation.

Business-to-Business Integration (see figure 2.6) If the systems that have to be integrated are not located in one business but distributed across different suppliers outside the company the term business-to-business integration is used. Additional problems like network unreliability and difficult negotiations about common data formats arise when applications from different companies have to be integrated.

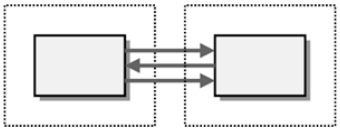

Figure 2.6: Business-to-Business Integration (Hohpe & Woolf, 2004).

2.3 Technical Integration Styles

Many integration solutions are ad hoc and have grown over time rather than being designed. These systems suffer from a wide variety of shortcomings including high costs for maintenance and change. The need for a more structured approach that addresses these shortcomings has led to the development of different styles of technical integration. Hohpe and Woolf (2004) categorize them into four different types, namely file transfer, shared database, remote procedure invocation and messaging.

2.3.1 File Transfer

As files are a common abstraction of data in many different systems and therefore supported by almost all platforms and languages they are a natural choice for integration between different applications. The knowledge about the internals of the applications needed is minimal, because the file formats that can be exported and imported represent their public interface. In addition to a common data format the participating applications have to coordinate how the files are named and where they will appear as well as how and when they might be accessed. After information was passed from producer to consumer someone has to delete the now obsolete files. As managing files and especially creating and changing them includes a rather large overhead the frequency of message exchange has to be well chosen. Too fine granularity, comparable to messaging systems, where every new information is shared immediately, is usually not possible because of performance constraints. Yet longer time frames between information exchange can lead to inconsistencies that are hard to resolve, like conflicting updates in different systems. Long update cycles often also compromise the original business goals of EAI as effective integration solutions need to be flexible and responsive.

File transfer is a simple integration style that is easy to understand and requires no external frameworks or additional tools. But performance issues, difficulties with access coordination and timeliness problems most often call for more sophisticated integration styles. In addition the negotiation and maintenance of a common message format is a complex task, where both technical and political problems have to be solved. This does not only include the definition of a common schema for the transfer files, but also coordination on the semantic level. For more information on semantic integration see section 3.

2.3.2 Shared Database

A shared database is a suitable integration solution if multiple applications have to access the same data. Modern databases provide good transaction support and access control. Data inconsistencies can be reduced to a absolute minimum and can be resolved easily. The main disadvantage of shared databases is the need to define one common data schema that is used by all applications. Such a common data schema can become complex if it is used by the different applications to store all their data. Often the usability of such a schema is bad, which makes application development more complex. There are also political reasons that make the introduction of a common data model difficult. Some application development teams might not be willing to take the additional effort involved in designing and using the common schema and the shared database. But not only the initial design of the schema is problematic, also changes to the schema or database are hard, because in the worst case all applications have to be updated. Another disadvantage of this integration style is that the shared database can become a performance bottleneck or that applications might deadlock as they lock parts of the database. If the database is replicated or distributed the performance of the network and distributed locking and synchronizing mechanisms might become a problem. Finally the usage of a common schema and a specific database is a hard to accomplish requirement if third party software is used. On the one hand most software products use their own way to store data and on the other hand most vendors reserve the right to change the way data is stored with every new version of their software.

Shared databases are a good solution if a common schema can be designed, which is usable for all involved applications and if there is enough control over the persistence mechanism of all applications. If one of this factors is not given, scalability is very important or functionality has to be shared rather than just data then a shared database is no suitable integration style.

2.3.3 Remote Procedure Invocation

Applications that need to share functionality in addition to data have to be integrated using the remote procedure invocation style also known as remote procedure call (RPC). By using RPC the data can be encapsulated within the application that owns it. Data retrieval and modification happen by direct calls from one application to the other. There are different technologies for RPC including CORBA, COM, .NET, Java RMI and Web Services. The complexity of remote calls is usually hidden by a middleware layer, which makes RPC calls appear like local calls to the programmer. Besides the advantage of better usability this can also be problematic, as programmers use RPC calls without having the performance and reliability issues in mind.

As the data is encapsulated within the different applications instead of a common data schema only the interfaces of the applications have to be negotiated. Although the RPC integration style reduces coupling compared to a shared database solution it still makes changes of a single part of the system hard. Even when the interface of an application stays untouched changes in

the internal implementation can lead to unexpected consequences, especially when it comes to sequencing (doing actions in a particular order). Another drawback of RPC style integration solutions is that many technologies for RPC calls do not work across different platforms.

The RPC integration style offers the possibility to integrate applications in a way that is natural to most programmers and therefore easy to understand. It enhances data encapsulation and enforces well defined interfaces between applications. The drawbacks of this methodology is that there is still a rather tight coupling between the applications and that software engineers tend to build integration solutions using the RPC style like single applications, which often leads to performance and reliability issues.

2.3.4 Messaging

In most integration scenarios the applications that have to work together use different platforms and different languages. Therefore an integration style that enables loosely coupled cooperation is needed. File transfer offers this feature but at the cost of timeliness, reliability and abstraction of the transport layer. Messaging tries to overcome these problems by sending small packets of data immediately in a reliable way. Messages are transmitted from sender to receiver asynchronously. They can be routed and transformed automatically while they are in the message channel or stored and retransmitted if the receiver is not available temporarily. Asynchronous communication is a main feature of messaging systems decoupling applications in terms of availability. One application can send information without having to wait for the receiver to consume it. Therefore the sender can immediately continue with its own tasks. But because most developers are not used to asynchronous messaging the development is harder, especially as asynchronous communication is hard to debug. Another feature of messaging frameworks is the possibility to transform messages while they are transmitted. This makes communication between two applications possible that do not even share the same message format further reducing coupling.

Application development and application integration are seen as two different tasks with their own set of properties and problems. Using the messaging integration style supports this clear separation much better than shared database or RPC. It allows the applications to use different conceptual models while providing a possibility to share data and functionality. This helps to create loosely coupled integration solutions, where each system has high cohesion but low adhesion to the other systems. Messaging also supports an event driven view of the integration system, which makes it easier to use for business analysts and project managers, who often have to map event driven business processes to the integrated IT system.

2.4 Architectures for Integration

Enterprise Application Integration can only work in an effective and efficient way if a suitable architecture for the integrated system is chosen. Often this does not impose a specific architecture on the involved applications, but simply defines the interface between the integration framework and the attached systems. According to Yan Du (2008) Service-Oriented architectures and the Enterprise Service Bus are the two architectures most commonly used to design enterprise integration solutions. Both architectures are based upon the messaging integration style. Therefore they support loose coupling between the integration system and the involved systems as well as asynchronous communication patterns. Nevertheless service-oriented architectures are often also influenced by the RPC integration style and provide RPC like interfaces for information exchange between integrated systems.

2.4.1 Service-Oriented Architecture

Service-Oriented architectures (SOA) are a language-independent conceptual model, that makes no assumptions about the underlying programming-model (Alonso, 2008). A service is a unit of functionality at a coarse enough abstraction level to provide business value. Services have a well defined interface and can be used without knowledge about implementation details (Dan & Narasimhan, 2009). In a SOA there are three different roles. Figure 2.7 shows the so called SOA triangle, which is comprised of service broker, service provider and service consumer. The service broker is responsible for storage and management of service descriptors and provides an interface to query and find services. A service provider contacts the broker to add its service and waits for service consumers. After a service was found in the broker by the service consumer it contacts the service provider directly to negotiate the service contract and call the service (Zhang Yi, 2009).

Web Services are one possibility to implement a SOA. Due to the heavy usage of standards and broad industry support Web Services have become the dominant form of SOA. The most important standards used by Web Services are UDDI for the service broker, WSDL for service interface description and SOAP for data transmission.

The term Service Oriented Integration (SOI) is used to refer to EAI using a service-oriented architecture (Zhang Yi, 2009). The basic principle is to add a service interface to an existing application and to build the integration based upon these high level interfaces. All applications that have to be integrated can be reached through these interfaces in a standardized way. The different services can be combined to implement business processes. The advantages of this solution are the standardization of the way services are described, found and accessed as well as the fact that existing applications only have to be extended to provide the data and functionality they want to share with other systems through a SOA interface. Yet the definition of the service interfaces can be difficult especially when the form and intensity of cooperation with other applications are not clear at design time. In addition the implementation of the SOI wrapper around

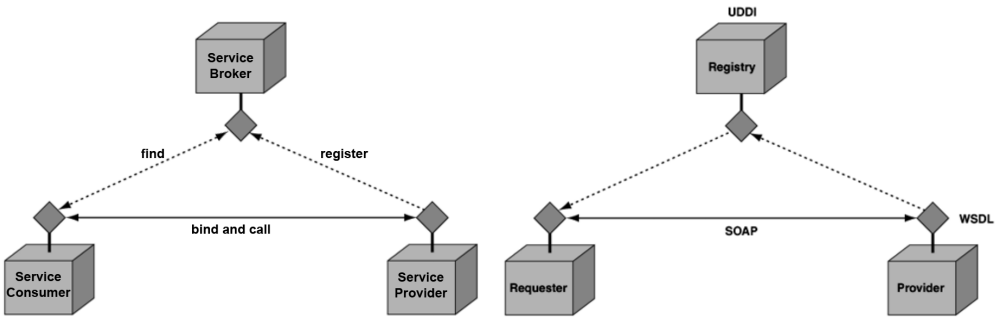

Figure 2.7: The SOA triangle visualized according to Zhang Yi (2009) and its representation for Web Services (Eric Newcomer, 2004).

some systems can be difficult for third party software and old legacy systems. Another problem with SOI is that many software developers think of SOA as another way of RPC which leads to very fine grained services, which suffer from the same problems as true RPC style solutions (see 2.3.3).

2.4.2 Enterprise Service Bus

The Enterprise Service Bus (ESB) concept builds upon other technologies like SOA, Message Oriented Middleware (MOM) and EAI solutions. Chappel (2004) defines the ESB as follows:

> *"An ESB is a standards-based integration platform that combines messaging, web services, data transformation, and intelligent routing to reliably connect and coordinate the interaction of significant numbers of diverse applications across extended enterprises with transactional integrity."*

By extended enterprise he refers to an organization, which has different logically and physically separated systems that have to be integrated, or which has to integrate its systems with those of business partners to achieve its business goals. As the definition already shows, the ESB concept was developed specifically for integration solutions. This makes it one of the most competitive integration architectures as many typical integration problems like the need for data transformation or the form systems connect to the integration middleware are already targeted at an architectural level. This reduces the amount of work the software architect designing an integration solution based upon an ESB has to do. Chappel (2004) defines amongst others the following characteristics of an ESB:

Pervasiveness An ESB has to be flexible enough to adapt to any form of integration environment including both local and global projects. Applications can connect in different forms

to an ESB using different protocols, while all systems plugged into the bus must be reachable in a standardized way.

Standards-Based Integration An ESB should make heavy usage of standards including Web Service as well as MOM and other standards. Data management and transformation has to be based upon XML standards. By using these industry standards the effort for any system to plug into the ESB should be kept low.

Distributed Integration The core services provided by an ESB like business process orchestration, routing and transformation have to be implemented in a way that allows distribution to increase the scalability of the ESB.

Data Transformation An ESB has to provide a transformation infrastructure, where any message can be transformed between sender and receiver, regardless of the location or other properties of the participation components.

Layered Services The ESB has to support the addition of customized layers for specialized business needs, like Business Process Management or collaboration servers. These should integrate seamlessly into the bus providing their functionality to the other components.

Event-Driven SOA An ESB should abstract away the details of communication and information delivery. That means that from a architectural standpoint each application plugged into the bus is seen as a service endpoint reacting to asynchronous events.

Operational Awareness Business Activity Monitoring is used to examine the state of business operations while they are in progress. By using XML standards for data management an ESB can provide real-time insight into the data processed by the ESB.

Incremental Adoption An ESB has to allow incremental adoption as big bang introduction of integration solutions usually do not work even for small companies and are completely impossible for extended organizations.

Figure 2.8 shows a possible technical structure of an ESB with message routing, validation and transformation capabilities.

2.5 Integration in the (software+) Engineering Domain

Although integration will be a key success factor in (software+) engineering as the involved systems and business processes become more and more complex and the growing tool landscape harder to maintain, there has been relatively little effort to apply EAI in this domain in a structured way. Most organizations rely on point to point integration between the different engineering tools, which is in some cases not even automated but done by humans. But as development becomes more difficult and time and cost effectiveness as well as high quality

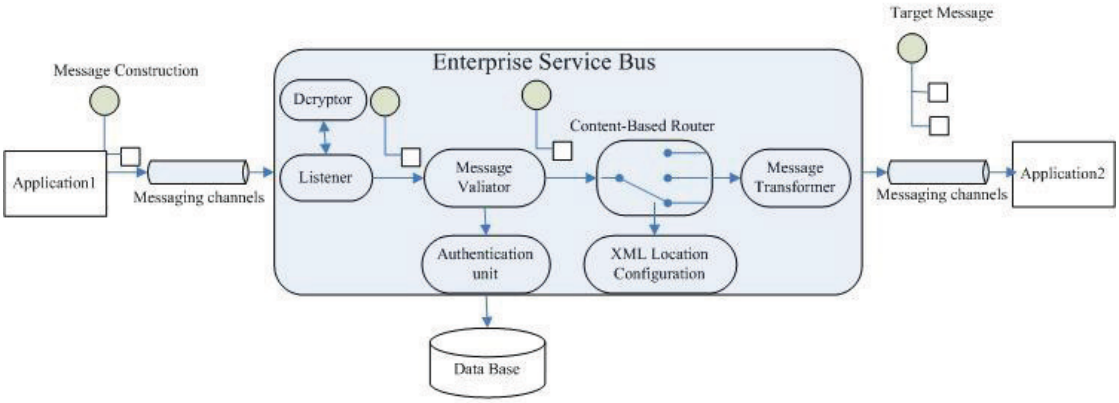

Figure 2.8: A possible technical structure of an ESB (Jieming Wu, 2010).

standards become more important due to increased competition in a globalized economy and many business leaders in this domain identified EAI as a key objective, companies are likely to increase their integration efforts in the near future.

Taking a closer look at EAI in the (software+) engineering domain makes clear that the systems that have to be integrated are mainly the development tools. In a typical (software+) engineering process some of these tools have always had to cooperate, but this integration is mainly achieved by hard to maintain, very specialized point to point integration solutions. In most cases some form of script-based approach is used to create these point to point links. The major problem of this point to point integration style is that it does not scale well. In software development integrated development environments (IDE) have developed in the last years, which offer some amount of tool integration. Yet while there are very mature and successful IDEs for software development, there is no IDE for (software+) engineering. Different projects have been started to target integration problems in the (software+) engineering domain. The following list describes some of them:

Cesar[1] The aim of this project, which is funded by the European Union, is to define a multi-view, quality assuring process for (software+) engineering projects. Although some interesting solutions have been proposed no implementation using the results of this project exists.

Modale[2] In this project supported by the German government a semantic integration framework for (software+) engineering tools has been developed, but no complete integration solution. Modale uses ontologies to design the semantics of the tools, which have to be integrated.

[1] http://www.cesarproject.eu/
[2] http://www.modale.de/

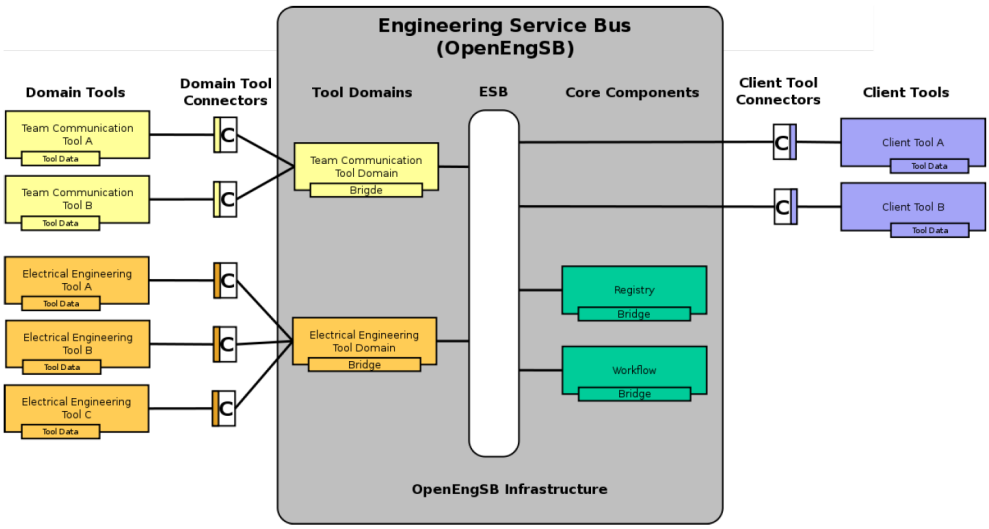

Figure 2.9: The architecture of the OpenEngSB (Pieber, 2010).

Medeia[3] The goal of this project is to develop an automation component model. Medeia is supported by the European Union. Like Modale it is a semantic integration framework with little technical integration support. To model the semantics of the integrated system UML based meta models are used.

Comos[4] An integration framework that uses XML files for data exchange between tools and provides integration for some important tools. The system is meant for large scale automation engineering environments and provides a unified view on the whole system. Comos is an all-in-one solution, which tries to incorporate all necessary components. Therefore customization is limited and can only be done using a predefined scripting approach.

OpenEngSB[5] Although this project is still in a rather premature phase it shall be discussed here as it is the target platform of this thesis. The Open Engineering Service Bus is an extended Enterprise Service Bus that was developed for the (software+) engineering domain. In this environment some of the characteristics of an ESB are especially important, although all of them contribute to a successful integration solution. Pervasiveness is necessary, because (software+) engineering projects are usually large scale projects, but companies usually try new technology in small projects before they use it in projects with significant business value. Data transformation is needed, because the data format of integrated tools often cannot be changed, as third party tools or other external components are used. Improved quality control and project management are key goals of system inte-

[3] http://www.medeia.eu
[4] http://www.comos.com/
[5] http://www.openengsb.org/

gration in a (software+) engineering environment, which makes operational awareness a critical feature of an integration solution.

The OpenEngSB uses different integration types to achieve technical integration. It uses aspects from shared business function, as the different tools provide their functionality to workflows and other tools via the bus. Furthermore as the OpenEngSB is based upon an ESB it uses aspects from service-oriented architectures, like service discovery and standardized service interfaces. Finally the OpenEngSB is designed to support distributed business processes using the central components, which extend the typical ESB functionality and like all ESBs the OpenEngSB also support business-to-business integration, as one typical characteristic of an ESB is distributed integration.

The major advantage of the OpenEngSB is that it is an open platform, which can be extended and adapted to suit the needs of different organizations. The OpenEngSB supports typical (software+) engineering tasks by providing basic components, like the workflow component for process support. The advantage of this direct integration into the infrastructure is the improved interaction possibility between the core system and these central services. To make tool exchange easy the OpenEngSB uses a two layer architecture to connect tools. A domain, which abstracts the typical functionality of tools of a given domain is used as an intermediate layer between system and tool. Domains define an interface, which has to be implemented by each tool connected to the domain. Therefore when business processes are defined they can interact with the domain interface. If a tool has to be exchanged no changes in the business process definition is necessary. Figure 2.9 shows the basic architecture of the OpenEngSB, with the core registry and workflow components and the domain based two layer tool connection structure.

3 Semantic System Integration

In this thesis a semantic integration solution for (software+) engineering is developed. Therefore this chapter discusses the theoretical background of semantic integration. Different types of semantic integration are identified and described. Finally the Engineering Knowledge Base (EKB) concept is explained in detail as it is the basis for the proposed integration solution for (software+) engineering.

3.1 Introduction and Definition

A major problem that occurs when different systems have to be integrated is data heterogeneity. According to Cruz et al. (2004) the mismatch in data representation can be classified into three categories: syntactic, schematic and semantic. Wile technical integration focuses more on syntactical and to some extent schematic issues and on providing a common message format to make communication between different systems possible, semantic integration focuses on the meaning of the data exchanged between those systems. This includes both schematic and semantic factors. Effective and efficient cooperation between different tools or systems is only possible if the semantics of source and target system are compatible (Rosenthal et al., 2004). Yet this compatibility is not given in most organizations, because applications are not developed with interoperability in mind and because new systems are added as a result of mergers or acquisitions (A. Halevy, 2005). Semantic integration tries to solve the problem of semantic heterogeneity by providing some form of intermediate layer that automatically transforms data between the different involved systems.

A. Halevy (2005) states that the main reason why semantic integration is hard to achieve are the differences between data sources. These differences result from different design goals. Therefore the same concept is often represented in different forms with varying levels of detail. A particular hard to resolve problem are overlapping or conflicting data representations. Furthermore in order to resolve semantic heterogeneities both technical and domain expertise is needed. Automatic solutions for semantic integration are usually not possible, because for machines the task to understand and align different data schemata is even harder than for humans as they do not have the necessary meta information about the intent of the schema designer.

subject	predicate	object
glass	be	empty
Alice	meet	Bob

Table 3.1: Statements represented as RDF triples.

3.2 Approaches for Semantic Integration

Much research in the field of semantic integration has been undertaken to solve the problem of conflicting data schemata and data semantics. The following two different solution categories are the most common solution types in literature. They are described in detail, because they have the most relevance for the proposed semantic integration infrastructure for (software+) engineering.

3.2.1 Ontology-Based Semantic Integration

In computer science an ontology is defined as representation of knowledge of a domain and relationships between these concepts in a machine understandable notation (Rebstock et al., 2008). Ontologies have originated in artificial intelligence, but have been used for integration solutions, because they provide a high level platform independent format for describing data models. Hakimpour and Geppert (2001) state that the correct understanding of the semantics of data is important for integration. They further state that formal ontologies are a promising way to describe the semantics of integrated systems in a application-independent way. Ontologies represent semantic information in a form that enables reasoning about the data and the automatic derivation of new knowledge.

One of the most important application areas of ontologies is the semantic web. Therefore many standards for ontology description and ontology languages have been developed in this research field. The most prominent among these ontology languages is the OWL Web Ontology Language, which is recommended by the World Wide Web Consortium (W3C). OWL is based upon the Resource Description Framework (RDF). In RDF statements are captured in subject, predicate object triples. The statements “The glass is empty.” and “Alice meets Bob.” for example are represented in RDF as the triples shown in table 3.1. Because of this triple structure the data model is represented as a labeled directed graph in RDF and allows the mixture of structured and semi-structured data. RDF can be serialized in different forms and is used as a model for data interchange on the Web (W3C, 2010b).

The current version of OWL (OWL 2) is available in different so called profiles, which are subsets of the full OWL language well suited for special use cases. These subsets are necessary because OWL is a very expressive language, which can be hard to implement if its full power is used. The profiles are:

OWL2-EL When OWL2-EL is used, reasoning can be done in polynomial time which is useful if ontologies are huge and performance is critical.

OWL2-QL OWL2-QL enables efficient querying of large numbers of individuals. It supports the usage of relational databases and SQL queries.

OWL2-RL This subset of OWL2 is used if lightweight ontologies are needed to organize large numbers of individuals and when rule-extended database technology is used.

The full OWL2 language is also called OWL2-DL for OWL2 description logic (W3C, 2010a).

Noy (2004) gives an overview about different semantic integration approaches using ontologies. She defines three dimensions of semantic integration:

Mapping discovery The process of determining similar concepts in different ontologies also called ontology alignment. A possible way to implement ontology alignment is the usage of a shared upper ontology. In this upper ontology basic concepts are defined, which are used to define the concepts of all dependent ontologies. The common upper layer can then be used to find possible mappings between upper level concepts and dependent concepts and also between different dependent concepts. Other approaches rely on heuristics or machine learning algorithms, which enable the utilization of information stored in the instances rather then explicitly modeled in the ontologies.

Declarative formal representations of mappings The way a mapping is defined in a machine understandable way that allows automatic reasoning with mappings. A possible way to represent mappings is to use an ontology of mappings. This has the advantage that no additional external language is needed. Another possibility is to use a view based mechanism for describing ontology mappings including both global-as-view and local-as-view based solutions.

Reasoning with mappings The process of deriving new knowledge about available data and relations between data elements using the mappings determined and defined in the previous steps. This includes automatic data and query transformation as well as ontology instance transformation. The reasoning mechanism used depends on the representation format used for ontology mappings.

These three dimensions show the basic mechanism of semantic integration solutions. First the semantics of the different systems are captured in ontologies, then mappings between these ontologies are defined and finally the information gained by the mapping process is used to derive further knowledge by reasoning and to perform typical integration tasks like data transformation.

The main drawback of ontology-based integration solutions it the complexity of ontology development and management. There are often not enough experts available, which have both technical knowledge about ontology creation and domain knowledge about the systems that have to

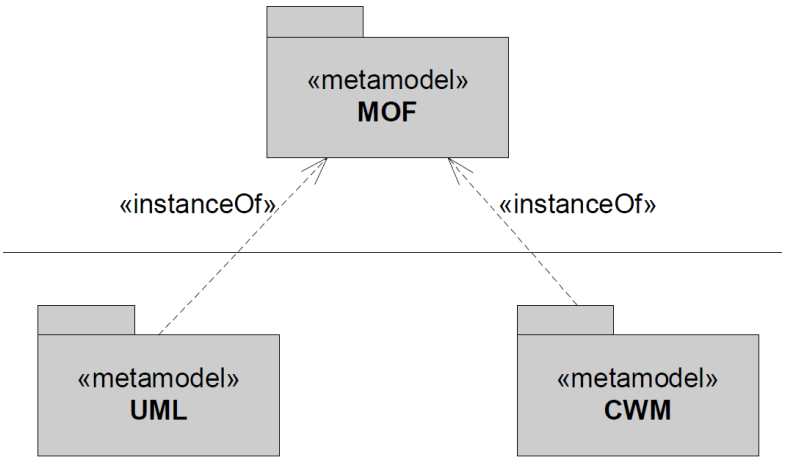

Figure 3.1: MOF as basis for other modeling languages like UML or CWM (OMG, 2010b).

be integrated. Further more ontologies are often hard to link to the actual data models used by the different systems, because they describe the data at a very high level of abstraction. Tool support for ontology creation and management has improved recently, but is not comparable to the sophisticated tools available for example in model-driven development.

3.2.2 Model-Driven Semantic Integration

The model-driven integration approach is closely related to the ontology-based integration approach. The main difference is the heavy usage of standards, tools and processes that have been developed in the context of model driven development and model driven architecture. Like in the ontology based approach semantic integration is realized in three steps. First a model for every system is developed, then the relationships or mappings between the model elements from these different models have to be discovered and represented in a machine understandable way and finally typical integration tasks like data transformation is performed based upon these mappings (Kramler et al., 2006).

