FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

An Integration Framework for Knowledge-Supported Project Management in IT Consortia

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Andreas Grünwald
Matrikelnummer 0827532

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Stefan Biffl
Mitwirkung: Dipl.-Ing. Dietmar Winkler

Wien, TT.MM.JJJJ

(Unterschrift Verfasserin) (Unterschrift Betreuung)

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

An Integration Framework for Knowledge-Supported Project Management in IT Consortia

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Andreas Grünwald
Registration Number 0827532

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Stefan Biffl
Assistance: Dipl.-Ing. Dietmar Winkler

Vienna, TT.MM.JJJJ

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Andreas Grünwald
Sechshauserstraße 24/14, 1150 Wien

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 Verfasserin)

Acknowledgements

I want to thank Dr. Stefan Biffl and Dipl. Ing. Dietmar Winkler for guiding me through the master-thesis, and supporting me with valuable feedback. Likewise, I want to thank the whole CDL-Flex crew for providing me with feedback and support throughout my work.

A big thanks goes out to my family (Mum, Dad, Herbert, Seppi, Rosi), especially to my parents, who always supported me throughout my education, and backed me up in Altenmarkt.

I want to thank all my friends, for distracting me from work, and for understanding, if I needed to lock up in my room on Friday nights.

I must also mention the scholarship office here, because of their extraordinary service and support throughout the past five years. Thank you!

I also would like to thank everybody who was involved at my semester abroad in Portugal, including the International Office, and all the fantastic professors and friends in Lisbon. Additionally, I want to thank the organizers of the I2C Innovation Curriculum at the TU.

Finally, I want to thank Susanna, for not giving up on inviting me for dinner.

Abstract

The application of an all-embracing project management software system (PMS) within a multi-organizational context is challenging: First, superior “all-in-one solutions” (PM suites) are difficult to assess, costly, and often infeasible for temporary project organizations. Second, the project partners have different local PMSs and expert tools in place, pursuing ambivalent objectives. Forcing them to switch to another system is likely to cause resistance, and disagreement. Third, some PM data, respectively knowledge, are residing at the local organizations and hence are difficult to identify / access by the project manager (a time and trust issue).

Available data integration solutions that (a) address the synchronization of data between local tools, and (b) into a centralized PM view, focus on data integration for large-scale enterprises, and are not appropriate for temporary project organizations. Ad-hoc solutions (e.g, Web-Dashboards) do not provide means to integrate data in a systematic, and traceable way.

This work proposes a systematic integration framework for knowledge-supported project management in IT consortia. The solution addresses integration issues at different organizational levels, is based on design-science, and leverages Model-Driven Software Engineering (MDSE), Semantic Web Technologies, and the OpenEngSb integration framework. The main research outcomes are (a) a *Consortium Negotiation and Integration Process* (CNP) to efficiently enforce the project manager’s integration requirements, (b) the *ITPM Dashboard* that leverages the project manager to efficiently configure, visualize and monitor aggregated PM data in a centralized view, and (c) the *Semantic Model Editor* as the missing link between the negotiation of common data models at the management level, and the technical integration.

The systematic integration framework is applied to an industrial-oriented ITPM consortium use-case (UC), in which four companies develop a joint software product. The configuration results in a set of UML-based data models, work-flows, and SPARQL queries that illustrate the semantic dashboard integration.

The evaluation results indicate a reduction between 25% and 80% of the project manager’s manual integration efforts (depending on the project size and the number of available connectors), and a reduction of 65% for the development of new OpenEngSb connectors. The research outcome is useful for IT project consortia, but also for researchers / knowledge engineers, who experiment with semantic technologies. For the later, the literature review revealed a strong need for efficient data integration and visualization tools.

Kurzfassung

Die Anwendung einer allumfassenden, organizationsübergreifenden Projektmanagement Softwarelösung (PMS) stellt sich als schwierig und herausfordernd heraus: "All-in-One Lösungen" (z.B. von Marktführern) sind schwierig zu evaluieren, meist kospielig und daher für temporäre Projekte unpassend. Zusätzlich verwenden die Projektpartner unterschiedliche lokale PMSs und Expertentools, und verfolgen ambivalente Ziele mit diesen. Diesen Partnern eine Lösung vorzuschreiben, kann auf starke Ablehnung und Widerstand stoßen. Schließlich befindet sich PM Daten und Wissen innerhalb von Organizationsgrenzen, und es ist nicht gewiss, ob, und wie der Projektmanager dieses Wissen erkennt, und darauf zugreifen kann (Zeit- und Vertrauensfrage).

Existierende Frameworks, welche sowohl die Datensynchronisation zwischen lokalen Tools, als auch die Darstellung von PM-Daten in einer zentralen Benutzeroberfläche unterstützen, fokussieren sich auf die großflächige Integration von unternehmensweiten Daten, und sind daher unpassend für temporäre Projektorganisationen. Ebenso wenig eignen sich Ad-Hoc Lösungen (z.B., Web-Dashboards), welchen es an an einer systematischen Integrationslösung, für nachvollziehbaren Datenaustausch, mangelt.

Diese Arbeit stellt ein systematisches Integrationsframework für wissensgestütztes Projekt Management in IT Konsorten vor. Die Lösung adressiert Datenintegration auf verschiedenen Organizationsebenen, basiert auf Design-Science, und setzt auf modelgetriebenene Softwareentwicklung (MDSE), Semantic Web Technologien, und das OpenEngSb Integrationsframework. Die wichtigsten Forschungsergebnisse sind: (a) der *Verhandlungs- und Integrationsprozess für Konsortium* (CNP), welcher den Projektmanager bei der effizienten Erreichung seiner Integrationsanforderungen unterstützt. (b) das *ITPM Dashboard*, welches dem Projekt Manager die effektive Konfiguration, Visualisierung, und Überwachung von aggregierten Projektdaten ermöglicht. Schließlich stellt (c), der *Semantic Model Editor* das Bindeglied zwischen der Verhandlung von gemeinsamen Datenkonzepten auf der Managementebene, und der Integration ebendieser auf technischer Ebene dar.

Das systematische Integrationsframework wird anhand eines industrieorientierten ITPM Konsortium - Anwendungsfalls (UC) durchgespielt und evaluiert. Der UC umfasst die Entwicklung eines gemeinsamen Softwareprodukts mit vier unterschiedlichen Firmenpartnern. Die Konfiguration resultiert in UML-basierten Datemodellen, Work-Flows, und SPARQL Abfragen, welche die semantische Datenintegration anhand des ITPM Dashboards verdeutlichen.

Die Evaluierung des UC zeigt, dass sich der manuelle Konfigurationsaufwand für den Projektmanager zwischen 25 und 80% reduziert (in Abhängigkeit der Projektgröße und der verfügbaren Konnektoren). Der Entwicklungsaufwand für die Neuimplementierung von OpenEngSb Konnektoren reduziert sich um 65%. Die Forschungsergebnisse dieser Arbeit sind sowohl für IT Projekt Konsortien, als auch für Forscher / Wissensingeneure, die mit semantischen Technologien experimentieren, relevant. Für die Letzeren zeigt zeigte die Literaturrecherche, dass ein starker Bedarf in Datenintegrations und Visualisierungslösungen besteht.

Contents

CHAPTER 1

Introduction

The application of an all-embracing Project Management (PM) software system within a multi-organizational context is both, costly and time intensive. Superior industry-solutions are typically difficult to assess, and imply high costs for tailoring, training, and maintenance. In addition, these systems are infeasible for temporary project organizations, such as consortia or research projects, in which local partners want to retain their specific PM systems and tools. During the conduction of the thesis, a software-framework for the efficient configuration and integration of heterogeneous PM data sources is developed, and applied to an industry-oriented ITPM consortium use-case. The hypothesis is that such a framework can significantly improve the efficiency of PM data integration, including data visualization, for heterogeneous project partners with ambivalent objectives, by retaining and utilizing semantic aspects of the underlying data models.

In recent times, IT projects are more and more considered as innovation boosters that help companies to remain competitive [108]. However, it is no secret that projects frequently result in large budget overruns, time delays, and a huge waste of development resources [38]. The inherit complexity of interrelations between heterogeneous stakeholders, project teams, and technology cause IT projects to have a high level of uncertainty, and make them difficult to manage. For instance, business requirements are often ill-defined, and the underlying complexity of IT artefacts (software, hardware, networks) is difficult to assess.

The CHAOS report states a project success rate of only 32% in 2009 [71]. [38] report that one out of six IT projects has a budget overrun of averagely 200%, and a schedule overrun of almost 70%. That is, although a ratio of 25% of the worldwide Gross Product (GP), are spent on projects each year [103, p.3]. Poor communication is a major reasons for IT project failures [49] [97], and appears in various guises, such as unfounded estimation values, or ambiguous communication across teams with different cultural backgrounds, technical jargon, or expert viewpoints. PM software stimulates communication effectiveness by unifying and integrating project data and PM knowledge [98, p.18ff]. However, the utilization of a tailor-made PM software solution is challenging, because the PM software market is very in-transparent, and comprehensive solutions are costly.

Two extreme scenarios are stated to visualize the solution space for software-supported PM in a traditional, hierarchically organized company. In the first scenario, an enterprise-wide PM product is in place (figure 1.1 a). This solution forces the project manager and her teams to adapt to the central software product, with all its benefits and shortcomings. The manager and the project teams are somewhat restricted in using their preferred management and reporting tools, but the data is integrated tightly, and reporting is conducted very efficiently. In the second scenario, the project manager applies a simple spreadsheet solution (e.g., MS Excel) for all planning and management activities[1]). The spreadsheets allow her to reuse proved PM techniques and data, such as pre-formatted responsibility matrices, cost formulas, or a set of wage rates. The degree of freedom is high. However, most PM data must be integrated and reported manually between the teams, the project manager, and the top-management. Hence inconsistent data emerge, and controlling and monitoring provisions are of little efficiency (figure 1.1 b).

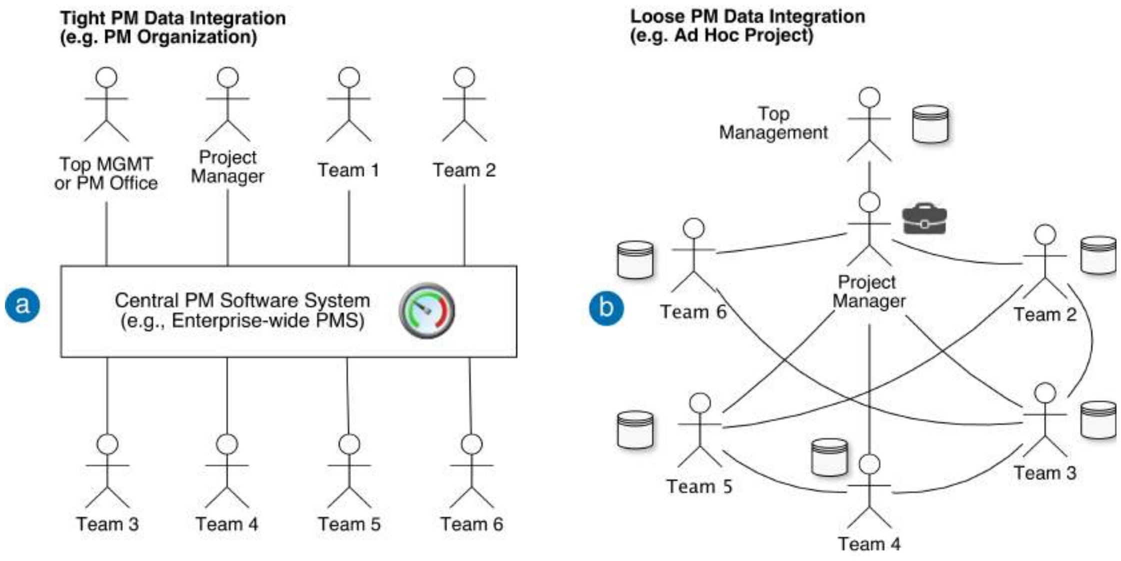


Figure 1.1: Comparison of a tight PM data integration system (a), and chaotic integration (b).

The focus of this work is on *IT consortium projects*. In a consortium, autonomous, heterogeneous organizations bundle their competencies, and form a (temporal) alliance to deal with an innovative, and challenging project. Examples are the development of a new open data standard, or the implementation of a regional ticketing system [86, p.195ff]. *Information Technology (IT) projects* involve using hardware, software, and/or networks to create an innovative product, service, or result [103, p.2]. PM is the application of knowledge, skills, tools, and techniques to guide the project team towards the successful implementation of customer requirements within the boundaries of *scope*, *time*, *costs*, and *quality* [94].

[1]Workshop and expert discussions revealed that spreadsheets are a wide-spread tool for project management (cf. also [103, p.32]

In the context of this work, *PM-related data*, abbreviated *PM data*, is defined as any computer-processable *data*, i.e., *observations or facts out of context* [128], which contributes to the achievement of PM goals either in its raw form, or via interpretation, aggregation, or in combination with other data. The meaningful accumulation with other data leads to the emergence of *knowledge* [128]. Examples for PM-related data/knowledge sources (abbreviated *PM data*) are

- PM software systems (e.g., Jira)
- PM tools (e.g. a simple earned-value calculator)
- structured or unstructured data sources (e.g., team data in spreadsheet format, or requirements in textual representation)
- knowledge sources, such as wage rates assigned to specific team roles
- domain-specific tools or systems that contain PM-related data, such as test management tool that provides test results for a specific software component.

If the scenarios introduced in figure 1.1 are relocated into a consortium environment, then neither of the approaches is efficient. The first approach (a) is infeasible, because each organization brings in its individual, predefined landscape of tools, PM-related processes and competencies. Resistance from one or several consortium members against the acquisition of a specific all-in-one PM product, is likely. The acquisition of a (temporary) but costly all-in-one PM solution with high training costs, and unknown user acceptance is fraught with risk. For the second approach (b), the numerous, repetitive and manual collection, integration and reporting tasks between various consortium members and sub-contractors, are time-intense and tedious for the project manager. In addition, these synchronization overhead detracts her from more important planning, management, and controlling routines.

The two scenarios motivate the need for an efficient and transparent integration framework for heterogeneous, autonomous project partners. Such a candidate solution would be describable as a transparent hood (consecutively referred to as *ITPM consortium hood*) that is put over the entire project consortium; automating tedious manual synchronization tasks between various tools and partners under the hood, and providing customizable reporting and controlling features for the management, over the hood. This solution should be applicable with low set-up costs, and should reinforce the re-utilization of integration artifacts for both, management and technician audiences.

The Open Engineering Service Bus (OpenEngSb) is an open source representative for a systematic tool Integration Framework (IF) [15]. It is actively developed at the Christian Doppler Laboratory for Software Engineering Integration and Flexible Automation Systems (CDL-Flex) (Vienna University of Technology (TUV)), and has been applied to various industrial fields, including mechanical engineering, automation, power engineering, and software development (not PM though). The OpenEngSb supports the integration of engineering and development tools based on *tool domains*, i.e., hierarchical coherent structures that encapsulate connectors based on their functional belonging [93, p.81]. An event-based work-flow engine enables traceability and automation features based on configurable business rules. The individual, local tools of engineers are connected with the bus in order to exchange data within and in-between different tool domains. Thereto, the local data representations, i.e. the *local concepts*, are transformed into abstract representations, i.e., *common concepts*, and aback. The *semantic* mapping, i.e., the preservation of the local concepts' meaning, is facilitated via the OpenEngSb-specific Engineering Data Base (EDB) and Engineering Knowledge Base (EKB) storage technologies [14] [87] [115] [114]. A typical Use Case (UC) for the OpenEngSb is the integration of two different Source Code Management (SCM) tools (Git and Subversion[SVN]). A workflow can be defined so that changes in either repository are propagated to the other repository, and a meaningful notification (email) is forwarded to the responsible team leader, including an assigned message priority, derived from the number of committed Lines of Code (LOC). The example includes *technical* and *semantic* integration challenges. Technically, tool connectors and business rules must be implemented. From a semantic viewpoint, the mapping between the technical jargon of the local tools, and the common concepts, needs to be performed. E.g., the concept of a "commit" in Git and SVN has different meanings. Analogously, the message priorities of the mailing programs have different ranges and meanings.

On the foundation of the beforehand introduced *ITPM consortium hood* and the OpenEngSb framework, PM integration challenges that appear on (a) the managerial, and (b) the technical level, are outlined. Figure 1.2 vividly illustrates (a) and (b), including the horizontal and vertical data flows. *Horizontal* integration addresses the data exchange between local PM systems and tools. *Vertical* integration relates to the meaningful representation of data and knowledge, within a centralized PM view.

Commitment for PM Data Integration (Figure 1.2 - 1) Lack of top-management support is a major reason for the failure of integration projects [77]. In a consortium project, the project manager must convince the executives that (a) a feasible and systematic integration approach is available, (b) the integration solution significantly contributes to the attainment of the project objectives, rather than distracting from essential project activities, and (c) the benefits of the integration approach compensate additional implementation costs. Benefits of PM data integration include more efficient reporting, the automation of manual synchronization tasks, the resolution of miscommunication between sub-organizations, and the effective utilization of local data, PM knowledge, and processes [86].

Analogously, relieving team members from tedious, repetitive reporting/synchronization activities is likely to increase the teams' motivation and commitment. Automated reporting provisions improve the accuracy and timeliness of figures, such as project forecasts (e.g., effort estimations), and the project status. Training and change management efforts appear, if sub-organizations (e.g., local software development teams) are forced to replace their preferred tools against a common Project Management System (PMS). (People) *change management* addresses management activities to dismantle cultural resistances of individuals, teams, and organizations, against a technological or cultural shift [62]. The challenge for the project manager is to minimize training and change management efforts, while maximizing automated report provisions.

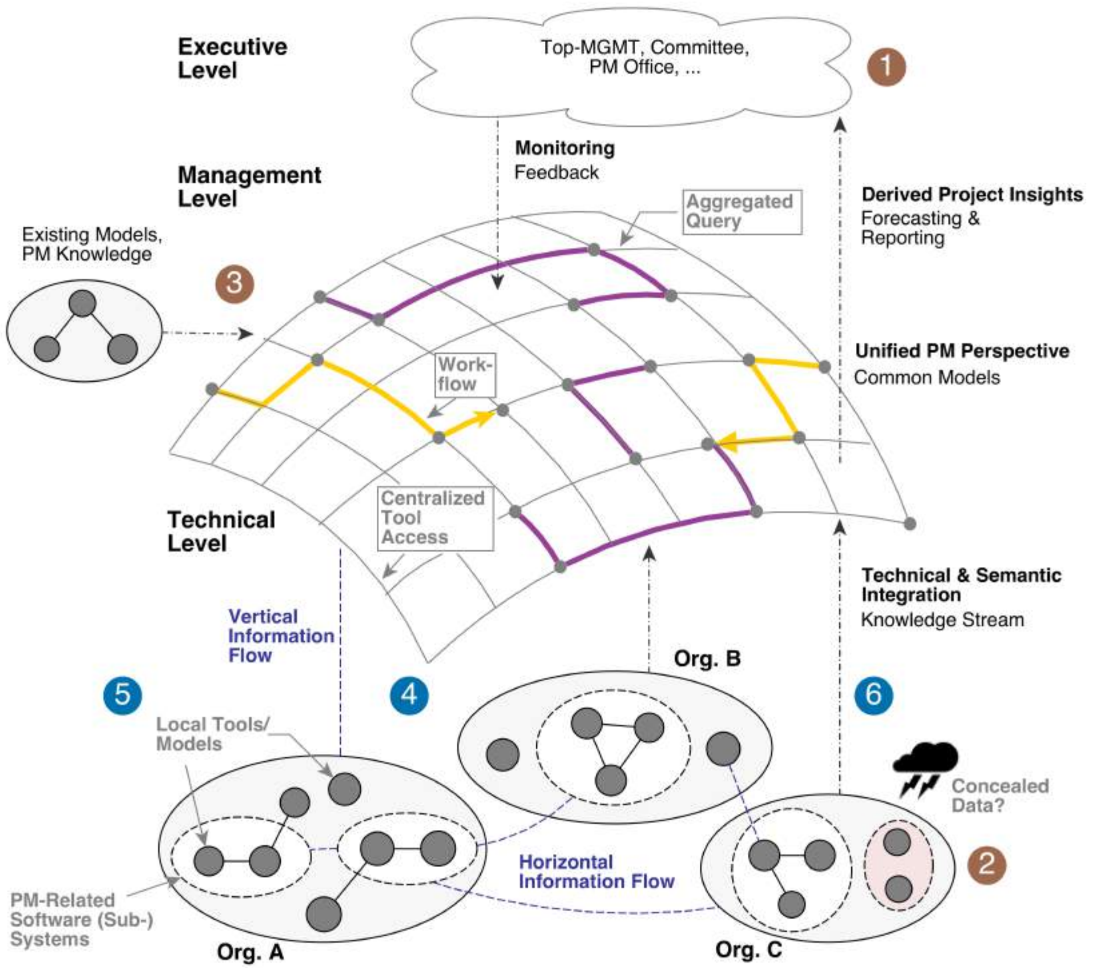


Figure 1.2: The semi-transparent ITPM consortium hood and its fibres, which visualize the capability to define work-flows and queries on aggregated information.

Identification of Data Sources, Models and Automation Scenarios (Figure 1.2 - 2) At the initiation of a consortium project (with new partners), the project manager is unaware of the consortium members' local tools, respectively *PM-related data sources*. Even if basic information about the main PMSs is exchangeable, there is a high probability that important PM data will remain undisclosed. One reason is that the consortium members do not exactly know what data is relevant and valuable to the project manager, and vice versa. Another reason is that local organizations can have incentives to hold back relevant PM data (e.g., administrative reasons, sensitive business knowledge, competitive advantages).

The challenges for the project manager are: (a) Convince the local organizations to actively disclose and share their data (e.g., convince them of the resulting benefits for them). (b) Identify potential automation and data integration scenarios, e.g., for the creation of automated work-flows. If the data sources and their accessible data structures are unknown, the manager cannot assess the costs and the benefits of their integration. (c) Once the available and relevant data sources have been identified, the related data instances should be easy to integrate (low manual effort), and accessible quickly. (d) *Models* abstract data structures and processes, help to condense relevant aspects (from the modeller's viewpoint), and accelerate communication [22, p.1ff]. The challenge is to provide a framework that lets managers and technicians efficiently share and reuse information about data sources and underlying models.

Derivation, Reuse, and Visualization of Knowledge (Figure 1.2 - 3) The derivation of knowledge based on integrated data must be simple. For instance, if a project manager repetitively needs to browse through various sources, and needs to talk to several technical experts of different sub-organizations to gather all relevant figures, which she has to manually combine into an aggregated table, or a formula to create a reporting figure or an estimation, then either (a) the reporting figure will be outdated, or less accurate, soon, or (b) the project manager needs to stage or cancel other relevant PM activities.

Existing knowledge, for instance the project manager's experience from past projects, or PM experience from consortium members, should be reusable and easy to integrate [32]. Analogously, automated work-flows, models, and rules for the derivation of knowledge should be transferable between projects, and other managers. Similarly, the representation of *information* should be persuasive and ideally reusable for reporting activities to the top management, and other stakeholders. Visualizations should support the project manager in monitoring and controlling activities, efficiently. For instance, the centralized visualization of escalations (e.g., the number of open features exceeds a certain threshold, based on an upcoming delivery date) let the project manager quickly explore project deviations that are difficult to detect, manually. The project manager should be in possession to perform basic data value modifications, or tweak reporting figures, efficiently.

The main technical challenges address (a) the efficient technical integration, and (b) the preservation of semantics throughout the synchronization process, including the maintenance of consistent models, accessible and understandable for the project manager and for technicians.

Horizontal and Vertical Integration of PM Data (Figure 1.2 - 4) Local PM data must be be integrated between the individual software systems of the local teams, ideally close to real-time (horizontal flow). In other words, the distributed (engineering) teams should be able to edit and access PM data in their preferred local PM systems and tools. The main challenges are (a) the integration between PM data of the same types (e.g., the synchronization of features or work-packages between two systems), and (b) the exchange and derivation of data for different data types based on work-flows (e.g., for each requirement in IBM Doors[2] an issue should be derived into Jira Issue[3]). (a) includes several semantic integration challenges such as the mapping between local concepts' attributes and id's , and the specification of machine-interpretable, reusable models [114].

[2]IBM Doors: `www.ibm.com/software/products/en/ratidoor`

[3]Jira Issue: `https://www.atlassian.com/software/jira`

Based on the local representations, a common view needs to be derived, as the foundation for the visualization of reporting data (Key Performance Indicator (KPI)s, diagrams, tables, graphs) and project monitoring (vertical knowledge flow). The common view should be configurable by the project manager, and/or a knowledge engineer. The data instances based on the common concepts must be browsable and query-able. Additionally, work-flows between the different types of common concepts should be definable.

The OpenEngSb approach to classify engineering tools into (functional) domains is not directly applicable to the integration of PM-software, because PM data has various cross-functional characteristics which are typically combined within a single PMS. In order to efficiently integrate PM data sources, a classification of PM software types and PM knowledge is required, and the appropriateness of the OpenEngSb must be verified.

Shareable Models / Support of Semantic Web Standards (Figure 1.2 - 5) The semantics of the local and common models should be well-defined so that these models are reusable and sharable across (a) project managers and engineers, and (b) PM and knowledge engineering communities. Ontology Web Language (OWL) ontologies are well-defined and machine-interpretable, but they lack of elaborate graphical model-editor support that represent models compactly, and understandably to laymen. In addition, PM models typically include several hundreds of classes and hence need to be designed and organized transparently [6].

The OpenEngSb facilitates semantic technologies based on the EDB and the EKB, but (a) the models and the data are separated, and (b) maintained in an internal format. Hence, the models cannot be shared, visualized, queried or reused, based on industrial standards (e.g., OWL, SPARQL).

Efficient Development of Connectors (Figure 1.2 - 6) The implementation of tool connectors in the OpenEngSb environment, as carried out by the CDL-Flex in cooperation with industry partners, involves several roles, such as a knowledge engineer for the modelling of the domain data models (UML, OWL), and a developer who is in charge to implement these models and their processing in Java. Additionally, managers provide the requirements for new connectors, and approve the delivered functionality. The challenges include (a) the efficient execution of the development process, e.g., automate the transformation of UML class diagrams into OWL and Java, (b) the consistent alignment of models and code, and (c) the maximization of reusability/maintainability of the models and connectors [28, 90ff.].

The thesis is based on a design-science approach, which is appropriate for complex research areas with practical relevance and incomplete requirements (cf. chapter 4). The central design artifact is the *ITPM Dashboard* for project managers, based on Model Driven Software Engineering (MDSE). The main requirements for the prototype are to let project managers (a) efficiently configure and integrate PM data sources, (b) monitor the synchronization process and the project state, (c) query and visualize data, and (d) automate the synchronization between local data sources so that local development teams can continue to work with their preferred tools. For instance, if organization A is applying *Jira Issue*, and organization B is utilizing *Redmine*

Project Management[4], then these organizations can retain their software systems while working on a joint project. The data instances (e.g., features or work-packages) are automatically synchronized in the background. The Web-based *ITPM Dashboard* User Interface (UI) enables flexible data source configuration, monitoring, and reporting features for the project manager. MDSE in this context means that (data) models are not discarded, but reused for the derivation of additional software artifacts, such as code or UI elements (cf. chapter 2).

The central meta-artifacts of the research are reusable, extensible PM *data models*, and (a) related *negotiation and integration process(es)* for the implementation of these models. The research work examines the required properties of such models and processes to make the integration of PM data in project consortia more efficient. More precisely, it is investigated how models can be used and integrated to (a) collect and unify PM vocabulary, (b) formulate and persist PM-specific structures and knowledge, (c) share and reuse information between managers, technicians, and organizations, and (d) (semi-)automate the implementation of PM connectors, including the querying, visualization, and modification of related PM data.

Based on the initial CDL-Flex requirements, conducted expert workshops, and the available test data, the UC *ITPM in a Consortium* is constructed. The UC describes a typical consortium setting, in which organizations are locally distributed, and have heterogeneous key competencies. The UC is settled in the Software Engineering (SE) domain.

The outcome of the research design process (software prototypes, processes, semantic models) is applied to the UC, and used for the configuration of the integrated ITPM dashboard. As a result, automation scenarios and data models are identified, for which semantic mappings, work-flows, and SPARQL queries are implemented. The scenarios include the following automation cases:

1. The synchronization between similar PM systems (e.g. Jira, Redmine).
2. The automation of integration scenarios, based on work-flows, between different PMS types (e.g. requirements are automatically pushed into Jira and Redmine).
3. The derivation of KPIs, and other status and progress information, based on constructed SPARQL queries.

To evaluate the UC configuration, a test environment, including all PM-related software systems and data sources, is set-up. The evaluation is performed within the prepared test environment, on two different levels:

Cost-Benefit Analysis at the Management Level The evaluation at the management level takes place in form of a cost-benefit analysis. I.e., the time efforts that are saved by automating integration tasks are compared against the additional integration (configuration) effort. Technical development efforts (the development of new connectors) are excluded from the evaluation (it is assumed that all connectors are available at hand). The anticipated result is that the manual integration effort should be reduced by at least 15-20%.

[4]Redmine: `http://www.redmine.org`

Comparison of Development Efforts at the Technical Level The costs for the development of two PM-related connectors is compared against the current OpenEngSb approach. Three connectors (retrieval and storage of *collaborator*, *team*, and *project* data, based on a customer's provided spreadsheets) have been implemented throughout an industrial use-case, in 2012. The development included a basic user interface for data visualization and modification. Because the working efforts have been recorded for the industrial UC, the efforts are accurately comparable against the proposed MDSE-based approach. In this case, the anticipated evaluation result is that the manual development effort can be reduced by at least 30-40%.

The remainder of the work is structured as follows. Chapter 2 summarizes related work, including (a) background knowledge regarding project management (frameworks, PMS, PM knowledge), (b) an overview and comparison on enterprise data integration solutions, and (c) MDSE based on Semantic Web technologies. Chapter 3 introduces the main research questions. Chapter 4 details the design-science research method. In particular, Hevner's *Information Systems Research Framework* (ISRF) is introduced, and the application of the framework for the thesis is outlined, including the evaluation concept. Chapter 5 describes the consortium use-case in detail. Chapter 6 presents the Systematic PM Integration Framework (SIFPM) as the main outcome of the design-science research. It is useful to keep the UC of chapter 5 in mind, although the solution approach is not tied to it. In chapter 7, the SIFPM is applied to the industrial UC. The ITPM dashboard, which is part of the SIFPM, is configured. The output of the configuration serves as the input for the subsequent evaluation. Chapter 8 discusses the benefits and limitations of the work, including threads to validity. Finally, chapter 9 concludes with the main research results, and summarizes future work.

CHAPTER 2

Related Work

In this chapter the theoretical background of the research is presented. Section 2.1 lays out the foundations of project management, by introducing the Project Management Body of Knowledge (PMBOK) as "the" standard framework for project management, and by visualizing typical communication issues in IT consortium projects. Section 2.2 provides a categorization of the various types and definitions of PM software, data, and knowledge. The aim is to (a) enable the discussion of related integration issues, (b) narrow down the research issues, and (c) facilitate the unambiguous referencing of terms throughout the remainder of this work.

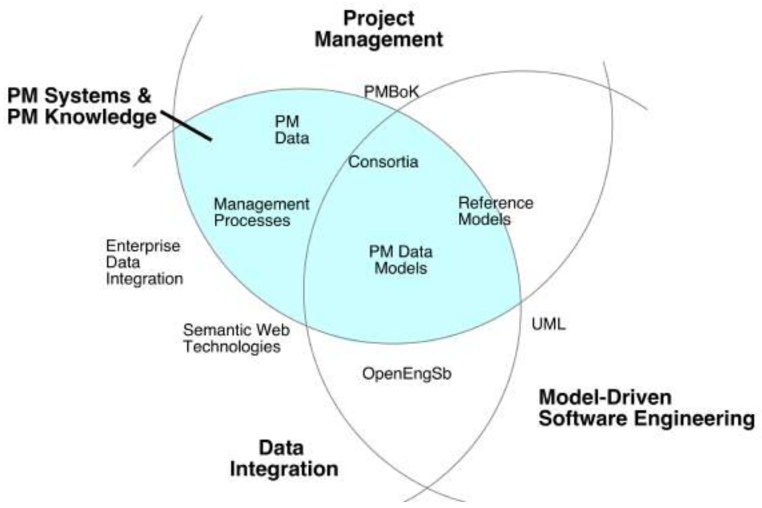


Figure 2.1: Visualization of the Central Research Areas and Related Topics.

Section 2.3 presents different (PM) data integration approaches, and discusses congruities, shortcomings, and complements. The most relevant research items in this section are model-based PM approaches, and Semantic Web technologies. Section 2.4 summarizes the potential of

MDSE, and code generation, to make data integration in IT, respectively SE project consortia, more efficient. Figure 2.1 provides a conceptual overview of the related work structure. The figure illustrates the four main sub-chapters, and emphasizes the central role of PM data models for the research.

2.1 Project Management in IT Consortia

The aim of this section is to provide the reader with a common view on PM terminology, and to exemplify typical issues of multi-organizational IT projects. On that score the PMBOK as a general PM framework is introduced, and typical consortia formations and related challenges are outlined.

2.1.1 Project Management Foundations

A project is *a temporary endeavor undertaken to create a unique product, service or result* [94]. Projects often initiate the development of new products or improvement of existing ones, and hence promote innovation and change. Additionally, projects tend to improve employee motivation, and ease managerial control. In contrast, *operations*, such as a manufacturing or a logistics process, are ongoing, repetitive activities, which focus on efficiency [103] [108].

Project management (PM) addresses the settlement of heterogeneous requirements to satisfy customers needs. It is *the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements* [94]. A project manager is responsible to guide an (often heterogeneous) team towards project success, which is achieved if the boundaries of *scope*, *time*, *costs* and *quality* are not exceeded (the PM Literature recurrently refers to these traditional success measures as the *iron triangle* [11]), and the stakeholders' needs and expectations are met. The *stakeholders* are the people involved in or affected by the project activities, including customers, the project manager, and development teams [103, p.40].

IT project management (Information Technology Project Management (ITPM)) is a subset of PM, and addresses the management of projects which involve hardware, software and/or networks to create a product, service, or any other particular result. Analogously, an *SE project* is an IT project that applies (and constructs) only software products.

A Systems Development Life-cycle (SDLC) divides a project into *phases* in order to make it manageable. The phases are executed sequentially, sometimes iteratively, but once at a time. For instance, the PMBOK divides projects into (1) initiation, (2) planning, (3) execution, (4) monitoring and controlling, and (5) closing. The two main types of SDLC are (a) *predictive life cycles*, and (b) *adaptive life cycles*. Predictive life cycles are appropriate for projects, in which the key performance indicators (KPI), associated to the "iron triangle" (scope, schedule, time, quality), are accurately predictable [103]. This class embraces traditional life cycle models, such as the *waterfall life cycle*, and the *V-Model*. In contrast, adaptive life cycles assume that the project scope can be determined only very limited, and incomplete, throughout early project phases. As a consequence, as soon as the development teams and the customer gain more insight about the project, they want to change and adapt the requirements. Hence adaptive life cycles, in software development often equated with *agile development*, foresee a flexible adaptation of the Project Life Cycle (PLC) after each development iteration. The aim is to maximize the business value for customers during each period. Representatives of agile development methods are *SCRUM* and *Extreme Programming (XP)* [36].

PM frameworks, respectively de-facto standards, such as the PMBOK or PRojects IN Controlled Environments 2 (PRINCE2), are industry recommendations, which describe *what* should be considered by project managers and organizations when performing projects in general environments [88] [94]. In contrast, a *methodology*, such as the SE-related methodology SCRUM, describes *how* a specific standard or best-practice suggestion should be applied to a particular business environment. The PMBOK is applied by 41% of all organizations, and by 11% of IT organizations worldwide, and hence by far the most applied standard [97] [79].

2.1.2 The PMBoK as “the” Industry Standard

The PMBOK [1] is described as a point of entry, and a framework, which is used for the later discussion of PM-related processes, PM knowledge, and PM data models. The framework is especially useful to summarize the most relevant PM competencies.

The PMBOK contains an exclusive subset of general industry PM knowledge. *General* means that the knowledge, respectively tools an techniques, are (a) commonly accepted, and (b) can be applied to any kind of project [94, p.3]. For instance, a critical path analysis, or a work breakdown structure is relevant for civil engineering, bio-medicine, and IT projects. This *general knowledge* is categorized into nine knowledge areas, and five project management process groups (initiation, planning, execution, monitoring and control, closing). In total, a number of 42 sub-processes is constituted (summarized in figure 2.2).

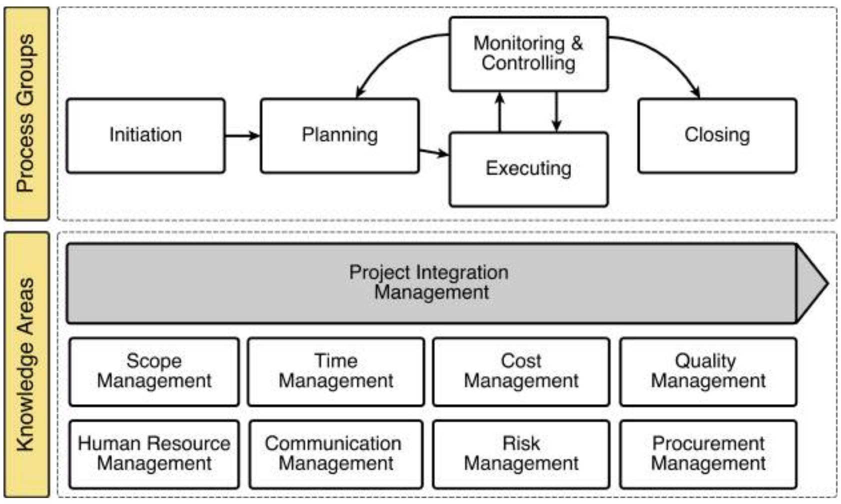


Figure 2.2: Overview of the PMBOK process groups and knowledge areas (adapted from [94])

.

[1]The fourth edition will be referred consistently, because version 5 was not available due to spring 2013. In PMBOK5, a tenth knowledge area for the management of “stakeholders” has been added.

Figure 2.3 exemplifies the process for collecting requirements, which is part of the *scope management* knowledge area. As any other PMBOK process, the “Collect Requirements” process involves *inputs* (from preceding processes), *tools and techniques*, and *outputs*, which serve as the inputs for succeeding processes. For the specific process, the proposed inputs include a project charter, and a stakeholder register. The proposed tools and techniques comprise interviews, focus groups, or facilitated workshops (not stated in the diagram). The process outputs include a requirements documentation, and a traceability matrix.

The PMBOK guide is complemented by several companions and secondary literature. Stackpole’s “A Project Manager’s Book of Forms” is of particular relevance for this work, because it provides a set of applicable reporting forms, which visualize the implicit data models that underlay the PMBOK [111]. For instance, the book includes

- a stakeholder register, including the names and descriptions of the columns / attributes (name, position, role, requirements, influence, etc.),
- a requirements traceability matrix (id, name of requirement, priority, Manifests in WBS deliverable, verification, etc.), and
- a change request form for the exchange of issues,
- a risk register,

in both, electronic (MS Word templates) and copy-able paper format.

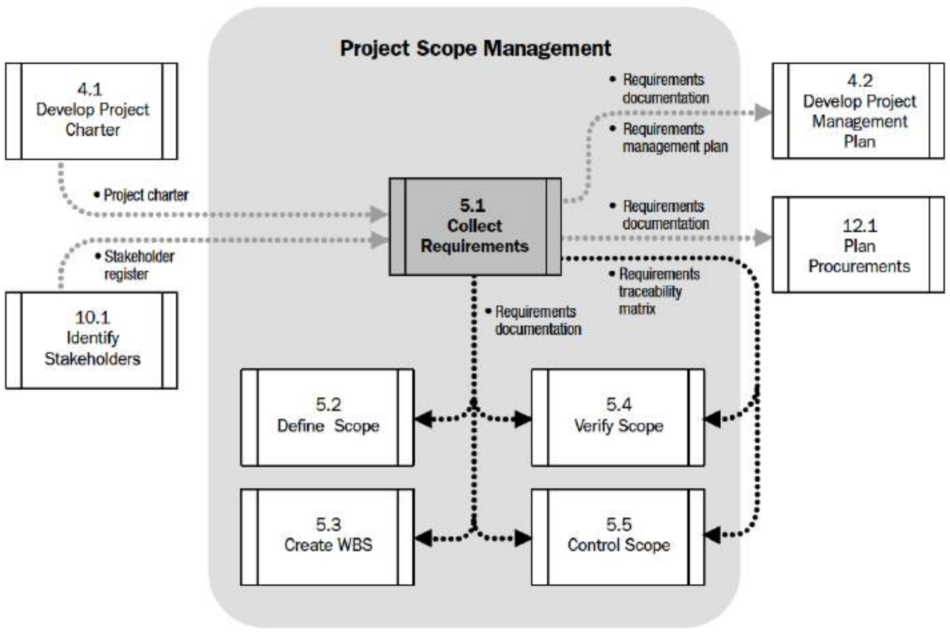

Project Scope Management
4.1 Develop Project Charter
• Project charter
• Stakeholder register
10.1 Identify Stakeholders
5.1 Collect Requirements
• Requirements documentation
• Requirements management plan
4.2 Develop Project Management Plan
• Requirements documentation
12.1 Plan Procurements
• Requirements traceability matrix
• Requirements documentation
5.2 Define Scope
5.4 Verify Scope
5.3 Create WBS
5.5 Control Scope

Figure 2.3: Example of one of the 42 PMBOK (sub-) processes (here: collection of requirements, including inputs and outputs between related processes) [94]

The PMBOK is a functional framework that can help to establish efficient project management and communication in consortia, but lacks of domain-specific methodologies and concepts. More precisely, the PMBOK offers a concise summary of *general PM knowledge*, and is a good framework to ensure the covering of the relevant project management knowledge areas, such as quality management, risk management, or Human Resource Management (HRM). In addition, the unified PM terminology can support the heterogeneous PM consortium partners to communicate "in one tongue". However, as the PMBOK only specifies general PM knowledge, i.e., methods and techniques that are applicable industry-independent, the vocabulary, and the tools and techniques, are quite limited.

2.1.3 IT Consortia Environments

A consortium is a *a group of people, companies, etc., that agree to work together*.[2] In other words, independent organizations bundle their resources to accomplish a joint project, which could only be realized very difficulty, or not at all, by a single party. A consortium typically includes a consortium agreement[3] which sets the terms and conditions for the exchange of knowledge, technology, resources and competencies. Examples of consortia with different scale and purpose are provided hereinafter.

- Between 1967 and 1973 a consortium of leading European aviation companies partnered to develop the Airbus A300, the first twin-engined widebody jet airliner. While the Airbus Division was responsible for assembling the airbus, France manufactured the cockpit. Hawker Siddeley provided the wings, Germany most of the fuselage, the Netherlands flaps and spoilers, Spain the horizontal tailplane, and Rolls-Royce delivered the engines.[4]

- The World Wide Web Consortium (W3C) was founded in 1994 by Tim Berners Lee with the goal to *lead the World Wide Web to its full potential by developing protocols and guidelines that ensure the long-term growth of the Web.*.[5] It includes 387 members such as Adobe, Dell, Facebook, Google and Microsoft.[6]

- The CDL-Flex cooperates with different industrial partners to develop complex automation systems like power plants or production plants. The building of a new system requires the bundling of engineering knowledge from several disciplines and organizations. For instance, mechanical, electrical, software, and civil engineers cooperate in a consortium-like setting, to implement a specific technology for a hydroelectric power station. [115]

- The Open Cities Consortium (OCC) is composed by 12 Core partners encompassing 4 European member states and 4 major cities: Berlin, Amsterdam, Paris and Barcelona. The OCC builds on significant expertise on Open Innovation, Living Labs and Smart

[2]Consortium Definition: `http://www.merriam-webster.com/dictionary/consortium`

[3]EU FP6 Consortium Agreement Checklist: `http://ec.europa.eu/research/fp6/model-contract/pdf/checklist_en.pdf`

[4]Airbus Narrative: `http://www.airbus.com/company/history/the-narrative/`

[5]W3C: `http://www.w3.org/Consortium/`

[6]W3C membership status in January 2014

Cities and existing platforms on Crowdsourcing, Open Data, Sensor Networks and Fiber to the Home networks. The OCC aims to build on both strands of expertise combining an interesting blend of socioeconomic and technological research with real-life pilots in major European Cities [7].

- The Austrian Research Fund (FFG) stimulates the cooperation of startups and research institutions with grants in the range of 10.000 and 12.500 Euros.[8] This is an opportunity for startups to partner up with one or more research or development institutions in a consortium. In a fictional case the startup is responsible for the development of the user front-end and marketing activities, while the research institutions (CDL-Flex, etc.) provide the high-tec components. All partners take advantage from the consortium: The startup receives technology and research results, while the research institutions gain industrial data and reusable prototypes.

2.1.4 Project Management Issues in IT Consortium Projects

The high failure rate of IT projects correlates to a large part with communication issues. IBM and PWC report that poor communication, either by the top-management, within the project team, or with external stakeholders, is a major cause of project failure [49] [97]. The Bull Report, carried out across the French computer manufacturer and system integrator industry in 1998, ranks *bad communication between the relevant parties* with 57% as the number one cause of project failure [29]. Flyvbjerg et al. report that 67% of IT project managers face major conflicts between project and line organizations [38].

Build Commitment for PM Data Integration

Moncrief emphasizes that communication issues are intensified in IT project consortia. Matters of trust, power, and commitment are omnipresent and often lead to a divergence of interests throughout the project. As he argues, it is important to ensure top-management support, establish win-win conditions, and address internal barriers [86].

Varian illustrates that consortia are often not cosy birthplaces of ideas and innovation, but rather constitute rough negotiation processes between self-centric agents with hidden intends. For instance, powerful market players, sometimes even competitors (e.g., Apple, Samsung[9]), team up to resolve inter-dependencies, or to cope with threats from other competitors (cf. [75] for more on classical negotiation theory) [120].

The works of Moncrief and Varian both suggest that an initial “integration effort” is required to (a) build a web of trust between the partners, and to (b) overcome internal barriers, including the integration of PM-related data, which are typically scattered across various local PM systems. (a) focuses primarily on the management activities that are required to build stakeholder

[7]Open Cities Factsheet (p.3): `http://slideshare.net` (last visit Feb. 2014)
[8]FFG Innovation Check: `https://www.ffg.at/innovationsscheck10000`
[9]Apple and Samsung Beginnings: `http://01787.blogspot.co.at/`

and team commitment in early project stages. (b) addresses the efficient, systematic integration of a central, consortium-widely accepted PM (information) system to ensure productivity and transparent communication across team members and organizations. (a) is discussed consecutively. As (b) involves difficult decision-tasks, such as the selection of an appropriate PM software, or the installation of a generic data integration framework, it will be detailed in the sections 2.2 and 2.3) [105].

Hiat and Creasey motivate (people) *change management* to gain the acceptance of individuals, teams, and enterprises in order to establish organizational, cultural, or technological shifts. Changes can either be performed incrementally, or radically, but the ultimate aim of change management is to get individuals out of their 'comfort zones", and to transform them into active "supporters" of a project rather than deniers or obstructionists. Change management in this respect must not be confused with the tracking of requirement updates throughout a project [62].

Boehm et al. introduce *Theory W*, and the collaborative *WinWin* negotiation approach. The *WinWin* approach is based on a spiral model process to negotiate the requirements of a project in a way that turns all key stakeholders into winners (customers, developers, managers). For instance, a useless customer requirement is likely to result in a sloppy implementation and a demotivated project team. If the requirement can be renegotiated, so that the customer recognizes the low business value of the requirement, then a win-win situation is achieved, leaving everyone better off (the customer saves money) [21] [18] [20].

Both approaches help to build commitment between project participants in order to succeed a project. People change management is particularly important in a consortium, because individuals from various organizations, with probably very heterogeneous competencies and cultural backgrounds, need to collaborate with each other. Likewise, the stakeholders need to agree on common PM methodologies, software systems, and data integration, which are critically for the PM to successfully direct a project. However, *change management* is a framework rather than a methodology. It is not explicitly addressed by the PMBOK, and there are no specific guidelines or proposals for its implementation within consortia projects. In addition, change management is associated with costs and effort. Hence it needs to be conducted by the PM very efficiently. The *WinWin* approach is much more precise about the implementation of a methodology for the negotiating of requirements. The authors even introduce a collaborative, software-based prototype (Easy Win Win (EWW), cf. [20], which is however not publicly available. Nevertheless, the the central question that arises from the study of the presented work is: Is it possible to leverage the *WinWin* approach as a methodology to facilitate change management activities in a IT consortia project, e.g. to negotiate a common PM system, and to initially convince the consortium members to willingly disclosure and integrate PM-related information?

Standardized Communication

Inefficient communication channels as well as differences and ambiguities in language, professional competences, and cultures, tempt to lose sight of the relevant project management tasks in order to achieve the joint consortium goals. Mc Connell argues that conflicting interests between stakeholders take place at both, inter-organizational (i.e., the consortium), and organization-internal level (team-internal). As he states, unequally distributed power across organizational levels, frequently lead to poor project forecasts. Mc Connell claims that 40% of all software errors in SE projects are caused by stress. For instance, customers put pressure on functional managers, who force project managers to accept unfounded deadlines. The game continues on lower hierarchical levels: Project managers persuade developers to accept unrealistic schedules, while at the same time developers averagely underestimate their implementation efforts by 30%.