Model-driven integration approaches rely heavily on standards, like Unified Modeling Language (UML), a specification of the Object Management Group (OMG). The Meta Object Facility (MOF) is the common basis for UML and other modeling languages like Common Warehouse Metamodel (CWM) and represents the foundation of Model-Driven Architecture as it is defined by the OMG. Figure 3.1 shows the connection between UML, CWM and MOF. The goal of this standards is to ease the process of software development by unifying the whole process from business modeling to deployment, evolution and integration (OMG, 2010a).

One advantage of model driven semantic integration is the good tool support for model driven development as well as the fact that there are many developers familiar with model driven data

modeling. The models can automatically be transformed into executable code, which reduces maintenance effort and increases flexibility with respect to model changes. In addition many data models are already available as UML models, which reduces the initial effort for model based integration (Amar Bensaber & Malki, 2008).

The most important problem of model driven semantic integration is the fact that most modeling standards that are in broad usage have not been developed to capture the semantics of data well. They are more focused on syntactic and schematic issues (Jamadhvaja & Senivongse, 2005). One possibility to add the semantic information is to define a specialized modeling language based upon MOF. Another possibility is to use ontologies to capture the semantic information and to define formal mappings between concepts in the ontologies and model elements in the MOF based models. There is some overlap between MOF and ontology languages like OWL, e.g. the possibility to define classes, subclass-of relationships or relationships between classes. Yet the focus of OMF and ontology languages is different. Wile OWL was developed by the semantic web research community to precisely define the meaning of concepts and to allow reasoning about statements containing information about those concepts, OMF has been developed for software engineering purposes (Frankel et al., 2004). As semantic integration of (software+) systems touches both areas, software engineering and the need to capture and reason about semantic information, it is a good application area for model driven semantic integration. For other application scenarios with a weaker connection to software engineering specialized ontology-based semantic integration solutions may be a better choice.

3.3 Engineering Knowledge Base

The Engineering Knowledge Base (EKB) is a semantic integration approach for tool and data integration in the engineering domain proposed by Moser (2010). The three main features of an EKB are:

1. data integration using mappings between different engineering concepts
2. transformations between different engineering concepts utilizing these mappings
3. advanced applications building upon these foundations

The EKB framework was developed for engineering tool integration. By providing an effective and efficient semantic integration layer it simplifies the process of engineering. Especially tasks that span different domains, where experts with different technical background have to cooperate can be performed with less effort if all involved tools are integrated semantically. Moser (2010) describes the main target of the EKB as follows:

> *"The EKB is used to facilitate the efficient data exchange between these engineering tools and data sources by providing a so-called "virtual common data model"."*

The common data model is called virtual, because the data is not stored and managed using this global knowledge model. The data of the involved tools is transformed to match the virtual common data model at runtime using mappings between the respective tool models and the common data model. Therefore the different tools are not directly bound to the common data model and do not have to use it to store their own data. Because of this abstraction of the virtual data model the problems of common schema negotiation become less critical. As the different tools can still build upon their own data schemata no difficult schema negotiations have to be undertaken between the different system designer. Changes to the common data schema have no influence to the data or functionality of the respective tools. Furthermore the complexity of the common data model does neither harm the performance nor complicate the development process of the integrated systems.

Although the EKB was developed as ontology-based integration solution, it is possible to use any format for knowledge representation including MOF based data models. The EKB framework manages engineering knowledge in a machine understandable format and performs data transformation based upon semantic mappings at runtime.

The major drawback of an EKB based integration solution is the necessary effort needed to model the virtual common data model and the mappings to the tool models. This process needs technical and domain knowledge and may include the usage of new technologies. Therefore the introduction of an EKB into a company may meet with political opposition, because developers need training or specialists have to be employed and new tools have to be introduced into the development process. Furthermore if ontologies are used to represent the semantic knowledge the lack of industry standard tool support in this domain further complicates the usage of an EKB. Finally there is currently no complete implementation based upon the EKB approach, although the OpenEngSB[1] project is currently evolving in this direction.

Figure 3.2 shows the internal structure of an EKB. The different parts are labeled with numbered tags in the the graphical representation.

(1) Tool Data Extraction Each tool provides its data by a connector to the EKB framework. The connector is responsible for extracting the data from the tool and making sure the data adheres to the tool specific knowledge model.

(2) Data Storage A central persistence service called Engineering Data Base is provided by the EKB framework for tools that cannot manage their data locally, because they are not always available, or do not support sophisticated data management tasks.

(3a) Definition of Tool Specific Local Knowledge The knowledge model of the tool has to be described. This must include the data structures the tool provides and consumes and can optionally also include the meaning of the model elements.

[1] http://www.openengsb.org/

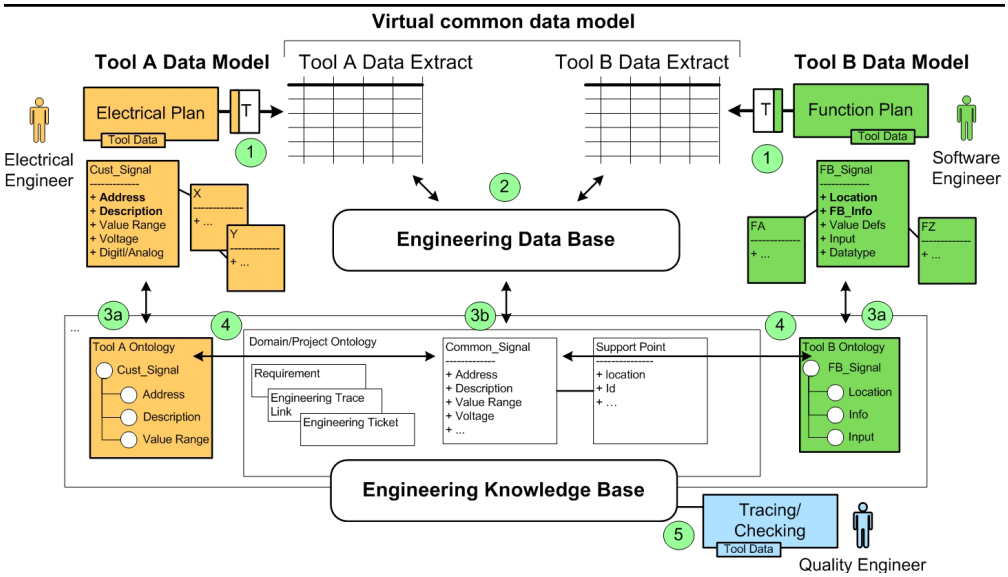

Figure 3.2: Use case scenario for an EKB based semantic integration solution (Moser, Biffl, et al., 2010).

(3b) Definition of Global Knowledge Like the local knowledge models the virtual common data model has to be defined. This step is of particular importance as the virtual common data model is used by all applications building upon the EKB integration framework. Usually the global knowledge model contains the common elements from the different tools that have to be integrated. This high level knowledge model should be independent as far as possible from the specific tools to reduce the costs of tool exchange.

(4) Mapping between local and global knowledge The mapping between the tools specific local knowledge models and the global knowledge model has to be defined. The semantics of the models have to be taken into account here. Using the mapping to the global data model semantically similar concepts in different tools can be identified. The mapping has to be stored in a machine readable format that allows automatic transformation from tool specific data to the virtual common data model.

(5) Interface for Advanced Applications The EKB framework provides an interface for extracting data from the different integrated tools using the virtual common data model. This enables advanced applications like change impact analysis, quality assurance, business process modeling and monitoring or project management.

Figure 3.2 shows a possible use case of the EKB framework. Tools from electrical and software engineering provide data, which is either managed directly by the tool itself or stored in the Engineering Data Base, which is a global persistence service. The experts from each respective domain do not need to understand the other domains or the integration framework to work with

the system, but can continue to use their well known tools to perform their tasks. One of the advanced applications based upon the EKB framework in this use case would be an end-to-end test using data from tools for creating electrical plans, configuration tools and software engineering tools to find out if the data type in the software is consistent with the physical component it represents. Currently such end-to-end tests can only be performed manually, or by using fragile and complex point-to-point integration between the different involved engineering tools. In section 6.1 such an end-to-end test is described in detail as a use case scenario for the semantic integration framework proposed in this thesis and in section 8.1 a prototypic EKB based implementation is evaluated.

4 Data Integration for (Software+) Engineering

In this thesis a semantic integration solution for (software+) engineering is developed. As a main part of the integration solution is the provision of the data of the integrated tools through a virtual common data model, this section shall give an overview about data integration and its theoretical background. Common problems and solution strategies as well as different integration techniques are discussed. Finally the topic of data virtualization and its relationship to data integration is covered.

4.1 Definition and Introduction

Lenzerini (2002) defines data integration in the following way:

> *"Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data."*

This definition already shows the general setup of data integration problems. There are different existing data sources with incompatible data models or even incompatible technologies and their data has to be offered to users through an uniform interface. The user should not realize that he is dealing with heterogeneous data sources, but the system has to react as if it uses a single data source. Data integration therefore also includes the topics schema mapping and query processing, which will be discussed in detail in sections 4.2 and 4.3. The focus in this section is on data integration based on mapping data from different sources to a global schema or model rather than on physically duplicating the data to store it according to the global schema. In a (software+) engineering environment the data stored by the different tools most often contains design or implementation information, which can change rapidly. These changes have to be consistent throughout the system. Duplicating this data would increase the probability of inconsistencies and would contradict the principle of a single point of change. Therefore the data-warehouse approach, where data from different sources is stored in one central place with respect to one

well defined global schema and where the integrated data is separated from the operative data is not covered in detail here.

One important trend in information technology in the last years has been the increasing importance of data (Brodie, 2010). This can be observed by the enormous amounts that have been paid for companies, whose main business goal has been the collection and connection of user data. In addition many sophisticated applications that are currently developed use different and often heterogeneous data sources, like for example semantic search engines. Data integration is an important topic in many application areas today including Enterprise Application Integration, Semantic Web or Cloud Computing. Many companies need to tap into different data sources to implement their business processes in an effective and efficient way. In science data from different researchers has to be collected to be analyzed and interpreted. Governments have to use different data sources to provide their citizens with convenient services, but also to check the results of public spending or to improve public security (A. Halevy et al., 2006).

There has been much research about data integration that has led to different solution strategies, algorithms and even some commercial products in this field. While at the beginning mainly database specialists where concerned with the research, today the development of the world wide web, which contains millions of heterogeneous data sources and large research areas like the semantic web or cloud computing have contributed to increased research and business activity in the data integration domain (Brodie, 2010). Some of the problems that occur in modern data integration scenarios are unknown data sources, incomplete data, hidden or unstructured data that has to be extracted or preprocessed and constant changing data sources or unreliable data sources (Cafarella et al., 2009).

4.2 Schema Mapping

The process of mapping the data models of the various data sources to one global data schema is called schema mapping. As this process is complex because of the syntactic and especially because of the semantic differences of the models it is usually done manually. Many different algorithms for semi-automatic and automatic mapping support have been proposed, but many of them rely on domain specific heuristics or special preconditions (see 4.2.4). Therefore no general-purpose industry standard solution exists today. Schema mapping mechanisms can be distinguished into global-as-view (GAV) and local-as-view (LAV) solutions. To make a clear distinction and a consistent definition of these two approaches it is necessary to formalize the notion of a data integration framework, queries and mappings. Therefore in this section the theoretical background of schema mapping is given. The formulae and explanations in this section are based on Lenzerini (2002) if not stated otherwise.

A data integration system that offers an interface against a global schema for different heterogeneous data sources can be defined as a system I of three components. The schema of the data

sources S, the global schema G and the mapping M. It can therefore be describes as the triple $\langle G, S, M \rangle$, where

- The global schema G is expressed in a language L_G over an alphabet A_G. For each element of G there is a symbol in the alphabet. This means that if G is expressed in an object oriented way, there is one symbol for each class in the alphabet.
- The source schema S expressed in language L_S over the alphabet A_S. Like for the global schema the alphabet again contains one symbol for each element of S.
- The mapping M between the sources schema and the global schema. The mapping is comprised of a set of assertions of the form $q_S \rightsquigarrow q_G$ and $q_G \rightsquigarrow q_S$. Where q_G and q_S are queries defined over the global or the source schema, which have the same arity. To express queries q_S the query language $L_{M,S}$ over the alphabet A_S is used, while queries q_G are defined in a query language $L_{M,G}$ over the alphabet A_G. Therefore $q_S \rightsquigarrow q_G$ means that the concept queried by q_S in the sources schema corresponds to the concept queried by q_G in the global schema.

Data is stored at the sources using the source schema. There is no data stored using the global schema, which means that the global schema can be called virtual. The only way to retrieve data using the global schema is to transform it using the mappings M. If the data integration system is queried a query language L_Q over the alphabet A_G is used. So only elements which are represented in the global schema can be queried.

Based upon this formal definition of a data integration system, a detailed definition of the two mapping approaches local-as-view and global-as-view can be given. The following sections briefly introduce the necessary concepts and give a comparison of the two mapping types.

4.2.1 Local-As-View

The global schema in local-as-view based schema mapping solutions are defined independently of the data sources. Therefore every source has to be defined as a view over the global schema. So in a data integration system $I = \langle G, S, M \rangle$ the mapping M connects each element in the sources schema S to a query q_G over the global schema G. This means that if the local-as-view approach is used the mapping consists of one assertion of the form $s \rightsquigarrow q_G$ for each element s of S. Local-as-view based systems are focused on extensibility, as adding a new data source can be achieved simply by adding one assertion to the mapping. In addition the definition of the mapping between the local and the global schemata is a rather straightforward process, where the content of the sources is defined with respect to the global schema. No knowledge about the other data sources is necessary to create the mapping for a given source. The global schema has to be well designed and has to remain rather stable over time. Therefore this mapping approach is often used if an enterprise data model or something comparable exists. The main drawback

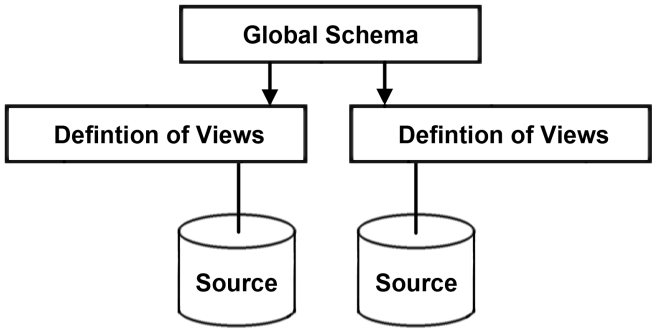

Figure 4.1: Structure of a local-as-view system based upon (Boulcane, 2006).

of local-as-view based approaches is the difficulty of query processing. The mapping gives little information about how a query against the global schema can be translated into queries for the source schemata. Figure 4.1 shows the basic structure of local-as-view based systems.

4.2.2 Global-As-View

A schema mapping solutions is considered to be global-as-view if the common data schema is defined in terms of the integrated data sources. From the theoretical point of view this means that for every element g in G a query q_S over S is included in the mapping M. In a global-as-view based approach the mapping M is a set of assertions of the form $g \rightsquigarrow q_S$, one for every element g of G. This means that every element of G is linked to a view q_S over the source schemata. The global-as-view based approach is therefore better suited if the sources are rather stable and many queries have to be executed against the global schema, because the mapping M defines how the data can be extracted from the sources. The drawback of global-as-view solutions is the influence each new data source may have on the global schema and all associated views, which can lead to a redefinition of these views. Furthermore the definition of the mapping between the sources and the global schema is harder than in local-as-view based systems, because the data extraction procedure has to be defined. Therefore global-as-view systems are said to correspond to a more procedural way of modeling data integration systems. In figure 4.2 the structure of a global-as-view system is depicted.

4.2.3 GLAV and BAV

To combine the advantages of global-as-view and local-as-view based solutions hybrid solutions have been developed, which are called GLAV for global-local-as-view. In GLAV both assertion types can be used to define the mapping between the source schema and the global schema. These hybrid systems are often also called mediated systems, because they usually use an additional mediation layer between the global schema and the sources. Boulcane (2006) describes

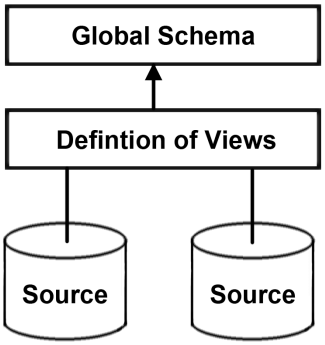

Figure 4.2: Structure of a global-as-view system based upon (Boulcane, 2006).

a hybrid schema mapping systems, which operates on several layers, which is called a multi-mediator schema mapping architecture. In this solution the global schema is connected to an intermediate layer using the global-as-view approach, while the intermediate layer is connected to the data sources using the local-as-view approach. Figure 4.3 shows the structure this mediator based system. McBrien and Poulovassilis (2003) propose another schema mapping approach called both-as-view (BAV). BAV is based on reversible schema transformations and provides the possibility to derive both a global-as-view and a local-as-view representation of the system. Therefore if BAV is used to model the mapping the advantages of local-as-view and global-as-view based solution can both be utilized. Further more the both-as-view solution supports the evolution of both the global schema and also the data sources.

In an EKB based integration solution the data integration layer can operate in any of the described schema mapping types. Therefore the EKB can be classified as a BAV solution, because it is possible to define the mappings between the local and the global schemata in a form which allows the deduction of both a local-as-view and a global-as-view model of the system. Nevertheless it is also possible to use an EKB in a strict global-as-view or local-as-view manner, but with the restriction that if a local-as-view approach is used the system has to be provided with enough information to process queries against the global data schema in an effective and efficient way.

4.2.4 Generating Schema Mappings

The generation of the mapping between the local and the global schema is a complex process that requires both theoretical knowledge about database systems and mapping generation and domain knowledge to be able to map the different schema elements correctly. Because of the tedious and repetitive character of this work many researchers have developed semi-automatic mapping support. These systems help to reduce the manual work and use different strategies to perform simple mappings automatically. As mapping generation is a NP complete problem and

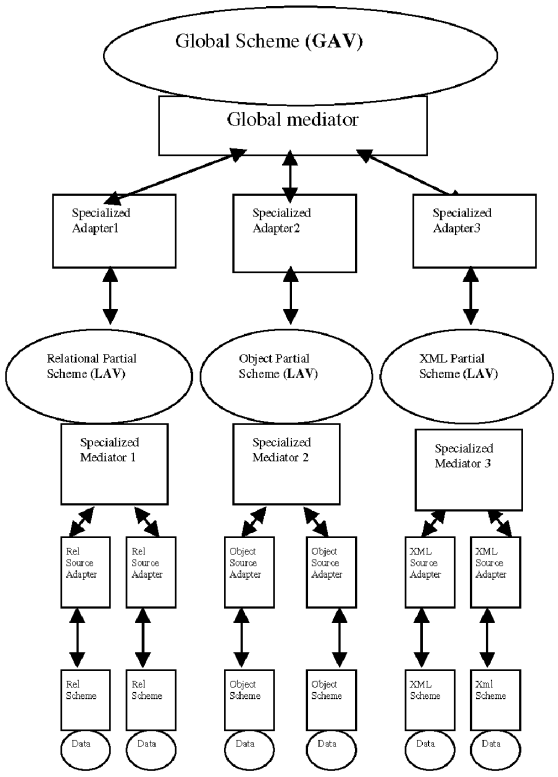

Figure 4.3: Structure of a multi-mediator based data integration system (Boulcane, 2006).

because of the huge amount of metadata which would be necessary, a fully automated mapping solution is not feasible (A. Halevy et al., 2006).

The semi-automatic mapping solutions deploy amongst others the following techniques to support the mapping process (A. Halevy et al., 2006):

Schema Similarities The elements in the source and the global schema that are alike are potential candidates for mapping. They can be similar with respect to syntax, like for example if the same modeling elements are used. Often also the usage of the same datatype or the same data structures is used to increase the mapping probability of two elements. The underlying assumption for this methodology is that syntactic similarity implies semantic similarity.

Naming If two elements have equal or similar names or labels they are candidates for mapping.

Combined To increase the performance and accuracy of the automatic mapping procedures many different techniques are combined to identify corresponding elements in the source and the global schema.

Machine Learning The repetitive nature of many tasks that have to be performed during schema mapping and the fact that models from the same domain often are relatively similar make the mapping process a suitable candidate for machine learning approaches. Here at first some mappings are performed manually and used as learning sequence for the system. The quality of the solution depends on the similarity between the input model and the training data.

There are many different approaches for automatic schema mapping support based upon ontologies, model-driven or directly operating on relational data schemata. Chen et al. (2009) for example propose an ontology based mapping mechanism, which extracts attributes from data sources on the web and evaluates the mapping probability using a semantic attribute matrix. This semantic similarity between attributes is calculated by WordNet and the matrix is later analyzed using a reverse backtracking algorithm to find semantically related attribute groups. This data integration approach is local-as-view based, but uses some concepts from mediated schema integration. Xiong et al. (2009) present a mapping procedure for XML data integration. They match similar elements after parsing and transforming the original XML schemata into an intermediate format. Therefore elements are mapped into the same concept in the global schema if they correspond in the intermediate format. The structure of the system proposed by Xiong et al. (2009) is mediator based, where the local XML schema is the mediator between the actual data source schema and the global schema. Hu (2006) propose a local-as-view based approach, which generates the global schema from the local views. The algorithm he proposes tries to resolve structural and semantic conflicts and also generates the mapping information needed for query processing parallel to the generation of the global schema. While the automatic generation of the global schema ensures a clear definition and perfect coordination with the mappings, it can lead to a schema that is hard to understand and difficult to use. As the user has to use the global schema to formulate queries its usability is of special importance.

Besides mapping the schema of data that has to be integrated often also the instance data has to be transformed before it can be presented to the user of the global interface. The reason is that although the same schema is used there can be semantic differences in the way the data is represented. For example could the same address be stored in different ways, or the same company name could be stored in different forms and the data integration system has to reconcile this data to collapse the duplicated data elements. Because the amount of data that has to be processed is usually very large automatic systems have to be used to perform these operations (A. Halevy et al., 2006). As these automatic methods for data integration have been rather successful some researchers are trying to use the research results from data mapping for fully automatic schema mapping. Gal (2008) discusses the problems and possibilities of fully automatic schema mapping for data integration. Furthermore he focuses on the uncertainty of the mapping process and the influence on the correctness of generated mappings.

Automatic mapping generation is an important part of an EKB based integration approach, as it reduces the initial effort for system integration. A combined automatic mapping procedure,

which uses syntactic, naming and other available information to create a mapping proposal is a feasible solution for an EKB based system. This proposal is then edited and corrected manually by domain experts. Machine learning approaches in the (software+) engineering domain suffer from the lack of sufficient training data and the heterogeneity of different tools and engineering domains. Also the automatic generation of the global view as proposed by (Hu, 2006) is not feasible, because the usability of the global schema is important as it is used for project management and quality assurance tasks.

4.3 Query Processing

Query processing for data integration systems generally deals with transforming a query against the global schema into queries against the schemata of the data sources and with transforming the answer of these sub-queries to provide a unified result for the original query. Because the behavior of the data sources and their features are not as clearly specified as for a homogeneous data storage system, query processing cannot be separated into a query optimization and a query execution step. The optimizer would not yield good results for two reasons:

1. The optimizer may not have enough information to create a good execution strategy for the query, because most data source specific properties are abstracted by the data integration system and therefore unavailable.

2. The strategy that seems optimal at optimization time may be arbitrarily bad at execution time. In addition to the incompleteness of the information about the sources the optimizer also has to deal with uncertainty with respect to the behavior of the sources. This means that if the sources behave differently than expected a query might take much longer than expected.

A solution for this problem is the usage of adaptive query processing, which combines optimization and execution (A. Halevy et al., 2006).

The query processing process is different for local-as-view and global-as-view systems, because the mapping contains a different amount of knowledge that can be utilized for the transformation of the queries in the two approaches. Therefore the following two sections describe the differences and special features of query processing using a local-as-view or global-as-view approach respectively.

4.3.1 Query Processing in LAV

In a local-as-view based system the sources are represented by views over the global schema. Therefore in a LAV environment the term view based query processing is used. This means

that the result for a query written against the global schema has to be computed using the views rather than directly the data in the sources. Two different solution strategies for view based query processing can be distinguished (Lenzerini, 2002):

View Based Query Rewriting Using this approach the goal is to rewrite the query against the global schema so that it only refers to the views for the sources. This process is independent of the target source. The result query language of the rewriting process has to be fixed and is usually the same as the query language used for the original query. This means that only the target of the query changes from the global data schema to the views over the data sources. If there is no exact representation of the original query over the views than the general strategy is to use a result that is as close as possible to the original query. This is called maximally contained rewriting. View based query rewriting is essentially a two step process of a.) query rewriting and b.) evaluation of the result of a.) against the views of the data sources.

View Based Query Answering In this solution strategy the result for a query is calculated by evaluating its certain answers. Certain answers are those that can be logically derived from the information at the sources to be a result for a given query. This means that query answering methods will try to use all available information to compute a result for a query directly.

4.3.2 Query Processing in GAV

In systems that use the global-as-view approach query processing is usually done by query unfolding. Because the mapping already contains information how to query an element of the global schema at the sources, each element that is part of the query can simply be queried at the sources using the mapping information. This means that every element in the original query is replaced by the result of the corresponding query against the data sources. Then the unfolded query can be evaluated without any further need to contact the sources. Yet this is only true if a plain global data schema is used without integrity constraints such as foreign key constraints. In such cases query answering becomes more complex, as an simple unfolding strategy would miss all answers derived using the foreign key information.

4.4 Data Virtualization

Data virtualization is closely related to data integration as the following definition from Jagatheesan et al. (2005) shows:

> *"Data Virtualization is the concept of bringing together different heterogeneous data and storage resources into one or more logical views so that the distributed*

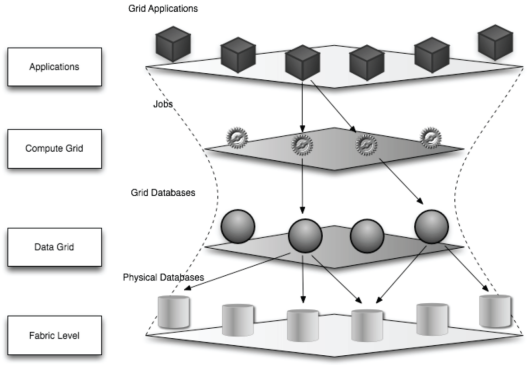

Figure 4.4: The hourglass model of a grid system (Fiore et al., 2009).

> *and replicated data appear as a single logical data source managed by a single data management system."*

The main difference between the two approaches is the technical background of the two research areas. Data virtualization has been developed in the field of grid computing and now cloud computing, whereas data integration has originated in database research. This leads to a slightly different point of view of the data integration or virtualization system. Data virtualization focuses more on the complete abstraction of the different data sources, their physical location and on the user's view on the system, whereas data integration focuses more on the technical and logical mechanisms needed for combining heterogeneous data sources. Weng et al. (2004) give a different definition of data virtualization, which emphasizes this small difference between data virtualization and integration:

> *"A Data Virtualization describes on abstract view of data. A Data Service implements the mechanism to access and process data through the Data Virtualization."*

Currently grid and cloud computing are the most important application areas of data virtualization, where it is only one part of a nearly complete decoupling of the logical view and the technical layout of a system. In these two application areas the abstraction of the physical location of the different parts of the system are especially important. Further more it has to be possible to replace parts of the systems or change their physical location without affecting applications based on the grid or cloud computing system. It even has to be possible to perform these changes at runtime.

Figure 4.4 shows the layout of a grid system, where data virtualization is done by a specific data grid layer. This data grid layer provides the compute grid with an integrated view of the data sources managed by the data grid. As it acts as a data management system the data grid has to fulfill other requirements besides virtualization, like robustness, transparency, efficiency and

security (Fiore et al., 2009). This makes the implementation of a data virtualization system in a grid or cloud computing environment a difficult task.

In many modern application areas data virtualization and data integration are both used to achieve a high level of data source abstraction. As applications are increasingly connected and therefore have to interact and to share data this trend is likely to continue in the future.

5 Research Issues and Approach

In this section the requirements of a semantic integration layer for a (software+) engineering framework are defined and based upon these requirements the research issues are formulated in section 5.1. In section 5.2 the research methods used to resolve the research issues are discussed and the two use cases used in this thesis are explained in detail.

(Software+) engineering is done by people with different technical backgrounds and different views on the overall system. The heterogeneity of the tasks they have to fulfill is reflected by the heterogeneity of the tools they use. These tools have not been built to integrate well with each other, but rather to solve a specific task, like creating a circuit diagram, planning the connectivity of a device or implementing its software in an effective and efficient way. The need for quality assurance and project management across tool borders as well as the introduction of iterative development paradigms resulted in the need for closer integration. As data is exchanged between tools in both directions of the production chain more often, manual coordination of the data transfer becomes more error prone and increasingly complex. This leads to a situation where a reliable change impact analysis is virtually impossible. Therefore, manual coordination is no longer a feasible option in a modern (software+) engineering environment. Existing technical integration solutions provide the infrastructure for data exchange and communication between tools, but without semantic integration the identification of common concepts and the transformation to a common data format from the tool specific data formats has to be done repetitively and in an uncoordinated way. A semantic integration solution for (software+) engineering has to solve this problem by providing a central unit that is responsible for managing the common data model. This common data model has no physical representation, because the data is provided by the tools in their specific format and has to be transformed to match the definition in the common data format. Therefore the term virtual common data model will be used throughout this thesis. The semantic integration solution has to provide infrastructure for storing and editing the virtual common data model. Transformations from tool specific data models to the virtual common data model have to be derived semi-automatically from the information provided in the data model. The semantic integration unit shall be the single point where changes to the common data model are done. In addition the life-cycle of the integrated tools has to be coordinated with the life-cycle of the semantic knowledge about these tools managed in the semantic integration layer. Therefore the semantic integration unit has to provide the possibility to add new tools, change the status of already connected tools or remove tools at runtime. A query interface is

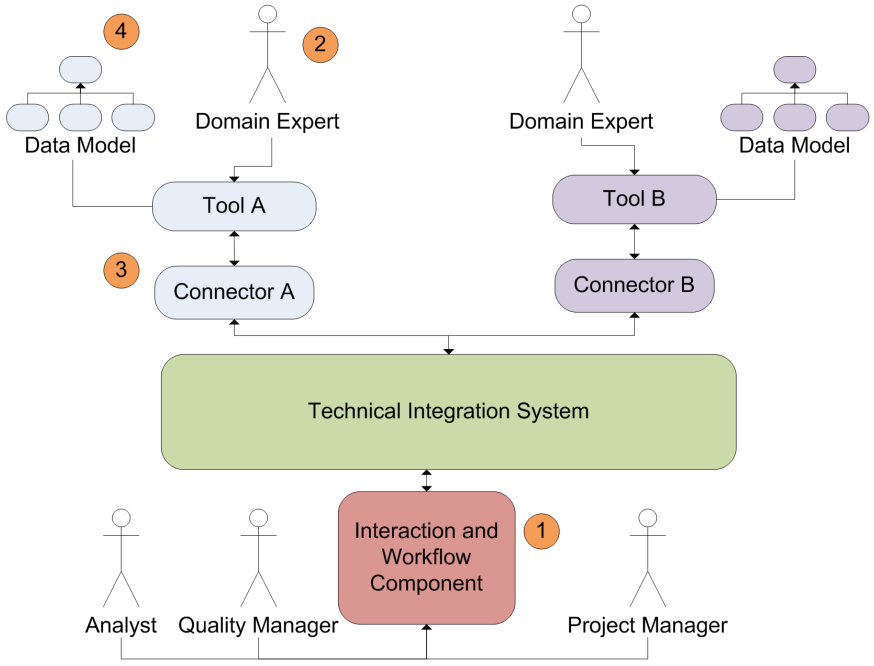

Figure 5.1: A technical integration solution for (software+) engineering.

an important part of the semantic integration infrastructure, which enables queries against the virtual common data model. Therefore the central unit, which is responsible for the management of the common data model, has to provide an interface, which supports data retrieval from the integrated tools.

Figure 5.1 shows a technical-only integration solution for the (software+) engineering domain. The different parts of this system are connected with the help of a technical integration framework (see section 2) with no specific support for semantic integration (see section 3). At four components in this figure the problems of a technical integration without semantic integration support can be identified. (1) Advanced applications like project management or quality management tasks are usually performed using the workflow and interaction interface of the integration system. But to perform these tasks it is often necessary to retrieve data stored at different integrated tools. In this technical-only integration solution there is no common data model, which can be queried to retrieve this data. Therefore it is necessary to design the advanced applications in a tool specific way to use the tool data. This makes tool exchange difficult and reduces the flexibility of the overall system. Furthermore this work cannot be done without domain knowledge, so domain experts (2) have to support project managers, analysts and quality managers to define and maintain their advanced applications. At the connector level (3) the infrastructure for data extraction based upon the local tool data model is also not part of a typical technical integration solution. Even if this infrastructure and a central unit for data extraction

would be implemented, without semantic integration the data models of the tools (4) and a common data model cannot be aligned. Therefore a semantic integration solution is necessary to provide effective and efficient support for advanced tasks like end-to-end tests or change impact analysis.

5.1 Research Issues

The main research issue of this thesis is to develop a semantic integration layer for a (software+) engineering framework. The integration solution has to be effective, efficient and robust. In addition it has to be operable by engineers with little experience in the field of semantic integration. Derived from the requirements above the research issues of this thesis can be stated more precisely:

RI-1 Feasibility of an EKB based solution for semantic integration: It has to be investigated whether an EKB based integration layer is a feasible way to provide semantic integration for a (software+) engineering environment. The typical requirements that can be derived from common use cases like data restriction enforcement or change impact analysis have to be identified and it has to be evaluated if the proposed approach is capable of fulfilling these requirements.

RI-2 Knowledge for semantic integration: Robustness in terms of error tolerance and stability is a critical requirement of an integration framework. Domain experts have to be confident about the results of automatic data processing or they will refuse to commit their work into the integration framework. A life-cycle for the management of semantic knowledge that is compatible with the life-cycle for the management of integrated tools has to be developed to provide the system with the necessary robustness. The life-cycle has to clearly define the behavior of the integrated tool with respect to the virtual common data model in every possible state.

RI-3 Derivation of transformation instructions: A knowledge model provides information about the tool specific data models, the common elements of these models and the dependencies between data elements from tool specific data models and common data elements. This knowledge has to be facilitated to semi-automatically produce transformations between the virtual common data model and the respective tool data models. Because manual transformation of the data is an error prone and time intensive task a feasible level of automation has to be reached to make the framework usable. In addition automatic generation of transformations makes the framework more flexible in terms of tool exchange and changes to the virtual common data model.

RI-4 Design of a query infrastructure for a virtual common data model: One of the main benefits of a working semantic integration solution is the possibility to query the virtual

common data model. The virtual common data model replaces a multitude of tool specific data models and the query interface provides a common interface for data retrieval from heterogeneous data sources. By providing the tool specific data in a standardized format and via a simple interface advanced quality assurance and project management tasks can be done with less effort. To facilitate these benefits an interface has to be defined which can be used to retrieve information from the different tools using the virtual common data model. The interface has to provide the information needed by a query engine that retrieves data using the virtual common data model and uses the transformation infrastructure provided by the semantic integration layer.

Furthermore, to be able to retrieve the information from the different integrated tools, an infrastructure has to be developed, which clearly defines the interface a connector has to implement to expose its data and which reduces the implementation effort for a connector as far as possible. As domain experts have to be able to create the connector implementation, the data retrieval interface has to be easy to understand and well documented. Nevertheless it needs to be flexible enough to support query optimization and other more advanced features in the future. Therefore, besides providing all data of a tool connector to the integration layer the infrastructure also has to provide the possibility to formulate restrictions reducing the result set before it is sent to the integration infrastructure.

5.2 Research Method

In this thesis three different approaches are used to address the research issues RI-1 to RI-4. To gain insight into the theoretical background of system integration from the technical, semantic and data driven point of view a literature research is done (see section 5.2.1). To show the feasibility of an EKB based semantic integration framework two real world use cases are implemented as prototypes (see section 5.2.2). These use cases are also used to gather empirical information about the efficiency and robustness and the usability of the proposed solution. Further more a comparison between the original OpenEngSB solution, which only provides technical integration to the proposed semantic integration framework with respect to the effort necessary for tool integration and exchange and available functionality will be performed (see section 5.2.3).

5.2.1 Literature Research

In this thesis an adapted version of a systematic literature review is used to gather information about the theoretical background of technical integration (see section 2) and semantic integration (see section 3). Then this information is summarized from a specifically data centred point of view in section 4. An effective and efficient semantic integration framework for (software+)

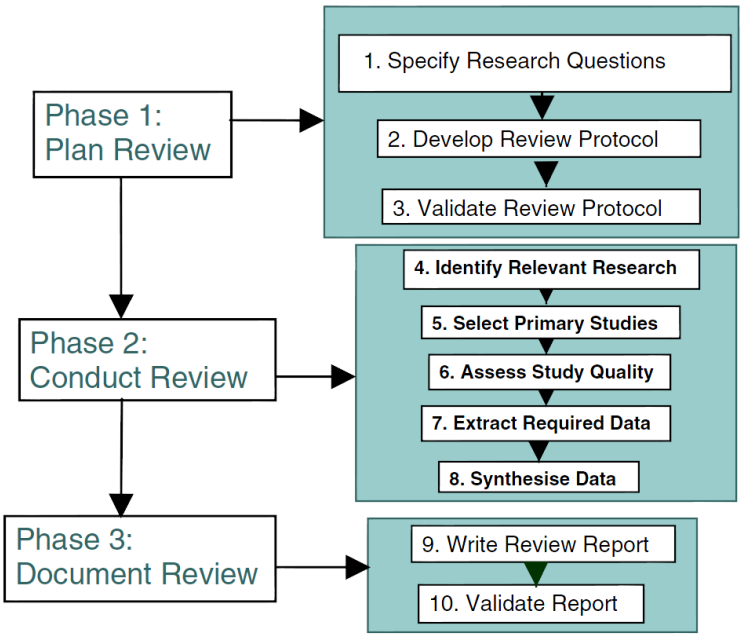

Figure 5.2: The different phases and steps of a systematic literature review (Brereton et al., 2007).

engineering can only be designed based upon these theoretical foundations. Furthermore different approaches that have been described in the literature have to be studied to avoid common pitfalls and to create a robust and scalable system.

Brereton et al. (2007) describe the systematic literature review process and define its main goal in the following way:

> *"Systematic literature reviews are primarily concerned with the problem of aggregating empirical evidence which may have been obtained using a variety of techniques, and in (potentially) widely differing contexts—-which is commonly the case for software engineering.*

They further state that the review is a secondary study, while those research papers reviewed are primary studies. While in other research areas, like medical engineering secondary studies play an important role, because of their summarizing and accumulating nature, they have rarely been used in the software engineering domain.

Figure 5.2 shows the different phases and steps, which have to be performed during a systematic literature review. In this thesis a similar process is used, but with less focus on documentation, as the results are documented in sections 2, 3 and 4. The three phases of a systematic literature review are:

Plan Review In this phase research questions and a review protocol are defined and reviewed. This provides the basis and the systematic framework for the review. Furthermore it is necessary to reduce the possibility of bias in the study. Included in the protocol are guidelines with respect to the process that will be used, the conditions that have to be applied when selecting the primary studies and quality issues. In this thesis only a very basic protocol is established, because of the limited size of the review and the fact that the review is conducted by a single person. Therefore the focus is less on the explanation of the process and mainly on the selection of primary studies.

Conduct Review The second phase of the review is concerned with the actual realisation of the review. In this step the relevant research is identified, primary papers are selected and interesting information gathered. This phase is conducted unaltered in this thesis. Primary papers are searched for using the ACM Digital Library[1] and IEEE Xplore[2]. Relevant search terms like "semantic integration" or "ontology + integration" are used. Then the results are filtered with respect to relevance for integration using as a first criteria the title and as a second criteria the abstract of the papers. Finally the content of the papers is reviewed and irrelevant papers are rejected. As the target of the systematic research is to identify the relevant sources for the theoretical background of technical and semantic integration, only papers which focus on these theoretical foundations are accepted.

Document Review In the final phase of a systematic literate review the gather information is summarized in a report, which is later validated using for example a peer-review mechanism. In this thesis the information is summarized in sections 2, 3 and 4. Therefore no additional report is generated.

The advantage of the usage of a structured approach based upon the process of a systematic literature review instead of doing an ad-hoc literature research is the reduction of bias and the possibility to gain a broader insight into the related work and the theoretical background of the research field. This is necessary to create an effective and efficient integration solution.

5.2.2 Use Case based Feasibility Evaluation

To investigate the feasibility of the proposed EKB based integration solution prototypes for two real world use cases based upon the requirements of industrial partners and common shortcomings in the (software+) engineering tool chain are developed and analyzed.

Definition of Quality Criteria Across Tool Data Models in Electrical Engineering This use case is an example of the integration of tools from different engineering domains including software development and electrical engineering. It shows the importance of semantic

[1] http://portal.acm.org/
[2] http://ieeexplore.ieee.org/

integration to provide a common data model across tool and engineering domain boundaries. The seamless integration of the different tools involved in this use case is the basis to provide advanced functionality like end-to-end tests. The power of the possibility to query the virtual common data model with the help of the EKB is shown by formulating a sample query against this data model that validates quality criteria for sensors across tool boundaries. For a more detailed description of this use case see section 6.1.

Change Impact Analysis for Requirement Changes Requirement traceability is a well known goal of software development processes, which makes the implicit interdependencies between requirements and other artifacts explicit. These interdependencies are semantic information, so the EKB based semantic integration layer can be used to implement requirement tracing. In this use case a change impact analysis is done for a changing requirement to find out which issues and developers are affected by the change request. This information can be used to mark all dependent artifacts for review and to contact all involved developers automatically. Furthermore it allows better estimates for the costs of the changes. A detailed description of this use case is given in section 6.2.

These two real world use cases are used to derive the requirements for a semantic integration framework in the (software+) engineering domain, which are described in detail in section 6. Based upon these requirements an EKB based integration solution is designed and implemented (see section 7). Then the integration platform is used to build two prototypes for the use cases, which are evaluated in section 8. Figure 5.3 gives an overview of the prototyping process conducted in this thesis.

It is important to notice that the notion of a prototype in software engineering is quite different to a prototype for example in the automotive industry. A software prototype is usually not interesting as a product or a blueprint for a product, but the process of the development of the prototype and the information which can be gathered during this process is important. This information is used to improve the quality of the final product, by providing early feedback about the effectiveness and efficiency of a system (Floyd, 1984). Furthermore in this thesis the prototypes are not meant to be blueprints for semantic integration solutions, but are designed to show the feasibility of the proposed approach and to validate the EKB concept in the (software+) engineering domain. So the most important goal of the prototypes in this thesis is to provide a practical demonstration of the feasibility of an EKB based semantic integration approach in the target domain. In addition the resulting prototypic systems are evaluated with respect to the following criteria to target the research issues defined in section 5.1:

Effectiveness This criterion tests whether the EKB based integration solution is feasible in the (software+) engineering domain and whether the proposed solution fulfills all requirements of a semantic integration framework.

Efficiency The semantic integration framework has to be efficient with respect to the effort needed for initial setup, maintenance and usage.

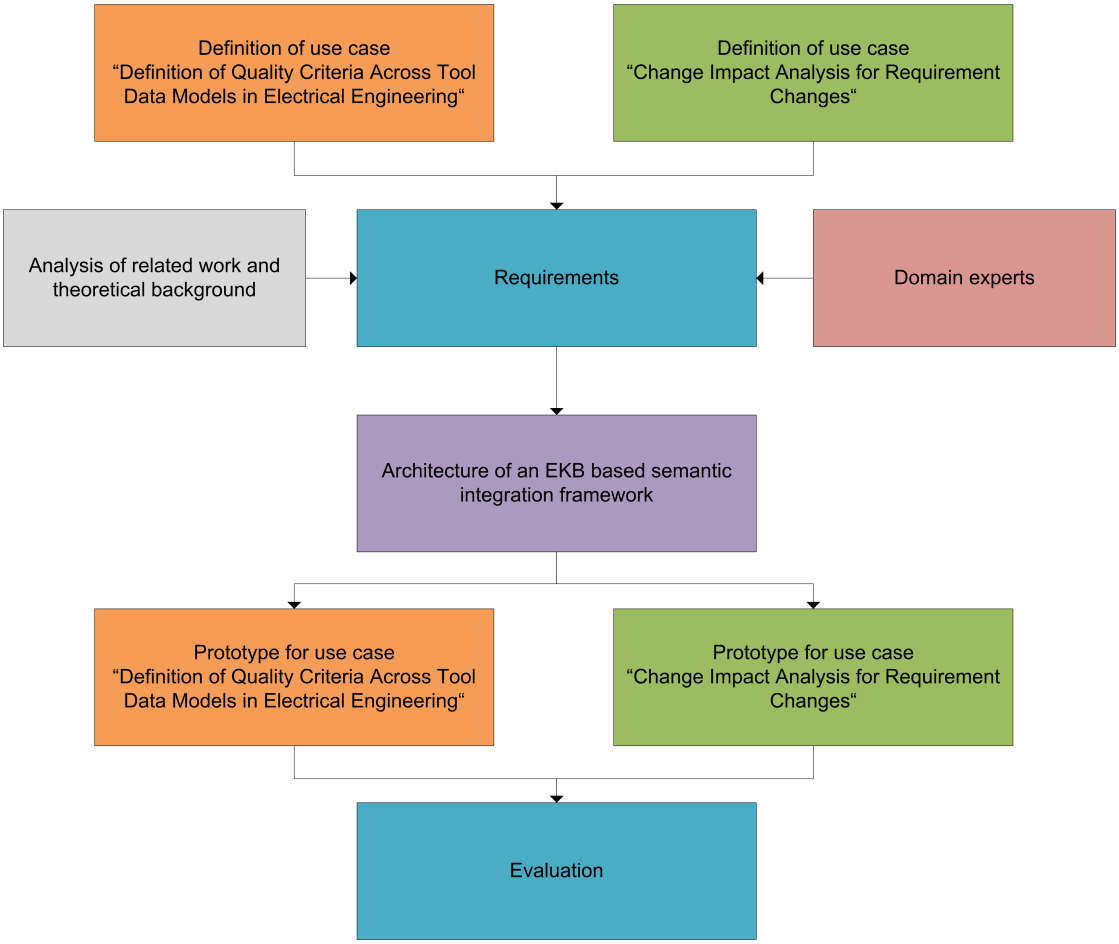

Figure 5.3: Overview of the prototyping process used in this thesis.

Usability The system has to be usable by domain experts with little or no knowledge about semantic integration without the need for time and cost intensive training. Advanced tasks based upon the semantic integration framework like end-to-end tests or project management tasks have to be easy to setup and perform. Therefore both the query interface of the semantic integration framework and the interface for the definition of the virtual common data model and the mappings to the tool data models have to be usable by domain experts. Furthermore the complexity of a typical tool connector is also part of the usability of the system, as these components have to be designed and implemented by domain experts.

Robustness The system has to be able to deal with unexpected input and defects in a well defined way. The of semantic knowledge about tool connectors and the tool connectors themselves have to guarantee that the system will not reach an undefined state.

The empirical evidence will be evaluated with respect to the guidelines for empirical research in the software engineering domain proposed by Kitchenham et al. (2002). Their guidelines cover the following six areas:

Experimental context There are three elements, which define the experimental context. The first is background information about the context in which an empirical study is performed. The second is the research hypotheses and how it was defined and the third is information about related work in this research domain. The goal of guidelines for the experimental context is to achieve a proper definition of the research objectives and an understandable description of the research as a whole.

Experimental design Guidelines in this area should improve the quality of empirical studies with respect to the products, resources and processes involved. Furthermore they should help to design the study in a way which ensures that the research objectives defined beforehand are achieved.

Conduct of the experiment and data collection It is necessary to clearly define how the outcome of the experiment is measured and how the experiment can be reproduced by other researchers. Furthermore every derivation from experimental plan has to be recorded.

Analysis Independent of the statistical approach used these guidelines try to ensure that the results gathered in the previous step are analyzed correctly. The analysis has to be conducted with respect to the study design and has to be powerful enough to cope with the requirements stated in the design. The analysis guidelines aim at the correct usage of statistical methods.

Presentation Besides understandability also the ability to reproduce the study is an important issue and has to be taken into account when the form of presentation of the study including its results is decided. To ensure that the study can be reproduced the presentation has to include amongst others design procedures, data collection procedures and analysis procedures.

Interpretation of the results The results of the empirical study have to be interpreted in a consistent and understandable way. No new information may be introduced at this stage of the study. In addition the guidelines try to support the researcher in the correct qualification of his results.

5.2.3 Comparison to Technical Integration

The proposed semantic integration framework for the (software+) engineering domain provides important features for the integrated system including the possibility to query a virtual common data model and data transformation based on semantic knowledge. But to use these features in an engineering environment additional setup is necessary. Further more the tool connectors

need to support the possibility of data extraction to act as data sources for the virtual common data model. To provide empirical information about the complexity and costs of these additional setup, maintenance and usage steps a comparison between the prototype developed in this thesis and the original system will be performed with respect to the following criteria:

Tool Integration The different steps necessary to integrate a new tool are identified and evaluated with respect to time consumption. This evaluation will be performed with the help of domain experts. In addition the technologies, which have to be understood to perform the tool integration will be compared.

Tool Exchange The process of tool exchange is compared with respect to time consumption, involved technologies and necessary steps to exchange one tool by another tool of the same tool domain.

Advanced Applications The effort needed to perform advanced applications like end-to-end tests or project management tasks, which use the features of the semantic integration framework to extract data from different engineering tools is evaluated and compared.

The goal of this comparison is to provide practitioners with information about the trade-offs of adding a semantic integration layer to a technical integration system. Furthermore it helps to show the feasibility of the proposed approach and gives information about the robustness and efficiency of an EKB based semantic integration solution.

6 Use Cases and Requirements of a Semantic Integration Solution for (Software+) Engineering

In this section a detailed description of the use cases "Definition of Quality Criteria Across Tool Data Models in Electrical Engineering" and "Change Impact Analysis for Requirement Changes" is given. Then requirements for a semantic integration framework in the (software+) engineering domain are derived from these descriptions. Further requirements are defined based upon the theoretical study in sections 2, 3 and 4 and upon the research issues (see section 5.1). The solution architecture described in section 7 is designed to address the requirements defined in this section.

6.1 Definition of Quality Criteria Across Tool Data Models in Electrical Engineering

Large scale (software+) engineering projects can only be realized if different engineering disciplines cooperate in an effective and efficient way. Some process models, like the waterfall model try to reduce this need for cooperation by defining a strict order in which the different parts of the system are developed. Thereby, the contributions of each respective engineering discipline is simply regarded as input for the next step in the engineering process, which can be conducted by another discipline. In such a model the need for cooperation is low and can be reduced to a clear definition of the expected results of each engineering step. Yet in practice the engineering process is not linear, but includes iterations, change requests and defect reports. Therefore parts of the overall system have to be updated at any time of the engineering process to repair defects or to implement new requirements (Moser, Waltersdorfer, et al., 2010).

Although in many projects rigorous quality assurance mechanisms are in place for the initial development, there is often little support for quality management when changes are performed at a later point in time. Poor system quality and a high failure rate are the results of change related bugs. To increase the stability of the overall system it is necessary to check specific restrictions

across tool and domain borders automatically each time a change is performed. Most involved tools support consistency tests for their specific view of the overall system, but no end-to-end tests across tool borders. Manual consistency checks are impractical, because of the complexity and the huge amounts of data that has to be analyzed. In addition, data from different domains has to be taken into account, which makes this task even more difficult for a single person. An integration system has to provide the possibility to conduct these consistency checks across tool and domain borders and to enforce restrictions in every part of the system.

In a (software+) engineering environment for electrical engineering tools, like ePlan[1] from various domains specialized for different purposes, like creating circuit diagrams or defining device connectivity have to be integrated with tools for software development, like Eclipse[2] or Maven[3]. Technical and semantic integration is necessary, because common concepts are used throughout the various domains in different ways. One example of such a common concept in electrical engineering are sensors. The concept of a sensor is reflected by a physical component in the technical device plan, by a connection of a special type in the connectivity plan and by a variable in the software that runs on the device. Although in all three cases the same sensor is referenced, it is modeled in different ways by each respective tool. The reason for this heterogeneity is the different view on the sensor concept based on the needs and properties of the different involved engineering domains. If an advanced application like definition of quality criteria across tool data models has to be designed, it is necessary to define the semantics of the concepts of the different tools. Using this semantic information it is possible to link the different representations of the sensor concept and to execute consistency checks across tool and domain boundaries (Biffl, 2010).