One strategy to homogenize communication is the rigor application of PM standards. The PMBOK provides a list of *communication dimensions* (e.g., internal and external, formal and informal; written or oral), and suggests techniques to actively manage project communication, e.g., via the creation of a *communication plan*. The communication plan addresses the formal communications between project members, but also the technological components and reporting artifacts that are used (e.g., PM software, document repositories; weekly status reports via email). In addition, the PMBOK provides a concise vocabulary, processes, and best-practices that help to resolve and unify ambiguities between organizational cultures [94] [103]. However, the frequent criticism about the PMBOK denounces that it only specifies general PM knowledge, and hence many domain-specific concepts are not addressed (e.g., the PMBOK does not address what a *user story*, a *sprint*, or a *backlog item* is). Moreover, discussions with experts revealed that the PMBOK (at least in Austria) is often disliked, because if forces organizations to adapt to the terminology of a new standard (additional effort), rather than vice-versa. In addition, the PMBOK provides only a scaffold rather than a precise methodology [94] [88].

PM systems are the second tool that help to unify and improve communication via a common platform. According to Shang and Seddon, Information System (IS) (a superclass of PMS), increase productivity and decision-making activities, make communication more efficient, and help to close cultural gaps between organizations and teams. However, they also conclude that IS can be expensive and that a Return of Investment (ROI) can take years [105]. Kannenberg and Saiedian state that traceability *makes project management easier by simplifying project estimates*, because it enables a PM to browse through related artifacts, and to gain more insights about risks, costs, and progress state. However, the authors mourn the poor tool support [74]. Heindl and Biffl provide a good example of "traditional" *requirements tracing*, i.e., the tracing between requirements, test cases, and code [56]. However, their work is a good state-of-the art representative to demonstrate, that most traceability approaches focus on the tracing of requirements from a Quality Management (QM) perspective, rather than a PM perspective (thus reinforcing Kannenberg and Saiedian's proposition about limited tool support) [80] [10].

2.1.5 Concluding Remarks / Approaching IT Consortium Issues

Based on the identified communication issues, two higher-level goals of a PM in a consortium are derived:

1. Build initial commitment to achieve the common project goals, and ensure that all consortium members share their PM-related data. More precisely, none of the partners should

withhold (uncritical) data only because of self-interests, e.g., to avoid administrative efforts. People *change management* and the *WinWin* approach are identified as possible approaches to address this goal.

2. Eliminate communication issues and cultural conflicts that occur due to heterogeneous, (multi-)organizational structures. Related work suggests (a) the rigor application of PM standards and best-practices to provide a common language (e.g., PMBOK). (b) The application of PM software to increase the efficiency of communication by providing (automated) best-practice tools, methods, and a unified and integrated view on project data.

The lack of domain-specific knowledge in the PMBOK is criticized by several authors (e.g., Morris [88], Huijbers et al. [67], and Gasik [40]), and has been found to be shared by practitioners. The criticism of Gasik and Morris addresses the issue that the PMBOK tries to approach any industry domain (e.g., construction engineering as well as bio-medicine, and IT) and hence fails in addressing a specific domain (e.g., IT), properly. From the viewpoint of the author, this criticism is justified. However, from a theoretical PM perspective the PMBOK is still very valuable, because it abstracts many domain-specific details away, which are, at the end of the day, irrelevant for the core PM activities. So what practitioners actually denounce is, strictly speaking, not a shortcoming of the PMBOK per se, but the lack of mappings between domain-specific concepts (e.g., SCRUM: backlog item, sprint, scrum-master) and the generic PMBOK concepts (e.g., work package, project phase, project manager), implying the unacceptably high implementation effort for such mappings. The criticism of Gasik addresses the lack of, what he calls *off-procedural knowledge*, e.g., wage rates, price lists, or skills registers that can be provided either industry-wide, or organization-internal. While this criticism is justified, PM integration solutions can help to integrate such knowledge from secondary sources, or historical data.

Related work shows that PM software is essential for efficient and productive project management. However, the selection of an appropriate PM software product is complex. Among the reasons are the in-transparent software market, the high costs of sophisticated solutions, and the fitness of a product to a specific type of project [105]. Details regarding PMS types and tools are discussed in section 2.2.

A consortium project typically does not require the integration of a single PMS, but rather a heterogeneous set of (already established) PMSs. Hence a *meta-integration approach*, i.e., the "integration of already integrated systems" is necessary. The temporary nature of consortia implies that such a framework must be applicable very efficiently. Potential integration approaches for heterogeneous organizations are outlined in section 2.3.

2.2 PM Systems, Data and Knowledge

The PM software market is vast and in-transparent. Standard literature is not very concise about the definition of PM software systems. Hence, a categorization of PM software is provided, and challenges in selecting and integrating PM software are identified (section 2.2.1). To provide a generic integration approach for PM software, and to establish a link to the PMBOK and

Semantic Web technologies, the discussion is continued at the underlying level of PM data, formats, and PM knowledge (section 2.2.2).

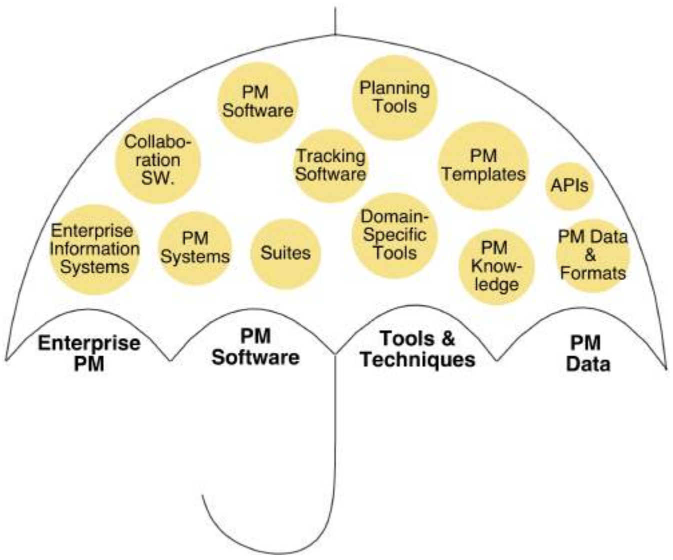


Figure 2.4: The "Umbrella" of PMS Terminology (extracted from literature).

Figure 2.4 visualizes a subset of the scatty PM terminology that frequently appears in literature and vendor-descriptions, under the "umbrella" of PM software.

2.2.1 PM Software

PM software, applied appropriately, boosts productiveness and improves communication effectiveness [98] [105]. However, selecting "the" appropriate PM software product has become a difficult task during the past decades. The nature of PM software has evolved; *from single-user planning tools to complex, distributed, multi-functional enterprise systems* [5].

PM Software Types

The PMBOK defines a *Project Management Systems* (PMS) as *the aggregation of processes, tools, techniques, methodologies, resources, and procedures to manage a project* [94]. In contrast, a *tool* is defined as *something tangible, such as a template or software program, used in performing an activity to produce a product or result* [94].

Kerzner identifies three level of *PM software*. *Level I PMS* provide basic planning capabilities for single projects, but require manual rescheduling of the plans after deviations (e.g., MS Project 2007). *Level II PMS* include additional features for semi-automated controlling, tracking and reporting (e.g. Redmine[10]). *Level III PMS* enable planning, monitoring and controlling for

[10]Redmine: `http://www.redmine.org`

multiple projects, including sophisticated cross-project mechanisms. Typical features of level II and level III software are Gantt charts, report generators, Work Breakdown Structure (WBS) visualization, cost control and resource allocation (e.g., MS Project Enterprise) [76].

Schwalbe defines three categories of PMS, which broadly correspond with Kerzner's levels. However, Schwalbe's categorization is based on the license costs of PMS per user: *Low-end tools* cost less than $200 (e.g., the minimum version of "Basecamp"[11] or add-in features and templates for MS Excel). *Midrange Tools* range between $200 and $600 (e.g. MS Project 2007). *High-end Tools* are those exceeding the $600 limit (e.g., MS Project Enterprise) [103].

Ahlemann and Backhaus identify five categories of PMS, based on the review of 34 industrial PM software products. Single-Project Management Systems (S-PMS) comprise single-desktop solutions. Multi-Project Management Systems (M-PMS) correspond with Kerzner's definition of level III software. Enterprise Project Management Systems (E-PMS) are adaptable, and configurable for institutional enterprise standards. They include Workflow-Management (WFM), e.g. for implementing specific SDLCs. Performance-Oriented Project Management Systems (PO-PMS) extend the functionality of E-PMS, but provide advanced Business Intelligence (BI) and reporting features (e.g., KPIs). Knowledge-Oriented Project Management Systems (KO-PMS) facilitate the reuse of knowledge, experiences and lessons learned. Opposed to Kerzner and Schwalbe, Ahlemann uses the term Project Management Software Systems (PMSS), respectively Project Management Information Systems (PMIS), accenting the enterprise-context in which PMSS operate, comprising organizational and technological components [5] [6].

Ahlemann reports that the functional focus of PMS is still on the traditional features of project planning and controlling, but that the trend is towards collaborative, team-oriented features, and workflow-based mechanisms. The presented PMS classification frameworks do neither reflect these trends, nor do they explicitly focus on the users' needs. Hence, a fourth, *user-oriented* perspective is presented, which bases on standard literature, the CDL-Flex's expert power, and Web resources (cf. figure 2.5)[12,13] [4] [42] [101] [58].

The *user-oriented* perspective comprises four different viewpoints. The *management-centric* view (1) represents the perspective of the project manager, who is mainly interested in applying the software for planning the project, and controlling / reporting to the project sponsors, respectively executives. UI features, availability (e.g. Desktop-based, Web-Client, Cloud-based) and the level of technological integration (e.g., automated data transfer) are of main interest [76].
The *organization-centric view* (2) addresses the needs of the top-management. Not the particular application features are of capital importance, but how the software system is embedded within the context of the organization. The objective is to leverage an efficient, central PMS, which can be reused by all project managers. A main requirement are powerful reporting and controlling

[11]Basecamp: `http://www.basecamphq.com`
[12]PMSdEMO: `http://pmsdemo.com/types-of-project-management-software`
[13]SQA: `http://www.sqa.org.uk/e-learning/ProjMan01CD/page_34.htm`

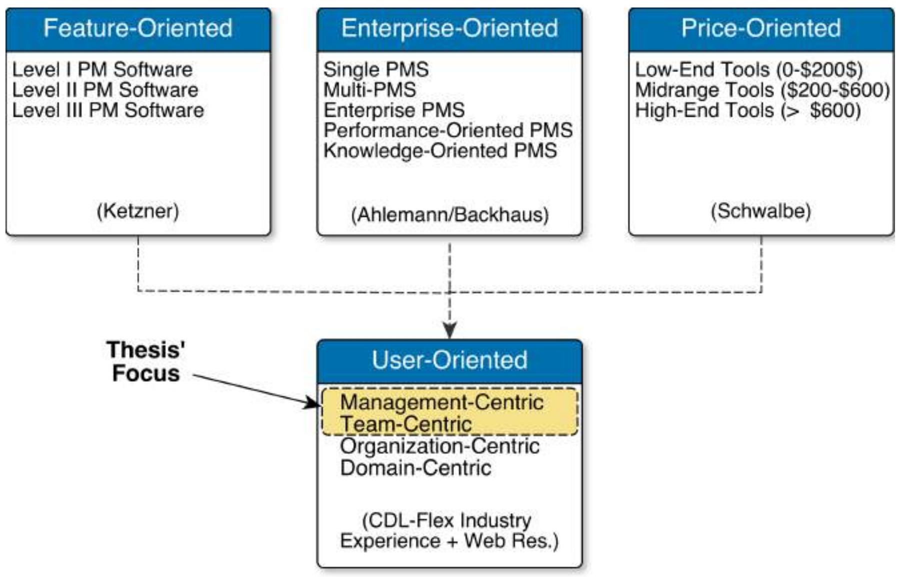


Figure 2.5: Summary of the Studied PMS Categorization Frameworks

capabilities, to aid the Project Management Office (PMO), a portfolio management team, the top management, or other key stakeholders (this view is not focused in the thesis).
While the first two classes address the management level, the *team-centric* view (3) dwells on software that supports project team activities. Project team members typically obtain the *Getting-things-done* perspective, but also require collaboration and tracking features. For instance, in major business projects between 25% and 30% of total project man-days are spent for collaboration [58] [101]. Collaborative agile tools (e.g. boards for SCRUM and/or KANBAN) become more and more important and fit into this category [78] [30] [42].
Finally, the *domain-centric* view (4) stresses that PM software often requires additional features that are only relevant for a specific industry, or an organization. For instance, companies which operate in the domains of mechanical engineering, power engineering, or automated engineering need to be in possession of additional operating schedules, e.g., to regulate a test plant, or to trigger the shut-down of a critical system. Analogously, in SE projects, software tracking systems, such as Jira[14], do not only provide task management features for the ongoing development, but tightly integrate SCM, and code documentation features. The corresponding user role of the *domain-centric* view is the domain-expert, but the domain-specific features are typically relevant for all participating roles, in particularly for the team [89].

Both, Schwalbe and Ahlemann independently report the predominant role of MS Project [103] [4]. Gartner reports a market share of 45% for MS Project (desktop version) for 2010[15] [113]. Additionally, *suites*, i.e., *all-in-one-solutions* that bundle coherent software products from a sin-

[14]Jira Issues: `http://www.atlassian.com/software/jira`
[15]additionaly, MS Enterprise Project had a market share of 12%

gle vendor, are often utilized for enterprise-wide PM [113]. There is a large gap between expensive high-end products and free or open-source solutions. High-quality open-source solutions, such as Redmine, are rare. Schwalbe notes that *these tools are (mainly) developed, managed, and maintained by volunteers. They often run on limited platforms and may not be well supported* [103].

Challenges regarding PM Software

The identified challenges regarding ITPM software address the application of PMS to optimally support the project manager and (virtual) project teams. Heterogeneous tools, organization-specific requirements, and related limitations must be considered. A wide range of commercial off-the-shelf PM software is available on the market. However, the selection of the "best" tool remains challenging.

Selection of PM Software The PM software market is vast and non-transparent. Hence the selection of the "best" PMS requires expertise and proper knowledge of the organizational environment, strategic goals, and a diligent assessment of the intended use. Market studies and vendor's product description are typically biased [5]. High initiation costs (consulting, installation, migration, training), and long roll-outs complicate the application of a central PMS, and make all-in-one solutions infeasible for temporary IT consortium projects [54] [5]. In addition, commercial software vendors (e.g. Microsoft, HP, Atlassian) sell PMS *bundles*, which increase the probability of lock-in, i.e., the costs from switching from one product to another, exceed the expected benefits of the new product [7] [120].

Escape the Project Manager Lock-In Lock-in effects also emerge at the site of the project manager, i.e., if a manager is forced to use a predefined, enterprise-specific product. Inefficiencies result either if the project manager is not familiar with the software, or even worse, if she dislikes it. Additionally, project managers who commonly work in heterogeneous, or inter-organizational contexts, have to jump between different tools [103].

Integration of PM Software and Knowledge Even if large enterprises strive for a fully integrated E-PMS solution, it is very unlikely that a single, coherent PMS, can tightly integrate all PM-related tools from several teams. In a consortium, the partner organizations will not give up their preferred localPMS, easily. Even if a central PMS is accepted, the communication across teams with different fields of scope (and their preferred expert tools) remains limited and cannot take place seamlessly (eg., [15]).
Comprehensible PM suites focus on the centralized management of a project, based on sophisticated UIs. However, these systems' API capabilities are limited. The systematic integration for a larger number of external sources is not sufficiently supported into both directions (typically the integration into the vendor's UI is focused; but the back-stream to other PMS is uncared-for, or rudimentary implemented [4].
Wicks investigated the tool landscape of a telecommunication provider in Scotland, and discovered *integration islands* across the SDLC. More precisely, he found that two SDLC phases, viz.,

defect tracking and change control, and *configuration management* were well-supported by software, while the other five phases (PM, requirements management, analysis and design, implementation, testing) were rarely considered. The same "island" phenomena has been observed for the companies of some interviewed project managers (two Austrian IT-companies in the banking sector, and the CDL-Flex). The 'ìslands" of the CDL-Flex look similar than those of the Scottish telco provider, but additionally the CDL-Flex utilizes a Continuous Integration (CI) server for testing updates in the central code repository. The resolution of the integration networks towards a more balanced approach across the overall SDLC is an important step to improve project communication, and to keep redundant PM artifacts up to date [124].

2.2.2 PM Data, Formats, and Knowledge

A complete definition of *PM-related data*, respectively *PM data* has already been provided in chapter 1. Zack defines data as *observations or facts out of context, and therefore not directly meaningful. Information* is the *result from placing data within some meaningful context, often in the form of a message* [128].

PM Data

PM data, in accordance with the PMBOK, either relates to one of the nine PMBOK knowledge areas, or is domain-specific, but contributes to PM in some kind. The related work about PM software (section 2.2.1) already suggested some types of PM data, such as data about resources, work packages, project members, stakeholders, risks, or requirements.

PM data (in digital form) resides within specific PMS, or tools (e.g., files), and can be accessed via a UI, or an Application Programming Interface (API)s. The UI can include detailed data views (e.g., an overview of all available requirements), and aggregated views (e.g., summaries about the requirement types, and information about the stakeholders how created them; diagrams; graphs). In addition, some PMS track historical data about changes.

PM Data Formats

Data can either be structured (relational database, table format), semi-structured (e.g., XML), or unstructured (textual representations) [65]. Ahlemann reports that the majority of enterprise application integration is undertaken using database interfaces (ODBC or JDBC), or flat file import/export. As he states, *only a few systems offer a documented API* [4]. However, Web-accessible Representational State Transfer Protocol (REST) APIs that provide semi-structured data, either in JSON or XML format, increased over the last years. Additionally, some PMS have been found to implement programmatic access via a Simple Object Access Protocol (SOAP) interface (e.g., Jira). Schwalbe states that spreadsheets (structured data) are a widespread tool for PM. Indeed, there is an own market for MS Excel PM plugins and templates [103].

Knowledge

According to Zack, *Knowledge emerges based on meaningfully organized accumulation of information (messages) through experience, communication or inference* [128]. Knowledge, analogously to data, appears either structured, unstructured, semi-structured, or human-based [64]. It also can be *general*, *domain-specific*, *enterprise-specific*, or even individualistic [94] [88]. *Human knowledge* addresses the expertise of the project manager, the team members, and the other stakeholders, and is not directly accessible. There are several types of *unstructured PM knowledge* sources available. For instance, PM books, best-practices, success (or failure) stories, PM patterns and anti-patterns, are types of *general* and *domain-specific* PM knowledge that help project managers to avoid common pitfalls (cf. [67] [88] [38] [26]). In addition, enterprise-specific (project knowledge), e.g., obtained from past projects, may be available, either in unstructured, or structured form.

Gasik addresses *PM knowledge* in the context of the PMBOK. He distinguishes between *processual*, and *off-processual* knowledge. Processual knowledge relates to the *general knowledge* that is defined by the processes of the PMBOK. *Off-processual* addresses generally-valid *facts*, such as wage rates, skill registers, quality metrics, or reusable report templates, but also enterprise-specific knowledge, such as human-resource registers, or a product database. Gasik terms *Knowledge-Oriented Project Management (KOPM)*, which *focuses on the utilization of the entire knowledge of PM collected by the PM or organization [40]* [40]. In a second paper, Gasik also defines task-specific *micro-knowledge*, and emerging *macro-knowledge*. The later is processed by a single entity, but transferred/accumulated across organizational levels. In this regard, Gasik distinguishes between the *individual level*, the *project team level*, the *organizational level*, and the *global (PM) community level* [41].

Zack classification into *procedural knowledge* and *declarative knowledge* takes a more theoretical perspective [128]. In PM, the procedural knowledge is visible in form of processes, such as *work-flows*, methodologies, or UI-based action sequences, but also in form of templates that stimulate a process [128]. *Declarative knowledge* is typically available in form of *rules*, or *ontologies* [64]. An ontology is *an explicit and formal specification of a conceptualization* [44]. In the context of the Web, ontologies provide a *shared understanding of a domain* [9]. An ontology consists of a finite list of terms and the relationships between these domains antoniou2004semantic. Hence, a *model*, i.e., a purposeful abstraction of the reality, with clearly defined semantics, is also declarative knowledge. Finally, the difference between knowledge and *facts* is, that a fact is *a rule without a body* [64], which equals a tautology. In the context of this work, a fact is defined to be *a possibly abstracted, specific, communicable figure that is assumed to be true*. For instance, wage rates, or the average project execution time of a specific type of enterprise-project, represent facts.

Challenges regarding PM Data and Knowledge Integration

Reuse of (Historical) Knowledge Dippelreiter states that most project managers do not reuse historical data, either because (a) the closure-phase of projects is not conducted properly, or (b) because the data are neglected throughout the init-phase of the new project [32]. Mc Connell notes that ignoring existing knowledge, such as available historical data or lessons learned from past projects, result in negative project outcomes. However, he finds that unfounded estimations, e.g., based on solely personal memories, are used for about 60 to 85% of all estimations, although the application of documented facts is negatively correlated with project overruns [83]. The reuse of knowledge implies that the knowledge must be transferable, and reusable, e.g., based on rules or models.

Derivation of Knowledge (Access to Data) Data is often available in text documents, spreadsheets, or within tools, but cannot be reused automatically, because of a lack of integration. As Ahlemann and Backhaus report, knowledge-based features, such as elaborate document repositories, or the documentation of lessons learned, are only available for very costly Knowledge-Oriented PMS (KO-PMS) [5]. The integration of data and knowledge should imply that knowledge easily accessible, and adequately pre-processed/represented for each project member, for instance in the local tool of the developer, or vividly aggregated as a KPI, in the central view of the project manager.

As this thesis aims to improve the integration of heterogeneous PM software components and data, it is important to be aware of the essential types of PM software, and the underlying data/knowledge structures. The research findings in section 2.2 support (a) the identification and isolation of a realistic project setting, (b) the effective inquiry of practitioners, based on a common language, and (c) the design a solution which is applicable / suitable for other software types, analogously.

2.3 PM Data Integration

Analogously to IT project management, data integration comprises a complex set of technical, organizational, and business components, which make it a hard problem. Main data integration issues include (a) the efficient integration of a larger set of heterogeneous, autonomously organized data sources[16], (b) the *mapping* between heterogeneous data (formats) in order to exchange data consistently, and (c) the access and useful visualization of the integrated data [52].

The aim of this section is to study related work for the efficient integration of PM data within temporary project structures, considering semantic aspects of the data. For this purpose, holistic, reusable, ideally lean approaches that enable the (centralized) synchronization, querying, and data visualization, are examined. Related work that addresses reusable models, ontologies and/or Semantic Web technologies, is of particular interest.

2.3.1 Data Integration

Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data [81]. From a theoretical viewpoint, data integration is

[16]already between 2 and 5 data sources can be inherently complex to integrate.

a set of mappings (i.e., queries) between a global (virtual) schema G, and a set of local source schemes S [81].

Doan et al. provide four main reasons that make data integration a hard problem [34]:

1. The data of disparate data sources must be queryable (and modifiable), ideally in real-time.
2. The number of sources complicate the integration. The integration challenges exacerbate as soon as the number of sources grows (already two sources can be challenging).
3. The accessed data sources and platforms are heterogeneous. Different file formats and API technologies need to be integrated. In addition, the semantics, i.e., the meaning of the sources' schemata and data are ambiguous and must be resolved.
4. The integrated sources are autonomous, which means that they are possessed and administrated by different organizations. The sources can change their data formats and access patterns at any time.

Sunindyo et al. classify related work into *technical* and *semantic* integration. Technical integration focuses on architectural aspects of the framework, including the systematic and efficient integration of disparate data sources. For instance, Sunindyo et al. apply the OpenEngSb (introduced in chapter 1) as a distributed integration framework for automated monitoring and tracing activities of SE-related build processes. As they state, the event-based architecture of the bus facilitates a proper separation of the integration logic (program code) and the business logic (work-flows) [114]. Semantic integration addresses the matching of disparate, local data schemes, the detection (and prevention) of data inconsistencies (such as duplicate records), and the modeling and mapping of complex relations in, and between vocabularies of heterogeneous data sources [114]. Semantic mappings help to prevent the *meaning* of data, when exchanged across different data sources and views.

In addition to Sunindyo et al., Halevy distinguishes between *schema heterogeneity* and *data heterogeneity*. *Schema heterogeneity* addresses ambiguities on the scheme-, respectively model-level. For instance, the same attribute may exist in two different schemes, but with a different name. Another example is the accomplishment of a mapping between two different ordinal data scales (e.g., priorities). *Data heterogeneity* addresses mismatches that a model can reflect only very difficulty. For instance, two data sources contain names of the same users, but the stored names vary slightly (e.g., “Andreas Gruenwald”, “Andreas Grünwald”, “A. Gruenwald”) and hence cannot be mapped unambiguously [50].

2.3.2 Semantic Web Technologies

Semantic Web Technologies are a set of W3C standards and recommendations that aim to resolve semantic integration issues on the foundation of machine-interpretable and shareable ontologies [9]. OWL ontologies, and SPARQL queries are the most relevant semantic artifacts for this work.