In this use case data from different tools is extracted using the virtual common data model. Then the different representations of the same sensor are compared with respect to compatibility. The sensor is stored in different formats at different tools. To find these different instances of the same sensor it is necessary to use the information in the virtual common data model to transform the tool specific sensor instances after they are loaded into a sensor concept. This transformation is done with the help of semantic information stored in the virtual common data model, which also contains the definition of the common sensor concept. After the transformation critical fields like sensor type or the unit of measurement can be compared. If there are inconsistencies or conflicts with restrictions for the sensor concept, like for example that a sensor that can physically only distinguish between two states, is used to measure a temperature (see figure 6.1), the responsible engineers are contacted with a change request. Furthermore, other restrictions can be checked, like for example if every software variable is linked to at least three sensors of the same type in safety critical parts of the system.

Figure 6.2 gives an overview about the setup in this use case. The definition of the quality criteria is done against the interaction and workflow interface of the integration system (1). This

[1] http://www.eplanusa.com/products/eplan-electric-p8.html
[2] http://www.eclipse.org/
[3] http://maven.apache.org/

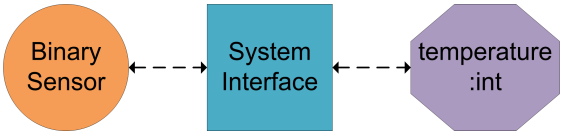

Figure 6.1: A binary sensor with an erroneous link to an integer software variable.

component uses the data retrieval component (2) to get the necessary data with respect to the virtual common data model (3). The data is collected from the tools using transformations between the tool data models (4) and the virtual common data model. This happens with the help of an data extraction interface (5) the respective tool connectors have to implement. Finally the quality criteria are checked as part of a predefined workflow. Two quality criteria will be formulated for sensors:

1. Each software variable has to be connected to at least three physical sensors. This is often a requirement in safety critical systems, as the malfunction of one sensor does not lead to wrong results if a majority vote is used to calculate the overall result from the values measured by the respective sensors.

2. The type and measurement unit of sensor and variable have to be consistent. Many tools provide the possibility to add small amounts of semantic information to the data. A software tool might for example provide the possibility to add a description to each variable, which states the planned usage of this variable. Likewise in a plan for the device the purpose of a sensor is described. Many development teams define a specific way how these descriptions have to be done. With the help of semantic integration this information stored in a tool specific way can be extracted and facilitated to test whether the planned usage of a sensor is consistent with the planned usage of the variable linked to the sensor. If for example a sensor is described to measure the pressure in Millibar, then the variable should also be designed to capture the pressure in Millibar. If the variable is designed to capture the pressure in Bar the system is likely to deliver wrong results.

These two quality criteria are hard to enforce manually, as the necessary information is distributed between different tools of different domains. Therefore in this use case an automatic enforcement of quality criteria based upon the proposed semantic integration framework is developed. This process has to be started automatically after each change to a relevant part of the system, like for example when a sensor type is changed, to deliver instant feedback about the validity of the new system configuration.

In this use case only two connected tools are used to reduce the complexity of the system and to be able to describe the features of the proposed semantic integration framework more clearly. In real world engineering projects the setup can be much more complex, a potentially large number of different tools has to be integrated into the system.

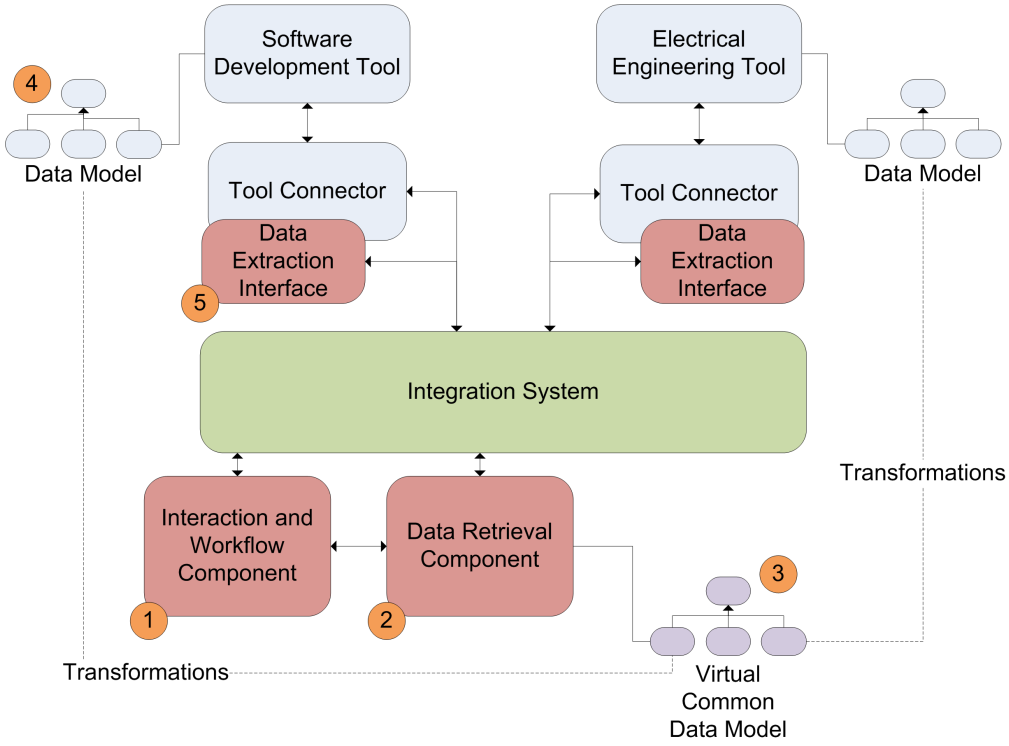

Figure 6.2: System overview for the "Definition of Quality Criteria Across Tool Data Models in Electrical Engineering" use case.

The main goal of including the "Definition of Quality Criteria Across Tool Data Models in Electrical Engineering" use case in this thesis is to show the importance of a semantic integration solution for advanced applications, where tools from different domains have to be coordinated and where it is necessary to tap into heterogeneous data sources.

6.1.1 Requirements

The following requirements can be derived from the use case description for "Definition of Quality Criteria Across Tool Data Models in Electrical Engineering":

Requirement 1 A suitable modeling language or API has to be developed to make the definition of the virtual common data model and its relation to the tool specific data models possible. This modeling language has to be powerful enough to capture the information necessary to derive transformations from tool data into the virtual common data model.

Requirement 1.1 The modeling language has to support the definition of the virtual common data model.

Requirement 1.2 The definition of the relation between tool data models and the common data model has to be supported by the modeling language. These relations have to be modeled in a formal way to allow semi-automatic transformation generation. It has to be easy to define default relations, which are handled by the system with automatically generated transformations, like type conversions or simple text manipulation. In addition, a possibility to define manual transformation instructions for more complex relations has to be provided. Especially the notion of a common concept that is represented in different ways by different tools has to be easy to model, as it is the most common case of relevant semantic information in a heterogeneous engineering environment.

Requirement 2 To manage a virtual common data model it is necessary to create an infrastructure to load and update the model elements using a suitable data format (see requirement 1). This infrastructure component is responsible for synchronizing the semantic knowledge with the actual status of the integrated tools.

Requirement 2.1 A management infrastructure for the virtual common data model has to be developed.

Requirement 2.2 The semantic integration solution has to support the management of different versions of the virtual common data model.

Requirement 2.3 The life-cycle of the tools has to be reflected by the life-cycle of the semantic knowledge about the tool data model. If the tool is removed, exchanged or updated the semantic information stored in the semantic integration framework has to be updated in a coherent way.

Requirement 3 Transformations between tool models and the common concepts in the virtual common data model have to be derived from the information stored in the management component for the virtual common data model. The language elements defined in requirement 1.2 have to be used to decide whether an automatic generation of the transformation instructions is possible or not.

Requirement 4 A data extraction infrastructure has to be developed including an interface tool connectors can implement to expose their data. This infrastructure has to provide the possibility to extract all data of a specific type from the connected tool, but should also be designed to support query like restrictions on the expected result set in the future.

Requirement 4.1 The extraction of all entities of a specific type has to be supported by the data extraction interface for tool connectors.

Requirement 4.2 The data extraction infrastructure has to choose the correct tool connectors to retrieve a specific global concept. Then it performs the necessary transformations from tool specific data into the common data model and finally collects and organizes the results.

Requirement 4.3 The architecture of the data extraction infrastructure and the connector interface has to be designed with extendability in mind. It has to support restrictions on the result set of data extraction calls in the future.

Requirement 5 An interface for data extraction against the virtual common data model has to be defined, which is usable for project managers, quality engineers and domain experts, who use the workflow and interaction interface of the integration system.

Requirement 6 To provide the system with the necessary flexibility it is necessary to determine at runtime which tools are currently capable of delivering a specific global concept or a sub-concept of the global concept. With this information the semantic integration framework can decide which tool connectors have to be contacted to retrieve all elements of a specific global concept. A mechanism for tool connectors has to be developed to allow them to publish the concepts, which are stored by the tool they connect to the integration system.

6.2 Change Impact Analysis for Requirement Changes

As engineering processes become more agile and customers are more closely integrated into the development process the focus of requirement engineering changes. The precise definition of the requirements up-front becomes less important than the consistent management of the requirements. Part of this consistent management is the traceability from a requirement to dependent engineering artifacts and vice versa. There are four different types of traces (Jarke, 1998):

Forward from Requirements Traces from requirements to dependent engineering artifacts. These traces are necessary to evaluate which changes have to be performed if a requirement is updated, so they are used for change impact analysis.

Backward to Requirements Traces from engineering artifacts to requirements. These are used to make sure that for every part of the developed system a requirement exists and no superfluous work is done.

Forward to Requirements Traces from high level project descriptions or design documents to derived requirements. If stakeholders change high level system goals these traces can be used to determine all affected requirements and finally all affected engineering artifacts.

Backward from Requirements Traces from requirements to high level project descriptions and design documents. These traces are important to evaluate the quality of requirements, as the business needs leading to the respective requirements can be identified.

It is possible to derive backward to requirements traces from forward from requirements traces and vice versa if the tracing system has full information about all trace links. This is done by

automatically establishing links in both directions if a link is created in one direction. Therefore many projects define trace links only in one direction, relying on the possibility to derive links in the other direction if they are needed. Likewise forward to requirements can be derived from backward from requirements and vice versa. This use cases especially focuses on the backward to and forward from requirements traces. The advantages of managing the implicit dependencies between requirements and engineering artifacts explicitly are on the one hand better decision support in every phase of development, because crucial information can be found directly and on the other hand easier change management, as affected artifacts can be identified with the help of trace links. Some process quality standards like CMMI[4] demand the establishment of a structured requirement management and tracing approach.

Despite all these advantages only few engineering projects use requirement tracing. The reason is the huge effort needed to capture and manage requirements and trace links to engineering artifacts. Although there are specialized tools for requirement tracing their major drawback is the missing integration with other development tools. Furthermore, when requirement tracing is done with a specialized tool the trace links have to be established in a dedicated work step. This is undesirable, because it has to be either done after the developer finished his work or parallel to the work. The first approach is impractical, because the effort for trace generation is considerably higher if tracing is done in an extra step after the development task is finished. The second approach reduces the time consumption for trace generation, but might break the workflow of a developer as he has to constantly switch tools during development. Integrated requirement tracing is an alternative, but is hard to accomplish in a heterogeneous environment like (software+) engineering. Therefore requirement tracing is often done ad-hoc, which makes it impossible to measure the costs or benefits precisely (Heindl & Biffl, 2005).

In the “Change Impact Analysis for Requirement Changes” use case the integrated environment with semantic integration support proposed in this thesis is used to extract requirement traces from available information. In many software engineering projects tracing information is captured in a structured and well defined, but informal way, which can be understood by humans, but which is not easily usable for automatic processing. By defining the semantics of tool data models including the tracing information it is possible to automate parts of a change impact analysis.

Figure 6.3 gives an overview about this use case. If a stakeholder files a change request (1) the requirement engineer has to identify the affected requirements using a requirement management tool (2). Then he can conduct a change impact analysis with the help of the interaction and workflow component (3). The change impact analysis is done for the affected requirements using the virtual common data model (4). As a first step data is retrieved from the requirement management tool (5) and by using the semantic information in the virtual common data model all dependent issues are identified. So traces of the type “forward from requirements” are used to navigate from the requirements to dependent issues. For this purpose the links need not actually

[4] http://www.sei.cmu.edu/cmmi/

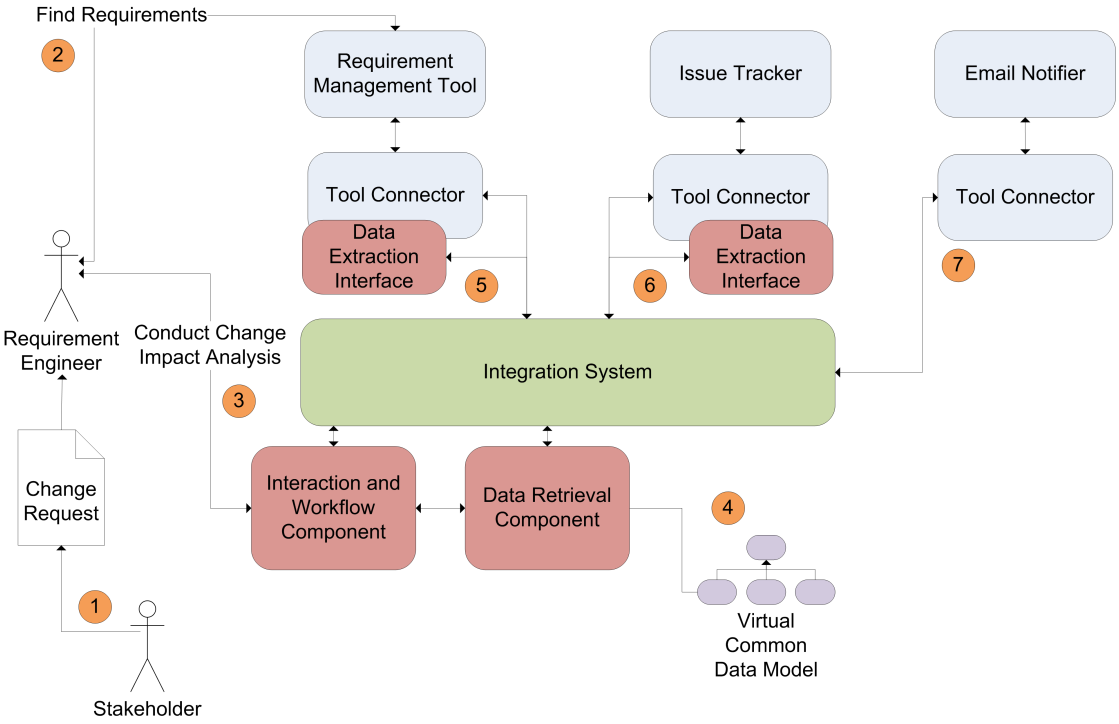

Figure 6.3: The "Change Impact Analysis for Requirement Changes" use case.

be available in this direction, as the integration system can use traces of the type "backward to requirements" to derive the trace links in the other direction. In this use case the link between the trace link between issue and requirement is defined by a reference to the addressed requirement in the issue description. Such informal or semi-formal forms of trace information are found in most engineering projects and can only be used if semantic information about the meaning of these links is available. The affected issues are retrieved from the integrated issue tracker (6) and separated into three groups:

Open Issues These issues have to be marked for review. They are likely to change as the underlying requirement has changed. After the issue is redefined according to the new requirements and its estimates updated the change impact can be evaluated.

In Progress Issues, which are currently in progress are the most critical group. All development team members, who are assigned to this issue have to be notified about the requirement change. They have to update the issue's definition and estimates, before they can continue with their work. Further more they need to contact the requirement engineer undertaking the change impact analysis and provide their opinion about the amount of work necessary to perform the changes. The integration of the affected team members into the change impact analysis is critical for its quality and correctness.

Resolved and Closed All issues, which are already finished have to be reopened and marked for review, as the underlying requirement has changed. The developers, which are assigned to these issues are notified that an already finished piece of work has to be reevaluated. Furthermore, they have to provide their estimates for the amount of work necessary to perform the requested changes.

To notify the affected team members again semantic information has to be used. The issues contain information about the assignee, which has to be extracted and mapped to a member of the development team. Then another mapping between the developer and its contact information has to be used to determine the recipient of the notification. An integrated notification component, like an email connector (7) sends the notifications. Finally the requirement engineer, which started the change impact analysis process is presented with a report containing all the information, which could be gathered automatically and a list of all persons, who have to provide manual feedback.

This semi-automatic form of change impact analysis for requirement changes provides a high quality result without the need to manually search and evaluate the trace links between requirements, issues and developers. Furthermore, it provides the possibility to inform all affected team members during the process and to automatically perform necessary project management steps in the issue tracker, like reopening already finished issues.

The goal of introducing the "Change Impact Analysis for Requirement Changes" use case into this thesis is to show that the proposed EKB based semantic integration solution for (software+) engineering can be facilitated to perform a difficult and complex software engineering task, like requirement tracing. Furthermore, it should show how the semantic integration framework supports the development and project management process by automating tedious manual tasks. Advanced applications like change impact analysis can be built based upon the semantic integration framework, which help to generate better estimates. This use case is designed to underline the importance of efficient cooperation between different team members to perform complex tasks, like the reevaluation of parts of the system after a requirement change and the crucial role of effective tool support during this process.

6.2.1 Requirements

The following additional requirements for a semantic integration framework in the (software+) engineering domain can be derived from this use case.

Requirement 7 The definition of informal semantic meta information has to be supported by the modeling language used for the virtual common data model (see requirement 1). Such meta information includes for example the trace links from requirements to issues in this use case. The semantic integration framework has to provide the infrastructure to manage and to de-reference these links.

Requirement 7.1 The modeling language has to support the definition of traces by semantic meta information in a simple format, which is both understandable by humans and automatically processable.

Requirement 7.2 The semantic integration framework has to support the management and usage of semantic meta information for traces between different engineering artifacts. More specifically this means that it has to support the de-referencing of the trace links and the localization and retrieval of dependent artifacts. This has to be done using a key or an identifier included in the trace link. The data extraction interface for tool connectors (see requirement 4) has to support the retrieval of a single element of a specific type by key or identifier for this purpose.

Requirement 7.3 The data extraction interface (see requirement 5) has to be extended to allow project managers, quality engineers and domain experts to use the tracing information for advanced applications, like change impact analysis.

6.3 Additional Requirements

Based on the theoretical foundations in sections 2, 3 and 4 and on the research issues in section 5.1 the following additional requirements for a semantic integration solution for the (software+) engineering domain can be defined. In this section the focus is more on the non-functional requirements for a semantic integration solution. Functional requirements have already been derived from the two use cases described in this thesis in sections 6.1 and 6.2.

Requirement 8 The semantic integration framework has to be efficient with respect to the amount of work necessary to model semantic information. The benefits of semantic integration like better support for advanced quality or project management applications has to significantly exceed its costs.

Requirement 9 Robustness is critical for the acceptance of the integration solution by practitioners. The semantic integration solution has to be robust with respect to failure management and stability. More specifically errors in the semantic integration framework should not bring the whole integration system down. In addition the system has to remain in a well defined state at any point in time. This is addressed by the synchronization of the tool life-cycle with the tool knowledge life-cycle (see requirement 2).

Requirement 10 The main concern of this requirement is the usability of the integration solution. The integration infrastructure has to provide usable interfaces for data retrieval and management of the virtual common data model. Furthermore the interfaces, which have to be implemented by tool connectors to facilitate the semantic integration solution have to be easy to understand and well documented. The framework should reduce the effort needed to write tool connectors as far as possible, as this is a repetitive task done by domain experts with little or no knowledge about semantic integration.

7 An EKB based Semantic Integration Framework - Concept and Architecture

In this section the architecture of the proposed semantic integration framework for (software+) engineering is described. At first section 7.1 gives an overview about the design and explains how the EKB concept is used to implement a semantic integration framework. In section 7.2 the architecture and design of the core components of the proposed solution are explained and evaluated with respect to the requirements stated in section 6 and research issues defined in section 5.1. Finally the integration into a technical integration framework is discussed using the example of the prototype implementation for the Open Engineering Service Bus (see section 7.3).

7.1 Concept and Architecture

Based on the theoretical foundations of semantic integration (see section 3) and especially the Engineering Knowledge Base approach, which is explained in detail in section 3.3, a semantic integration framework for (software+) engineering is designed and implemented. This framework has to realize the three main features of an EKB (Moser, 2010):

1. data integration using mappings between different engineering concepts
2. transformations between different engineering concepts utilizing these mappings
3. advanced applications building upon these foundations

Figure 7.1 shows the externally visible parts of the EKB based semantic integration framework. The EKB component is part of the OpenEngSB and provides three different public interfaces for tree different usage scenarios:

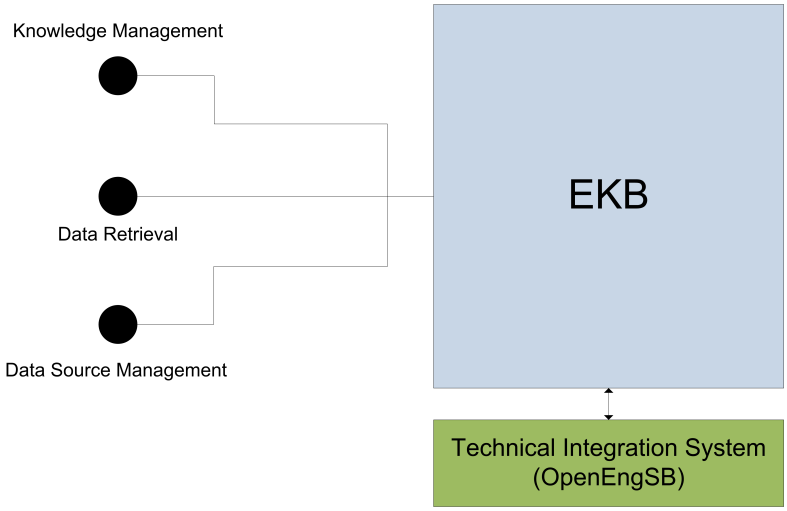

Figure 7.1: Overview about the public interfaces of the proposed EKB based semantic integration solution.

Knowledge Management This is the interface where the semantic information stored in the EKB can be managed. It addresses the first main feature of an EKB as it provides the possibility to define the mappings between different engineering concepts. Furthermore the second main feature, which is performed by the EKB internally is also controlled by the semantic information managed with the help of this interface.

Data Source Management The different integrated engineering tools act as data sources for the semantic integration framework. Because a typical (software+) engineering environment is dynamic and tools join and leave the system during development, the tools have to be managed dynamically by the EKB. Therefore this interface provides the functionality for dynamic data source management.

Data Retrieval This interface addresses the third main feature of an EKB. It provides a usable interface for other components of the integration system, like workflow and interaction components. Project managers and quality engineers can use the data exposed through this interface to build advanced applications, like end-to-end tests across tool boundaries.

Figure 7.2 shows the internal architecture of the proposed solution in more detail. The core components shown in this figure are described in detail in section 7.2. The EKB is designed to be part of the technical integration system (1), which is in this case the Open Engineering Service Bus. So all external interfaces of the EKB, which are shown in the balls and sockets notation of UML collaboration diagrams, are available for every integrated tool in the technical integration system.

The virtual common data model and its relation to the tool data models is analyzed by an external component (2). The design rationale behind this solution is explained in detail in section 7.2.1.

The model analyzer component updates the semantic information managed by the EKB using its Knowledge Management Interface (3). These parts of the system, which are responsible for the management of the virtual common data model, realize the first feature of an EKB by providing a possibility to capture the mappings between different engineering concepts (see section 7.2.2).

Parallel to this model management responsibilities is the management of the data sources, which are in the (software+) engineering domain the tools. To expose their data to the integrated system the tool connectors use the data source management interface (4) to publish the concepts they support (see section 7.2.3). Every time either the virtual common data model is updated (3) or a data source changes its state (4) the life-cycle manager (5) is notified and checks the internal state of the EKB for consistency. For more information about this process see section 7.2.4. Constant consistency checks provide the system with the necessary robustness and address research issue RI-2. The components described until now form the management section of the EKB, which stores the semantic knowledge of the integrated system and keeps track of available data sources and their properties.

The technical integration system's interaction and workflow component can communicate with the EKB through the data retrieval interface (6). This is the interface for advanced applications defined in research issue RI-4 and as third main feature of an EKB. The data retrieval interface has two different use cases. It provides the possibility to extract information about the virtual common data model and the dependencies between the different concepts. Furthermore, it makes it possible to query tool data using the virtual common data model. The internal data retrieval infrastructure then coordinates a query for data by performing the following steps:

- To be able to choose the respective data sources it is necessary to query the virtual data model for the respective concepts and sub-concepts (7).
- Then information about the data sources is gathered using the data source management component of the EKB (8). This information is used to transform the query for the respective tool connectors. For more information about this process see section 7.2.5.
- A specialized component attached to every tool connector is used to extract the data from the tool (9).
- The data is transformed to match the virtual common data model if necessary. This process is performed by a specific component, which uses the information from the data model to generate transformation instructions semi-automatically (10). It is explained in detail in section 7.2.6. The transformation infrastructure addresses research issue RI-3 and realizes the second main feature of an EKB. The additional information for the creation of all transformation instructions has to be provided by the user attached to the virtual common data model through the knowledge management interface (3).
- The results are gathered and returned to the interaction and workflow component of the integration system.