OWL Ontologies

The *Ontology Web Language* (OWL) enables the formalization of knowledge within ontologies. More precisely, classes, properties, and advanced logical relationships between these concepts can be modeled and persisted (e.g., hierarchies, domain restrictions, functional relationships). In addition, OWL ontologies can be populated with *individuals*, i.e., instances, that represent case-specific facts. Antoniou summarize the main benefits of OWL ontologies. First, OWL bases on a stack of W3C markup languages (e.g., Resource Description Framework (RDF), and XML), which makes the ontologies serializable, sharable, and (machine)-interpretable. Second, OWL rests on well-understood formal semantics (declarative knowledge), which enables reasoners to infer conclusions from the given knowledge, automatically. In addition, Noy et al. report that the design of ontologies supports people (but also software agents) in explicitly building, organizing, and sharing a common understanding of domain knowledge [9] [91].

There are several shortcomings of the OWL and ontologies. First, there is a trade-off between a reasoner's runtime complexity, and the expressiveness of the OWL dialect (the thesis does not focus on these aspects; OWL2 DL is consistently referred throughout the remainder). Second, the capabilities of the non-commercial reasoners are very limited. Third, rules in OWL are *monotonic*, which means that no new individuals/entities can be derived based on reasoning (some vendors address this issue by providing vendor-specific extensions; cf. StarDog[17]). Fourth, the storage technologies are still in their infancy. More precisely, the capabilities of the available Open Source (OS) solutions (e.g. Jena[18], OWLAPI[19]) are very limited, with respect to performance, stability and documentation. Fifth, the available ontology editor support is limited. For instance, it is very difficult to model / visualize an ontology compactly, e.g., such as engineers are used to express data models in compact UML class diagrams. In addition, expert know-how must be acquired to operate the complex UIs, and to design consistent ontologies. Bizer, Heath, and Berners-Lee identify the weak ontology editor support as a main research challenge of Semantic Web [17]. For the remainder of the work, Protégé[20] as a state-of-the-art ontology editor is referred [64] [65].

SPARQL and Linked Data

Linked Data is an excrescence of the ongoing *Semantic Web* research, and has emerged from the necessity to access, combine, and query ontologies from different sources. The main relevant difference between Semantic Web technologies, and Linked Data is that the latter is based on RDF and SPARQL, rather than on OWL (OWL builds on top of RDF though). The SPARQL is the query language for RDF data and is based on graph patterns and sub-graph matching. Contrary to SQL, a SPARQL query is constructed of basic graph patterns which are composed of triple patterns containing query variables at the subject, predicate, and object position [55]. In addition, SPARQL is designed to also query the model elements (the terminology, respectively

[17]Stardog: `http://www.stardog.com`
[18]Jena: `http://jena.apache.org`
[19]OWLAPI: `http://owlapi.sourceforge.net`
[20]Protege: `http://protege.stanford.edu`

T-Box) rather than only the data instances (the assertions, called *A-Box*). For instance, a query could ask for all sub-classes of a particular ontology class. SPARQL closes the gap of a powerful, missing query language for OWL [17] [95] [17] [64].

Semantic Storage Technologies: OWLAPI and Jena

The parallel evolution of OWL-centered technologies and Linked Data makes the selection of an appropriate technology a difficult task. OWL provides better tool-support, has a more expressive syntax RDF, and is appropriate for reasoning. Several reasoners OWL (e.g., Pellet [92], and HermIt [107]) are available. SPARQL has powerful capabilities to load and query data from distributed sources, overcomes the monotonic nature of OWL rules (e.g., SWRL), but Linked Data lacks of powerful reasoning mechanisms. As Polleres argues, SPARQL is an efficient rule language by itself [95].

The differences between OWL and RDF-centric approaches are reflected at the level of the specific data storage solutions, including their APIs. The main shortcoming of the OWLAPI is that the data is persisted in serialized files. DB-mapping approaches, such as the *OWLDB*, introduced by Henss et al. [59], or by Redmond [100], have been found incompatible with the latest version of the OWLAPI [3.4]. Additionally, the OWLAPI does not support SPARQL natively. There are working plugin solutions available, though (cf. Sirin et al. for [109]).

Jena, on the other hand, is RDF-centric, and provides only limited OWL-support. The main advantages of Jena are that data can be stored in scaleable data stores (Jena TDB or SDB), and extensive SPARQL support is available. The disadvantage is that there are no UI tools to browse, visualize, or administrate the persisted data [66] [69].

2.3.3 PM Data Integration Approaches

Currently no research solution addresses the all-embracing integration of PM data and knowledge for temporary project organizations, respectively IT project consortia, sufficiently. Nevertheless, several options to deal with the synchronization of heterogeneous data between local organizations exist; ranging from primitive, manual synchronization, till the reuse of comprehensible, large-scale enterprise integration frameworks. Related work is summarized in figure 2.1, and detailed subsequently.

Research Item	**Relevance**
Traditional PM Approach: No Integration	Low
All-In-One PM Software	Low
Dashboard-Based Ad-Hoc Integration	Some
SE Tool Integration Frameworks	Medium
Data Warehousing Management Processes	Some
Model-Based PM Integration	High
Semantic PM Approaches	High

Table 2.1: Main research items for PM data integration, and their research relevance.

Traditional PM Approach: No Integration

The traditional PM approach does not address the automation of data integration (trivial case). The project manager works with her preferred (and available) tools, and must manually communicate instructions and status updates (e.g., via e-mail, or by adding updates to the teams' local PMS). Likewise, reporting data, such as status reports and time-sheets, are communicated manually between (possibly several) hierarchical levels.

This solution can provide the PM and the teams with a high level of freedom. For instance, the PM can apply MS Project, MS Excel, or any available PMS, or community template[21], to create and maintain PM artifacts (e.g., a project plan). Analogously, the local teams are allowed to stick with their familiar PMS, such as project trackers, or agile PM software. However, the PM can be also restricted, for instance if the organization denies the provision of demanded licenses. The disadvantage of this approach are the high manual synchronization and coordination efforts that distract the project members from their assignments. At the same time, the communication is inefficient, because the heterogeneous PMS imply ambiguous vocabulary and processes. Moreover, there is no systematic way to ensure that the PM and the teams reapply (available) organizational standards and knowledge, and that the learned experiences "flow back" into the organization, facilitating its reuse for future projects. At the end of the project, the planning artifacts are typically out of date. Additionally, as Dippelreiter states, there is little time left to collect and pass the learned lessons, in a reusable form [32] [103] [88].

All-In-One PM Software

The application of an all-in-one PM solution means that a central PMS is in place, which is accessed by the PM and all project team members (e.g., engineers). Instructions, progress states, insights, and other PM data, are exchanged via a central UI, respectively system.

The complexity of all-embracing PM solutions has already been addressed in section 2.2. The main problems are the high selection and rollout costs (e.g., selection, migration, training, configuration, licencing), lock-in hazards, and the uncertainty how to systematically integrate inevitable systems and tools of local consortium partners.

This thesis does not evaluate commercial software products because, as already said, the biased product descriptions and the nontransparent market, prevent well-founded scientific statements. Instead, the discussion of related issues bases on two (lightweight) PMS solutions, viz., *Redmine* and *Jira Issue*. As Redmine is an OS solution, a free demo version was available for detailed studying. *Jira Issue* is applied by the CDL-Flex for industrial projects, so extensive expert knowledge about the product was available.

Redmine Redmine is a high-quality OS-PMS. Its main features are project planning (e.g., Gantt Chart), issue tracking, and document management (e.g., Wiki), including a user management system. In addition, Redmine supports the integration of third-party tools via plugins (e.g.,

[21]OpenPM: `http://www.openpm.info`

integration of SCMs, messengers, or agile methods[22]). While Redmine aims to provide useful features for efficient project management, and is based on a intuitive, Web-UI, there is no systematic approach to integrate other PMS. More precisely, the UI-centric integration approach only works well as long as the number of integrated tools is small (e.g., 2, 3), and the integration only addresses the visualization of data within the Redmine UI. If the number of tool increases, and complex cross-domain workflows are required, then Redmine does not scale [82].

Jira Issue Jira Issue is a popular commercial PMS for planning and tracking projects in teams[23]. Jira provides similar features as Redmine. It allows the creation of embeddable plugins, and provides accessible APIs (e.g., REST, SOAP). Analogously to Redmine, this approach is sufficient for a small set of point-to-point communications, but as soon as the number of integration sources increases, the approach does not scale. The flexible pricing model of Jira is noteworthy. The cloud-based PMS is free for single-users, but the license costs increase with the number of added users, and installed plugins. The application of Jira as a central PMS implies two problems: First, learning efforts appear due to the change of all members to a single platform. Second, the implementation of missing plugins (those not available within Jira's central plugin repository) can be costly.

Jira very visually exemplifies the hazards of lock-in: As the numbers of users and plugins grow, more and more data is pushed to Atlassian, and, due to the multiplicity of plugins, is scattered across various places. As Jira also provides a tightly integrated source code repository (Bitbucket[24]), and a Requirements Management (RM) platform (Confluence [25]), the more plugins and data are added to the system, the more difficult it will become to migrate to another system [120].

Dashboard-Based Ad-Hoc Integration

Another integration approach is to take the existing PMS environment for granted, but initiate data integration activities on demand, e.g., when manual synchronization effort become unbearable, or when monitoring or reporting efforts are very inefficient.

Several innovative and lean integration approaches, mainly based on Web-Technology, are available. One key technology in this regard are HTTP-based protocols, particularly the REST protocol. Currently, various PMS and tools implement a REST API. These interfaces enable the efficient retrieval of data in semi-structured format (e.g., XML or JSON), and the triggering of basic, tool-internal actions, such as the creation of a new users or work items.

The integration of two or more tools based on REST requires a central application, such as a *dashboard*. The dashboard is used either to integrate the data within a central view, or to trigger the execution between heterogeneous tools. The central application is necessary, because two

[22]Redmine Backlogs Plugin:`http://www.redminebacklogs.net`
[23]Jira Issue: `www.atlassian.com/jira`
[24]Bitbucket: `http://bitbucket.org`
[25]Confluence: `http://www.atlassian.com/confluence`

(or more) tools cannot exchange data without a mediator.

There are several innovative dashboard-approaches available that ease the visualization of data based on prepackaged, customizable templates. *Twitter Bootstrap Dashboard Themes*[26] provide prepackaged visualizations, including elements for the visualization of project progress, such as KPIs, tables, graphs, diagrams, and notification items. The templates are based on CSS and are typically available for less than $30.

Programmableweb[27] collects numerous of API implementations and *mashups*, i.e., dashboards that integrate and combine different HTTP-accessible data sources (either web-services or websites). These mashups provide higher-level views for managers and other roles. A typical mashup example would be the integration of social network streams (e.g., Twitter, Facebook) and Google Analytics KPIs, to represent a Website's, or a marketing product's regional popularity based on a central map. There are few mashups available for project management though.

Yahoo Pipes allows the mixing of popular data feeds to create data mashups via a visual editor. A pipe is a data processing pipeline (hence the name "pipe"), which consists of one or more data sources (e.g. RSS/Atom feeds or XML sources) and a set of interconnecting operators, each of which performing a specific task. There are operators for manipulating data feeds (e.g. sorting or filtering) and operators for features like looping, regular expressions, or counting. More advanced features such as location extraction (e.g. geo coordinates identified and converted from location information found in text fragments) or term extraction (e.g. keywords) are also supported. The goal of Yahoo Pipes is to enable users to design data processing pipelines that filter, transform, enrich, and combine data feeds and are again exposed as XML streams [127] [96].

One approach that tangents PM is provided by Biffl et al. The authors utilize semantic technologies to extract PM-related data from OS Web communities, and summarize the aggregated data in a centralized cockpit. While the visualization results are convincing, this approach, as many others, does only consider the extraction of data, but not the modification, i.e., data backflows. In addition the cockpit is not flexibly configurable [16].

While dashboards and mashups are very efficient for data visualization, they are mainly "one-way" integration solutions. In other words, they are good for visualization, but are not appropriate for complex synchronization activities. These systems neither provide a systematic integration approach for larger amounts of data sources, nor do they provide stable mechanisms to "control" the access and the data flow of sensitive data sources. In addition, it is very difficult for the PM to gain access to a local data base, administrated by a different organization or subentity. Moreover, it is very unlikely that an administrator will grant *write permission* to access a central PMS. The reason is that adhoc-integration jeopardizes the mix-up of entire databases in a non-reconstructible way. More precisely, no *journal* is maintained that summarizes which

[26]Bootstrap Dashboard Themes: `https://wrapbootstrap.com/tag/dashboard`

[27]ProgrammableWeb: `http://programmableweb.com`

events where executed by whom.

Enterprise Information Integration (EII) is a trend that relates to mashups and Web-dashboards. EII *provides uniform access to multiple enterprise data sources without having to loading them into a data warehouse* [27]. According to that definition, EII represents the missing link between the centralized management dashboard, and the flexible integration of a number of decentralized data sources. However, EII addresses the overall entire enterprise context, and has not been applied for more small-scale (temporary) PM organizations so far. Moreover, EII has been found to be a paradigm, rather than a ready-to-go solution [27] [70] [52].

EII based on Linked Data is probably one of the most promising and tangible approaches: Linked Data facilitates the querying of distributed data sources based on RDF and SPARQL. More precisely, the data sources are wrapped into autonomous Semantic Web services that provide access to the underlying ontologies. A central SPARQL query can then be leveraged to access, load, and query the distributed sources based on RDF and the http-protocol. The disadvantage of the Linked-Data approach is that there are no endeavors to integrate large sets of (PM) data in a systematic, traceable way. In addition, the configuration of a set of autonomous (Semantic) Web services takes effort [70] [72] [51].

SE Tool Integration Frameworks

Tool integration frameworks focus on the systematic cross-linkage of larger, heterogeneous sets of (engineering) tools. They represent complex *middleware* systems, consisting of sophisticated messaging, mediating, and event handling components.

The historical impact of SE is observable within nowadays' tool integration solutions. Most frameworks address a set of similar issues, such as the integration between RM, QM, and (software) engineering tools, and the traceability of related artifacts. This work does not investigate the wide-ranging integration approaches from various industrial domains (cf. [57] [10] [123] [125] on more details). Instead, the OpenEngSb is used as a representative state-of-the-art framework for the SE domain [14] [16].

The OpenEngSb The high-level architecture of the OpenEngSb is explained based on a typical bus configuration in a SE project setting (cf. figure 2.6).

The service bus component (1) enables the dynamic deployment of any type of connectors, and the systematic message passing between these. For instance, security provisions are considered. The EDB is an integral part of the service bus, which is leveraged to persist any kind of user (access) data, tracing information, or *common concepts* that are exchanged between tools (for instance data about software bugs). The EKB (2) is leveraged to store specific semantics (i.e., metadata) about common concepts. The work-flow engine (3) enables the configuration of complex work-flows to exchange data between different tools (based on Drools[28]). Finally, the event engine (4) processes and schedules events that are thrown by any registered tool connector, or component.

[28]Drools: `http://www.jboss.org/drools`

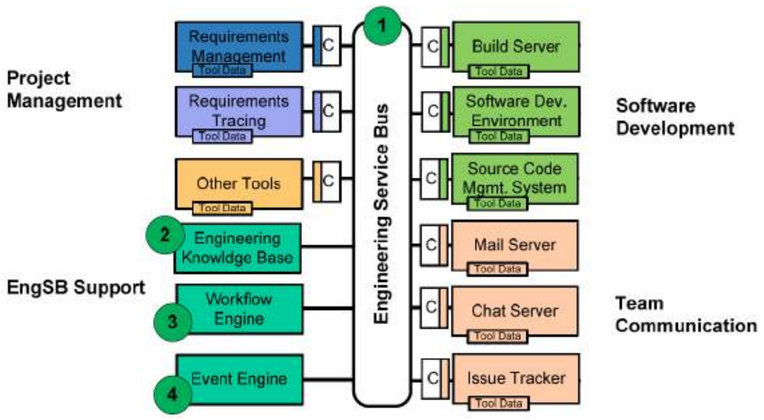


Figure 2.6: High-Level View on Tool Connections with the OpenEngSb [114]

To integrate a new tool (e.g., a bug tracker) a new connector must be implemented. *Tool domains* enable the generic access to similar types of tools. Each connector belongs to a specific tool domain (e.g., PM, SE, team communication). The tool domains abstract the implementation details of specific tools, and enable the generic access to similar types of software (e.g., Mantis and Bugzilla are both issue tracking programs). Work-flows are used to exchange data across different tools and domains, based on triggered events.

The main strength of the OpenEngSb is the systematic integration, based on structured *tool domains* and work-flows. The event-based approach ensures that all registered actions become traceable. Traceability is a main advantage compared with ad-hoc integration solutions, and particularly important for safety-critical applications [74].

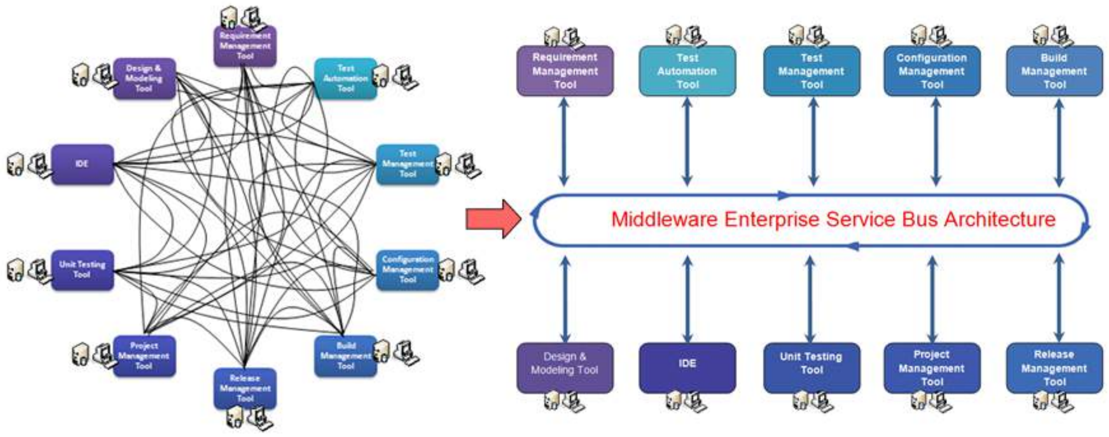


Figure 2.7: Chaotic vs. Systematic Tool Integration (adapted from Kovair [2])

The resolution of a chaotic tool integration based on the application of a systematic integration approach, based on tool domains, is visualized in figure 2.7. The figure is adapted from Kovair's *Omnibus integration platform*, but complies with the approach of the OpenEngSb [2].

The current shortcomings of the OpenEngSb are that the platform requires significant installation and configuration efforts, and high training costs for newcomers. While the framework is well-suited for system-critical applications and mature enterprise applications, the development of innovative prototypes is not very efficient. Among the reasons are: (a) The implementation and execution of (Java) unit tests is very time-consuming. (b) Although the OpenEngSb provides a systematic UI integration approach (based on Wicket), the UI must be implemented manually. There is no possibility for managers to customize their view, e.g., via arranging items in a dashboard. In addition, each single UI update requires a restart of the related OpenEngSb component, which is ineffective and unfeasible for the development of JavaScript-intense applications. (c) The OpenEngSb has not been applied for PM so far. Hence the fitness of the *tool-domain* concept, and the implementation of new PMS connectors, must be verified. (d) The bus currently provides no permanent synchronization mechanism. I.e., there is no mediation approach for the ongoing, permanent data exchange between the local tools. (d) The OpenEngSb does not implement Semantic Web standards, such as OWL or SPARQL. As a result, (external) knowledge engineers cannot query (and analyze) stored data models, and instances.

PM Integration Tools In addition to the OpenEngSb, two (simple) industry tools for the integration of PM data are identified. The first one is *Task Adapter*[29] and provides a set of synchronization mechanisms (e.g., between Basecamp, Jira, MS Project, Redmine). A demo version of the product is downloadable. The synchronization is based on very basic UI mappings between fields of different PM artifacts. For instance, MS Project tasks (summary, description, start date) can be transferred into Redmine or Jira. The tool provides no automation mechanisms and must be triggered manually. *The Connector*[30] provides data integration between Jira and MS Project. The tool is similar to *Task Adapter*, but much more limited.

Data Warehousing Management Processes

Data warehousing (DWH) is a comprehensible, heavyweight solution for the integration and analysis of large amounts of enterprise data. DWH is not appropriate for PM per se. However, it addresses management issues that analogously appear in heterogeneous project environments. As a consequence, some Data Warehouse (DWH) approaches help to inspire the development of innovative PM integration solutions, including relevant management aspects.

Integration Management Process Projects with heterogeneous, autonomous partners, have to deal with similar integration issues than DWH projects, especially in case of consortium projects (cf. section 2.1). McFadden describes a typical high-level management process for a DWH data integration [84]:

[29]Task Adapter: `http://www.taskadapter.com/`
[30]The Connector: `http://www.the-connector.com`

1. *Requirements Analysis*. Members of the executive teams must be brought together to determine the main goals and requirements of the integration process. The step includes the identification of the most important (distributed) data sources, data entities, and attributes.

2. *Data Model Design*. The data model of the central data store is designed. Entity Relationship (ER) or UML class diagrams are suitable.

3. *Data Mapping*. The local data sources and the global scheme must be mapped.

4. *Data Management*. The revised collection, distribution and synchronization of data must be established.

5. *User Interface (Customization)*. A user (management)-centric UI must be provided to guarantee the usefulness of the central data store, including predefined information visualizations, ad-hoc query tools, and support for the analysis of aggregated data and trends. Han et al. name some additional requirements for the UI: high flexibility, end-user autonomy, and the visualization of data on different levels of granularity (associated with *drill down* and *roll-up* in DWH) [53].

ETL Process The Extraction Transformation Loading (ETL) process is the standard process for the integration of data into warehouses. It comprises (a) the extraction of data from the local sources, (b) the transformation of the data into the central format, including data cleansing (resolve inconsistencies, deduplication, etc.), and (c) the loading into the central warehouse (aggregation of data, etc.). Setting up a transformation process for a DWH is a time-consuming task. However, there are also small-scale solutions available that enable ad-hoc integration. RapidMiner[31] provides extensive integration, transformation, and analysis features based on an intuitive graphical UI.
ETL processes are not further considered for several reasons. First, most ETL processes provide only "one-way" synchronization. I.e., the data flows from the local data sources into a central warehouse, but not vice-versa. Hence, real-time synchronization across multiple tools is not considered. Second, a ETL does not fit into the architecture of the OpenEngSb. Third, ETL is a very *process-oriented* approach, while ontologies aim a more model-based perspective [52].

Model-Based PM Integration

Reference models are reusable and communicable items that help to accelerate the development of IS. These models typically do not directly contribute to the resolution of a business problem. In other words, they are not executable software applications. However, (reference) models effectively *communicate innovative ideas and best practices* [6] and therefore serve as the starting point for the implementation of IT systems [6].

[31]RapidMiner: `http://rapidminer.com/`

RefModPM The *reference information model (RefMod^{PM})* is a collection of UML diagrams that simplify the implementation of new E-PMS, based on industrial PMS standards (developed by Ahlemann). The model (consecutively referred to as *RefModPM*) has been derived by reverse-engineering 28 commercial PMIS products. It consists of of 18 UML class diagrams, and 10 activity diagrams. Ahlemann applied a conceptual architecture called *M-Model*, to structure the processes and data models into different hierarchical levels and project phases. Hence, although the RefModPM covers all PMBOK knowledge areas, it is not structured accordingly. The RefModPM is a very vivid approach, because the UML diagrams are compact and can be communicated very effectively. Practitioners can reuse the models and adapt them to their needs, i.e., by adding or removing diagram details. However, as UML diagrams are difficult to exchange in electronic format (cf. [48]), and must be manually remodeled and implemented [6].

PROMONT Abels et al. introduce *PROMONT*, which is a PM ontology that builds upon the RefModPM. More precisely, the authors transform a subset of the RefModPM into OWL, to improve the models' reusablility, and to make them machine-interpretable [3]. The evaluation of PROMONT is based on a virtual PM scenario, in which three project partners jointly develop a UI-based software system. The authors demonstrate the information exchange across the local systems, based on a common ontology. The structure of this ontology (PROMONT) must be implemented by the partner's PMS. Abels et al. demonstrate the exchange of *events*, such as milestones, or the completion of a development activity, across project partners. In addition, they suggest machine-interpretable rules to automate the verification of constraints. For instance, overlapping project phases can be detected via reasoning. PROMONT is discussed on a conceptual level. The authors remark that a prototype must still be implemented [3].

Mohammadi and Khalili propose a Semantic Web service architecture that rests on PROMONT and the Web Service Modeling Ontology (WSMO). However, the authors do not specifically address aspects of PROMONT. Instead, they mainly refer to PMBOK for structuring the Web services, and the flow of the business logic. The decision-making component is the core of the framework. The component tries to dynamically discover solution scenarios, based on received events, such as an information that the planned project costs have been exceeded. As the authors state, the architecture is designed for *open, unregulated, and often chaotic [PM] environments* . The authors do neither provide an evaluation of their framework, nor do they provide evidence of the existence of a prototype implementation. In addition, their framework does not address UI visualization aspects [85].

The *normative* approach of the RefModPM is helpful for the integration of new PMS, because the application of reference models prevents practitioners from common pitfalls. However, the normative approach forces organizations to adapt to the model, rather than vice versa. The RefModPM is a "best-of" collection of existing industry standards, and has been found suitable for for technological-oriented audiences, and the development of new products. However, it is less appropriate for project managers, because the used vocabulary is unconventional (e.g., "Initiatives" as the central concept for any type of action), and forces managers to adapt to another standard.

In addition, the UML diagrams of RefModPM and the PROMONT OWL ontologies coexist, which makes them difficult to maintain. The UML diagrams are compact and suitable for project managers, while the ontologies are only appropriate for knowledge engineers. Another issue, not answered by Abels et al., is the negotiation of a common data model between project partners. As Ahlemann states, the RefModPM must be adapted before it is suitable for a specific system. If several partners simultaneously adapt the reference models to their needs, then a conflicting mismatch results, i.e., the models are not adapted to synchronization. Finally, the implementation of RefModPM (either based on UML or OWL) implies several (redundant) implementation efforts, which could be improved via the re-utilization of the model artifacts (cf. section 2.4).

PMBOK Ontology by Szwed Szwed modeled a comprehensive PMBOK-based ontology[32], including 340 total classes. The ontology is very concise. However, Szwed's ontology focuses on a process-oriented view of the PMBOK and does not include class attributes, respectively data properties. The central ontology class is the *pmbok.Process*, which has inputs (*hasInput*), outputs (*hasOutput*), as well as tools and techniques (*hasToolsAndTechniques*). The rest of the ontology is divided into five main classes:

- ProjectFramework (contains the two main classes *ProcessGroups* and *KnowlegeAreas*)
- Processes (the 42 PMBOK processes)
- Artifacts (objects that are produced and consumed by PM processes)
- Tools & Techniques (e.g. *SWOTAnalysis*, *EarnedValueManagement*, *QualityAudits*)
- Roles (classify project members and teams according to their functions)

Szwed applies the PMBOK ontology for the mapping between a traditional PM methodology (waterfall) and an agile methodology (SCRUM). He describes a formal approach for the mapping of the SCRUM vocabulary to the vocabulary of the PMBOK ontology. For instance, the companion piece of the PMBOK *Project Initiation* phase is the *Sprint Planning* phase of SCRUM. Analogously, the PMBOK *Closing* phase relates to the concept of a *Sprint* review. Szwed concludes that mappings between different methodological concepts are especially useful for *IT enterprises running projects in heterogeneous environments*, because they help to achieve a suitable level of alignment between different methodologies [117] [118].

The comparison of Szwed's PMBOK ontology with PROMONT leads to three conclusions. First, the utilization of PMBOK vocabulary without auxiliary concepts increases the readability for project managers very much. Second, the design of an ontology depends very much on the viewpoint of the modeler. Because Szwed focuses on the process aspects of the PMBOK and does not include any attributes, the ontology cannot be reused for this work. Third, the PMBOK ontology has been found available, while PROMONT is not.

[32]PMBOK Ontology: `http://pszwed.ia.agh.edu.pl/ontologies/pmbok-ref/`

OSLC Open Services for Lifecycle Collaboration (OSLC) is an open project with the aim to develop an integration framework for software development and Application Life Cycle Management (ALM), respectively Product Lifecycle Management (PLM) tools. In contrast to PM, ALM addresses the entire life-cycle of a specific application, and hence focuses on release planning, and QM aspects. OSLC is based on Linked Data (RDF, SPARQL). The current version includes ontologies for the traditional SE domains RM, QM, and change management (in this context: modifications in a product's requirements). However, the current version of OSLC has been found to be very incomplete. For instance, the QM and change management ontologies are almost empty. The RM ontology is the most complete one. However, this ontology contains a large number of technical attributes that are conceptually irrelevant (e.g., *createdAt*, *lastUpdatedAt*, *createdBy*, *updatedBy*). These attributes decrease the readability of the model [126].

Semantic PM Applications

Finally, PM research prototypes with UI provisions, to access/execute semantic features, are presented. Applications that focus on the improvement of semantic search features (e.g., via keywords and/or tagging) are not discussed.

Historically, there exists a set of knowledge-based approaches with a strong background in Artificial Intelligence (AI). These approaches typically focus on traditional aspects of project planning, and try to improve specific scenarios via (semi-)automated support. Muñoz-Avila et al. provide a simple UI for the semi-automated decomposition of work packages in the aerospace industry. The project manager is guided through the decision process, by answering a set of yes/no questions. Based on previous decisions, or training data, a heuristic is applied that applies the most adequate case based on a similarity measure [90]. *LaSSIE* is a knowledge-based system with the goal to improve the visibility of software artifacts in large systems. The system provides a UI to give programmers direct access to a set of semantically-based views. A developer can enter a query based on natural language processing, to explore software dependencies, and knowledge about specific software components (e.g., a software library) [31].

More current approaches focus on the querying and derivation of explicit PM knowledge based on OWL and related rule languages.

Dong et al. utilize Protégé and SPARQL to design queries for the retrieval of PM team data. The main result of their work are four SPARQL queries, which are applied for the retrieval of very simple (real-world) PM data (project-related employees, hierarchical relations, and assigned criteria) [35].

Ruiz-Bertol et al. design a simple PM ontology for the detection and querying of *management issues* (e.g., the absence of a staff member). They utilize Semantic Web Rule Language (SWRL)[33] to define (a) two *query rules*, and (b) three *reasoning rules*. SWRL rules consist of a header, and a body, and enable the inference of explicit knowledge from existing OWL ontologies.

[33]SWRL: `http://www.w3.org/Submission/SWRL`

Query rules consist of a "pseudo" rule-body. I.e., the rule-body does not add new knowledge to the ontology, but specifies a user view for the temporary displaying of records (e.g., `org:ProjectTeamMember(?p) => sqwrl:select(?p)`). The work is based on Protégé and the Jess reasoning engine [102].

Settas and Stamelos provide a conceptual framework for the design of a *PM antipattern* ontology based on *Bayesian Networks (BNs). Antipatterns* describe a *commonly occurring solution to a problem that generates decidedly negative consequences* [25]. *Gold Plating* is an example of a frequently occurring antipattern. It addresses the adding of (irrelevant) feature specifications (either by a committee, or by developers) based on proprietary interests. For instance, developers are often playful and add irrelevant features, or committees demand unrealistic requirements [25]. BNs are probabilistic networks that help to model relationships between different types of antipatterns. They are suitable to depict PM antipatterns and *can be used by project managers to illustrate the effects of of uncertainty on a PM antipattern* [104].

Settas and Stamelos express antipatterns as OWL ontology rules. Project managers can query the knowledge base via a graphical UI, to retrieve information about specific (anti-)patterns. A utility value is assigned to each query to assess the usefulness of the result, and the collection of feedback is facilitated. For each antipattern, at least one BN is associated with. The BN expresses the relationships of different antipatterns, based on probabilities. These probabilities are either collected manually, or obtained by using text classification programs [104].

Dippelreiter utilizes DotProject[34], which is a Web-based OS PM software product , and enriches it with semantic features, to improve project management. Her work is based on three main user scenarios, which she derived from interviews with Austrian project managers. In the first scenario, a project member asks the project manager to take vacations at short notice. Dippelreiter argues that based on the project (resource) plan and assigned work packages, semantic technologies can be leveraged to determine whether the absence of a team member critically delays the project. The second scenario addresses the linkage of lessons from past projects, during the creation of a new project. The project manager can search for similar tasks, artifacts (software modules, documentation), or lessons so that they become reusable. The third scenario addresses the composition of the "optimal project team", based on the available team members and their competencies. Dippelreiter designed a PM ontology, and implemented several DotProject plugins as a prerequisite to evaluate the scenarios. The evaluation consists of a set of resulting SPARQL queries, and a prototype evalution with six test persons. Dippelreiter concludes that the test persons found the project very useful, but that the scenarios only addresses a very small area of PM, and that the approach lacks of a real-world scenario.

The first scenario is evaluated very detailed, but the other, more meaningful scenarios (especially scenario 2), are discussed very cursorily. The approach of enhancing existing PMS with semantic technologies is an interesting, efficient approach. However, the DotProject integration is only suitable for basic research and of little use for real-word projects. Moreover, the project is of no reuse for this thesis, because it does not address integration issues, and is based on simple PHP plugins [32] [33].

[34]DotProject: `http://www.dotproject.net`

Related work reveals a lack of comprehensive, reusable semantic PM solutions for several reasons. First, there is definitely a lack of suitable UIs that enable the communication and validation of research results to/by practitioners. Most approaches rest on Protégé, OWL, and utilize SPARQL for the evaluation. Second, OWL ontologies (e.g., represented in Protégé) are not appropriate for the communication of research results. Most researchers invest additional effort to translate their ontologies into UML class diagrams. UML diagrams are simple, widely accepted, and compact. Third, Semantic Web research focuses on very small, artificially-designed scenarios without real-world data. As a consequence, most research is centered around PM team data, which is (a) easily accessible, constructible, and communicable, and (b) well-supported by the graph-structure of ontologies. An efficient framework for data integration can significantly contribute to the design and evaluation of realistic, more comprehensive research scenarios.

Research Item / PM Integration Approach	**Description**
Traditional PM Approach: No Integration	High manual synchronization and coordination efforts. Low reuse of organizational knowledge. High user-freedom (?)
All-In-One PM Software (Commercial Suites)	Tight integration and sophisticated UIs. High installation, maintenance and training costs with lock-in hazards. Long roll-out durations. Limited integration capabilities for third-party tools. Teams need to adapt to a single solution.
Dashboard-Based Ad-Hoc Integration (REST)	Quick and dirty solution. Good visualization results. Low scalability. No synchronization mechanisms.
SE Tool Integration Frameworks (OpenEngSb)	Systematic/scalable integration in the enterprise context. Not optimized for PM. OpenEngSb: no support OWL, SPARQL). No PM connectors available. Limited synchronization mechanism. Not appropriate for lean development. Limited UI integration.
Data Warehousing Management Processes	Mature and Well-Documented. Address data integration and visualization of aggregated, manager-centric data. Focus is on large-scale integration projects.
Model-Based PM Integration (RefModPM, PM Ontologies)	Efficient reuse and communication, but not executable per se. Large parts of the integration must be performed manually. RefModPM: Communicable to managers (UML diagrams). normative; not machine-interpretable; addresses the roll-out of new E-PMS. Ontology Approaches: machine-interpretable, but not communicable to the PM.
Semantic PM Approaches (SemProM)	Few, immature, solutions. Machine-interpretable. Optimized for specific scenarios. Lack of integration and real-world data.

Table 2.2: Existing (Project Management) Integration Approaches

Table 2.2 summarizes the related work of section 2.3. The central statement is that powerful integration approaches are existing, but scattered across various fields of applications. Most related work focuses on large-scale enterprise integration, while lean, Web-based solutions lack of systematic data integration.Model-Based approaches are very efficient for (human) communication. However, the integration of models into an executable IT environment requires significant programming efforts.

2.4 Model-Driven Software Engineering with Semantic Technologies

The previous chapter motivated the need for technologies that leverage the efficient re-utilization of existing (data) models. This chapter introduces Model-Driven Software Engineering (MDSE), and enumerates the issues and solutions regarding the efficient application of semantic technologies for MDSE.

2.4.1 Overview

Models are commonly regarded as efficient and easy-to-communicate design artifacts in SE projects. However, data models, such as UML class diagrams, are rarely leveraged for the automated derivation of software artifacts. One major reason is the poor data exchange standard of UML (cf. [47]). OWL ontologies provide a precisely specified format, are machine-interpretable, and make several traditional implementation tasks, such as the explicit creation of a database scheme, obsolete. However, the application of ontologies introduces a set of new problems. First, the design of ontologies requires profound expert know-how, which practitioners do not have. Second, the specific storage technologies and their APIs (e.g. Jena, OWLAPI) are often incompatible, and difficult to use.

2.4.2 MDSE and Tool-Support

MDSE is *a methodology for applying the advantages of modeling to software engineering* [22]. *Models* are purposeful abstractions of reality, enabling the cost-efficient sharing of mental representations. In SE, models are widely regarded because they efficiently communicate ideas, drafts, architectural decisions, processes, data schemes, or UI sketches.

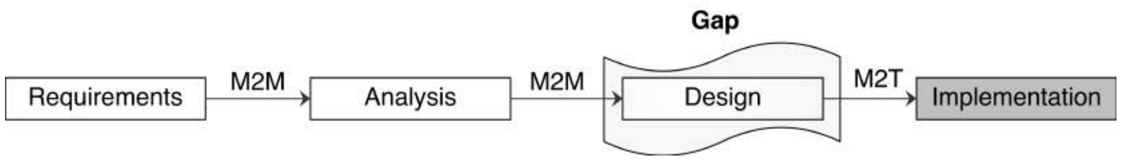


Figure 2.8: A typical MDSE-based software development process (adopted from [22])

In MDSE, *everything is considered as a model*, i.e., diagrams, specifications, but also software artifacts. Models become reusable by (a) transforming certain aspects into other model representations (for instance UML to OWL), or (b) by deriving functional artifacts (code, tests, documentation). The first type of transformation is called *(Model-to-Model (M2M)*, while the later is referred to as *Model-to-Text (M2T)*. A usual MDSE-based development process is illustrated in figure 2.8.

EMF The Eclipse Modeling Framework (EMF)[35] consists of a set of MDSE tools and plugins. It is the predominant OS modeling framework. ECore is EMF's central data format for maintaining models and resembles a simplified version of UML class diagrams. Based on ECore, Eclipse provides a set of tools for (a) the specification of domain specific syntax and languages (Xtext[36]), (b) the transformation between models (e.g., ATL[37]), and (c) code generation (e.g., xtend2[38]). Based on the Model-Driven Architecture (MDA), a conceptual Object Management Group (OMG) framework for applying MDSE in industry, the EMF also provides basic features for OCL[39], which is a language for verifying constraints based on a meta-model, during runtime. The EMF is a very powerful, but restrictive framework: (1) The various tools (ECore, ATL) work hand in hand, as long as the developer is working inside Eclipse. However, the integration of external tools (e.g., vendor-specific UML editors) is error-prone. (2) The EMF is a heavyweight solution, unsuitable for sharing results with managers, interactively. (3) The EMF and the OWL have several areas of tension, which complicate the integration of OWL into EMF (not discussed here; cf. [47] and [129] for details). As a result, only the code generation approach (*xtend*), which can also be executed outside the EMF, is considered for this work [22].

umlTUowl UML class diagrams, compared to the ontology visualization of state-of-the-art editors (e.g., Protégé-based, TopBraid Composer[40], represent models more compact. In addition, practitioners are familiar with the notation of UML class diagrams. *umlTUowl* facilitates the creation of OWL ontologies based on the automated transformation of UML class diagrams. Practitioners use their preferred vendor-specific UML editors to model class diagrams, and can semi-automatically derive an OWL ontology at the end of the modeling session. *umlTUowl* is a simple, compact tool, which has been developed by Grünwald and Moser in 2011 *umlTUowl* [47]. Since its release, the tool has been downloaded more than 600 times, and received more than 40 positive responses by researchers[41]. Several of these responses included requests for extensions, because *umlTUowl* currently supports only three UML editors (MS Visio, Visual Paradigm, ArgoUML), and needs to be started from command line *umlTUowl* [47]. The change requests mainly asked for the integration of additional UML editors, or the provision of a central, UML-like ontology editor, for the creation of OWL ontologies, but also other artifacts (e.g., code).

2.4.3 Code Generation

Leveraging MDSE for the generation of software artifacts is beneficial for both, development and maintenance activities. First, the design artifacts remain synchronized with the code artifacts, and hence serve as a high-level documentation of the code. Second, the data access implementation effort can be cut significantly, because beans, DAOs, unit tests, and even simple

[35]Eclipse EMF: `http://www.eclipse.org/modeling/emf`
[36]Xtext: `http://www.eclipse.org/Xtext`
[37]Eclipse ATL: `https://www.eclipse.org/atl`
[38]Xtend: `https://www.eclipse.org/xtend`
[39]OCL: `http://www.eclipse.org/modeling/mdt/?project=ocl`
[40]TopBraid Composer:`http://www.topquadrant.com/composer`
[41]umlTUowl: `http://sourceforge.net/projects/uml2owl/`

UI elements, can be generated on-the-fly. According to Albert et al., *the Create / Read / Update / Delete (CRUD) operation accounts for staggering 80% of the overall software functionality in typically data-intensive applications* [8] [22, p.27].

The application of a semantic data store requires significant expert know-how to (a) select the appropriate storage technology, and (b) to implement the programmatic access to store and retrieve data instances, properly. For instance, the available semantic APIs (e.g., OWLAPI, Jena) address the persistence, modification, querying, and reasoning of semantic data very heterogeneously. A powerful code generator can reduce the costs for the implementation of a semantic data store significantly. First, a large part of the data access code is generated automatically. Second, the specific low-level APIs are abstracted, ensuring unified data access. Third, the risk of selecting the wrong storage technology in initial project phases, is reduced. Finally, errors in the generated code feedback and increase the stability of the model [46].

Semantic Code Generators The range of existing semantic code generators is very limited.

Jastor Jastor[42] is a OS generator for Java and Jena. It is based on a paper of Kalyanpur et al. [73]. Jastor generates interfaces, implementations, factories, and listeners, based on an ontologies' properties and class hierarchies. The listeners are used to verify ontological restrictions, such as min/max cardinalities of data properties, or functional object properties. For instance, if a property is defined as *functional*, then there must be not two elements that link to the same entity (e.g., the same person cannot have two distinct social security numbers). Jastor is not maintained any more, and the sources are difficult to adapt.

RDFReactor RDFReactor is a code generator for Jena and Sesame[43], which already has been applied for several semantic-based research projects (e.g., SemWiki, SemVersion). The main features of the RDFReactor are the creation of Java classes, Data Access Object (DAO)s, and their access via http (*jREST*). Two main components of the RDFReactor are *RDF2Go* and a *Velocity*[44]-based *template engine*. RDF2Go is an abstraction layer that provides unified, implementation-independent access to the underlying semantic data stores. The template engine enables the formatting of the Java output. The RDFReactor does not support the OWLAPI. In addition, OWL is only supported partially and experimentally [122] [121].

Semantic Web Builder The Semantic Web Builder (SWB) facilitates the generation of a data access layer, based on predefined ontologies. By populating the ontology scaffold with instances and transformation rules, structural and behavioral elements can be injected into the generated code. Compared to Jastor and the RDFReactor, the SWB enables the management of behavior, and the partial creation of graphical UI elements. This however trades off in additional configuration effort (the code is not generated from the ontology structure itself, but from the instances of the ontology). In addition, the SWB is not available for further evaluation [112] [46].

[42]Jastor: `http://jastor.sourceforge.net`
[43]Sesame: `http://www.openrdf.org`
[44]Velocity: `https://velocity.apache.org`

2.4.4 Ontological Engineering Methodologies

The application of semantic technologies for MDSE facilitates the reuse of proved design methodologies. *Ontological Engineering Methodologies* (OEM) enable the efficient the design of sophisticated data models and ontologies, in knowledge-intense projects. The available OEM propose design processes that depend on the problem context, the application area of the ontology, and the availability of reusable ontologies. They address different scopes, and levels of granularity. Some popular OEMs are the *METHONTOLOGY* (begins with the creation of a glossary, and a taxonomy; ends with the population of the ontology), the *KACTUS* methodology (start with identifying motivating scenarios; ends with the specification of completeness theorems), and the *Ontology 101* [37] [91].

Ontology 101 The *Ontology 101* is chosen as a representative of OEM, which is analogously applicable for the creation of UML-based ontologies. The *Ontology 101* consists of seven well-documented steps, which Noy and Guiness exemplify with the support of Protégé [91]:

1. Determine the domain and scope of the ontology.
2. Consider reusing existing ontologies.
3. Enumerate important terms in the ontology.
4. Define the classes and the class hierarchy.
5. Define the properties of classes—slots (define and assign data and object properties).
6. Define the facets of the slots (define the cardinalities, data types for data properties / attributes, and the domains/ranges for object properties / associations)
7. Create instances (populate the ontology).

Applied to UML-based data modelling, in step (4), the UML classes and their generalizations would be defined. In step (5), the most important attributes and associations are added to the classes. Finally, in step (6) the data types of the attributes, and the cardinalities of the associations, are refined. Step (7) is obsolete.

2.5 Concluding Remarks

Both, PM as well as data integration involve complex systems, including technical, social, and organizational factors. Related work about PM software shows that sophisticated all-in-one PM software products exist, but the selection of an appropriate solution is linked with high costs. Data integration solutions, such as DWH or EII solutions enable the integration of multiple PM software instances, but focus on the enterprise-context, and have not been considered for temporary project organizations (or consortia) so far. One key to more efficient PM data integration is the consideration of knowledge, which is implicitly available in distributed data sources and PMS, and also addressed by the PMBOK. MDSE can increase the visibility of such knowledge, and helps to improve the efficiency of software engineering processes, via reusable models.

CHAPTER 3

Research Issues

Based on related work, the main research issues are identified, and refined. The central research question is: How can project managers be aided in integrating, synchronizing, and monitoring project data effectively and efficiently, especially if the project organization is temporary and heterogeneous?
The discussions performed with PM experts and researchers (workshops, etc.) show that project-related data is often available, but is not reused efficiently. The main reasons are (a) the lack of an efficient integration process for (temporary) project organizations, (b) the heterogeneity and the distributed nature of the available data, and (c) the believe that manual data integration activities are pointless, because they are costly, and error-prone one-time activities (missing automation alternatives). The hypothesis is that the provision of (user interface supported) software components for the configuration, semi-automated integration, and visualization of project (management) data, leads (a) to a higher degree of integrated project knowledge, (b) makes monitoring and reporting activities more efficient, and (c) fosters autonomous organizational units to report more efficiently. In this respect, the potential of Semantic Web technologies and model-driven software engineering, are researched.

RI-1 Efficient Integration of PM Software and (Abstract) Data Concepts in Temporary IT Project (Management View) A framework for the efficient identification of PM software and knowledge sources, the negotiation and agreement on (abstract) common data concepts, and the technical integration of those across project (sub-)organizations, needs to be examined and elaborated. The integration of PM software and data, based on (knowledge) models, must be considered at various organizational levels.

RI-1.1 Negotiation and Integration of PM Data at the Management Level The requirements for data integration at the top-management level are mainly to explore and discover potential integration scenarios and solutions, to develop and negotiate a common set of PM-related models, and to identify costly manual integration tasks with optimization potential.

A related issue that has been stressed by practitioners during the conducted expert discussions, is a lack of the project (sub-)organizations' commitment and end-user acceptance, for the introduction of a new PMS or data integration solution.

RI-1.2 PM Data Integration / Mapping Between Local PM Systems To reduce the manual synchronization activities for both, project managers and teams, the automated data exchange between local PMSs, tools, and data needs to be examined. The main problems that must be addressed are that (a) data needs to be integrated and frequently synchronized between tools of the same domain, (b) data between the local systems must be exchanged/mapped unambiguously, and semantically correct (mapping between different concepts, attributes, data types, etc.) (c) the level of the data integration completeness should be expandable, so that project teams can retain their local tools without loosing important features, (d) the project manager should be in control of the synchronization process, and (e) configurable work-flows should enable the exchange of PM-data across different tool domains for specific projects. In addition, the manual configuration effort should be kept minimal.

RI-1.3 An Integrated PM View to Improve Monitoring and Reporting Activities This issue addresses the reduction of manual synchronization and reporting activities of the PM, based on the development of a prototype (ITPM dashboard). The project manager should be in possession to browse across distributed data in a central view, and to navigate to the linked local representations and tools, on demand. In addition, the manager needs to be capable (in collaboration with a knowledge engineer) to derive aggregated knowledge from distributed PM data, such as KPIs, graphs, or notification tables. The feasibility of aggregated queries across multiple domains, the dynamic configuration of PM visualizations, and the identification of an appropriate (semantic) integration model needs to be conducted. "Appropriate" in this respect means (a) easy to configure, (b) traceable, (c) flexible, and (d) enabling near real-time synchronization.

RI-2 Efficient Utilization of Data Models for PM Data Integration (Implementation View) This issue addresses the efficient reuse and communication of models (a) across different organizational levels and projects, and (b) the derivation of software artifacts based on these models.

RI-2.1 Efficient Technical Implementation of PM Data Integration Scenarios Based on the requirements of the top-management, a development team needs to implement the code for the integration of the identified PM software. This research issue focuses on the efficient reuse of negotiated data models, (a) across various organizational levels, and (b) for the derivation of additional code artifacts (the potential and commonalities of UML class diagrams, *umlTUowl*, OWL, and MDSE are investigated in this respect). For instance, at the CDL-Flex, the common models and the description of related tools, are currently passed textually, or via UML class diagrams to the developers, who manually need to transform them into OWL ontologies and code.