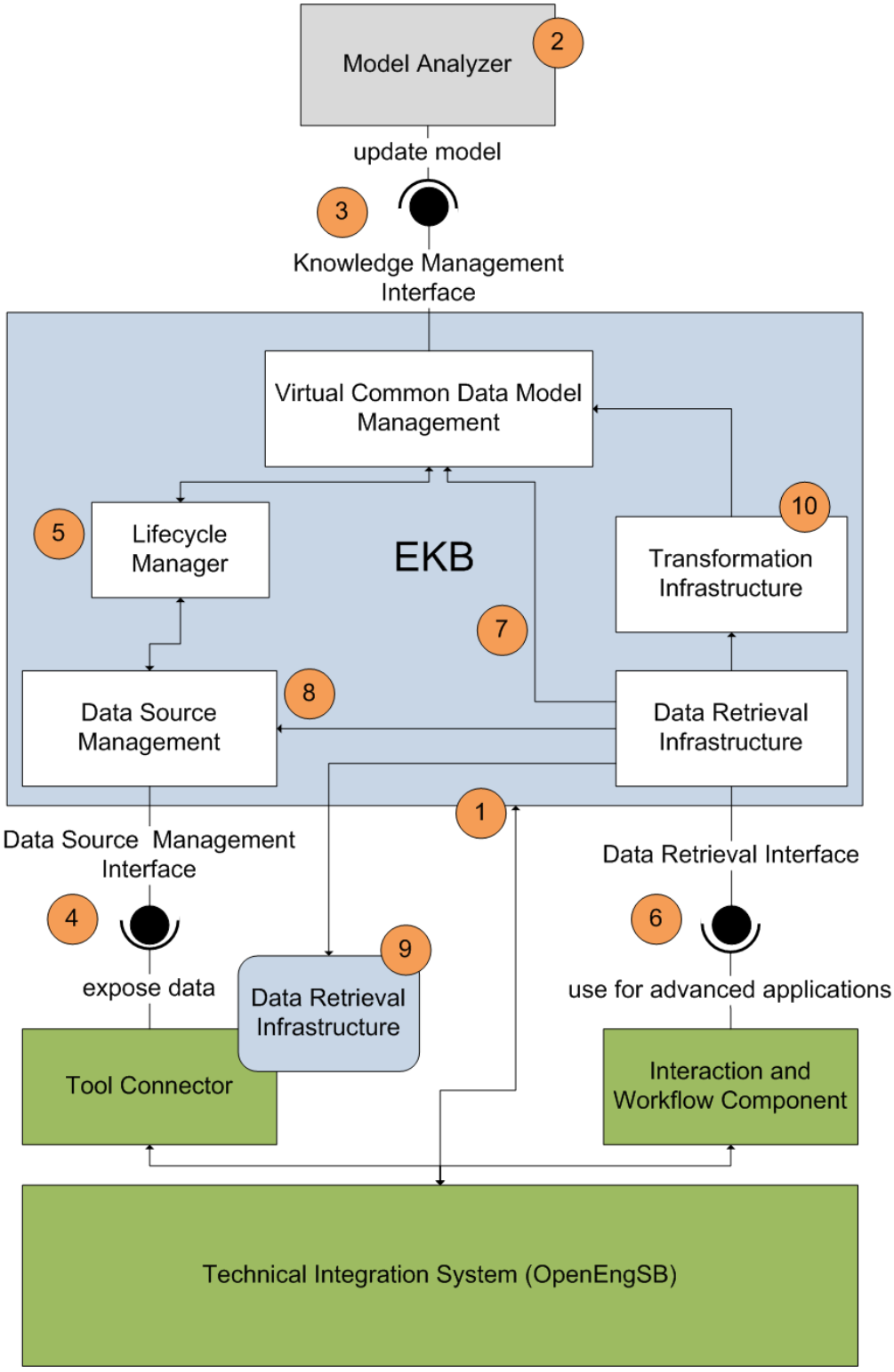

Figure 7.2: Architecture of the proposed EKB based semantic integration solution.

To implement an effective and efficient EKB based semantic integration solution for (software+) engineering it is necessary to consider the drawbacks of semantic integration in general and especially of an EKB described in section 3.3. One of the most important disadvantages is the increased effort for configuration, maintenance and usage of the integration system. Therefore the architecture of the proposed solution specifically aims at reducing this effort. This is done by defining simple, well documented interfaces the tool connectors have to implement. Furthermore, to make the process of managing the virtual common data model easy and flexible it is split into two parts. The first part is an external component responsible for analyzing the virtual common data model and to generate API calls against the second part of the management infrastructure, the virtual common data model management component, which is a part of the EKB. The external component can be modified to accept different input formats, or can be replaced by a system that accepts user input through a graphical interface and generates the API calls directly. Section 7.2.1 describes the different possible setups in detail.

Data source management is an important part of every data integration system and has to be done in a way suitable for the target domain. In (software+) engineering tools have to be easy to exchange and may be only available in a specific phase of the development. Therefore the data sources have to be managed dynamically and independent of the semantic knowledge. In the proposed architecture the two aspects of data integration, model or schema management and data source management can be performed independently. The life-cycle manager is responsible for keeping the EKB in a consistent state, when either the data model or the data sources change. Decoupling these two aspects of data integration also reduces the effort for model management, as the model is updated automatically when the data sources change (see section 7.2.4).

The data retrieval infrastructure is designed for extendability with respect to the introduction of a query engine. The data retrieval interface can be used as basis for a query engine, as it provides all necessary information. Furthermore, the part of the infrastructure located at the tool connectors can be extended to support restrictions and other important features necessary for efficient data retrieval. With a query engine as interface to the data provided by the EKB the usability can be further increased.

7.2 Core Components

The proposed EKB based semantic integration framework consists of several more or less independent parts, which address different requirements. This section gives a detailed description of the different components shown in the architectural overview (see figure 7.2). Usage scenarios and different possible configurations are explained. In addition the design rationale behind the different parts of the system and alternatives are discussed.

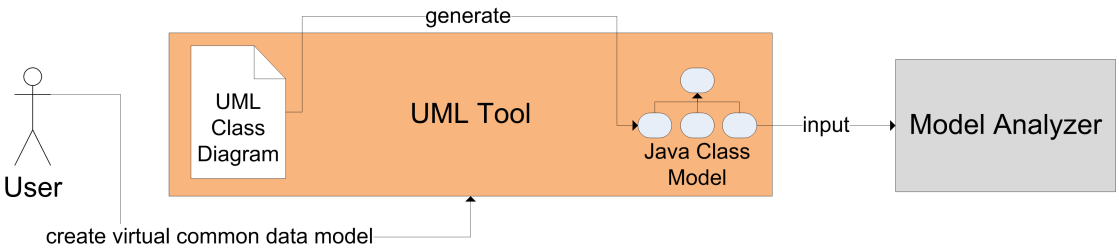

Figure 7.3: Usage of a UML tool and a Java class model for the definition of the virtual common data model.

7.2.1 Model Analyzer

The model analyzer is an external component that is responsible for loading the virtual common data model and its dependencies to tool models. The reason why the model analyzer component is not situated directly in the EKB is that it has to be easy to exchange this part of the system. As a result it is possible to support different ways to define and maintain the virtual common data model. Amongst others the following scenarios are possible:

Java Class Model In the prototypic implementation of a semantic integration framework a simple Java class model is used as input for the model analyzer, which stores the information in the EKB using the knowledge management interface. This setup is used, because of its simplicity and the fact that Java class models are easy to generate, maintain and modify. There are different possibilities how the class model can be generated. It can be written directly or generated from any other form, like for example from an UML class diagram. This means that the tool support, which is available for generating Java class models and UML class diagrams can be used to support the user during the process of creating the virtual common data model. Figure 7.3 shows a possible workflow for the generation of the virtual common data model.

DSL Another possibility is the usage of a domain specific language (DSL). A domain specific language has a clearly identified target domain, whereas general purpose programming languages can be used for any domain. A DSL defines the basic concepts, abstractions and possible relations in a domain. DSLs are closely related to model driven engineering, where metamodels are defined for a specific domain. Models, based on these metamodels correspond to programs written using a DSL (Kurtev et al., 2006). Therefore the advantages and disadvantages of model-driven semantic integration described in section 3.2.2 also hold for semantic integration with the help of a DSL. As it is closely related to software engineering, the (software+) engineering domain is a good application area for model-driven or DSL based semantic integration solutions. Figure 7.4 shows a possible setup for a DSL based solution. The user creates the virtual common data model with the help of a graphical editor, which stores the model using a DSL. The model analyzer is

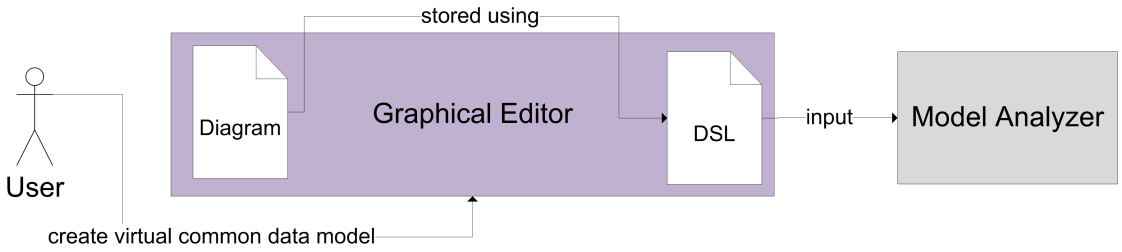

Figure 7.4: DSL based definition of the virtual common data model.

adapted to accept the DSL and stores the model information in the EKB using the knowledge management interface. Alternatively the user can also directly use the DSL to create the virtual common data model. In addition it is possible to transform the DSL model into a general purpose programming language, like Java. In this scenario the model analyzer component can still operate on a Java class model independent from the DSL.

Direct Another option is the implementation of a tool, which directly uses the knowledge management interface of the EKB. In this case the intermediate steps and transformations between different formats are not necessary. Therefore the user can interact with the EKB directly and prompt feedback about the validity of model changes can be given. The drawback of this solution is the additional development effort for the creation or adaption of a suitable modeling tool. Figure 7.5 gives an overview about this scenario. The model analyzer component is not necessary as it is integrated into the modeling tool.

The model analyzer addresses requirement 1 (see section 6.1.1) and requirement 7.1 (see section 6.2.1). The prototypic implementation, which uses a Java class model approach for the definition of the virtual common data model fulfills these requirements, because a general purpose programming language can be used to model anything including the virtual common data model, dependencies to tool data models and trace links using semantic meta information. Furthermore, the knowledge management interface specifically supports the management of semantic information of this type. This interface and the structure of the semantic information is explained in detail in section 7.2.2.

7.2.2 Virtual Common Data Model Management

The virtual common data model is one of the most important parts of the EKB. It contains semantic knowledge about common engineering concepts and their relation to tool specific concepts. Furthermore it provides the foundation for advanced applications, which use the semantic information stored in the common data model. To manage the virtual common data model it is necessary to define the notion of a concept and its relation to other concepts. Figure 7.6 shows the definition of the concept class used in the prototype.

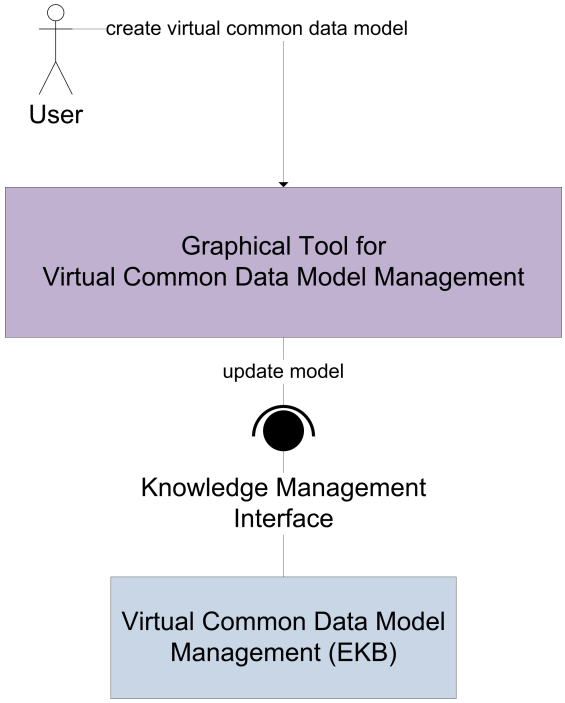

Figure 7.5: Definition of the virtual common data model using a tool, which is directly connected to the EKB.

The virtual common data model contains information of two types:

Schematic The schematic information is captured with the help of a suitable data model. If the integration system is implemented in an object oriented languages a class model can be used. The prototype for example uses a Java class model to capture the schematic information for the virtual common data model.

Semantic The semantic information is meta information, which has to be attached to the data model used to represent the schematic information. The proposed EKB based prototype manages this information with the help of specific concept classes, which are directly attached to the respective data classes. To provide a usable mechanism for attaching the semantic meta information to Java classes the model analyzer (see section 7.2.1) of the prototype uses annotations on the Java class model to generate the respective concept classes.

Both types of information have to be facilitated to provide a consistent view on the virtual common data model and to achieve semantic integration.

A tree structure is used to capture the hierarchical structure of concepts, like for example when different tools represent the same common concept in different ways. Therefore a concept can

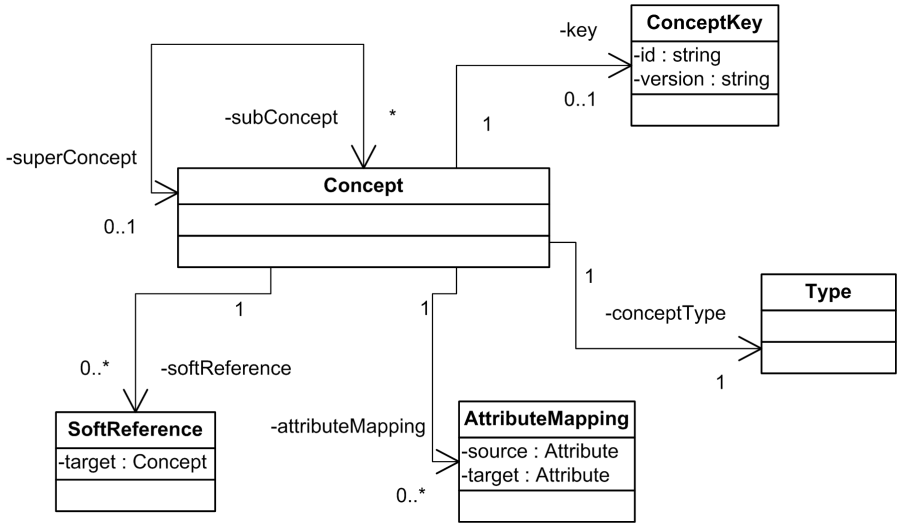

Figure 7.6: Formalization of a concept and its relations.

define a super-concept. The relation to this super-concept is modeled with the help of attribute mappings. Attribute mappings define which attribute of a tool concept maps to which attribute of a common concept. Furthermore transformation information can be attached to attribute mappings if an automatic transformation is not possible. The attribute mapping and transformation process is explained in detail in section 7.2.6.

Relations between different concepts, which are represented by semantic meta information, like for example trace links (see section 6.2) are modeled as soft references. The goal of soft references is to provide a possibility to establish links between two different concepts based on informal semantic information. In contrast to hard references soft references are not directly included in the data model in the form of a specific reference attribute. This means that they are part of the semantic meta information and not included in the common data schema. To establish a soft reference the following steps are necessary:

- The target concept has to define a key attribute. This key attribute is used to identify the target instance of the reference. The data extraction infrastructure also uses the key attributes to load specific data items (see section 7.2.5).

- The source concept has to define the target of the soft reference and the attribute, which contains the soft reference. In addition the mechanism for the extraction of the actual reference from the content of this attribute has to be defined. The extraction process is adaptable to make it possible to handle different forms of semantic meta information. In the prototype a regular expressions based solution is implemented, which makes it possible to extract the reference from an arbitrary textual source. A regular expression

based soft reference definition contains the target concept and the regular expression for the key extraction.

- To establish a concrete reference the content of the respective attribute of the source concept has to include the reference, which has to be extractable using the mechanism explained in the previous step.

Figure 7.7 gives an overview about the different steps necessary for the definition of a soft reference between an issue and a requirement. First the requirement concept has to define an identifier (1) and the requirement type, which is the attached data type of the requirement concept has to define a key attribute (2). In this example the requirement number attribute is used as key. Then the soft reference between the issue concept and the requirement concept can be defined. In the example shown in figure 7.7 this is done with the help of a regular expression based soft reference. Besides the identifier for the target concept (3), also the regular expression for the extraction of the reference key has to be defined (4). Note that when only the identifier of the target concept is defined and the version is omitted then the version of the source concept is also used for the target concept. If an issue has a description which contains a reference (5) then the semantic integration infrastructure can de-reference this link and load the respective requirement (6).

In this example the linking process is relatively simple, but it can become very complex and difficult to define and manage trace links. Different default soft reference solutions for common types of links defined by semantic meta data have to be defined in the future and integrated into the proposed solution to make it possible for the user to define the relationship between different engineering concepts.

The virtual common data model management component has to be able to deal with different versions of semantic tool information. Therefore, to be able to distinguish different versions of the same concept a binary concept key is used for the identification of concepts. The first part of the key is a textual identifier, while the second part contains version information.

This core component of the proposed EKB based semantic integration solution realizes requirement 2.1 by providing a management infrastructure for the virtual common data model. The formalization of a concept and its definition is derived from requirements 1.1 and 1.2. The version information in the concept key is introduced to resolve requirement 2.2 (see section 6.1.1). Soft references address requirement 7.2 (see section 6.2.1) by providing a possibility to use semantic meta information to create references between two different concepts. In addition requirement 2.3 is linked to virtual common data model management, as changes of the model trigger consistency checks by the life-cycle manager, which is described in detail in section 7.2.4.

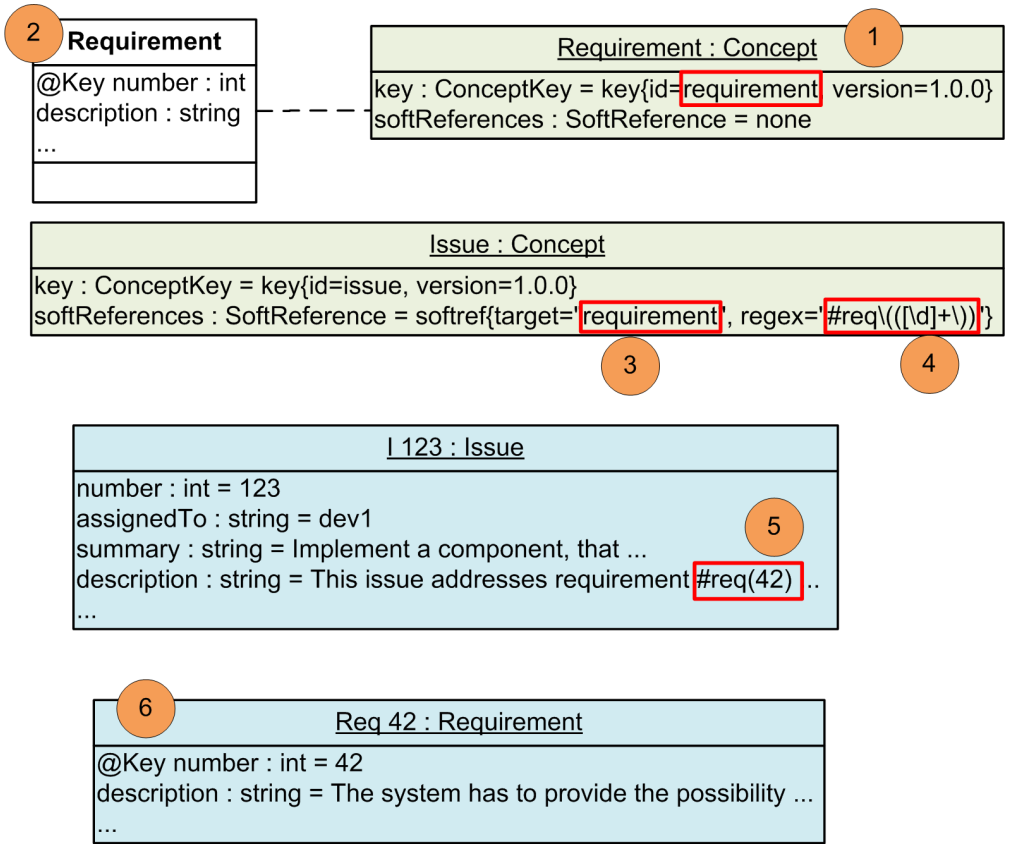

Figure 7.7: Overview about the soft reference definition process.

7.2.3 Data Source Management

The proposed semantic integration solution is capable of using tools connected to the technical integration system as data sources. The management of this sources is an important part of data integration. In a (software+) engineering environment tools have to be easily exchangeable. Therefore, flexibility is a critical feature of an effective and efficient integration solution. To provide this flexibility it has to be easy for data sources to join and leave the system and to expose the data they want to share. The actual data retrieval process is handled by the data retrieval infrastructure (see section 7.2.5).

The data source management component of the proposed EKB based semantic integration framework keeps track of all available data sources and provides an interface for status changes. The data source manager distinguishes two different states of data sources:

Active The data source is available and can be queried for data. If a tool is in this state it has to be connected to the integration system and has to respond to data retrieval calls.

Inactive The data source is either temporarily or permanently not available. To reduce the complexity of the system no distinction between these two possibilities is made. The system can be extended to distinguish between permanently and temporarily unavailable sources and can allow temporarily unavailable sources to be queried, waiting for the tool to become active again and respond to the data retrieval call. The advantage of such a distinction is that tools, which are not available all of the time, like external tools or shared resources can be supported more easily. One major drawback of temporary unavailable data source support is that the response time can vary and is hard to predict. This can lead to unexpected behavior of workflows and other advanced applications using these data sources.

Tools, which are offline but provide their data by opening their private data storage are treated as **active** by the system, as their data can be retrieved. The actual retrieval process is handled by the data retrieval stub of the tool connector. As it is concerned with data integration the semantic integration framework does not need to distinguish between tools that only provide their data or also their functionality through an active connection to the integration system.

When data sources become active they have to publish the concepts they can provide. Version information is included during this publishing process to make it possible for sources to determine which versions of a concept they support. There are two different possibilities how a source can provide a specific concept:

Direct The concept is directly supported by the data source. This means the the model type attached to this specific concept is used by the tool and can be retrieved using the data retrieval interface of the tool connector. No conversion of the result is necessary.

Sub-concept A sub-concept of the concept is supported. In this case the concept is supported by transforming the data of the type attached to the sub-concept to the type attached to the concept. This can include more then one transformation, as transitive transformation is possible, when for example a sub-sub-concept is supported by the tool. The mapping information stored in the virtual common data model is used to generate the transformation instructions, which are carried out by the transformation infrastructure (see section 7.2.6).

For advanced applications there is no difference between these two different types, as the data retrieval and transformation process is completely hidden by the semantic integration infrastructure.

Tools, which act as data sources in the (software+) engineering domain are integrated into a technical integration system. Therefore it is possible to define and enforce a common interface for all sources. This interface has to be implemented by the tool connectors and hides the heterogeneity of the data sources. Hence data management and retrieval can focus on the semantic differences between the data of the respective sources without having to deal with technical integration issues.

The data source management component addresses requirement 4 and 6 (see section 6.1.1) and resolves these issues in combination with the data retrieval infrastructure described in section 7.2.5. The data source management is related to the data extraction infrastructure defined in requirement 4, as it keeps track of all possible data sources. In addition it realizes requirement 6, as it provides the possibility to evaluate at runtime which tools are currently capable of delivering a specific concept. The main goal of the data source management infrastructure is to make it possible for tool connectors to publish the concepts they support. Furthermore, because changes to the data sources lead to consistency checks performed by the life-cycle manager (see section 7.2.4), it is also linked to requirement 2.3 as described in section 6.1.1.

7.2.4 Life-cycle Manager

In order to provide the semantic integration framework with the necessary robustness it is important to define the possible states of tools and semantic tool data and to keep them synchronized. When a tool joins or leaves the integration system it has to be checked whether the semantic information for the tool is available and consistent. In addition changes to the virtual common data model can lead to inconsistencies with already connected tools. So the semantic integration system has to make sure that tools, which act as data sources and semantic knowledge about these tools are managed in a coordinated way. Defining the different states a tool and the semantic knowledge of a tool can reach is equivalent to defining its life-cycle. Therefore the life-cycle manager component was included into the proposed EKB based semantic integration solution to synchronize data sources and the virtual common data model.

The possible states of data sources are described in section 7.2.3. To make it possible to find out which concepts are currently supported by the active data sources, the semantic information managed by the EKB also changes its state. Figures 7.8 and 7.9 give an overview about the possible states of data sources and concepts. The life-cycle of concepts reflects the life-cycle of data sources with the difference that concepts may be supported by more then one source in which case they are in the multiple support state. If they are supported by only one active data source they are in state **supported** and otherwise in state **unsupported**.

The life-cycle manager is activated when either the virtual common data model or the data sources are changed. There are two possible state transitions of a data source (see figure 7.8, which trigger the consistency check:

Active → Inactive If an active data source becomes inactive then all concepts, which are only provided by this data source have to be checked. They can either move from multiple support to simple support or from simple support to the state **unsupported**, depending on whether other data sources also support this concept. Note that if a data source supports a concept it automatically also supports each super-concept of this concept, because the super-concepts can be derived using the transformation infrastructure (see section 7.2.6).

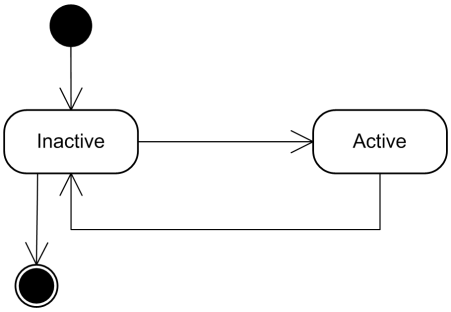

Figure 7.8: State diagram for data sources.

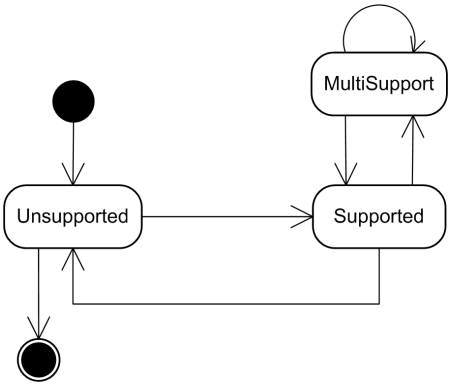

Figure 7.9: State diagram for concepts.

Therefore the consistency check has to include all super-concepts of the concepts supported by the respective data source.

Inactive → Active If a data source becomes active the concepts it supports and all its super-concepts change their state from **unsupported** to **supported** or from **supported** to **multi-Support** depending on the respective source state. If they are already supported by more than on source then they stay in the multiple support state.

Besides clearly defining the possible states of data sources and semantic information managed by the EKB the synchronization of the two life-cycles also makes it possible to take a snapshot of the currently supported concepts. Therefore besides the complete virtual common data model, which contains all concepts that can possibly be supported by the integration system, also the actual virtual common data model, which is currently supported is available. Project managers and quality engineers can use this information to plan or schedule advanced applications, which facilitate semantic information. In addition this information is also valuable for data retrieval, as it defines which concepts are currently available for data retrieval. Furthermore the current state of the system is easier to evaluate and incorrect configurations or errors in the virtual common data model can be found more easily.