RI-2.2 UML Class Diagrams for Efficient Communication and Integration The efficient reutilization of UML class diagrams and Semantic Web Technologies (OWL, SPARQL), for the communication of PM models and knowledge, and the automated derivation of software artifacts, are investigated. UML class diagrams are widely accepted by technicians and managers in IT projects, compactly communicable, but lack of re-usability (no vendor-independent exchange language). In contrOWL ontologies are machine - interpretable, but require expertise in the field of knowledge engineering, and have only limited editor support. *umlTUowl* tries to overcome some of these gaps, but (a) is tied to a small set of vendor-specific editors, and (b) does not facilitate the collaborative development and the further reuse of models.

RI-2.3 Maintaining Shareable and Adaptable ITPM Reference Models Based on the existing PM frameworks and reference models, a framework for designing, maintaining, organizing, and sharing semantically enriched PM models, (a) across organizations, and (b) across the ITPM community, needs to be developed. This issue picks up the practitioners' criticism (mainly feedback from Austrian project managers) that best-practice frameworks, such as the PMBOK, or the existing reference models (e.g., RefModPM, PROMONT), force an organization to adapt to a normative, possibly unsuitable approach, rather than vice versa. In addition, the existing reference models have been found to be (a) not available in digital, machine - interpretable, and human-friendly format, and (b) depend very much on the modeler's viewpoint, and the application purpose.

CHAPTER 4

Research Approach

The Information Systems Research Framework (ISRF) is applied for the scientific design and evaluation of a Systematic Integration Framework for Project Management (SIFPM). The main design artifact is the central *ITPM Dashboard*, which facilitates the efficient integration, visualization, and synchronization of organization-specific PM systems, semantic models, and data.

Hevner's ISRF is design-science oriented, and especially useful for research problems which are characterized by unstable requirements, complex interactions among sub-components, and a critical dependence upon human cognitive and social abilities [61, p.80f].

Section 4.1 outlines the IS research framework (ISRF). Section 4.1.1 introduces the Three Cycle View of Design Science Research (3CV) as an extension of the ISRF. Section 4.1.2 summarizes the seven guidelines as the foundation of the ISRF. In section 4.2, the 3CV, and the design-science research guidelines are applied to describe the thesis' research method [60].

4.1 The Design-Science based IS Research Framework

Unlike in natural science, in information science, the outcome of a conducted research project, is initially often unclear; requirements and constraints are unstable, and the environmental problem context is ill-defined. Complex interactions among sub-components of the problem and its solution, and a critical dependency upon human and IT, restrict the application of behavioral research methods. In design science, the built (designed) artifacts take the center. A research outcome in design-science is profound, if the developed artifacts are innovative, applicable, and relevant. In this way, the researcher is prevented from performing excellent, but impractical academic work.

Figure 4.1 visualizes the concept of the ISRF. The framework takes a middle ground between *design science* (left hand side), and *behavioral science* (right hand side). Behavioral-science *seeks* *to develop and justify theories that explain or predict organizational and human phenomena*, and is mainly descriptive (theories and proofs) [61]. In contrast, design-science is *fundamentally a problem-solving paradigm* [61]. Its focus is on the discovery, and the utility of new artifacts.

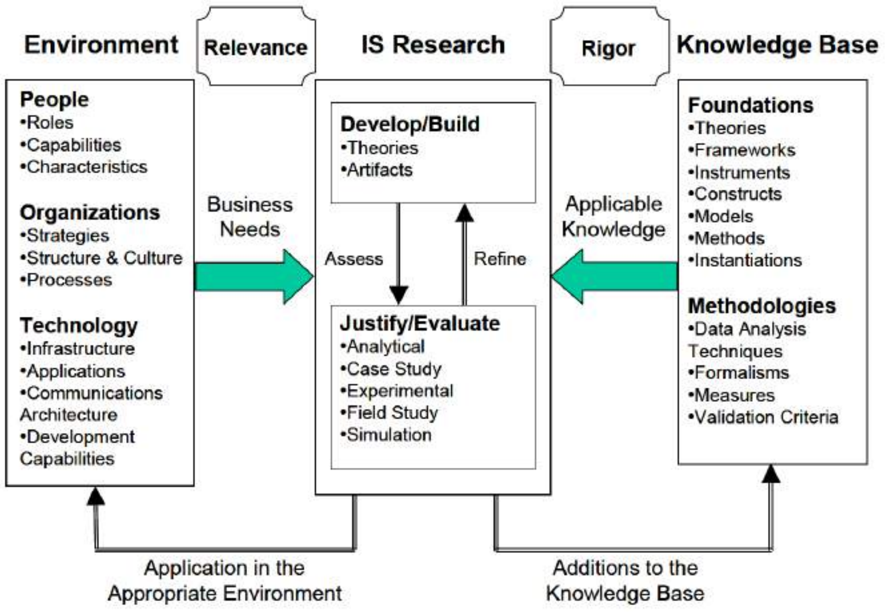


Figure 4.1: The Information Systems Research Framework (ISRF) [61].

Figure 4.1 visualizes the impact of the specific business environment for the relevance of developed artifacts (design science related). Although the ISRF is design-driven, researchers must ensure the rigor application of scientific foundations and methods. The center of figure 4.1 illustrates the relevance of the scientific knowledge base, for (a) the design, and (b) the evaluation of new artifacts.

4.1.1 The Three Cycle View of Design Science Research

Based on an extension of the ISRF framework, Hevner describes three research cycles, which must be simultaneously and continuously executed, to ensure the relevance of design-science based research [60] (called the *3CV*).

Throughout the *design cycle* new artifacts are built and designed. The design process of an innovative artifact requires the identification of business requirements, and the periodical evaluation of its utility in the problem domain (*relevance cycle*). Nevertheless, the development of the artifact must obey the theoretical foundations, and best practices in order to assess its scientific validity. The feeding-in of the scientific grounding from the common knowledge base is represented by the *rigor cycle*. Interactions take place between all three cycles; for instance, the design of new parts of the artifact requires additional acquisition of knowledge. Likewise, design decisions may occur, which require feedback from the business environment, such as additional information about a technical system, or empirical data regarding the organization. Both, the environment and the knowledge base, are implicated into the evaluation of the artefact throughout the design cycle. The results of the design cycle will feed back into the two other cycles; in form of emerged knowledge (rigor cycle), or superior technologies and/or business recommendations (relevance cycle), which again, trigger the synthesis of new technologies in future projects.

4.1.2 The Seven Guidelines for Design Science in IS Research

The seven *Guidelines for Design Science in IS Research* represent the foundation for the application of the ISRF, and are summarized in table 4.1. The guidelines support the rigor application of all elements of the 3CV, and ensure the practical and scientific relevance of the research results.

Guideline	**Description**
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7 : Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Table 4.1: Design-Science Research Guidelines of the ISRF [61].

4.2 Application of the IS Research Framework Guidelines

Consequently, the seven design-science guidelines are applied to elaborate the research method for the thesis. Figure 4.2 visualizes the application of the seven guidelines on top of the 3CV, and provides an overview of the design search process (design artifacts, environmental factors and scientific knowledge are stated exemplary).

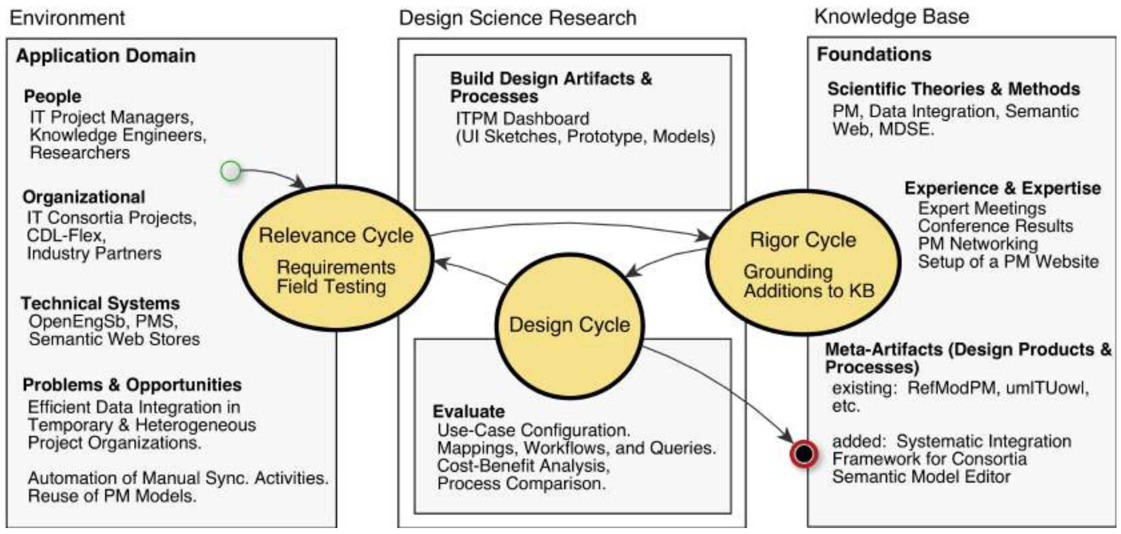

Environment
Design Science Research
Knowledge Base
Application Domain
People
IT Project Managers, Knowledge Engineers, Researchers
Organizational
IT Consortia Projects, CDL-Flex, Industry Partners
Technical Systems
OpenEngSb, PMS, Semantic Web Stores
Problems & Opportunities
Efficient Data Integration in Temporary & Heterogeneous Project Organizations.
Automation of Manual Sync. Activities. Reuse of PM Models.
Relevance Cycle
Requirements
Field Testing
Build Design Artifacts & Processes
ITPM Dashboard (UI Sketches, Prototype, Models)
Design Cycle
Evaluate
Use-Case Configuration. Mappings, Workflows, and Queries. Cost-Benefit Analysis, Process Comparison.
Rigor Cycle
Grounding
Additions to KB
Foundations
Scientific Theories & Methods
PM, Data Integration, Semantic Web, MDSE.
Experience & Expertise
Expert Meetings
Conference Results
PM Networking
Setup of a PM Website
Meta-Artifacts (Design Products & Processes)
existing: RefModPM, umlTUowl, etc.
added: Systematic Integration Framework for Consortia Semantic Model Editor

Figure 4.2: Application of the Three Cycle View (3CV) for the thesis (exemplary; adopted from [60]).

Subsequently, the seven design-science guidelines are applied for the design of the research method. The conduction of the research, including the prototype implementation, emerged in new, partly unanticipated, research results and feedback. Hence, the research method needed to be successively re-evaluated and refined. As a consequence, the application of the guidelines is discussed in retrospective.

Problem Relevance The high industrial relevance of IT projects, and the trend towards IT project cultures due to innovativeness and competitiveness, are motivated in the chapters 1 and 2. The low performance of IT projects, and integration challenges in consortia, such as power struggles, commitment, and ambiguous organizational cultures, were discussed (chapter 2.1). In addition, the disadvantages of enterprise-wide PM software, and all-in-one suites have made clear (high costs, lock-in, PMS / tool silos, incompleteness of integration features; restricted integration autonomy for project managers). The complexity of data integration is another reason to investigate PM integration scenarios (semantic mappings, workflows, etc.), at various hierarchical levels.

Semantic integration solutions, respectively Linked Data, are increasingly gaining enterprises' attention. However, there is still a relatively little body of acquired knowledge regarding the efficient application of those technologies in industrial projects. In addition, research about semantic technologies or knowledge-based research results, when applied to PM, often address a very limited, artificially confined problem space (cf. chapter 2.3).

Another relevant issue is the modeling of semantic structures. More precisely, the semantic community demands efficient, user-friendly ontology editors (defined as a research challenge by

Bizer, Heath, and Berners-Lee [17]). For instance, there exists is no appropriate, community-based UI editor for the Jena API.

Design as a Search Process Initially, the main design requirements were collected, and a rough research plan, for the design of a semantic *ITPM dashboard* was created. The knowledge transfer took place in form of meetings with developers, researchers, and CDL-Flex-related industry partners (PM experts). Considerable effort was dedicated to study and understand the OpenEngSb development environment, and to acquire knowledge about the fields of research.

Several UI sketches for the implementation of a Web-Based *ITPM Dashboard*, and a UML-based ITPM reference model were created. The results were communicated to PM experts (e.g., via a specifically created Website[1]), and discussed in workshops.

The outcomes were presented at the Software Quality Days 2013 (SWQD2013): In a workshop-like setting, project managers provided information about their organizations' PM systems, and described their organizational integration needs and shortcomings. The main outcomes were two industrial case-studies, including detailed communication plans, and a list of applied PMS and tools.

Based on the case-studies, the prototype was designed. The main steps included (a) literature researches, (b) the design of an architecture on top of the OpenEngSb, (c) the setup of a test environment, and (d) the integration of existing MDSE, and semantic technologies (OWLAPI, Jena, umlTUowl). As no test data were available for the industrial case-studies, the studies were (a) merged into a single, essential UC, and (b) transferred into the CDL-Flex environment, in which the usefulness of the approach was assessable efficiently (based on existing data).

Considerable effort was investigated into the implementation of a reusable prototype, including the integration of the OpenEngSb framework, and the design of a configurable UI. The quality of the prototype was successively improved, by integration research results and expert feedback, e.g., after each live demonstration of the prototype.

The search process revealed demand in several additional meta-artifacts, e.g., based on expert requests, or gaps in related work, which were originally not considered. For instance, a consortium negotiation and integration process, and a UML-based editor, for the efficient modeling and reuse of UML-based data models, emerged.

Design as an Artifact The central viable artifact is the *ITPM Dashboard* prototype, which integrates all developed software and user interface modules. Other important design outcomes are (a) a reusable IT data integration and negotiation process for IT project consortia, (b) a technical solution for the modeling and sharing of common project knowledge across the ITPM community (*Semantic Model Editor*), (c) an integrated and extensible, model-based code-generation approach for semantic technologies (OWLAPI, Jena), and (d) several architectural extensions and

[1]ITPM Project Website: `http://www.it-project-management.net`

contributions to the OpenEngSb, and the *umlTUowl* open source projects. The application of the prototype for the use-case leads to further results: a set of work-flows and semantic data mappings that describe the automation of manual synchronization scenarios; a set of cross-domain SPARQL queries; ITPM demo data, and reusable ITPM connectors (e.g., for the integration of requirements, issues, quality metrics, and other PM data), are the most important representatives. To indicate the comprehensiveness of the design artifacts, the entire prototype comprises ~350.000 Java LOCs, and more than 3000 unit and integration tests.

Research Contributions The main research contributions are: (a) A generic negotiation process for the integration of PM data in project consortia (Consortium Negotiation and Integration Process (CNIP)). (b) The provision of an ITPM dashboard for more efficient configuration and monitoring of distributed project data. (c) A MDSE code generation approach for the access and retrieval of semantic data. The main contribution in this regard are, the Semantic Model Editor for knowledge engineers, and the mapping between UML class diagram elements, OWL ontologies, UI elements, and Java code. (d) A proposal for the structuring, organization, and persisting of reusable ITPM models.

In addition, the UC-based results include semantic PM data mappings between PMS and PMBOK knowledge areas (common concepts), work-flow implementations, a set of SPARQL queries, and visualizations.

Research Rigor The construction of the main design artifacts relied upon a rigor application of scientific knowledge, and expert power. Scientific knowledge includes related work in the areas of PM, data integration, Semantic Web, and MDSE (cf. chapter 2). Expert power addresses the application of industry knowledge, derived from (a) a number of expert discussions (e.g., 32 face-to-face discussions with PM and QM experts), (b) active networking (e.g., linkedin.com, Xing, Webinars), (c) the frequent communication of intermediate results, and expert feedback (e.g., initiated PM website; 20 responses regarding the UI sketches, and feature requests), (e) PM case studies and survey results, and (f) PM software product evaluations.

The foundation for the consortium negotiation and integration process (CNIP) are Boehm's Win-Win approach, DWH management processes, and ontology engineering methods.

The interface of the ITPM dashboard was iteratively designed, after feedback regarding UI sketches, ITPM reference models, the UC, and interactive demo presentations, was received. The software architecture is designed and evaluated based on *attribute-driven design* [13, p.171 f.], and on other common architectural design practices, such as architectural views (module view, component view, deployment view), and tactics [13] [28] [24]. The relevant quality attributes are testability, maintainability, usability and stability, and were iteratively evaluated.

The mapping between model elements (UML class diagrams, OWL ontologies, Java code, UI elements) builds upon existing semantic approaches (umlTUowl, Semantic Web Builder, RDF-Reactor, RefModPM). In addition, the research incorporated feedback from specialized knowledge researchers at the CDL-Flex, and insights from active participation in various semantic Web platforms and mailing lists (e.g. Apache Jena mailing list).

The design of a sharable ITPM model repository is based on the PMBOK knowledge areas, the consideration of data quality attributes (Information Quality Evaluation Framework [99], ISO 25010 standard [12], and ISO 25012 standard [12]), and existing PM modeling approaches (e.g, RefModPM, PROMONT, Szwed's PMBOK ontologies [117] [118], OSLC). Criticism regarding the PMBOK (e.g., [40]) flew into the construction of the *ITPM Dashboard*, and the design of the semantic ITPM model repository.

The evaluation of the solution approach is based on the configuration of a UC, and the derivation of (SPARQL) queries, including visualization results. This evaluation method has been found to be commonly applied in semantic research (cf. related work about the construction of semantic PM applications in chapter 2.3.3). The application of a cost-benefit analysis for the tracing of requirements is successfully demonstrated by Heindl and Biffl [56]. Gable describes the utilization of (cost-benefit) analysis for pilot studies, which serve as an efficient and controlled starting point for the successful evaluation of prototypes in larger contexts (multiple case studies) [39].

Design Evaluation The evaluation of the design artifacts takes place at two levels. At the management level, the utility of the approach is evaluated. For that reason, the ITPM dashboard is configured for the presented use-case, and a cost-benefit analysis is performed. The analysis aims to demonstrate that the application of the integrated ITPM dashboard (a) enables the synchronization of data between local PMS, (b) reduces the manual efforts of the PM significantly, and (c) makes monitoring and reporting activities more accurate and insightful. The main steps of the evaluation are: (a) the identification of a set of use-case scenarios. (b) the implementation of these scenarios based on the ITPM dashboard approach (semantic mappings based on UML class diagrams, implementation of work-flows and SPARQL queries), (c) the comparison of the configuration costs against the saved manual synchronization efforts.

At the technical level, the model-based approach for the development of PM software connectors, is compared against the current OpenEngSb integration approach. In detail, the process steps and the time efforts from the perspective of a developer are evaluated. For that purpose, historical data from an industrial project is applied.

Communication of Research The communication in the form of insightful research results, and applicable technological artifacts, was an important concern throughout the execution of the research project. Intermediate research results were communicated at the SWQD2013 (handouts, poster), and in form of conference paper submissions ([45] [46]). A website[2] has been created to stimulate community building, and to present the project progress. The site includes the case study descriptions, UI sketches, and the possibility to leave feedback. The final prototype is

[2]ITPM Project Website: `http://www.it-project-management.net/`

functional, and freely available on the Web[3]. Additional effort has been spent to ensure that the prototype is easy to set-up so that project managers or knowledge engineers are actually willing to test it. All source code and documentation has been handed over to the CDL-Flex development team and flows into industrial projects, research, and open source projects (OpenEngSb, *umlTUowl*).

Chapter 4 summarized the research approach of the thesis, based on the design-science based IS Research Framework (ISRF). The ISRF accounts for unstable requirements, and (initially) hidden IS complexities. In the context of the thesis, the ISRF helps to balance the research efforts (collection of literature results, gathering of expert know-how, prototype design), and ensures the relevance and research rigor of the designed prototype.

[3]ITPM Prototype: `https://bitbucket.org/agruenwald/custom-itpm-builds/downloads`

CHAPTER 5

Use Case: ITPM in a Consortium

The use case (UC) for the joint development of a software product in a consortium project is presented. Section 5.1 introduces the participating organizations, and the main project roles. Additionally, the project phases and the core PM activities are outlined. Section 5.2 investigates the project manager's main integration tasks, based on the consortium's communication plan. Specific process steps are extracted, to illustrate the manual synchronization overheads. The chapter concludes with the summary of PM-relevant data sources, tools, and concepts, in section 5.3.

5.1 Consortium Project Structure

The UC includes four organizations, which join a consortium, to implement a novel high-tech software. In addition to the provided consortium definition in chapter 2.1.3), the following assumptions are taken: First, there is no dominant partner, which means that no single partner dictates the requirements which the other members must agree on. Second, one member is the *champion*. The champion is responsible for the management and the execution of the IT project [86, p.195].

The name of the project is: *Prototype Implementation of an ITPM Dashboard based on Model-Driven Technologies.* Attentive readers may realize that the definition is self-reflexive. Indeed, the project refers to the prototype implementation conducted during the master thesis. The advantages of this approach are:

1. The reader gains awareness of the product functionality and its complexity (cf. chapter 7).
2. The development of the prototype resulted in IT and PM artifacts which directly flow back into the use case evaluation.
3. The size of the project is convenient for the use case evaluation (small set of traceable requirements, limited resources for the evaluation).

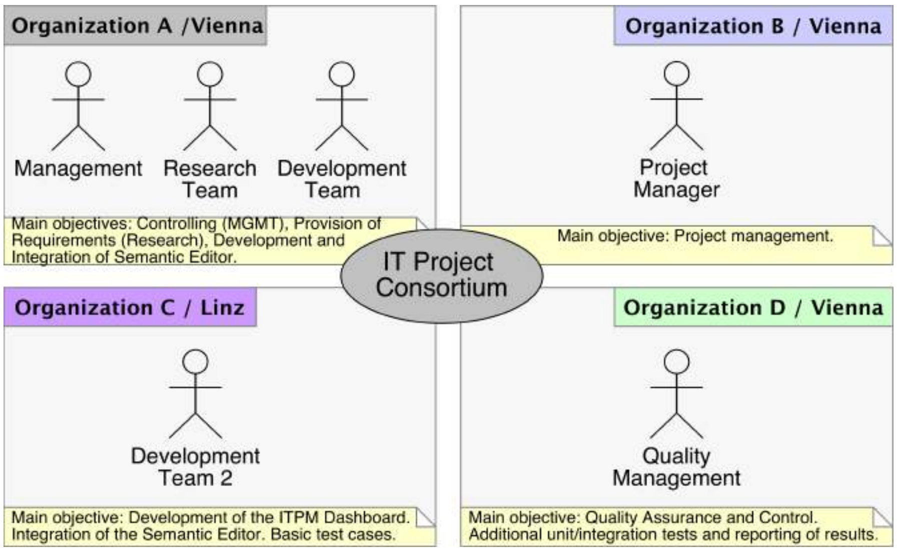

Organization A /Vienna
Management
Research Team
Development Team
Main objectives: Controlling (MGMT), Provision of Requirements (Research), Development and Integration of Semantic Editor.
Organization B / Vienna
Project Manager
Main objective: Project management.
IT Project Consortium
Organization C / Linz
Development Team 2
Main objective: Development of the ITPM Dashboard. Integration of the Semantic Editor. Basic test cases.
Organization D / Vienna
Quality Management
Main objective: Quality Assurance and Control. Additional unit/integration tests and reporting of results.

Figure 5.1: Overview of IT Consortium project partners.

4. The quality of the prototype is improved by including additional PM and QM measures, and evaluation feedback.

5.1.1 Organizational Structure

Four organizations form a consortium in order to realize the software product (cf. figure 5.1):

Organization A is the champion of the consortium and thus the main driver of the project. The organization provides three teams:

1. A team of independently working researchers. The main goal of the researchers is to determine the requirements from marketing studies and research surveys up-front. The mission of the researchers throughout the project is mainly to consult the project manager in difficult questions and to adapt / extend the requirements if necessary.
2. The steering team consists of a small team of top managers, and involves participants from the other organizations. During the ramp of the project, the management communicates the main requirements to the researchers, and keeps them on track. Throughout the project the purpose of the steering team is to control the management of the project.
3. A team of developers. The development team consists of employees and of external developers who deliver work on base of working contracts.

Organization B An experienced, independent project manager has been chosen to manage the project.The duty of the project manager starts with the unification and analysis of the identified requirements and ends with the handover of the project to the management team.

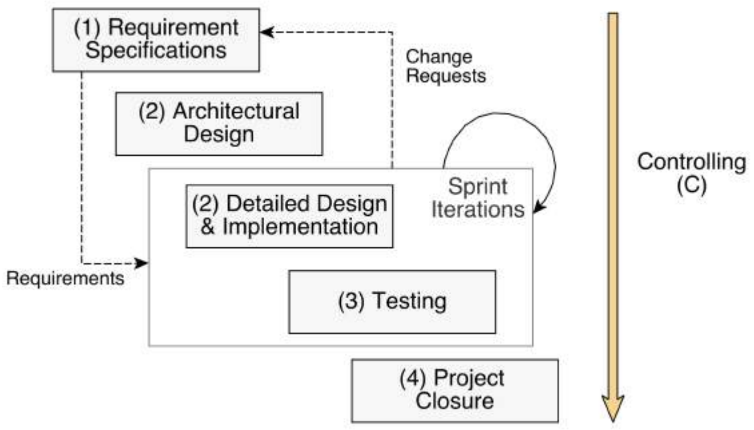


Figure 5.2: The SDLC of the IT consortium project (waterfall and agile).

Organization C A small development team which already conducted several projects for organization A. While organization A is located in Vienna the team is located in Linz / Upper Austria. Thus the communication with the project manager mainly takes place virtually (Skype Meetings).

Organization D is an independent department of quality engineers at the Technical University of Vienna. Although it is has relationships with organization A, D is an independent legal entity. The task of organization D is to provide high quality test cases for the developed software components and to report test results and code metrics to the project manager.

All organizations intend to reuse the final product or distribute it to their customers.

5.1.2 Software Development Life Cycle

The project is based on a traditional waterfall life cycle model, enriched with agile methods (cf. [116], [63], [110]). The SDLC is illustrated in figure 5.2. The requirements specification (1) and the architectural design (2) are conducted by the research team of organization A. The main outcomes of these phases are, (a) a list of technician-oriented requirements, and (b) the specification of the overall software architecture, including development guidelines. The requirements are specified developer-oriented, so that can be passed directly to the development teams. The central product is built and tested in (3) and (4). These two phases are managed agile (SCRUM) and are chunked into two-week sprints. Due to the agile project nature, change requests and design updates are anticipated. The waterfall model is not executed in a strict sequential order, but embraces back flows from the phases 2 and 3. The project closure includes the handover of the final product, the documentation, and the source code, to each of the consortium members.

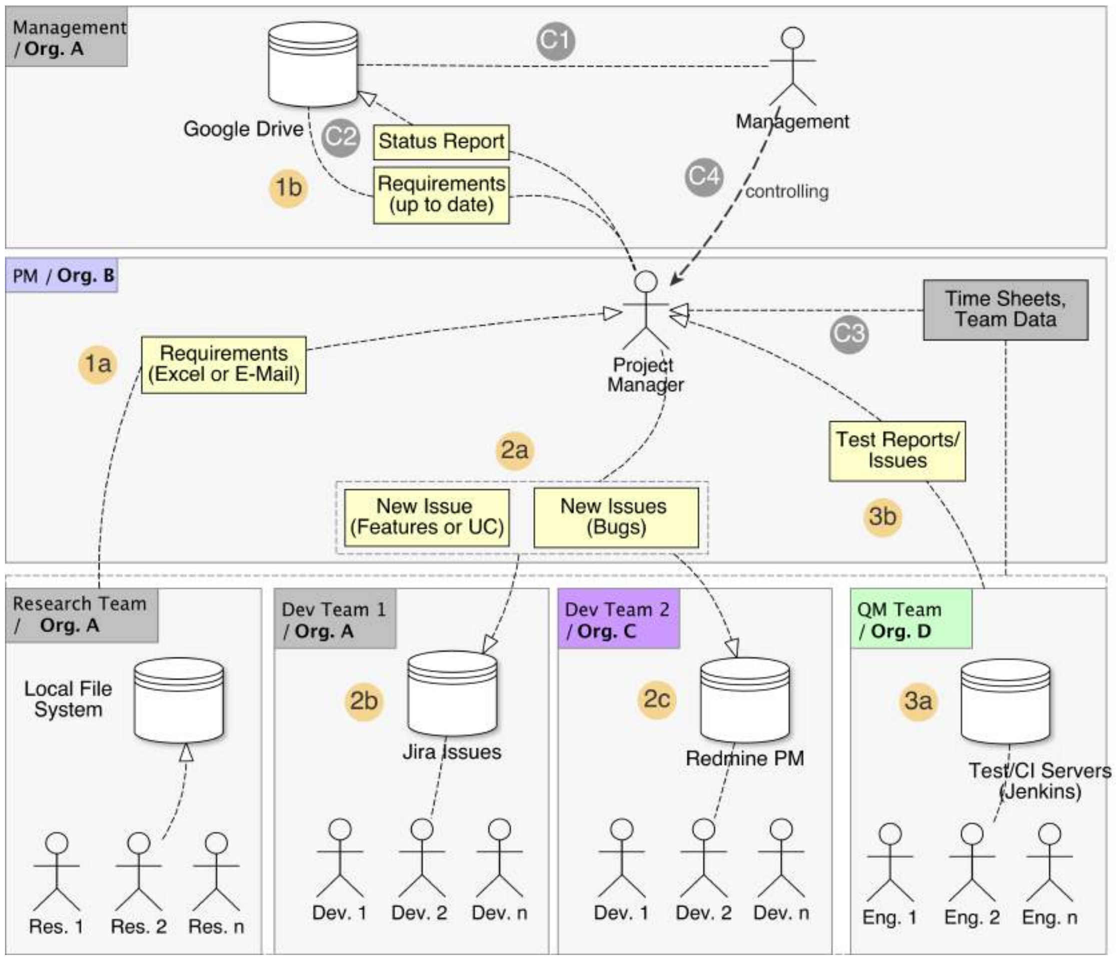


Figure 5.3: Project communication plan for the IT consortium project.

5.2 Project Communication Plan

The consortium members have distinct organizational cultures, and are used to predefined PMS, tools, methodologies and processes. Figure 5.3 visualizes the flow of the most relevant PM communication artifacts and the different PM systems that are utilized by the consortium members. The order of the encircled numbers (1-4; C1-C2) indicates the relation between the communication flow, and the sequential order of their execution throughout the SDLC in figure 5.2. The data integration steps with the most significant manual efforts are summarized below.

Collection and Transfer of Requirements The PM collects the identified requirements from the research team (organization A), during the requirement specification phase. The requirements are already preprocessed and provided within a single spreadsheet (MS Excel). The researchers email the document to the project manager or put it on a shared drive (e.g. Dropbox)

(1a). The project manager reviews the requirements and adds minor improvements. Throughout the project, a number of change requests appears. New requirements are identified by the researchers and the pool of requirements is updated (1b). Changes are emailed to the project manager. The reason why the requirements cannot be updated directly in the central spreadsheet is that the project manager needs to be aware of updates. In addition, the management of organization A requested access to the list of requirements. Because the steering team wants to save time, they insisted that the requirements are provided on their central Google Drive, in Google Spreadsheet format (1b). The PM can copy the requirements from the Excel sheet to the Google sheet, but must perform minor format changes (different columns; transformation of internal stakeholder abbreviations of organization A). Because the management adds or updates requirements from time to time (C1), the synchronization between the versions of the management, and the research team (organization A) is difficult.

Creation of a Product Backlog based on Sub-Requirements The PM translates the requirements into a product backlog (2a). The product backlog consists of a list of issues, which are assigned to the specific developers (2b, 2c). In the presented use case, two different software products, viz., *Jira Issues* and *Redmine*, are utilized by the organizations A and C. Both PMS are project tracking tools with similar capabilities. The central planning artifact are issues, which must be resolved / implemented by the local teams. An *issue* is a very generic concept. They are defined by their issue type, such as *feature* or *bug*. The type list is extensible, to support additional PM functionality, such as the application of agile management methods (e.g. types *use case*, *epic*).

The transformation between requirements and issues takes place as follows: All requirements at the under-most level (requirements can be defined hierarchically), are added to the issue tracking program by the PM. None of the field content (e.g., title, description, priority) is changed. Each issue is of the type *feature*. For each issue, the due date needs to be assigned manually, according to the project plan of the manager. Accordingly, each issue is assigned to a developer, or to a team leader, depending on the priority and the sprint iteration throughout the project. If requirements are added or updated, then the PM needs to manually synchronize the related issues in the backlog. Note that the project manager needs to request access for any local PM system. In the presented UC, all organizations are willing to grant administrative permissions to the (external)manager.

Tracking of the Development Status Assigned issues are implemented by a responsible developer. The project manager receives feedback in form of status updates, which are visualized in the local issue tracking tools. The presence of two different project tracking systems requires additional synchronization effort by the project manager: First, she has to resolve dependencies which occur between the isolated systems (e.g., a software component may depend on another software component, which is developed by the second team, and thus unknown within one of the local systems). Second, the application of different systems results in additional learning efforts. Third, the reporting is more complicated because the status information of the two systems has to be combined.

Communication and Reporting of Test Results The software quality team (QM team) ensures that all modules are covered by test cases (3a). Jenkins is utilized as a build server[1]. The function of a build server is to compile and execute code and automated tests periodically, and to inform the QM team about failures. Jenkins enables the installation of additional plugins, e.g., for the retrieval of code metrics. However, the test results and software metrics of Jenkins are not suitable to be directly reported to the project manager, or executives. Thus, the quality team has to summarize and report the results manually (e.g. in a text document or presentation) (3b). Bugs and test failures, which are reported by the QM team, must be resolved. Because the QM team of organization D does not have access to the systems of organization A and B, identified change requests and bugs are forwarded to the project manager via email. The manager must then add issues (typically bugs) in the specific PM tracking system of the responsible organization (development team one or two) (2b, 2c).

Reporting / Project Controlling The project manager must periodically report to the top management, i.e., at the end of each week (C2). The creation of the status report requires the extraction of tracking information from the local PMS (Jira, Redmine). The information must be linked with the developed software modules, test results, and requirements, in order to derive meaningful controlling figures.

All project team members track their efforts in template-based spreadsheets. The time-sheets are forwarded to the project manager at the end of each week (C3). The manager must extract the work logs for each sheet, and accumulate the time efforts for team members, software modules and teams. The derived efforts are essential to control the project costs, trigger re-planning activities, and report to the top management. Feedback from the top-management flows back into planning process for the next sprints (C4).

Typical questions that need to be addressed by the status report are:

1. How is the test distribution/coverage across all requirements?

2. How many tests have been implemented per module and what is their state?

3. How many lines of code / function points have been produced so far?

4. How many features / bugs are currently open?

5. Are there any open features that have not been implemented closely before the end of the current sprint?

6. Are the working efforts of the team members balanced?

7. What is the total effort that team members worked last week?

In addition to the management reporting, the PM permanently needs to be in control of the project progress. For instance, the PM checks at the end of each day:

[1]Jenkins: `http://jenkins-ci.org`

1. How many lines of codes / function points exist or have been added, since the last time I checked?
2. How many test failures are there currently?
3. Were critical updates in any of the available PM artifacts performed (requirements, issue systems, test reports)?
4. What are the current activities of my team members?
5. Which software modules have been implemented/tested so far (overview)?

5.3 Shared PM Concepts

Table 5.1 summarizes the most important PM concepts of the UC, and their origin tools. The project plan (which has not been mentioned so far) is created by the PM. It contains a list of the main project phases, including start and end dates. The last column of table 5.1 contains the role names that can directly access the tools, respectively files.

Concept	**Tools**	**Roles**
Requirement	MS Excel, Google Spreadsheet	Researchers, MGMT, PM
Issue	Jira Issues, Redmine	PM, Dev (Org. A), Dev (Org. C)
Test Results	Jenkins	QM
Code Metrics	Jenkins	QM
Software Modules	Jenkins	QM
Team Members	MS Excel	PM
Project Plan (Phases and Dates)	MS Excel	PM
Time Sheet	Google Drive	PM, each team member has access to his/her personal time sheet
Team Data	MS Excel	PM, Dev (Org. A), Dev (Org. C), QM (Org. D)

Table 5.1: Summary of the concepts and tools in place.

CHAPTER 6

Solution Approach

In this chapter the *Systematic Integration Framework for Project Management* (SIFPM) is presented. The SIFPM is designed to overcome PM data integration issues for temporary, heterogeneous project organizations, i.e., consortia. The main design artifact is the *ITPM dashboard*, which leverages the central configuration, control and visualization of PM data (sources). Semantic data models represent the central meta-artifact. They enable the efficient communication of PM integration requirements across various organizational levels, and facilitate the efficient technical integration.

The main components of the SIFPM are summarized in figure 6.1 and correspond with the structure of the chapter:

1. The consortium negotiation and integration process (CNIP) provides means for the efficient integration of PM data at the management level (section 6.1).
2. The *ITPM Dashboard* aids managers and technicians, in jointly executing the CNIP. In section 6.2 the most important components of the *ITPM Dashboard* are introduced, and the process steps of the CNIP are mapped onto the prototype components and related data (and code) artifacts. The configuration of the dashboard to aid project managers is left out for chapter 7.
3. Section 6.3 provides a mapping of the derived artifacts, based on MDSE, and the *Semantic Model Editor*. A mapping between UML class diagram elements, OWL ontology concepts, generated Java code, and UI forms is presented.
4. Section 6.4 introduces the *PM Model Repository*, for the organization of small-scaled, reusable, PM models. The aim is make common models, tool connectors, and PM knowledge available for further projects. The efficient development of PMS connectors, based on MDSE, respectively code-generation, and the OpenEngSb architecture, is outlined.

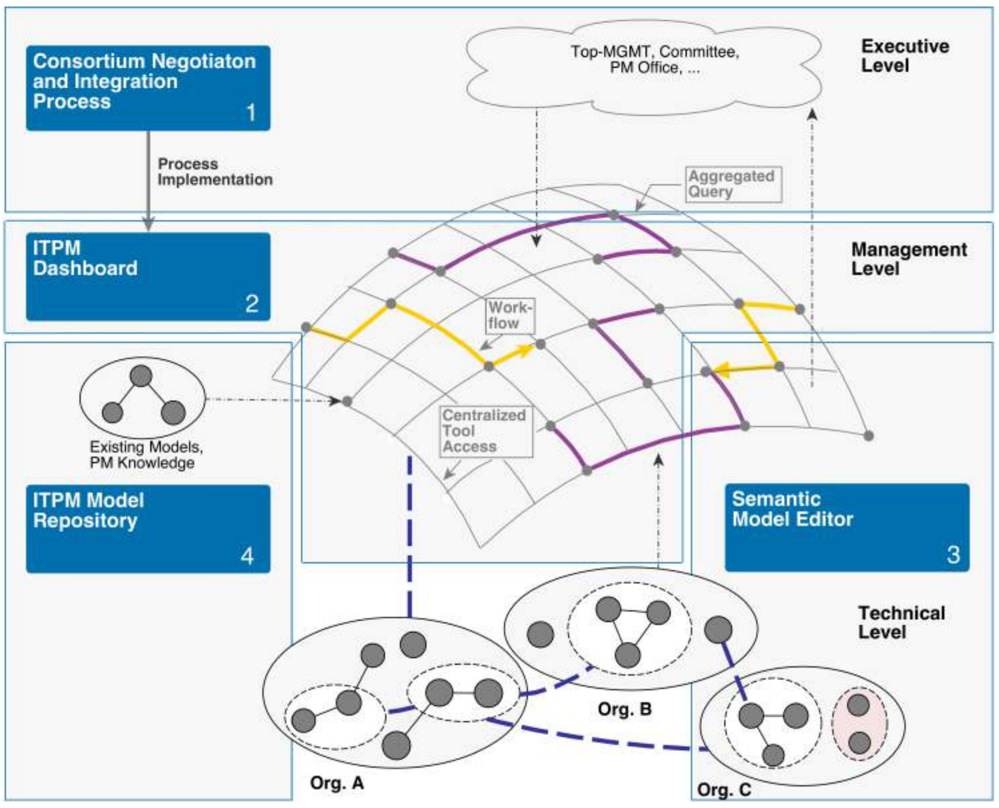


Figure 6.1: Overview of the solution approach (cf. ITPM consortium hood in chapter 1).

6.1 The Consortium Negotiation and Integration Process

To integrate PM software, tools and data sources efficiently, the *Consortium Negotiation and Integration Process* (CNIP) is proposed. Opposed to heavyweight enterprise integration solution, the CNIP focuses on temporary project organizations with heterogeneous, autonomous partners. The aim is to provide a systematic off-the-shelf framework, to (a) shorten roll-out duration and costs (compared with EII approaches), (b) enable the local teams to continue using their preferred, PM systems and tools, and (c) ensure efficient, guided integration, based on reusable design outcomes (data models).

The CNIP is inspired by Boehm's *Theory W* [18] [19], and the *WinWin approach* [21]. The WinWin approach uses a spiral model process to negotiate the requirements between the most important project stakeholders such as customers, developers, and users. The best alternatives are compared to resolve conflicting interests, and to establish *Win-Win* conditions between stakeholders. The outcome is refined gradually.

The technical foundation of the CNIP are the OpenEngSb framework, and the *ITPM Dashboard*. The OpenEngSb enables the integration of tools and data based on the implementation of tool domains and connectors. The ITPM dashboard builds upon the OpenEngSb. It provides tool-support to increase the efficiency of the CNIP. For the CNIP, the OpenEngSb can be replaced by any state-of-the-art integration framework. The application of the ITPM dashboard is recommended, but optional, and hence not detailed before section 6.2.

6.1.1 Process Goals

The proposed negotiation process of the CNIP is similar to the WinWin approach. However, instead of requirements, the (optimal) integration of heterogeneous PMS, data sources, and the degree of the integration are negotiated. The degree of the integration addresses the *completeness* of concepts, attributes, work-flows and queries.

The main goals of the CNIP are:

1. Inform the participating consortium members about the intended use of PM software and artifacts.

2. Agree on a common language / vocabulary. Clarify ambiguous terms and refine definitions.

3. Increase awareness of the pursued PM integration process (cf. [86]).

4. Negotiate the tools and sources to be integrated. Agree on the concepts (and attributes) that will be exchanged.

5. Provide a framework for the transformation of business management requirements into semantic models and technical integration artifacts.

6.1.2 Involved Roles

The process comprises the following roles:

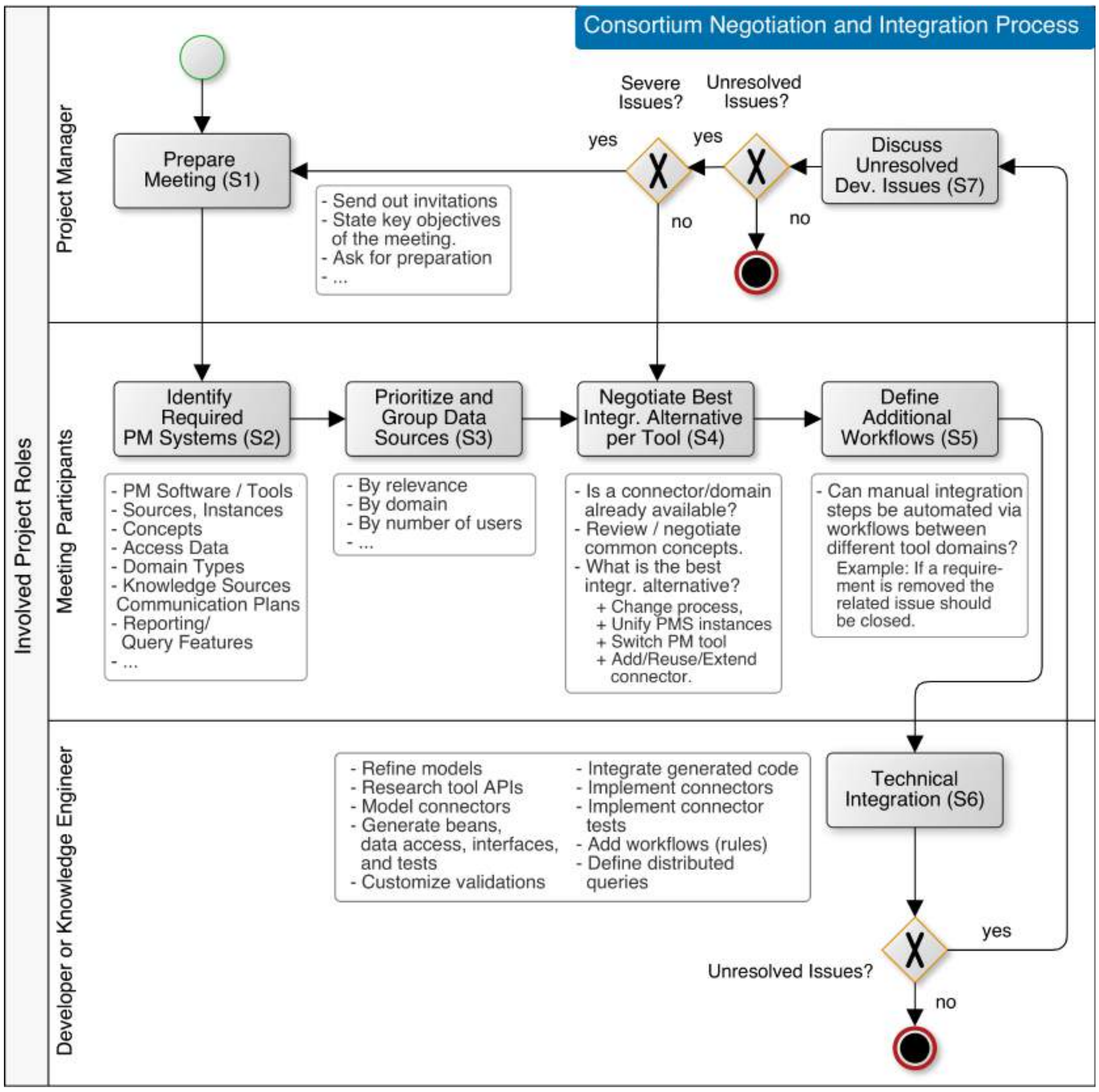
Project Manager The project manager is the driving force, which starts and terminates the process, and is in control of the negotiated artifacts.

Meeting Participants A key representative of each consortium member with basic knowledge about the PM process, a key representative of each PM domain area (e.g., a quality engineer), and a knowledge engineer or developer, with skills as described next, should be present in the meeting. To communicate sponsorship of the project, the participation of executives is crucial (they are likely to be the key representatives of the consortium members anyway).

Developer or Knowledge Engineer The engineer should be familiar with the *ITPM Dashboard* environment, and needs to be experienced with UML class diagram annotation, and Java. In addition, the engineer should have basic knowledge about business rules (Drools) and SPARQL.

6.1.3 Process Description

The main steps of the CNIP are summarized in figure 6.2. The Business Process Management (BPM) notation is used to model the processes.



Consortium Negotiation and Integration Process
Involved Project Roles
Project Manager
Meeting Participants
Developer or Knowledge Engineer
Prepare Meeting (S1)
- Send out invitations
- State key objectives of the meeting.
- Ask for preparation
- ...
Severe Issues?
Unresolved Issues?
yes
yes
no
no
Discuss Unresolved Dev. Issues (S7)
Identify Required PM Systems (S2)
- PM Software / Tools
- Sources, Instances
- Concepts
- Access Data
- Domain Types
- Knowledge Sources Communication Plans
- Reporting/ Query Features
- ...
Prioritize and Group Data Sources (S3)
- By relevance
- By domain
- By number of users
- ...
Negotiate Best Integr. Alternative per Tool (S4)
- Is a connector/domain already available?
- Review / negotiate common concepts.
- What is the best integr. alternative?
+ Change process,
+ Unify PMS instances
+ Switch PM tool
+ Add/Reuse/Extend connector.
Define Additional Workflows (S5)
- Can manual integration steps be automated via workflows between different tool domains?
Example: If a requirement is removed the related issue should be closed.
- Refine models
- Research tool APIs
- Model connectors
- Generate beans, data access, interfaces, and tests
- Customize validations
- Integrate generated code
- Implement connectors
- Implement connector tests
- Add workflows (rules)
- Define distributed queries
Technical Integration (S6)
Unresolved Issues?
yes
no

Figure 6.2: Overview of the Consortium Negotiation and Integration Process (CNIP).

Prepare Meeting (Figure 6.2 - S1) The project manager convenes a meeting, and asks the invitees to prepare. Each consortium partner should create a list including the following items: requirements regarding PM software and tools, an overview of the current PM tool landscape within their organization, a description of the most relevant data exchange and integration processes (either manual or automated processes), concepts, and the roles involved in the PM process.

Identify Required PM Systems (Figure 6.2 - S2) Based on the information collected upfront, the consortium members disclose their PM landscapes and requirements during the meeting. Misunderstandings are resolved, synergies explored, and a common language is elaborated, e.g. in form of a glossary. Some of the questions that should be answered for each PMS, respectively data source, are:

1. Of which type is the software product (e.g. planning, collaboration, testing)?
2. How many instances of the software product are available? Where are they located (e.g. URL)?
3. How many licenses are available for each software instance? Are there any privacy restrictions or can anybody access the tool?
4. What is the domain type of the software? I.e., what are the common concepts on which the software type focuses (e.g. issue; backlog item; test record; project phase)?
5. How can the project manager, respectively the teams, access the data source? Share the access data and assign administrative permissions to the project manager if possible.

In addition, the following questions should be asked.

1. How do the tools relate / integrate with each other? Sketch a communication plan for the consortium project.
2. Is there any auxiliary PM knowledge available that is accessible and easy to integrate? Stimulate creative thinking. Do not only think about PMS, but include expert tools, files, and Web resources, to reveal useful historical PM knowledge. Examples include the reuse of wage rates (either from historical files, or public Web sites[1]), function point calculation tables, risk registers, etc.