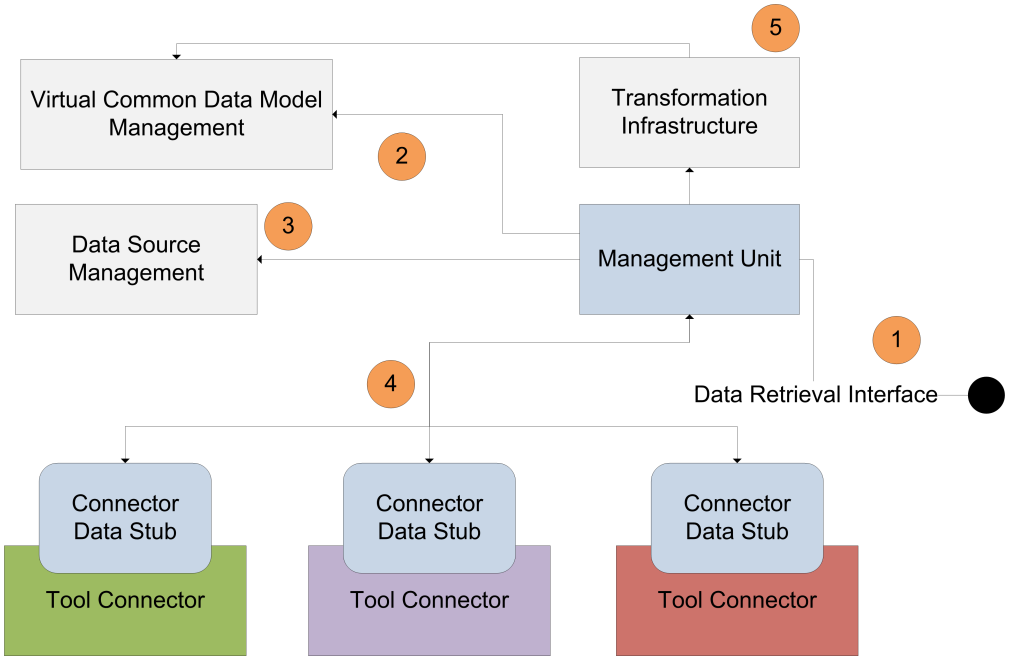

Figure 7.10: The data retrieval infrastructure of the proposed semantic integration solution.

The life-cycle manager component addresses research issue RI-2 as defined in section 5.1. It defines the possible states of data sources and semantic knowledge stored in the virtual common data model and makes sure that the system is in a consistent state. As a result the life-cycle manager contributes to the stability and robustness of the system. It resolves requirement 2.3 and is linked to requirement 6 (see section 6.1.1), which is resolved in cooperation with the data source management component (see section 7.2.3).

7.2.5 Data Retrieval Infrastructure

While the virtual common data model management component is responsible for providing a common data schema, which can be used to formulate queries against heterogeneous data sources using semantic information, the actual process of data retrieval is conducted by the data retrieval infrastructure. This component is responsible for coordinating the whole process of data extraction, including the transformation of the original request for the different tool connectors as well as the gathering and unification of results. Furthermore it has to provide a usable interface for data retrieval, which can be used by domain experts as well as project managers or quality engineers. This interface is especially important as it is the main connection between the EKB based semantic integration framework and the technical integration system.

To achieve a clean separation of concerns the data retrieval infrastructure consists of three parts (see figure 7.10):

Data Retrieval Interface The external interface of the EKB, which can be used to retrieve data against the schema defined by the virtual common data model. The data retrieval interface has to hide the complexity of the retrieval process and the fact that heterogeneous data sources are used as far as possible. In addition it has to fulfill high usability requirements, as it is used by domain experts, project mangers and quality engineers to conduct advanced applications like end-to-end tests across multiple engineering tools and domains. In the future the usability of this component has to be further increased by the introduction of a query engine, which makes it possible to formulate SQL like queries. The query engine increases the usability of the semantic integration framework as it provides a well known data retrieval interface. To ease the integration of a query engine the data retrieval interface is designed for extendability. This means that all information needed to build a query engine based upon the data retrieval interface is available. In addition the data retrieval interface supports the usage of soft references (see section 7.2.2).

Management Unit The management unit is the central component of the data retrieval infrastructure. Its main responsibility is the coordination of the entire data retrieval process. A data retrieval inquiry is processed by this component by performing the following steps shown in figure 7.10. A retrieval call at the data retrieval interface (1) has to be analyzed and transformed into calls to the respective tool connectors. First the involved concepts are retrieved from the virtual common data model (2). Then the data source management component is contacted to evaluate which connectors need to be queried (3). For each connector the data retrieval call is adapted. If a connector supports for example only a sub-concept of the concept that has to be retrieved the retrieval call is formulated for the sub-concept rather than for the original concept. This means that tool connectors only need to provide data in the format they are familiar with and do not need to know any details about the virtual common data model and common concepts. Then the data retrieval stubs at the connectors are called by the management unit (4). When the results are returned by the connectors they are gathered by the management unit and if necessary transformed into the original concept with the help of the transformation infrastructure (5). Finally the results are collected and unified and returned to the caller.

Connector Stub The connector stub is responsible for the data retrieval at the connector. Currently its main purpose is to provide a common interface to all integrated tools for the management unit of the data retrieval infrastructure. The data retrieval stub has to coordinate between the tool and the semantic integration system. The prototypic implementation of the proposed semantic integration framework supports two different types of data retrieval. The first is the retrieval of all instances of a specific concept and the second is the retrieval of a specific instance of a given concept. The latter is realized with the help of a key attribute, which is also used to define the target of soft references (see section 7.2.2). In addition to this minimal functionality the data retrieval infrastructure is designed for extendability. By attaching a data retrieval stub to each tool connector the code a tool connector has to implement to support data retrieval is minimal. All advanced features like

restriction management can be handled by the stub. Therefore the introduction of new functionality is relatively easy, because instead of changing the code of all connectors only the code of the stub has to be changed. If a tool connector needs to provide a high performance data retrieval solution it can intervene at any point of the data retrieval process and replace the default implementation with an optimized solution. It is also possible to provide different default implementations for the most common persistence solutions, like relational databases or XML based persistence to load the data directly from the data source without the detour through the tool connector. Nevertheless the default implementation provides full semantic integration support with the least possible implementation effort at the connector, which makes the system more flexible in terms of tool exchange, maintenance and evolution. One possibility for evolution of the data retrieval infrastructure is the support of advanced querying mechanisms, like the possibility to formulate restrictions on the expected result set.

From a data driven point of view the proposed EKB based semantic integration framework can be seen as a both-as-view or global-local-as-view approach (see section 4). As a transformation based system with no restrictions on the direction or form of transformations the system can be interpreted as global-as-view and as local-as-view. Yet the functionality described in this section is only available if at least the global-as-view information is present, which means in this case that the tool specific concepts can be transformed into the common concepts. This is a concept to super-concept transformation. The common concepts together constitute the global data schema, which is used by advanced applications. The EKB manages the common data schema together with the mappings to the local schema elements in the virtual common data model management component (see section 7.2.2), whereas data sources are managed separately. This distribution makes the system more flexible in terms of tool exchange and provides good support for the sharing of semantic information.

As the global-as-view information needs to be present, the transformation of data retrieval calls can be performed using an adapted unfolding strategy (see section 4.3.2). The adaption is that the transformation infrastructure is used to transform the results and that meta information about the data sources is facilitated to translate the retrieval call for each respective data source.

The data retrieval infrastructure addresses research issue RI-4 as defined in section 5.1. Furthermore it resolves requirement 4, 5 and 6 (see section 6.1.1) together with the data source management component (see section 7.2.3) by providing a possibility to extract data from heterogeneous data sources using the virtual common data model. It supports the retrieval of all instances of a given concept (requirement 4.1) as well as of a specific element to resolve requirement 7.2 defined in section 6.2.1. The data retrieval interface was specifically designed with usability in mind (requirement 5) and supports besides simple data retrieval tasks also the usage of soft references to use semantic meta information to connect different concepts (requirement 7.3). The tool connector data stub design makes it possible to improve the data retrieval infrastructure in the future with restriction processing and other advanced data retrieval features, without having to change the tool connector code (requirement 4.3).

7.2.6 Transformation Infrastructure

The transformation infrastructure component of the proposed EKB based semantic integration solution is responsible for the execution of transformations between different related engineering concepts. The relationship between the concepts is defined in terms of the virtual common data model and this information is managed by the virtual common data model management component (see section 7.2.2). Based upon the mapping information in the common data model transformation instructions are generated. For simple transformations this is done automatically. Complex transformation have to be defined manually. Therefore the transformation instructions are generated semi-automatically.

The basic principle, which is used for the concept transformation is an attribute mapping approach. An attribute mapping is defined between two concepts, which are in a concept – super-concept relation. Generally mappings in both directions are possible, but to provide the functionality described in this section and to realize the two use cases discussed in this thesis and most other advanced applications at least the mappings from sub-concept to super-concept are necessary. A concept can be transformed to any other concept which is higher in its concept hierarchy by repeating the transformation process as often as necessary. Therefore each data source supporting a specific concept automatically also supports all super-concepts of this concept.

An attribute mapping consists of three parts:

Source Attribute The source attribute of the mapping defines the type and the value which is used as input to the transformation.

Target Attribute The target attribute of the mapping defines the type of the transformation result.

Transformation Instruction The transformation instruction is optional. If it is not present the system tries to derive the transformation automatically. The different supported automatic transformations are described later in this section. If a transformation instruction is present it is always used to perform the attribute transformation, even if it is possible to derive an transformation instruction for the given source and target attribute automatically.

The prototype of the proposed semantic integration framework is currently capable of deriving the following types of transformations automatically:

Identity Transformation The simplest form of transformation, which simply copies the source attribute to the target attribute. This transformation is used if the source type and the target type are equal.

Simple Type Transformation Simple type transformations are performed if simple types, like primitives in Java are used as attribute type of both source and target attribute. Type transformation is performed using the standard platform mechanism.

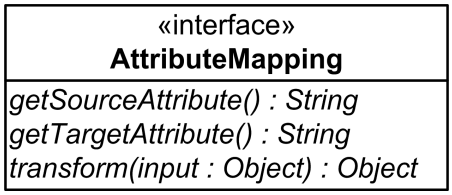

Figure 7.11: The interface custom transformation instructions have to implement.

String Parsing If the source type is a simple text, transformations to numeric values and other simple types are automatically derived. Note that string parsing only works if the standard platform mechanism is capable of processing the input value.

It is possible to create and use customized transformations by implementing the AttributeMapping interface shown in figure 7.11. Customized transformations can be shared throughout the system, if they may be used repeatedly, like for example text manipulation transformations, like text splitting or joining. Custom transformations can also perform external service calls to integrate a full fledged external transformation system. In addition it is possible to chain different transformations, which makes it possible to define an intermediate format and only transform to and from this format. This reduces the amount of necessary transformations and makes it easier to reuse already existing attribute mapping implementations. Generally the framework supports complex transformation, which are performed via multiple intermediate steps, but as the use cases described in this thesis show, in most cases either the automatic transformations provided by the system or very simple custom transformations are sufficient.

In section 7.2.1 the model analyzer component of the EKB is described as well as the the different possible configurations, which can be used to generate and manage the virtual common data model. If a setup with a graphical editor, a model or a DSL based approach is used then typical transformations for the target domain have to be supported. This makes it possible to define the attribute mappings together with the custom transformation instructions and reduces the effort necessary for the manual transformation definition. As a result the usability of the semantic integration system is increased and the configuration effort is reduced.

The transformation infrastructure addresses requirement 3 defined in section 6.1.1 together with the virtual common data model management component (see section 7.2.2) and the model analyzer component (see section 7.2.1). It derives transformation instructions from the information stored in the virtual common data model and provides a possibility to add manual transformation instructions for customization and for complex relations between different concepts. Furthermore it addresses research issue RI-3 described in section 5.1. The automatic derivation of simple transformations reduces the effort for the setup of the semantic integration system, thus making the system more usable and flexible in terms of tool exchange and changes to the virtual common data model.

7.2.7 Core Component Overview

In this section an overview about the different core components and the requirements they address is given. Table 7.1 shows which requirements (see section 6) are implemented by which components. The additional requirements defined in section 6.3 are not listed in the table or specifically mentioned in the description of the core components, because they are non-functional and concern the whole integration system. They have been taken into account during the whole design and implementation process of the proposed semantic integration solution and its prototype.

7.3 Integration into the Open Engineering Service Bus

The proposed semantic integration solution is based upon a technical integration system, which provides a common message format as well as the technical infrastructure necessary to integrate different tools (see section 2). Basically it can cooperate with any technical integration solution, but it is specifically designed to work together with an Enterprise Service Bus based solution. To work in other environments, like a pure service oriented technical integration system some adaptions to the concept and architecture described in this section may be necessary. The event driven and service oriented character as well as the focus on integration of an ESB reduces the design and implementation effort for the semantic integration framework. The Open Engineering Service Bus (see section 2.5) is an ESB based technical integration system for the (software+) engineering domain. The prototype of the semantic integration framework proposed in this thesis is developed for this system, as it is specifically designed for the target domain.

The following list describes the most important concepts and components of the OpenEngSB according to Pieber (2010) and how they are affected by the introduction of a semantic integration framework.

Core Components Core components provide functionality for all members of the integration system. They are specifically developed for the OpenEngSB, because no ready to use external solutions exist, which fulfill all requirements. The EKB based semantic integration framework is represented as a core component inside the OpenEngSB, as it provides data retrieval functionality for heterogeneous data sources to other components of the OpenEngSB. Furthermore it needs to be easy to use within the OpenEngSB, because other core components like the workflow engine need to interact with the EKB in advanced application scenarios like end-to-end tests across tool borders.

Tool Domains Tool domains in the OpenEngSB formalize the common functionality and concepts of a specific group of tools. The issue domain for example defines a common interface for various different issue tracker tools. Domains can provide functionality for all tools of a domain, like event triggering mechanisms, process support or a common data

	Requirement	Model Analyzer	Virtual Common Data Model Management	Data Source Management	Life-cycle Manager	Data Retrieval Infrastructure	Transformation Infrastructure
1	Semantic Knowledge Modeling	x	x				
2	Virtual Common Data Model Management		x	x	x		
3	Transformation Generation						x
4	Tool Data Retrieval Stub			x		x	
5	Data Retrieval Interface					x	
6	Data Source Management		x	x	x	x	
7	Soft Reference Modeling	x	x			x	

Table 7.1: Overview about the realization of requirements by core components.

pool. The semantic integration framework builds upon the semantic information already contained in the domains. Typically the virtual common data model will contain the concepts defined by the domain rather than the actual tool concepts, as the tools already have to support the domain concepts to be integrated into the OpenEngSB. This reduces the effort for the creation and maintenance of the virtual common data model and makes tool exchange easier.

Domain- and Client Tool Connectors Tool connectors act as interface between the OpenEngSB and any tool integrated into the system. Every tool connector is part of a tool domain and therefore has to implement the domain interface and support the domain concepts. The EKB based semantic integration solution extends the connectors with a data retrieval infrastructure, which makes it possible for tool connectors to publish which concepts of the virtual common data model they support. Furthermore the data retrieval stub at the connectors contains most of the functionality needed for a tool connector to act as a data source and as a result reduces the implementation effort necessary to create the actual connector.

Figure 7.12 shows the architecture of the Open Engineering Service Bus and how the EKB based semantic integration solution is incorporated into this system. Furthermore it depicts the data integration stub at the connectors and their connection to the EKB, which is used for data retrieval. It shows that the integration of the EKB does not lead to major changes of the underlying technical integration system. The EKB is added as a core component. The only part of the system, which needs to be adapted are the tool connectors, which need to be extended with data retrieval stubs. These stubs are directly connected to the EKB using the messaging mechanisms provided by the service bus of the technical integration system. This direct connection is used for data retrieval and data source management. Other features of the technical integration system, like service calls to integrated tools are not affected by the semantic integration component.

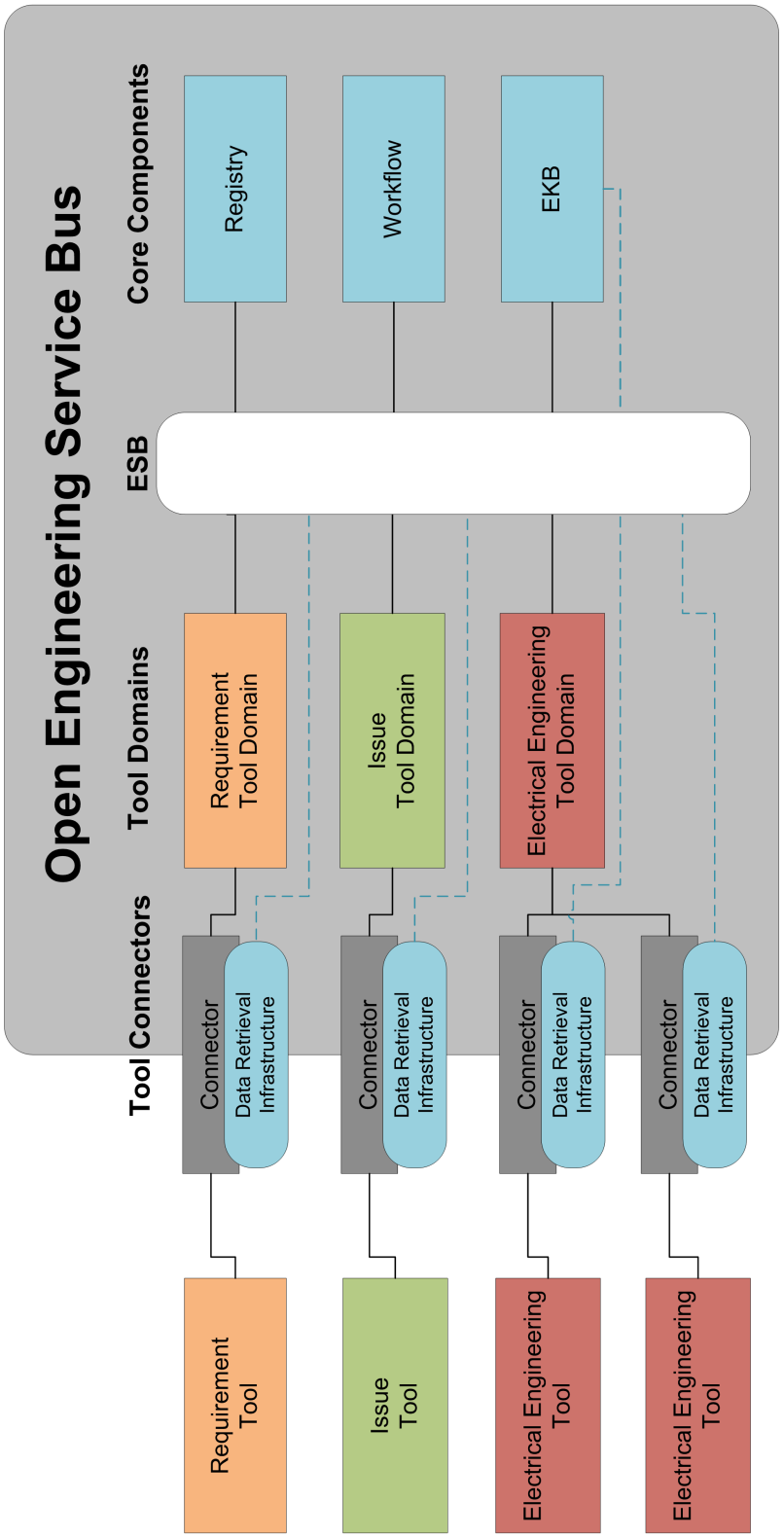

Figure 7.12: Overview about the architecture of the Open Engineering Service Bus (based upon Pieber (2010)) with integrated EKB component.

8 Evaluation

For the evaluation of the proposed EKB based semantic integration framework for (software+) engineering a prototype is realised and validated with the help of two real world use cases. The "Definition of Quality Criteria Across Tool Data Models in Electrical Engineering" use case is implemented according to the description in section 6.1 and the feasibility and efficiency of the proposed solution compared to a technical-only integration, as it is provided by the OpenEngSB, is evaluated (see section 8.1). The second use case, described in section 6.2 shows how the proposed semantic integration solution can be used for complex software engineering problems, like requirement tracing. Again an evaluation of the feasibility of the proposed solution, as well as a comparison to the performance and usability of a technical-only integration framework is done (see section 8.2).

8.1 Definition of Quality Criteria Across Tool Data Models in Electrical Engineering

As described in section 6.1 this use case deals with heterogeneous representations of the sensor concept in different engineering tools from the electrical and the software engineering domain. In this use case the properties **unit of measurement** and **measurement type** of the sensor concept are of interest. To check the consistency of these properties it is necessary to identify the different representations of the same sensor instance stored at different engineering tools and to compare the respective properties. Furthermore, each software representation of a sensor has to be connected to exactly three physical sensors modelled by the electrical engineering tool, as this use case is taken from a safety critical system.

The validation process is triggered on any change performed either by the electrical engineering tool or the software engineering tool. More specifically an event raised by one of the involved tools triggers the validation process. The actual process is modelled as a workflow in the technical integration system, which makes heavy use of the features provided by the EKB. These assumptions are derived in cooperation with industry partners and show a possible real world system setup.

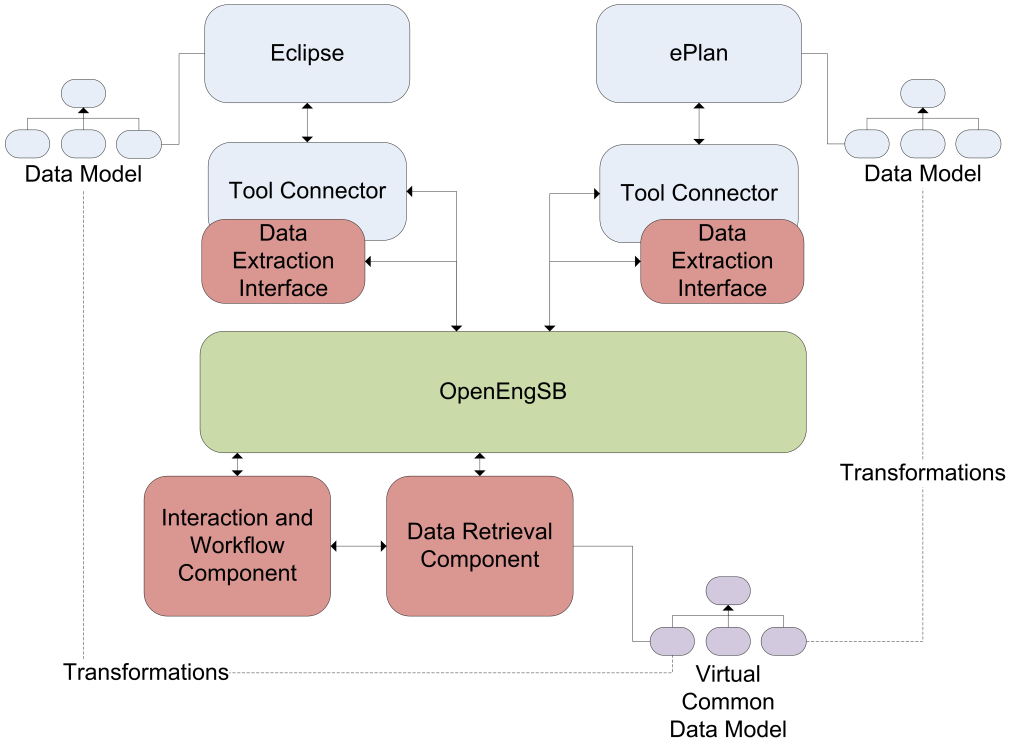

Figure 8.1: Possible setup for the "Definition of Quality Criteria Across Tool Data Models in Electrical Engineering" use case.

Figure 8.1 shows the necessary setup to conduct this use case. Source code introspection functionality is provided with the help of a code analyzing connector, which can interact for example with an IDE like Eclipse[1]. This tool provides information about the software representation of sensors. In addition the information stored in an electrical engineering tool, like ePlan[2] has to be provided. In both cases only the necessary functionality for this use case is realized, as a full fledged software introspection or electrical engineering connector is out of scope of this thesis. Yet this neccessary reduction of the connector complexity shows that the proposed semantic integration solution can be introduced gradually. Only the functionality necessary for the respective use cases has to be realized. From a data integration point of view these two connectors are data sources for the validation process, which enforces the restrictions on the sensor concept.

In figure 8.2 the two different sensor representations involved in this use case and their relation to a common sensor concept is shown. The validation process should operate on the common sensor concept to make it independent of tool changes or changes of the tool specific representation of a sensor. Therefore it is necessary to map and transform the respective sensor representations to the common sensor format. Tables 8.1 and 8.2 show how the attributes of the different sensor

[1] http://www.eclipse.org/

[2] http://www.eplanusa.com/products/eplan-electric-p8.html

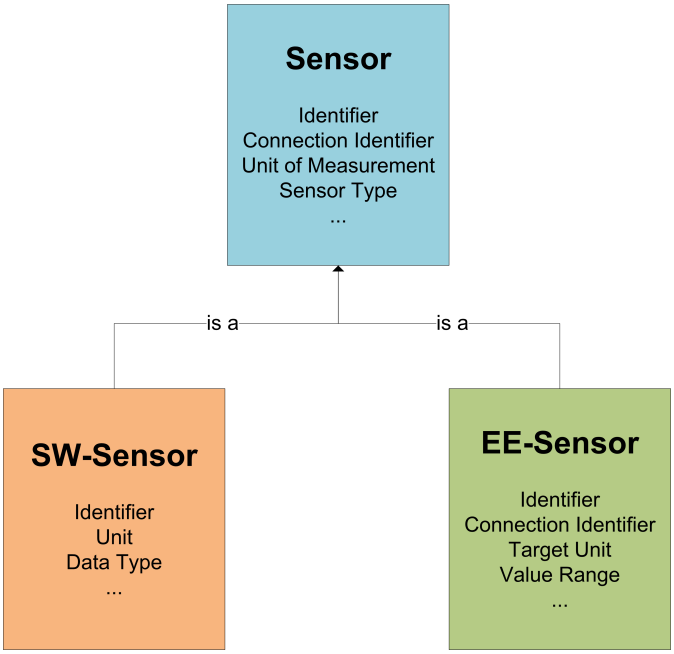

Figure 8.2: Common sensor concept and its relation to a software and an electrical engineering sensor.

Sensor	EE-Sensor	
	Mapping Attribute	Transformation
Identifier	Identifier	Simple/Automatic
Connection Identifier	Connection Identifier	Simple/Automatic
Unit of Measurement	Target Unit	Simple/Automatic
Sensor Type	Value Range	Complex/Manual

Table 8.1: Mapping of sensor attributes from electrical engineering sensor representation to common representation.

representations relate to the common sensor concept and how complex the necessary transformation process is.

In figure 8.3 an example for the transformation of a sensor representation from the software and the electrical engineering domain to a common sensor format is given. While the attributes **Identifier**, **Connection Identifier** and **Unit of Measurement** can be derived automatically from respective attributes of the source concepts, the **Sensor Type** attribute has to be transformed with the help of a specific transformation instruction. Simple or automatic transformations are only possible if the source and the target type are equal or if an automatic type transformation can be performed (see section 7.2.6). Although no automatic transformation for the attribute **Sensor Type** is possible, the transformation instruction is straight forward and can easily be formulated

Sensor	SW-Sensor	
	Mapping Attribute	Transformation
Identifier	Identifier	Simple/Automatic
Connection Identifier	Identifier	Simple/Automatic
Unit of Measurement	Unit	Simple/Automatic
Sensor Type	Data Type	Complex/Manual

Table 8.2: Mapping of sensor attributes from software engineering sensor representation to common representation.

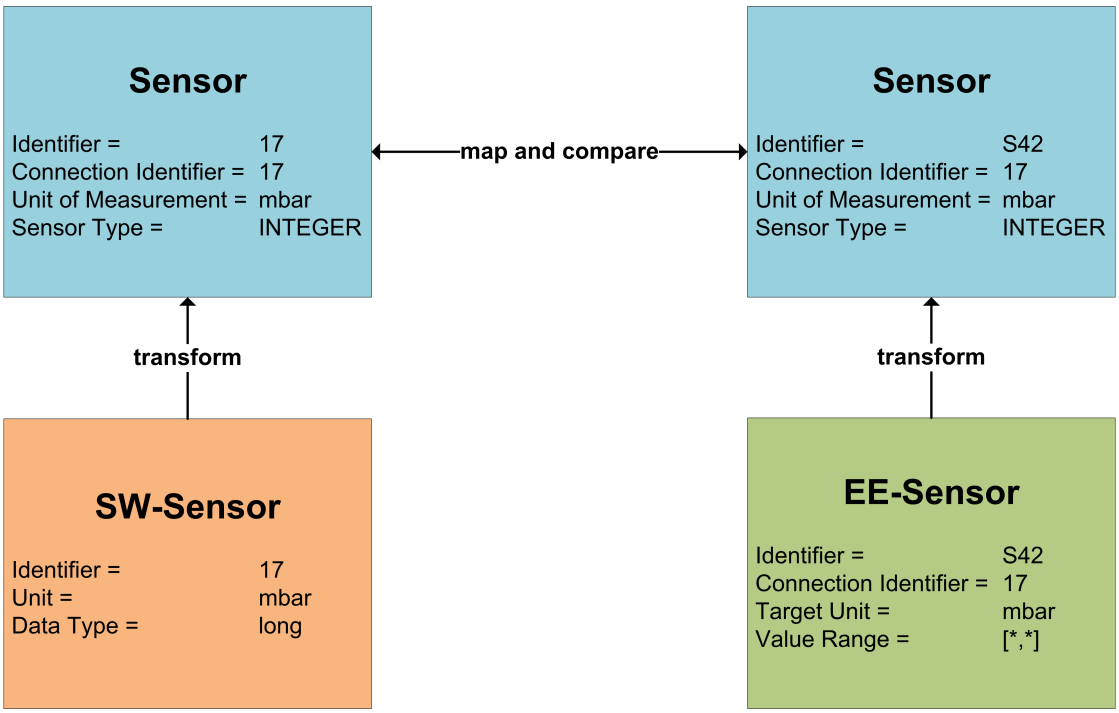

Figure 8.3: Example for mapping and transformation of software and electrical engineering sensor representation to common sensor concept.

by a domain expert. After the transformation the resulting sensors are in the same format and can be mapped and compared to find inconsistencies and other errors.

8.1.1 Feasibility Evaluation

With the help of the EKB based semantic integration solution the “Definition of Quality Criteria Across Tool Data Models in Electrical Engineering” use case is implemented in two steps. The first step is the connection of the respective tool connectors to the semantic integration system

and the second step is the definition of the validation process in a workflow. This validation process is then performed by the technical integration system using the features provided by the EKB.

The main configuration and setup effort has to be performed in the first step, because the connector has to be implemented to provide the tool specific data to the integration system. Furthermore, the semantic information has to be modeled and stored in the virtual common data model or more specifically the tool concepts and their relation to common engineering concepts have to be defined. As these steps happen at development time of the respective tool connector, they can be performed by domain experts, who know how to map the tool specific sensor concept to the common concept. This knowledge is available, because domain experts already need to perform the mapping manually to document their work and to provide information for other development activities building on their results. In addition the architecture of the data extraction infrastructure of the EKB reduces the effort for the implementation and integration of a tool connector into the OpenEngSB. This means that the additional effort for the connection of a tool to the semantic integration system is reduced to the implementation of one data extraction interface and a simple configuration of the meta data used by the data extraction system to find out which data sources can provide which concepts.

The fact that domain experts can perform the necessary mapping definition and tool integration during development time, as well as the minimal effort for tool integration show that the proposed semantic integration solution is efficient in terms of initial setup. As the tool connectors are usually also maintained by domain experts updates of the tool connector and the semantic information stored in the virtual common data model can be performed efficiently. The usability of the system for developers, who are no semantic integration experts is guaranteed by the fact that no knowledge about semantic integration is necessary to implement a tool connector. The definition of the mappings between the tool concepts and common concepts used throughout the integration system can either be supported by a specific tool, or with the help of an integration expert if the development of such a tool is out of scope of the project. Although this is a rather complex task, which makes initial tool integration, tool exchange and tool evolution more difficult, it is worth the effort as the realisation of advanced applications like the validation process in this use case becomes much easier.

The second step of the realisation of this use case is the definition of the validation process as workflow. This task is usually performed by a quality engineers or someone else, who is not an expert for each involved domain. Using the EKB based solution the validation process is defined without references to tool specific representations of the sensor concept. This means that this workflow is independent of the involved tools. Therefore the integration system can be used by quality engineers and project managers without the need for extensive support of domain and tool experts. The data extraction and transformation process is completely hidden from the designer of the advanced application. This leads to portable and robust process definitions, which can be reused in other projects.

In terms of robustness the system is designed to cope with tools that dynamically join and leave the system as well as with incomplete information in the virtual common data model. If such a problem occurs, the process stops and information about the problem is provided in the system log. It is critical that the semantic information stored in the virtual common data model is managed and updated when tool data models change. Inconsistencies do not bring the system down, but can reduce the functionality of the semantic integration system so much, that it cannot be used for advanced applications. The discovery and correct handling of inconsistencies, which is needed to provide the system with the necessary robustness is possible by defining and synchronizing the life-cycle of data sources and semantic tool information.

8.1.2 Comparison to Technical Integration

In this section a comparison between the necessary effort to integrate a tool, to exchange a tool and to perform advanced applications between the original OpenEngSB and the proposed EKB based integration solution is performed. The different steps necessary to execute these three different tasks as well as the technical expertise necessary to perform them is compared.

Tool Integration For tool integration in the original OpenEngSB it is first necessary to identify the correct tool domain for the tool connector. If such a domain is not available it has to be implemented and integrated into the technical integration system. The domain defines the service interface the connector implements to provide functionality to other components in the integration system. In the proposed semantic integration solution this step does not change if the domain does not define a data model for its connectors. If a data model is defined, then the semantic information about the concepts in the domain data model have to be stored in the EKB. This process is rather complex and needs some knowledge about semantic integration as well as knowledge about the tool domain and usually takes between ten to thirty minutes per domain specific concept for an integration expert and a domain expert. In the optimal case they work together to perform the necessary changes to the virtual common data model. As only mappings relevant for advanced applications and other use cases have to be modelled the costs for the definition of the mappings is not too high, although the necessary time for the definition of mappings does not scale linearly.

Then the connector is implemented and integrated into the technical integration system based on the domain interface. In the semantic integration system in addition to the implementation of the service interface also a data extraction interface has to be implemented by the tool connector and it has to publish which concepts it supports. The additional effort for this tasks depends on the tool, which has to be integrated. If the tool data is accessible the data extraction interface can be implemented in less than one developer work day. Note that in most cases the data extraction from the tool to the connector is the more complex part of the data integration process. But this step needs to be performed in any

case to provide the tool data to the integration system, regardless if a semantic integration solution is used or not.

One additional step is necessary for semantic integration. The semantic information about the concepts provided by the tool, which was connected to the OpenEngSB has to be modeled and stored in the EKB. This process is not necessary if the tool connector uses the data model provided by the tool domain. If the virtual common data model has to be adapted the same steps are necessary as described above, when a domain has to be implemented, which provides a data model for its connectors. So in this case the additional effort is again between ten to thirty minutes per tool specific concept for an integration expert and a domain expert.

Tool Exchange If a tool needs to be exchanged for another tool of the same tool domain in the original OpenEngSB it is necessary to implement the new tool connector as described in the tool integration section above. If the proposed semantic integration solution is used, in addition to the data integration tasks, also the virtual common data model needs to be updated. Yet this step is only necessary if the connector does not use the data model provided by its domain and in most cases only minor updates, like changes to outdated transformation instructions have to be performed. This means that compared to a technical-only integration in the worst case tool exchange takes one additional workday for a domain expert and one between ten to thirty minutes per tool specific concept for a domain expert and a semantic integration expert, while in the best case only one additional workday for a domain expert has to be invested.

Advanced Applications If advanced applications in the original OpenEngSB require data stored in specific tools, they need to either use the tool specific data model or manually transform the tool data to some common format. The semantic knowledge about this process is stored in the workflow definitions of the respective advanced application, which introduces a dependency from the process definition to the involved tools. Because of this dependency every time a new tool is integrated or a tool is exchanged the workflows for advanced applications have to be reviewed and possible adapted to the new data models of the involved tools. This process is complex and requires the involvement of experts from different domains as well as a data integration expert. The workflow definitions cannot be used easily in other projects or other contexts, because of their dependency to the involved tools. This often makes automatic end-to-end tests across tool boundaries or other advanced applications infeasible.

When the proposed semantic integration solution is used, advanced applications can rely on the data and semantic integration features of the EKB. This means that they can be implemented against common concepts independent of the involved tools and tool data models. Therefore neither integration nor domain experts are strictly necessary to define and maintain the process definitions of advanced applications like end-to-end tests across tool and domain boundaries, although the definition of the workflows is usually done in

cooperation with domain experts. The additional development effort during tool integration and tool exchange can be tolerated if advanced applications have to be performed in the (software+) engineering project.

The introduction of the proposed semantic integration solutions shifts the data and integration effort from the definition of advanced applications to the implementation and maintenance of the tool connectors. The advantage of this shift is that advanced applications can be defined more abstract and therefore can be reused in other projects. Furthermore, during development of the tool connectors domain experts are available which can easily perform the data integration for the respective tools and together with a semantic integration expert the update of the virtual common data model. Advanced applications can be implemented by quality engineers and project managers without domain knowledge. This shows that the proposed semantic integration solution is an effective and efficient solution for advanced applications across tool and domain boundaries.

8.2 Change Impact Analysis for Requirement Changes

In this use case the semantic integration infrastructure is used to conduct requirement tracing, an advanced software engineering task. A semi-automatic change impact analysis for requirement changes is performed using trace links between requirements, issues and developers. Team members, which are affected by the requirement change are notified and project management tasks like issue updates are performed automatically. Figure 8.5 shows a possible setup for this use case. A requirement management tool, like Rationale RequisitePro[3] is connected to the integration system, as well as an issue tracker like Trac[4]. In addition a connector for email notifications and a tool for the management of developer contact and identity information has to be available. A detailed description of requirement tracing and this use case can be found in section 6.2.

In this use case trace links between issues and requirements are established using informal semantic information in the issue description. Each issue, which is related to a requirement has to include a reference to the respective requirement using the format "#requirement(<requirementId>)", with the respective requirement identifier in its description. Issues can either be related to no requirement, to a single requirement or to multiple requirements. This informal semantic information is used by the integration system to establish trace links. Other forms of trace links between requirements and issues are possible, like for example explicit mapping tables, but for the sake of simplicity only this simple form of direct references is used. Links between the issue and developer are established with the help of the assignee attribute of the issue concept. Figure 8.4 shows the references between these three concepts

[3] http://www-01.ibm.com/software/awdtools/reqpro/
[4] http://trac.edgewall.org/

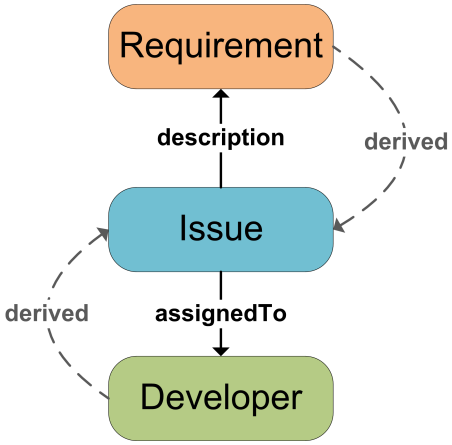

Figure 8.4: Trace links between requirement, issue and developer concept.

with actual and automatically derived backward links. Although they are not explicitly modeled the backward links can be navigated like normal traces and are automatically managed by the integration system.

The change impact analysis process is triggered by an requirement change request of a stakeholder of the (software+) engineering project. The requirement engineer analyses the change request and identifies affected requirements. These requirements are used as input for the change impact analysis. In a first step all related issues are identified. Depending on their current status they are simply marked for review, or reopened and marked for review. All affected developers are notified about the changes and asked for their estimates. Finally a report is generated, which contains information about affected issues and developers and who will need to contact the requirement engineer for a re-evaluation of the requirement estimate. For a detailed description of the change impact analysis process see section 6.2.

Only the functionality necessary for this use case is realised in the connectors for the respective tools to show that the proposed solution can be introduced step by step only modeling interesting parts of the system and because a full fledged requirement or issue tracker connector is out of scope of this thesis.

8.2.1 Feasibility Evaluation

Using the infrastructure provided by the proposed EKB based semantic integration solution the “Change Impact Analysis for Requirement Changes” use case can be realised with a simple configuration step during development of the tool connectors and by using the workflow and interaction component of the technical integration system in combination with the features provided by the EKB.

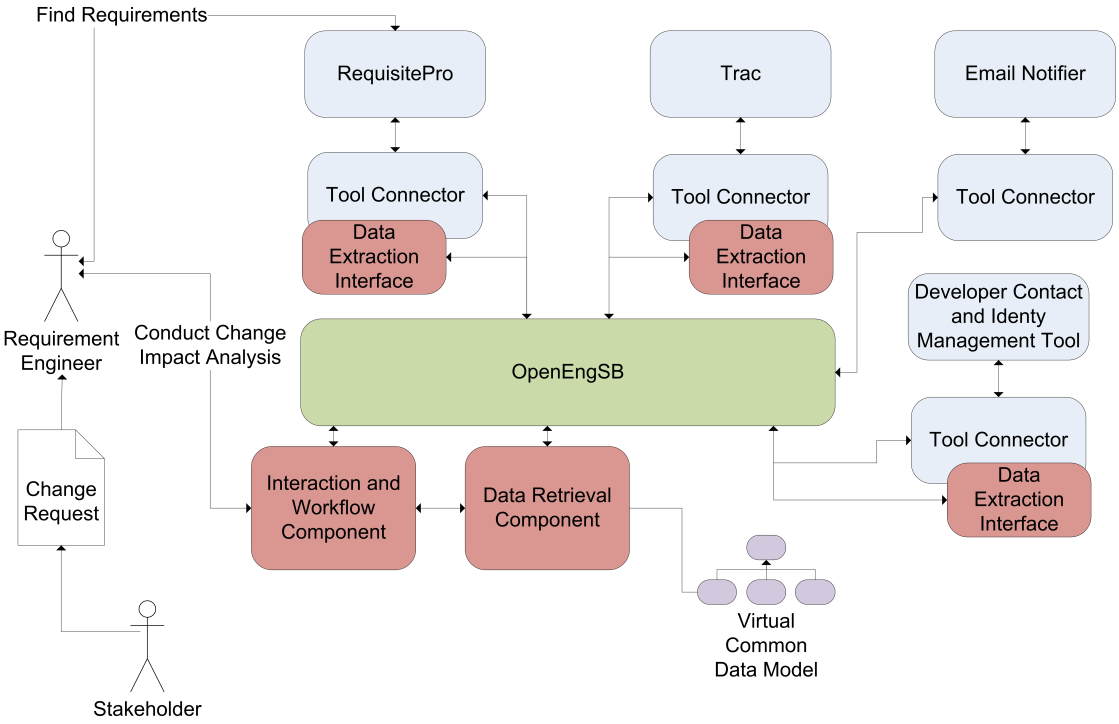

Figure 8.5: Possible setup for the “Change Impact Analysis for Requirement Changes” use case.

The definition of the relation between two different engineering concepts based on informal semantic information is modeled in the virtual common data model. For this purpose the soft reference mechanism of the EKB described in section 7.2.2 is used. In this use case the description attribute of an issue includes soft references to requirements and the attribute assignedTo is a soft reference to a developer. Listing 8.1 shows the definition of the soft references using an annotation based concept definition mechanism. As regular expression based soft references are used, the definition includes beside target concept also the regular expression for the extraction of the actual references. The configuration effort for this use case is reduced to the definition of the attribute containing the reference and a mechanism for extracting the actual reference from the field content. The EKB based semantic integration solution supports the usage of other solutions for the extraction of the references. Even very complex scenarios, where an knowledge system is used to find references between elements is possible, but a regular expression based approach is sufficient for this use case.

```
@ReferenceId(targetConceptId = "developer",
        targetConceptVersion = "1.0.0",
        regexp = ".+")
private String assignee;
```

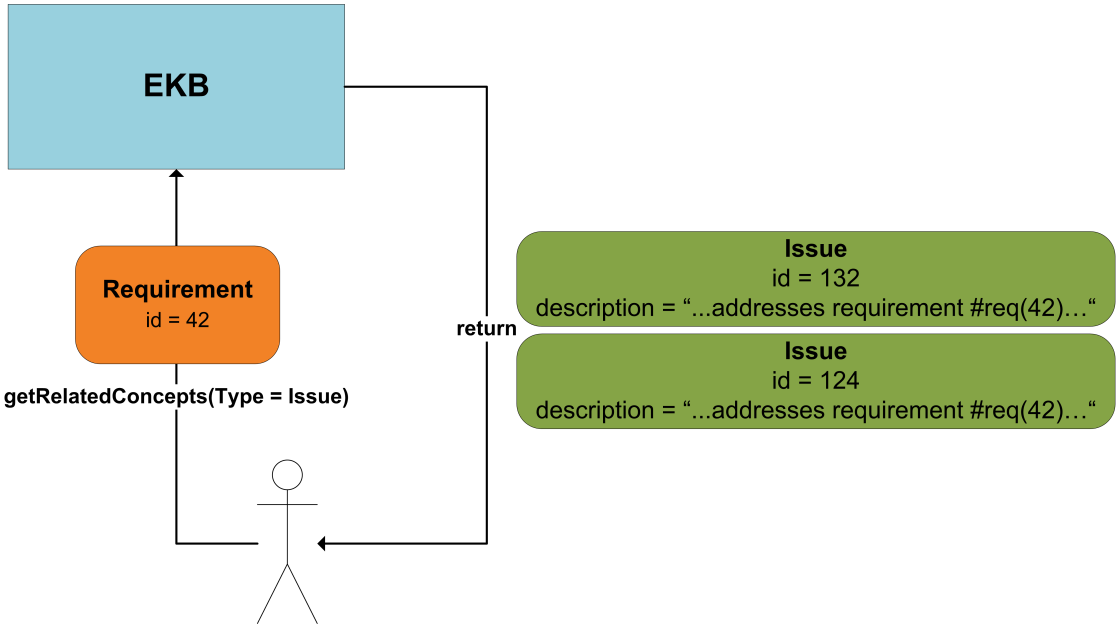

Figure 8.6: Example for a soft reference query with the help of the EKB.

```
@ReferenceId(targetConceptId = "requirement",
        targetConceptVersion = "1.0.0",
        regexp = "#requirement\\(([^\\)]+)\\)")
private String description;
```

Listing 8.1: Definition of soft references from issue to requirement and developer concepts.

Figure 8.6 shows an example of a query for issues related to a given requirement. The related issues are identified using the informal reference to the requirement identifier in the **description** attribute with the help of the soft reference mechanism of the EKB.

Building upon this configuration the workflow for the change impact analysis is implemented using the soft reference mechanism of the EKB. Based on the relation between requirements, issues and developers defined as soft references and stored in the virtual common data model, the EKB can derive actual trace links between instances of these concepts. So the requirement engineer defining and using this process does not need to know how requirements are actually linked to issues or developers. This knowledge is contained in the definition of the soft references, which are performed parallel to the introduction of the guideline for semantic meta information they originate from. The guideline in this use case is that an issue has to reference related requirements in its description. The requirement engineer can query the system for related issues and developers and use this information to conduct a high quality change impact analysis and to inform all affected team members.

As the setup to use informal semantic references between different engineering concepts is minimal, the proposed semantic integration solution is an effective and efficient solution for advanced applications like requirement tracing. Using the EKB, the usage of relations between concepts is decoupled from the actual form of the relation. This means that if the way requirements are linked to issues is changed, the change impact analysis process does not need to be adapted. The requirement engineer can perform the change impact analysis without knowledge about the issue domain and without tedious manual research which issues and developers are affected. This means that the system is also usable for stakeholders, which are not part of the development team, or not experts for all tools and tool domains involved in the development process. No additional effort is necessary for trace link generation, as the semantic meta information, which is already available due to specific project guidelines is used to trace from requirements to issues and from issues to developers.

Stakeholders, who are not part of the actual development team, usually do not know about team-internal guidelines, like the way requirements are referenced in issue descriptions. Yet they need to perform high level quality assurance and project management tasks, like the change impact analysis described in this use case. Using the proposed EKB based semantic integration solution, they can perform advanced applications using informal relations between different engineering concepts, without the need to know how the different concepts are actually connected. The robustness of the whole system is influenced positively by the fact that the semantic relations between concepts are defined directly in the respective domains and can be updated every time the underlying guidelines, a tool or a tool domain changes.

8.2.2 Comparison to Technical Integration

The differences in setup and implementation of the tool connectors are similar to the "Definition of Quality Criteria Across Tool Data Models in Electrical Engineering" use case and are described in detail in section 8.1.2. Therefore this section should only briefly cover the effort necessary to model the semantic relation information between different engineering concepts during tool integration or exchange. It focuses on the implementation of a change impact analysis with the help of the features provided by the EKB compared to a solution for a technical-only integration system.

The proposed EKB based semantic integration framework needs some additional setup, before soft references between different engineering concepts can be used. Basically two simple steps have to be performed to make references between concepts possible:

Definition of the soft reference The source concept has to define which attribute contains a soft reference and a mechanism for extracting the actual reference from the attribute value. The prototypic implementation of the EKB supports a regular expression based mechanism for soft reference definition. Listing 8.1 shows how this definition can be performed with the help of an annotation based concept specification mechanism. Other types of soft

references can be implemented and integrated into the EKB easily using a plugin style architecture.

Definition of a key attribute at the target concept The target concept of the soft reference needs to specify a key attribute. This has two specific implications. On the one hand the value of this key attribute is the actual reference value and used like a foreign key in a relational database. On the other hand the tool connector of the target concept needs to support queries for a specific element based on this key attribute.

These two steps are usually performed during integration of an engineering tool into the integration system. Domain experts and integration experts work together to identify key attributes and relations between engineering concepts. Furthermore, project guidelines, which are already in place and define how for example issues have to be described when they are created, give hints where semantic meta information is available and can be used to link between engineering concepts. Once the domain expert and the integration expert have an overview about potential references between different concepts the actual definition of the references can be done in a very short time. The whole process including review of project guidelines and other available meta information usually can be performed in one to two developer workdays for both domain expert and integration expert. If the project is smaller and each member has a good overview about the semantic meta information less effort is necessary for this task.

Using the proposed semantic integration framework the definition of the change impact analysis does not include any details about the relation between requirement and issue, besides the fact that such a relation exists. This means that a requirement engineer designing or using the change impact analysis workflow does not need to know any details about the relation between requirement and issue concept and its actual representation. The requirement expert can concentrate on the task of change impact analysis, which is decoupled from the actual linking mechanism between different engineering concepts. The semantic linking information between engineering concepts is defined directly in the virtual common data model and can be used by any advanced application. This means that this knowledge is not duplicated throughout the system and can be changed or updated easily. As a result the team is flexible and can decide to change specific guidelines or standards for semantic meta information, if they feel the necessity to do so. Evolution of the system is possible without the fear of breaking high level process definitions. Therefore tasks like tool exchange or the introduction of new engineering tools are much easier to accomplish.

The explicit nature of the reference definition in the virtual common data model is also a very useful documentation of semantic dependencies between different engineering concepts and can be analyzed by integration experts or domain experts to better understand the current setup of the system and to find potential inconsistencies or other problems.

To perform this use case in a technical only integration framework, like the original OpenEngSB without semantic integration it is necessary to use the semantic information in the change impact

analysis process definition. This means that in the process implementation the semantic relation between issues and requirements and issues and developers is explicitly used. The drawback of such a solution is that the actual process of using the semantic information has to be duplicated in every advanced applications which needs to facilitate the trace link between requirements and issues. In addition it is very hard to perform changes of the layout of the semantic meta information, as there is no single point of change, but all workflow definitions have to be reviewed and possibly updated.

Compared to a solution based on a technical-only integration system the definition of workflows in the proposed EKB based semantic integration system is much more portable and reusable across different projects. Due to the fact that only the existence of a link between two concepts regardless of the actual form of the link is needed to build advanced applications, workflows can be reused in other projects, where relations based on informal semantic information are represented differently. Therefore it is possible to use a common set of workflow definitions for typical project and quality management tasks recurring in all kinds of different projects operating with the same engineering concepts.

These differences show that the proposed semantic integration solution is much better suited for advanced software engineering applications like requirement tracing than a technical-only integration solution. The possibility to define and use semantic meta information to model relations between engineering concepts is a very flexible mechanism, which can also be used for other advanced applications, like for example automatically linking commits to related issues. The main advantage of this solution is the simplicity of the setup and the single point of change making the overall system easier to change and to maintain. Similar to the “Definition of Quality Criteria Across Tool Data Models in Electrical Engineering” use case described in section 8.1 the main drawback of the EKB based semantic integration solution is the additional configuration and setup effort during development and implementation of the tool connectors. But if advanced applications, like requirement tracing for change impact analysis have to be performed regularly, the additional effort during system setup pays off.

9 Discussion

In this section the results of this thesis with respect to the research issues defined in section 5.1 are discussed. The basis of the discussion is on the one hand the review of the theoretical foundations of integration from the technical, semantic and data driven point of view in sections 2, 3 and 4 and on the other hand the evaluation of the proposed semantic integration solution in section 8. Two real world use cases are studied in this thesis to define the requirements and perform an evaluation (see section 6). In section 7 the main features and architecture of the EKB based integration framework and its core components are explained in detail.

The research issues of this thesis can be summarized in the following way:

RI-1 – Feasibility The feasibility of an EKB based semantic integration framework in the (software+) engineering domain especially for advanced applications like end-to-end tests across tool and domain boundaries has to be evaluated.

RI-2 – Life-cycle Life-cycles for semantic tool knowledge and the tool connectors have to be developed and synchronized to provide the system with the necessary robustness to be accepted by practitioners.

RI-3 – Transformation Transformation instructions are used to transform data from the tool specific representation to a virtual common data model during runtime. These instructions have to be generated semi-automatically to make the system easy to setup and use.

RI-4 – Query Infrastructure A query infrastructure with both a usable frontend for developers with little knowledge about semantic integration and and a backend for data extraction from integrated tools has to be developed.

9.1 Feasibility of an EKB based semantic integration framework for (software+) engineering

In this section research issue RI-1 as defined in section 5.1 is discussed. The proposed EKB based semantic integration solution for (software+) engineering has to support project managers

and quality engineers in their task of designing and implementing advanced applications, like end-to-end tests across tool boundaries or requirement tracing. Two real world use cases are analyzed to find and define the specific requirements of such a system and to evaluate a prototype. The results show that the introduction of the proposed solution leads to higher setup and implementation costs for tool connectors. In addition the setup of a connector involves the integration of semantic tool knowledge into the virtual common data model, which is a complex task, where the support of a semantic integration expert may be necessary. Yet compared to other semantic integration solutions, which are for example ontology based, the additional technical expertise is minimal, as simple annotation based, DSL based or even graphical approaches for the modeling of the semantic knowledge are possible. The prototype uses a simple annotated Java class model to define the virtual common data model, which is sufficient for the use cases described in this thesis. In the future an easy to use DSL based or graphical approach for the management of the virtual common data model is highly desirable as it would decrease the initial setup costs and reduce the severity of the main drawback of the proposed solution. The evaluation of the two use cases shows that the increased effort for the creation of the tool connectors pays off when advanced applications are defined. The EKB based approach makes it possible to define advanced applications in an abstract way, without dependencies to semantic information or specific tools. Therefore process definitions for advanced applications can be shared by multiple projects reducing the overall development and maintenance costs. Furthermore, no knowledge about the target tools is necessary to implement the advanced applications, which makes the system usable for project managers and quality engineers, who are often not experts for all involved engineering domains.

The flexibility of the proposed solution in terms of semantic knowledge modeling is an important advantage compared to other semantic integration solutions, which only support a specific modeling language. This makes it possible to use integration expertise already available in engineering teams and to build a customized modeling solution, which suits the respective team's needs.

The evaluation of the two use cases from the software engineering and the electrical engineering domain clearly shows, that if advanced applications have to be performed across tool boundaries, or need to facilitate semantic meta information, the proposed EKB based framework is a feasible solution. The main advantage of the EKB based solution is the reduced effort for the design, implementation and maintenance of advanced applications, especially compared to technical-only integration systems.

9.2 Design of a robust semantic integration system based on a synchronized life-cycle model

This research issue aims at the development and implementation of a life-cycle system for both semantic tool knowledge and the actual tool connectors, which act as data sources for the seman-

tic integration system. The life-cycles are defined in section 7.2.4 and their relation to each other is explained. The synchronisation of the semantic knowledge of a tool with the tool connector state leads to a more flexible system in terms of integration of new tools during runtime, tool exchange and the deactivation of tools. Compared to other semantic integration solutions, where the semantic tool knowledge is directly stored at the tool, the decoupling of semantic information from the actual tool instance allows for a flexible update of either the semantic information or the tool instance. In addition it makes it possible to facilitate the tool domain concept of the OpenEngSB, which creates the possibility to define the data model used by tools of a certain tool domain directly in the respective domain implementation. As a result, no update of the virtual common data model has to be performed if a tool is exchanged or a new tool instance is integrated. Unfortunately the separation of a tool from its semantic information also leads to higher maintenance efforts, as changes at the tool connector can lead to changes in the virtual common data model and vice versa.

Analysis of the evaluation of the two use cases shows that the life-cycle model leads to a more robust integration system, as it provides a clear definition of tool and semantic knowledge state. The reaction of the system to configuration changes is more predictable. An additional benefit of the proposed solution is the possibility to gain knowledge about the actual system state through comparison of the active part of the virtual common data model, or in other words the concepts of the virtual common data model currently provided by active tool instances, compared to the complete virtual common data model, which may also contain concepts that are completely unavailable. This comparison makes it possible to perform a validation of the system setup and the current system state. Furthermore it improves the probability of configuration problem detection. Practitioners, who use the semantic integration system, need to know the life-cycle of tools and semantic tool knowledge to understand the system's reaction to configuration updates. Therefore this feature of the system needs to be well documented. In addition it would be desirable to have a clear documentation of the system state changes. Currently they are logged in the system log file. The development of a readily available and understandable system state overview would further increase the usability of the system.

In terms of robustness it is critical to manage the virtual common data model effectively and to update it if the underlying systems and concepts change. Inconsistencies between the actual system and the model stored in the EKB do not bring the system down, but if they are too frequent they can reduce the functionality of the semantic integration system until it is virtually useless. Therefore it is important to use the possibility to check the active part of the virtual common data model as explained above to gain insight into the current state of the system. In addition to the automatic consistency check already performed by the integration system, after a tool or the virtual common data model is changed, a more sophisticated system state analysis would lead to even higher robustness of the integration system and should be integrated into the EKB based system in the future.

9.3 Semi-automatic transformation instruction derivation based on semantic knowledge

The third research issue of this thesis addresses the semi-automatic transformation instruction generation based on semantic knowledge. In order to provide the system with the necessary usability in terms of setup time and maintenance it is critical that simple transformations from tool specific concepts to common concepts are performed automatically, while more complex transformations can be defined manually. The proposed semantic integration solution derives transformation instructions of three different types automatically (see section 7.2.6): a.) identity transformation, b.) simple type transformation and c.) string parsing. In addition it is possible to split an attribute to different attributes of the target concept or to merge one or more attribute values together to get the target attribute value. The two use cases evaluated in section 8 show that these three simple cases are sufficient for most mappings between source and target concepts. Furthermore, the proposed semantic integration framework provides good support for the definition of manual transformation instructions. They can be defined in plain Java code and can be attached directly to the concept definitions for the virtual common data model. This approach is especially well suited if an annotated Java class model is used for the modeling of the virtual common data model. If a graphical or DSL based approach, or even an ontology based approach is used, the efficient integration of the possibility to define manual transformation instructions is critical.

Compared to other semantic integration systems the automatic transformation derivation capabilities of the proposed solution are rather simplistic. To further increase the usability of the system it is important to develop more advanced automatic mapping mechanisms, like name and type based automatic attribute mapping procedures. It is also desirable to have a well integrated automatic mapping support in the tool used for the definition of the virtual common data model. Especially the possibility to automatically generate mapping candidates, which are approved or updated manually is an important feature, which would further increase the usability of the system and which is also part of other semantic integration frameworks.

The automatic transformation instruction generation capabilities of the proposed EKB based semantic integration solution are simple, but sufficient for the use cases described in this thesis and many other real world use cases. Therefore the system is usable and it is adequately easy to model the mappings between tool concepts and common concepts. The evaluation shows that the integration of new tools and tool exchange can be performed in acceptable time, although the update of the virtual common data model including the transformation definition has to be performed without explicit tool support.

9.4 Development of a usable query infrastructure

In this section research issue RI-4 is discussed. An effective and efficient semantic integration solution for (software+) engineering needs to provide a usable interface for data extraction from heterogeneous sources, which can be used by project managers and quality engineers, who are neither domain nor integration experts. Furthermore, a data extraction infrastructure has to be provided which makes it easy to connect tools as data sources to the integration system and which makes it possible for tools to publish which engineering concepts they can provide.

The proposed EKB based integration solution provides an interface for data extraction based on the common engineering concepts modeled in the virtual common data model. This allows the definition of advanced applications based on these common concepts without dependencies to tool specific data models. Therefore it is not necessary for project managers or quality engineers to have domain knowledge in order to design and implement advanced applications, like end-to-end tests across tool boundaries using the data extraction capabilities of the EKB based system. The interface is flexible and especially built to serve as basis for a query engine for data extraction. With the integration of such a query engine the usability of the system can be further increased and the necessary technological know-how to use the system is reduced. The evaluation shows that the definition of advanced applications like “Definition of Quality Criteria Across Tool Data Models in Electrical Engineering” or “Change Impact Analysis for Requirement Changes” can be performed with less effort than in a technical-only integration system. Especially the portability and re-usability of the process definitions for advanced applications is an important advantage of the abstraction provided by the data integration infrastructure of the proposed integration solution.

At the connector level the infrastructure for data extraction needs to provide simple connection points for tools, which want to provide their data to the integration system. The proposed semantic integration framework solves this problem by providing a connector extension that handles most data extraction related issues and provides a very simple interface, which has to be implemented by the tool connector. This approach has the advantage that it is very flexible and that it is easy to add more advanced data retrieval capabilities as long as they can be implemented in the data retrieval stub provided by the integration system. To perform such changes no update of the tool connectors is necessary. The major disadvantage of this solution is that it is not possible to use the capabilities of the tools for efficient data extraction. Therefore the integration system is designed to support the possibility to define a custom data extraction stub, which uses for example the features provided by a relational database storing the tool specific data. An important feature of the proposed solution is the possibility for tools to dynamically join and leave the integration system during runtime. The EKB automatically performs consistency checks in such a case and updates the state of the semantic knowledge of the tool.

As tool connector and the semantic information about the tool are separated, the implementation of the tool connector and the connection to the data extraction stub of the semantic integration framework can be performed without knowledge about semantic integration. This reduces the

development time and costs for tool connectors. The modeling of the semantic information is then performed in a separate development step, which is usually performed by domain and semantic integration experts together. Although this clear separation has some advantages it also reduces the locality of the tool related information. In other words, if a tool connector has to be updated or changed, it is also necessary to review and possibly update the virtual common data model, as tool related semantic information might have changed. Yet, if a technical-only integration system is used the semantic information has to be included in the process definitions of advanced applications. Therefore these process definitions are dependent on specific tool connectors. If such a connector changes, all dependent process definitions have to be updated. Therefore, instead of having to update the system at two well defined places, multiple artifacts have to be reviewed and updated. This shows that the proposed semantic integration solution reduces the maintenance effort compared to a technical-only integration framework.

From the data integration point of view the proposed solution can be seen as a both-as-view or global-local-as-view solution (see section 4). This means that both a global-as-view and a local-as-view representation of the system can be derived using the transformation instructions stored in the virtual common data model. To implement the use cases described in this thesis and other advanced applications especially the global-as-view representation of the system is important, as it allows easy transformation of tool specific data into the virtual common data model. Therefore it is necessary that transformations from tool specific concepts to common concepts are available in the virtual common data model, while the reverse direction is seldom needed in typical engineering projects and therefore does not need to be modeled.

10 Conclusion and Perspectives

In this section a short summary of this thesis is given as well as concluding remarks regarding the proposed semantic integration solution in section 10.1. In addition possibilities to improve the proposed EKB based solution are described in section 10.2.

10.1 Conclusion

Development of complex software intensive systems requires the cooperation of experts from different domains. These experts use different tools, which are well suited for their specific purpose, but usually do not provide sufficient mechanisms for cooperation with other engineering tools. Especially the integration of tools from different engineering domains, like electrical engineering or software engineering is problematic.

Literature research shows that technical integration systems try to provide a platform for tool integration, which defines a common message format as well as the necessary infrastructure to use services from other integrated tools and to share data with other tools. Based on technical integration systems advanced quality assurance and project management applications like end-to-end tests across tool boundaries are possible. Yet because technical integration systems do not take the semantics of tool data models into account the definition of advanced applications is difficult and often requires both domain and data integration expertise. Therefore these systems are often hard to use for quality engineers and project managers.

To overcome this problem semantic integration solutions can be used to model the semantic information about tool models and to use this information for advanced applications. The Engineering Knowledge Base (EKB) concept is one approach to provide semantic information for engineering projects. An EKB has three main features: a.) data integration using mappings between different engineering concepts, b.) transformations between different engineering concepts utilizing these mappings and c.) advanced applications building upon these foundations.

In this thesis a semantic integration framework for (software+) engineering is developed based on the EKB concept and the OpenEngSB project, a technical integration solution for engineering projects. The proposed solution supports the modeling of semantic tool knowledge and its

usage for advanced applications like end-to-end tests across tool boundaries. In addition data integration tasks like schema mapping through transformation instructions and the possibility to integrate different heterogeneous data sources are provided by the EKB based semantic integration solution. The real world use cases *Definition of Quality Criteria Across Tool Data Models in Electrical Engineering* and *Change Impact Analysis for Requirement Changes* are used to define the requirements of a semantic integration solution for (software+) engineering. Additional requirements with respect to effectiveness, efficiency, usability and robustness are formulated as research issues for this thesis. Based on these requirements and the theoretical foundations of integration from the technical, semantic and data integration point of view an architecture for an EKB based semantic integration solution is developed. Again addressing these two use cases prototypes of the EKB based solution are developed and evaluated against a technical-only integration solution, like it is provided by the original OpenEngSB.

Compared to a technical-only integration solution the EKB based semantic integration system leads to less complex process definitions of advanced applications, as neither domain nor semantic integration knowledge and functionality has to be included. As a result the process definitions can be shared between different projects, because they do not contain any dependencies to project specific tools. Another advantage of the higher abstraction level of these process definitions is the fact that they can be designed and implemented by project managers or quality engineers, who have little knowledge about the involved domains, tools, or data integration. The semantic integration solution provides an efficient possibility to define semantic meta information and to use it to transform tool specific concepts to common engineering concepts and to define informal relations between different engineering concepts. The major drawback of the proposed solution is the additional effort for tool connector implementation, as the connection to the semantic integration system for data sharing has to be implemented and the semantic tool information has to be modelled. Nevertheless, taking the reduced effort for design, implementation and maintenance of advanced applications into account, the proposed EKB based framework is an effective and efficient solution for semantic tool integration for (software+) engineering.

10.2 Future Work

Semantic integration for (software+) engineering is a complex task, which defines hard to achieve requirements with respect to usability, robustness and flexibility of the resulting system. As the development of a complete solution is out of scope of this thesis, some possibilities to improve the proposed semantic integration framework or the overall integration system are described in this section.

Advanced Modeling Support In section 7.2.1 the model analyzer component of the proposed EKB based semantic integration solution is described. While this component operates on an annotated Java class model in the prototype developed in this thesis, the usage of an

advanced modeling mechanism with the help of a DSL or a dedicated graphical editor would increase the usability of the system. Basically one of the scenarios described in section 7.2.1 should be implemented to reduce the effort for the management of the virtual common data model and thus reduce the severity of the main disadvantage of the proposed solution, which is the additional setup and configuration effort during tool integration and exchange.

Query Engine The interface of the EKB, which allows the retrieval of data from different tools using the virtual common data model is a good basis for advanced applications, like end-to-end tests across tool boundaries. Yet the usability of the system could be further increased if a query engine is developed based on the data extraction interface of the EKB. This query engine would make it possible for users to extract data without even realizing that a semantic integration solution is used to provide the data. As a result the definitions of advanced applications would become even more abstract leading to higher portability and reduced maintenance effort.

Advanced Data Extraction Infrastructure Parallel to the introduction of a query engine as frontend for the semantic integration solution the improvement of the data extraction stubs at the connectors should be performed. This is on the one hand necessary to support advanced querying mechanisms needed for the query engine and on the other hand could increase the performance of the overall system. Common persistence solutions like relational databases should be supported using their power to perform queries in an efficient way.

Further Automation of Transformation Instruction Derivation The proposed semantic integration solution only provides simple automatic transformation instruction derivation capabilities. One possibility to improve the system, would be to implement more advanced automatic mapping algorithms, which use attribute names and attribute types. Especially the development of a dedicated tool for the modeling of the virtual common data model as described at the beginning of this section would make it possible to improve the mapping mechanism. The tool could propose probable mappings between attributes of related concepts, which can be accepted by the user or updated manually.

Advanced System State Monitoring Tool The life-cycle model used for semantic tool knowledge stored in the virtual common data model and actual tool instances makes it possible to take a snapshot of the active part of the virtual common data model and to compare this snapshot to the complete virtual common data model (see section 7.2.4). To make it easier for users to take this snapshot and to perform the comparison, a simple to use management interface for the EKB could be developed. With the help of such a management interface the detection of configuration faults or other problems would become much easier.

References

Alonso, G. (2008). Challenges and opportunities for formal specifications in service oriented architectures. In *Petri nets '08: Proceedings of the 29th international conference on applications and theory of petri nets* (pp. 1–6). Berlin, Heidelberg: Springer-Verlag.

Amar Bensaber, D., & Malki, M. (2008). Development of semantic web services: model driven approach. In *Notere '08: Proceedings of the 8th international conference on new technologies in distributed systems* (pp. 1–11). New York, NY, USA: ACM.

Biffl, S. (2010). *Software engineering integration for flexible automation systems.* Presentation for Opening of Christian Doppler Laboratory SE-Flex-AS.

Biffl, S., Schatten, A., & Zoitl, A. (2009). Integration of heterogeneous engineering environments for the automation systems lifecycle. In *Indin 2009 7th international conference on industrial informatics* (pp. 576–581). IEEE Computer Society. (Vortrag: IEEE International Conference on Industrial Informatics (INDIN), Cardiff, UK; 2009-06-24 – 2009-06-26)

Boulcane, F. (2006, April). An approach of mediation. In *Information and communication technologies, 2006. ictta '06* (Vol. 2, pp. 3546–3551).

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., & Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. *J. Syst. Softw.*, *80*(4), 571–583.

Brodie, M. (2010, April). Data integration at scale: From relational data integration to information ecosystems. In *Advanced information networking and applications (aina), 2010 24th ieee international conference on* (pp. 2–3).

Cafarella, M. J., Halevy, A., & Khoussainova, N. (2009). Data integration for the relational web. *Proc. VLDB Endow.*, *2*(1), 1090–1101.

Chappel, D. A. (2004). *Enterprise service bus* (1st ed.). Sebastopol, CA, USA: O'Reilly Media, Inc.

Chen, J., Liang, H., & Mao, Y. (2009). Mapping mechanism based on ontology extended semantic related groups. In *Wism '09: Proceedings of the 2009 international conference on web information systems and mining* (pp. 45–48). Washington, DC, USA: IEEE Computer Society.

Cruz, I. F., Xiao, H., & Hsu, F. (2004). An ontology-based framework for xml semantic integration. In *Ideas '04: Proceedings of the international database engineering and applications symposium* (pp. 217–226). Washington, DC, USA: IEEE Computer Society.

Dan, A., & Narasimhan, P. (2009). Dependable service- oriented computing. *IEEE Internet Computing*, *13*(2), 11–15.

Davis, J. (2009). *Open source soa*. Manning Publications Co.

Eric Newcomer, G. L. (2004). *Understanding soa with web services*. Addison-Wesley Professional.

Fiore, S., Negro, A., & Aloisio, G. (2009, November). Data virtualization in grid environments

through the grelc data access and integration service. In *Internet technology and secured transactions, 2009. icitst 2009. international conference for* (pp. 1–6). London.

Floyd, C. (1984). A systematic look at prototyping. *Approaches to Prototyping*, 1–18.

Frankel, D. S., Hayes, P., Kendall, E. F., & McGuinness, D. L. (2004, July). A model-driven semantic web: Reinforcing complementary strengths. *MDA Journal, Business Process Trends*.

Gal, A. (2008, April). Interpreting similarity measures: Bridging the gap between schema matching and data integration. In *Data engineering workshop, 2008. icdew 2008. ieee 24th international conference on* (pp. 278–285).

Gruninger, M., & Kopena, J. B. (2005). Semantic integration through invariants. *AI Mag.*, *26*(1), 11–20.

Hakimpour, F., & Geppert, A. (2001). Resolving semantic heterogeneity in schema integration. In *Fois '01: Proceedings of the international conference on formal ontology in information systems* (pp. 297–308). New York, NY, USA: ACM.

Halevy, A. (2005). Why your data won't mix. *Queue*, *3*(8), 50–58.

Halevy, A., Rajaraman, A., & Ordille, J. (2006). Data integration: the teenage years. In *Vldb '06: Proceedings of the 32nd international conference on very large data bases* (pp. 9–16). VLDB Endowment.

Halevy, A. Y. (2005, October). Why your data won't mix: Semantic heterogeneity. *Queue*, *3*(8), 50–58.

Heindl, M., & Biffl, S. (2005). A case study on value-based requirements tracing. In *Esec/fse-13: Proceedings of the 10th european software engineering conference held jointly with 13th acm sigsoft international symposium on foundations of software engineering* (pp. 60–69). New York, NY, USA: ACM.

Hohpe, G., & Woolf, B. (2004). *Enterprise integration patterns : designing, building, and deploying messaging solutions* (I. Pearson Education, Ed.). Addison-Wesly.

Hu, G. (2006). Global schema as an inversed view of local schemas for integration. In *Sera '06: Proceedings of the fourth international conference on software engineering research, management and applications* (pp. 206–212). Washington, DC, USA: IEEE Computer Society.

Jagatheesan, A., Weinberg, J., Mathew, R., Ding, A., Vandekieft, E., Moore, D., et al. (2005). Datagridflows: Managing long-run processes on datagrids. In J.-M. Pierson (Ed.), *Data management in grids: first vldb workshop* (pp. 113–128). Springer-Verlag.

Jamadhvaja, M., & Senivongse, T. (2005). An integration of data sources with uml class models based on ontological analysis. In *Ihis '05: Proceedings of the first international workshop on interoperability of heterogeneous information systems* (pp. 1–8). New York, NY, USA: ACM.

Jarke, M. (1998). Requirements tracing. *Commun. ACM*, *41*(12), 32–36.

Jieming Wu, X. T. (2010, March). Research of enterprise application integration based-on esb. In *Advanced computer control (icacc), 2010 2nd international conference on advanced computer control* (pp. 90–93).

Kaushal, C., & Saravanan, S. (2004, July). Demystifying integration. *Communications of the ACM*, *47*(7).

Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., Emam, K. E., et al. (2002). Preliminary guidelines for empirical research in software engineering. *IEEE Trans. Softw. Eng.*, *28*(8), 721–734.

Kramler, G., Kappel, G., Reiter, T., Kapsammer, E., Retschitzegger, W., & Schwinger, W. (2006). Towards a semantic infrastructure supporting model-based tool integration. In *Gamma '06: Proceedings of the 2006 international workshop on global integrated model management* (pp. 43–46). New York, NY, USA: ACM.

Kurtev, I., Bézivin, J., Jouault, F., & Valduriez, P. (2006). Model-based dsl frameworks. In *Oopsla '06: Companion to the 21st acm sigplan symposium on object-oriented programming systems, languages, and applications* (pp. 602–616). New York, NY, USA: ACM.

Lenzerini, M. (2002). Data integration: a theoretical perspective. In *Pods '02: Proceedings of the twenty-first acm sigmod-sigact-sigart symposium on principles of database systems* (pp. 233–246). New York, NY, USA: ACM.

Linthicum, D. S. (2000). *Enterprise application integration*. Boston, Mass: Addison-Wesley.

McBrien, P., & Poulovassilis, A. (2003, March). Data integration by bi-directional schema transformation rules. In *Data engineering, 2003. proceedings. 19th international conference on* (pp. 227–238).

Moser, T. (2010). *Semantic integration of engineering environments using an engineering knowledge base*. Unpublished doctoral dissertation, Vienna University of Technology.

Moser, T., Biffl, S., Sunindyo, W. D., & Winkler, D. (2010, February). Integrating production automation expert knowledge across engineering stakeholder domains. In *Proceedings of the 4th international conference on complex, intelligent and software intensive systems (cisis 2010)* (pp. 352–359). IEEE Computer Society.

Moser, T., Waltersdorfer, F., Zoitl, A., & Biffl, S. (2010). Version management and conflict detection across heterogeneous engineering data models. In *Industrial informatics (indin), 2010 8th ieee international conference on* (pp. 928–935).

Noy, N. F. (2004). Semantic integration: a survey of ontology-based approaches. *SIGMOD Rec.*, *33*(4), 65–70.

OMG. (2010a). *Uml by omg.* Last visited July 13, 2010, online: `http://www.uml.org/`.

OMG. (2010b, May). Unified modeling language (uml) infrastructure specification version 2.3 [Computer software manual]. Available from `http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/`

OpenEngSB-Project. (2010). *Official open engineering service bus website.* Last visited April 20, 2010, online: `http://openengsb.org`.

Pieber, A. (2010). *Flexible engineering environment integration for (software+) development teams*. Diplomarbeit, Institute of Software Technology and Interactive Systems Vienna University of Technology.

Rebstock, M., Fengel, J., & Heiko, P. (2008). *Ontologies-based business integration*. Springer.

Rosenthal, A., Seligman, L., & Renner, S. (2004). From semantic integration to semantics

management: case studies and a way forward. *SIGMOD Rec.*, *33*(4), 44–50.

Ullman, J. D. (1997). Information integration using logical views. In *Icdt '97: Proceedings of the 6th international conference on database theory* (pp. 19–40). London, UK: Springer-Verlag.

W3C. (2010a). *Owl2.* Last visited July 13, 2010, online: `http://www.w3.org/TR/owl2-overview/`.

W3C. (2010b). *Rdf.* Last visited August 17, 2010, online: `http://www.w3.org/RDF/`.

Weng, L., Agrawal, G., Catalyurek, U., Kurc, T., Narayanan, S., & Saltz, J. (2004). An approach for automatic data virtualization. In *Hpdc '04: Proceedings of the 13th ieee international symposium on high performance distributed computing* (pp. 24–33). Washington, DC, USA: IEEE Computer Society.

Xiong, F., Han, X., & Liqun, K. (2009, August). Research and implementation of heterogeneous data integration based on xml. In *Electronic measurement and instruments, 2009. icemi '09. 9th international conference on* (pp. 711–715).

Yan Du, L. Z., Wuliang Peng. (2008, December). Enterprise application integration: an overview. In *Intelligent information technology application workshops, 2008. iitaw '08. international symposium on intelligent information technology application workshops* (pp. 953–957).

Zhang Yi, Z. J. (2009, December). Research and implementation of eai based on soa. In *Computational intelligence and software engineering, 2009. cise 2009. international conference on computational intelligence and software engineering* (pp. 1–4).

List of Figures

List of Tables

A License