Prioritize and Group Data Sources (Figure 6.2 - S3) Based on the initial discussion, and the identification of PMS, and data sources, the list is grouped and prioritized. This step ensures that all members agree on a common list. Based on the length of the list, some tools may turn out to be irrelevant; likewise, some elements may turn out to be missing. The items have to be grouped, based on the (main) domain type (e.g. PM tracking system; QM software).

[1]Payscale provides accurate wage rates for various countries and professions: `http://www.payscale.com`

Negotiate Best Integration Alternative per Tool (Figure 6.2 - S4) This process step facilitates the actual bargaining process, in which the meeting members try shall find an agreement, how the integration of each PM tool instance will be pursued.
There are several scenarios that can apply. In the ideal case, the PM tool, and also all tools of the same domain group, are already supported by the *ITPM Dashboard*, respectively the integration framework in place. In that case, the configuration and integration can be performed by the project manager (or a technician), immediately. No implementation effort results. In the second case, the tool (or a tool of the same group) is not known yet. In that case, the attendant developer (or knowledge engineer) and the rest of the meeting participants should undertake a brief analysis of the tool and its programming interfaces: Are there already implementations for tools of the same domain type available? Is it possible to reuse parts of the implementations? Of which concepts and interfaces does the tool/domain consist of, and which are relevant for the consortium project? Is there an API accessible so that the integration can actually be conducted?

Top - Management / Executives	**Project Manager**
• Does the solution improve the reporting process / results?	• Do I save manual synchronization or configuration effort?
• Are the cost savings of the reduced synchronization effort (PM) higher than the adaptation / configuration costs?	• Will information be synchronized/distributed quicker and/or more accurate to me and/or the project teams?
• Do I save manual synchronization effort?	• Will the solution increase the motivation of my project teams?
• Does the solution increase the team motivation?	• Does the solution reduce annoying polling activities on multiple PM software instances?
• Does the PM data consistency increase significantly?	• Is it possible to integrate data from different domains so that I can improve my reporting efficiency/effectiveness?
• Can the results be reused in other/future projects?	• Does the solution include reusable PM knowledge?
• (-) Does the implementation of the integration delay the project or increase the project costs?	• Can I reuse connectors in other projects?
Developer	• (-) If a connector is missing: Are my developers distracted from important development activities?
• Will I be able to reuse my preferred PM software (e.g. issue tracking)?	• (-) If a connector is missing: Do the available developers have the knowledge to implement a new connector?
• Will manual reporting activities be reduced (e.g. writing a weekly progress report or reporting vacations)?	• (-) Is ensured that no important data will be overwritten or lost?
• (-) Do we need to implement connectors by ourselves?	
• (-) Will features be cut due to limited integration capabilities?	

Table 6.1: Win conditions for different project roles (analogously to Boehm et al. [18])

The implementation of a specific solution has different outcomes and implications for the participating project roles and consortium members. Table 6.1 illustrates, how conflict areas arise from the ambivalent interests of different project roles. For instance, a project manager will typically stand up for the integration of a new PMS, unless it does not reduce her project budget or delay the project. The project owner, or in case of a consortium, the top management, will appreciate the integration only, if the integration effort is beneficial for the project outcome and/or future projects. Finally, developers will welcome any integration solution that cuts manual reporting efforts, and provides them with more freedom to use their individual PMS and tools. However, they may reject the integration, if the solution (e.g., development of a connector) must be implemented by themselves, and requires high training and testing efforts. In any case, an optimal win-win solution within the restricted solution space has to be pursuit. In other words, the best alternative, which contributes to the achievement of the project objectives, and is acceptable by all participants, should be chosen.

Among the action alternatives, not only the data integration via the applied data integration framework (e.g., the ITPM dashboard) should be considered. Table 6.2 exemplifies that it is also possible to adapt the PM process, or to install a unified software instance, which all team members can access. The appropriate solution depends on the costs of the integration alternatives. If the negotiators cannot agree on a specific solution, then games, such as the *Planning Poker* (not be discussed here; cf. [43]) help to determine a joint solution.

The negotiation process includes a joint modeling session, which is supported by the *Semantic Model Editor*. The mapping between the CNIP and the implemented prototype modules is discussed after the introduction of the *ITPM Dashboard* (cf. chapter 6.2).

Define Additional Workflows (Figure 6.2 - S5) The integration between tools of different domains should already be kept in mind, during the process steps (S1 - S4). In any case, after the agreement on a common set of PM tools to be integrated, work-flows between different tool domains must be explicitly specified (e.g., textual, or via BPMN, or UML activity diagram annotation). Work-flows enable the automation of manually conducted data synchronization activities, which require several steps of human mapping, transformation, and analysis. Step (S5) also involves the identification of advanced analytic queries, which the project manager or the top management require, to ease controlling and reporting activities (e.g., an aggregated view on the project state).

Technical Integration (Figure 6.2 - S6) After the meeting, an assigned developer or knowledge engineer needs to implement the negotiated extensions (if there are any). In particular, modifications performed on common concepts (tool domains) and connectors must be coded. The sketched work-flows must be translated into executable rules. Additional requirements of the project manager, such as the retrieval, aggregation, and visualization of distributed data also must be satisfied, e.g., via the implementation of queries.

Discuss Unresolved Development Issues (Figure 6.2 - S7) The project manager discusses the implementation progress with the developer. If non-resolvable issues appear during the

[2]Costs arise only for one specific member. Additional risk of installation delays.

Resolution	Example	Cost Range
Reuse Connector	An available Jira Issue connector is reused for the project.	-
Extend Connector	The available Jira connector does not support the exchange of the due date. Either the connector and/or the tool domain must be extended.	€
Add new Connector	Redmine has to be integrated, but no connector exists. However, as Redmine belongs to the PM issue tracking domain, and a similar Jira connector is already available, the common concepts are reusable, and the Jira connector serves as a reference implementation.	€ - €€
New PM Tool Domain	A Kanban tool is integrated for which neither a connector nor a domain exists. The common concepts are modelled and the connector needs to be implemented and integrated in order to tie the new software to the integrated tools.	€€ - €€€€
Switch PM Tool	Consortium member X wants to integrate their own PM solution. It turns out that the development of an integration tool is too costly. The representative of another consortium member Y proposes X to use the same software they apply (product ABC), which they are very satisfied, and for which a connector already exists. X relinquishes to integrate their own tool. Instead, they are curious to give product ABC a try.	€ - €€ [2]
Unify Software Instances	Consortium A uses a premium issue tracking solution. Consortium B uses a naïve freeware solution, for which no connector is available. A offers B free access to their premium solution.	€
Change Process	The PM wants to integrate the internal time tracking solution of consortium member X. No connector is available yet. As it turns out, all consortium partners use the same issue tracking solution. The other partners track their efforts directly within the issue tracking system. X was not aware of this solution, and after some explanation and clarification, X agrees that this approach results in virtually no additional effort for the team members. The project manager agrees that this solution is sufficient.	0 - €€€€€

Table 6.2: Action alternatives for the resolution of PM integration issues.

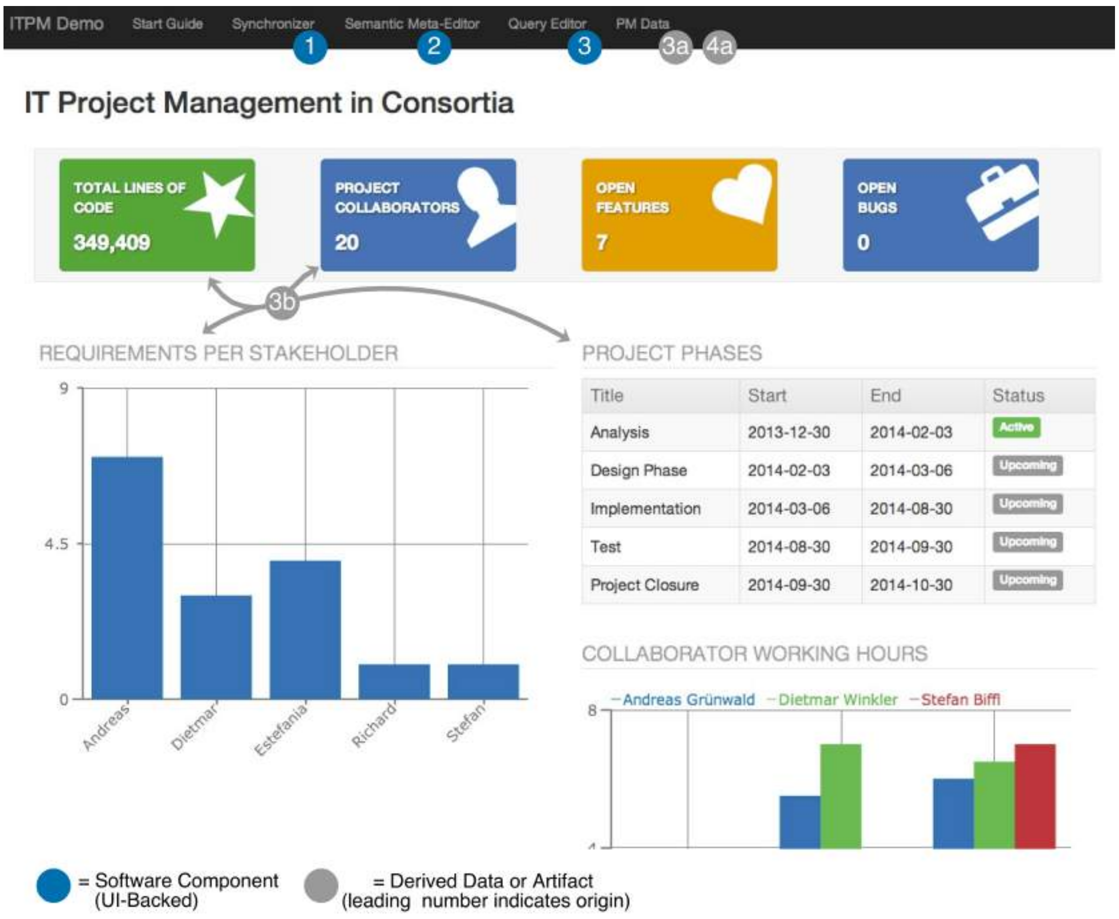
development of a connector, then the manager and the developer try to resolve the issue jointly. In case that the resolution requires severe changes in commonly negotiated concepts, or demands additional knowledge, then the consortium partners need to be informed. The partners may acknowledge the change after a quick communication (e.g. via email), or a new negotiation process, possibly a new meeting, must be set up.

6.2 Semantic PM Data Integration based on the ITPM Dashboard

Consecutively, the implemented *ITPM Dashboard* prototype is presented. The essential user interface functions and the related software modules are enumerated in section 6.2.1. Section 6.2.2 describes the support of the CNIP by the dashboard.

6.2.1 The Central ITPM Dashboard

The main function of the dashboard is to support the project manager in integrating and visualizing distributed and heterogeneous PM data. Additionally, the MDSE-approach eases the collaborative negotiation of common models, and serves as the starting point for the technical integration. Subsequently, the implemented software modules of the ITPM dashboard prototype are summarized.



Title	Start	End	Status
Analysis	2013-12-30	2014-02-03	Active
Design Phase	2014-02-03	2014-03-06	Upcoming
Implementation	2014-03-06	2014-08-30	Upcoming
Test	2014-08-30	2014-09-30	Upcoming
Project Closure	2014-09-30	2014-10-30	Upcoming

Figure 6.3: Visualization of the central ITPM dashboard. The numbers serve as a reference for the latter process implementation of the CNIP (figure 6.4).

Synchronization Cockpit facilitates the registration and monitoring of distributed PM software instances and their data sources. The UI enables the project manager to activate/deactivate specific sources, select the type of streaming (upstream, downstream), and keep track of the latest synchronization activities, including info messages. The synchronization procedure can be started on demand, or is permanently executed in background. Upstream means that the local data sources are pushed into the central data store. Downstream relates to the back-flow of changes performed in other tools, to a specific local tool.

Semantic Model Editor Aids managers, knowledge engineers and developers in documenting, refining, and sharing knowledge about specific tools (connectors), and ITPM domains (e.g., parts of PMBOK knowledge areas), based on UML class diagram notation and OWL.

Model-Driven Code Generator aids technicians to fluently refine and translate models into reusable code, to store entities in semantic data stores (OWLAPI, Jena). By automating the generation of Java beans, DAOs, and unit tests, the programmatic integration effort is minimized. Additional features, based on the Semantic Model Editor, smooth the integration into the OpenEngSb environment.

Semantic Query Editor Manages a set of semantic rules and queries, which are used to aggregate PM knowledge across multiple domains. Supports syntax highlighting to simplify the design of SPARQL queries, auto-completion (integrated with the Semantic Meta-Model Editor) and the visualization of information (tables, graphs, diagrams, KPIs). The reuse of queries and rules is facilitated by organizing the rules based on the PMBOK knowledge areas.

PM Connectors and Synchronization Logic Several PM domains and connectors were studied and implemented to demonstrate the integration of PM data across, and within PM domains (results are presented in chapter 7). A (naive) synchronization algorithm has been developed to enable the data integration of local sources, close to real-time (e.g., scan for local changes each 2 minutes).

ITPM Dashboard combines the above mentioned modules within a single Web-Based UI. In addition, browsing, data modification, and visualization features are provided. The configurable home page visualizes the most relevant PM data based on the defined rules and queries. The dashboard is pictured in figure 6.3. The numbers in figure 6.3 are used to provide back-links for the upcoming mapping between the CNIP and the software components, data, and artifacts through runtime.

6.2.2 Process Implementation: CNIP Support based on the ITPM Dashboard

The CNIP is backed by several UI features of the ITPM Dashboard. The MDSE approach eases the collaborative negotiation of common models, and reduces the gap between management integration requirements and technical integration.

Figure 6.4 visualizes the facilitation of the CNIP by the ITPM dashboard, and depicts the flow of the most relevant data artifacts (e.g., code, queries, models, data instances).

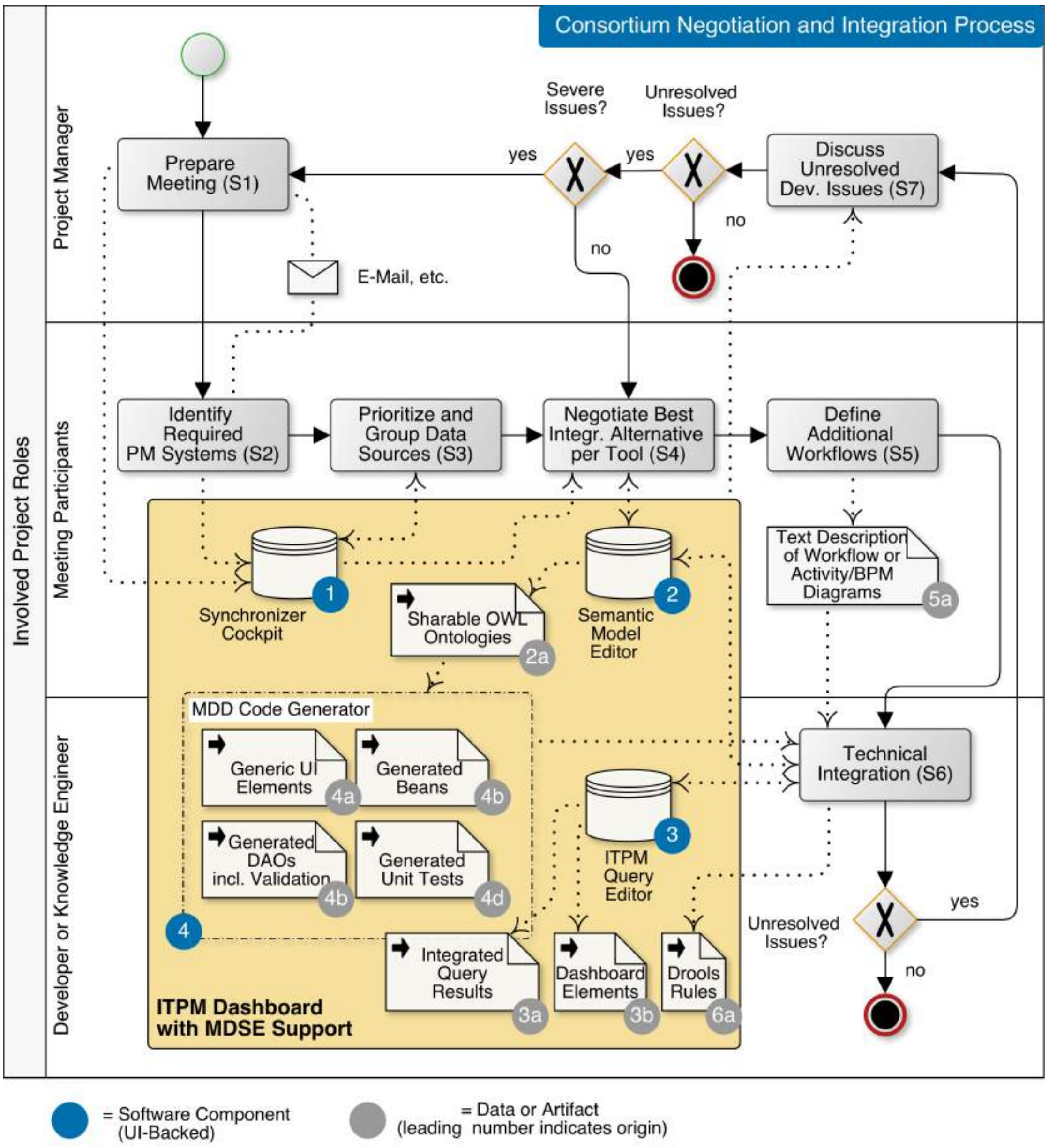

Consortium Negotiation and Integration Process
Involved Project Roles
Project Manager
Meeting Participants
Developer or Knowledge Engineer
Prepare Meeting (S1)
E-Mail, etc.
Severe Issues?
Unresolved Issues?
yes
yes
no
no
Discuss Unresolved Dev. Issues (S7)
Identify Required PM Systems (S2)
Prioritize and Group Data Sources (S3)
Negotiate Best Integr. Alternative per Tool (S4)
Define Additional Workflows (S5)
Text Description of Workflow or Activity/BPM Diagrams
5a
Synchronizer Cockpit
1
Sharable OWL Ontologies
2a
Semantic Model Editor
2
MDD Code Generator
Generic UI Elements
4a
Generated Beans
4b
Generated DAOs incl. Validation
4b
Generated Unit Tests
4d
4
ITPM Query Editor
3
Integrated Query Results
3a
Dashboard Elements
3b
Drools Rules
6a
ITPM Dashboard with MDSE Support
Technical Integration (S6)
Unresolved Issues?
yes
no
= Software Component (UI-Backed)
= Data or Artifact (leading number indicates origin)

Figure 6.4: Relation between the CNIP, and the ITPM dashboard components and artifacts (extension of figure 6.2). Relevant data flows are visualized.

Subsequently, the CNIP is revised, to establish a link between (a) the process steps, (b) the ITPM dashboard components, and (c) the project roles. Note that the numbers in figure 6.4 correspond with the UI elements of figure 6.3 (visualization of the central ITPM dashboard).

Prepare meeting (Figure 6.4 - S1) Although the project manager or a developer can prepare several models and configurations in the ITPM dashboard (e.g., via the Semantic Model Editor), and share the link with the meeting participants, the current prototype version does not actively support this step. Relevant members are manually identified, and invited via email.

Identification and Grouping of PM Data Sources (Figure 6.4 - S2, S3) The synchronizer cockpit (1) supports the project participants to collect, group, refine, and maintain information about the available data sources (e.g., type of tool, main concepts, access data). Whether the adding of the identified data sources to the ITPM dashboard starts at process step S1, S2, or S3 is up to the project manager. In any case, after the execution of S3, all relevant data sources must have been added to the cockpit.

Negotiate Best Integration Alternative per Tool / Common Concepts (Figure 6.4 - S4) The Semantic Model Editor (2) facilitates the collaborative design and negotiation of common concepts, based on UML class diagram annotation. The meeting participants negotiate and document their integration needs in form of accessible, reusable PM data models. The models are refined, and technically integrated into connectors, by developers.

If the meeting participants agree to utilize the available data integration middle-ware (ITPM dashboard, OpenEngSb) for integration of a specific PM tool, then four scenarios appear (cf. figure 6.2).

1. Reuse Connector: In the simplest case, a connector is already available. The concepts are documented in the Semantic Model Editor, and available for review.
2. Extend Connector: If the review reveals that the connector does not support parts of the business requirements (for instance, the project manager cannot perform a specific reporting query, because of a missing data attribute), then the connector model has to be updated. The change requirements are documented within the connector model. The specific implementation, based on the available functionality of the underlying API, is typically left to the responsible developer. Severe modifications require the redesign of related common concepts, and tool domains. In that case, the meeting participants should review the affected models, to ensure that data changes of local PM tools are appropriately propagated.
3. Add New Connector: Like the extension of existing connectors, the creation of a new connector is mostly developer-driven. The meeting participants (managers) document the main concepts of the PM data source within a (new) ITPM data model. The model is then refined with the support of the present developer(s) (review model updates, research connector API, refine model). Depending on the scope of the integration, and the extend of the connector, the latter activity either takes place during the meeting, or afterwards.
4. New PM Tool Domain: In the worst case, the management realizes that an entire tool domain needs to be added, respectively that an integral part of the central ITPM model is missing, or not usable. In that case a systematic joint modeling session is required.

The suggested solution approach follows a slightly adapted *Ontology 101* process [91]: First, the scope of the common concepts is defined, by identifying the most important concepts (classes) . Second, the models for other tool domains are reviewed, to determine reusable parts and patterns for the new concepts. Third, the common concepts are refined, by adding attributes, hierarchical relations, and associations between classes. Finally constraints, rules/queries, and comments (i.e., annotations), which serve as the input for the MDSE approach, and for the technical implementation, must be documented. The documentation of the model, and the definition of basic (data) constraints is directly supported by the Semantic Model Editor's UI. In addition, the systematic adding of sophisticated, machine-interpretable expert rules, on top of the ITPM models (i.e., OWL ontologies) is facilitated.

The outcome of a modeling session are shareable models, respectively OWL ontologies (2a). These models serve as the input for the code generator (4). The Semantic Model Editor provides the interface between the management, and the technical integration. The generator automates and simplifies the code implementation, and provides an additional step to verify the quality of the ITPM models. The generated artifacts include generic UI elements (tables, forms) (4a), beans (4b), data access objects (DAOs) (4c), and unit tests (4d). The DAOs enable the abstracted access to semantic data stores (OWLAPI, Jena). In addition, the DAOs provide default data validation mechanisms, and patterns to inject additional data constraints (in conformance with OWL facets).

Define Additional Work-flows (Figure 6.4 - S5) The (graphical) description of work-flows is not supported by the current version of the prototype. Thus a textual description or activity diagrams (7) need to be created and handed over to the developer who is in charge implement the work-flow integration.

Technical Integration (Figure 6.4 - S6) The technical integration is based on the results of the modeling session (2a), and the code generation (4). The developer needs to identify and implement the demanded connectors and extensions. The generated artifacts simplify the integration and validation cycle of external PM concepts. In addition to the basic data integration, rules (6a) based on work-flow descriptions (5a), and ITPM queries need to be implemented.

ITPM queries are demanded by the project manager, to aggregate and visualize PM data of various knowledge areas. The queries simplify monitoring and reporting activities, and help to acquire additional project knowledge. The *ITPM Query Editor* enables the efficient design of ontology queries and rules, to create aggregated data visualization for the dashboard (3a, 3b). The project manager can choose, to display important query results within the central ITPM dashboard (home page; 3b). All queries are organized within in an internal ontology, according to the PMBOK knowledge areas.

Discuss Unresolved Development Issues (Figure 6.4 - S7) The Semantic Model Editor utilizes the communication of implementation issues between the developer and the project manager, based on model annotations and comments. Analogously, the Semantic Query Editor enables the communication of query results an data visualizations, based on ITPM dashboard visualizations (3a, 3b). The code generator supports the developer in translating model updates automatically into code. Based on the MDSE approach, the implementation effort is reduced, and manual modifications (programmatic validations, ontology extensions) are retained.

The underlying technical processes, including the integration of connectors, the definition of cross-domain work-flows, and the implementation of data visualizations based on SPARQL, are detailed in appendix A.1. The relation between the components' artifacts (models, ontologies, code, UI elements), and their efficient re-utilization, based on MDSE, is outlined next.

6.3 The Semantic Model Editor and Derived Artifacts

The Semantic Model Editor is a Web-based UML class diagram editor, enriched with documentation features, code generation capabilities for semantic data stores, and tools that facilitate the efficient integration of PM systems, based on the OpenEngSb framework. The structure of the models is based on the suggested knowledge by the PMBOK. These knowledge areas were extended, to cover additional requirements for specific industrial PM tools.

Subsequently, the MDSE capabilities of the Semantic Model Editor are outlined. First, the relation between the various types of models, artifacts, is described at the level of the UI. Next, the existing umlTUowl is extended for MDSE.

6.3.1 UI Support for Model-Driven Software Engineering (MDSE)

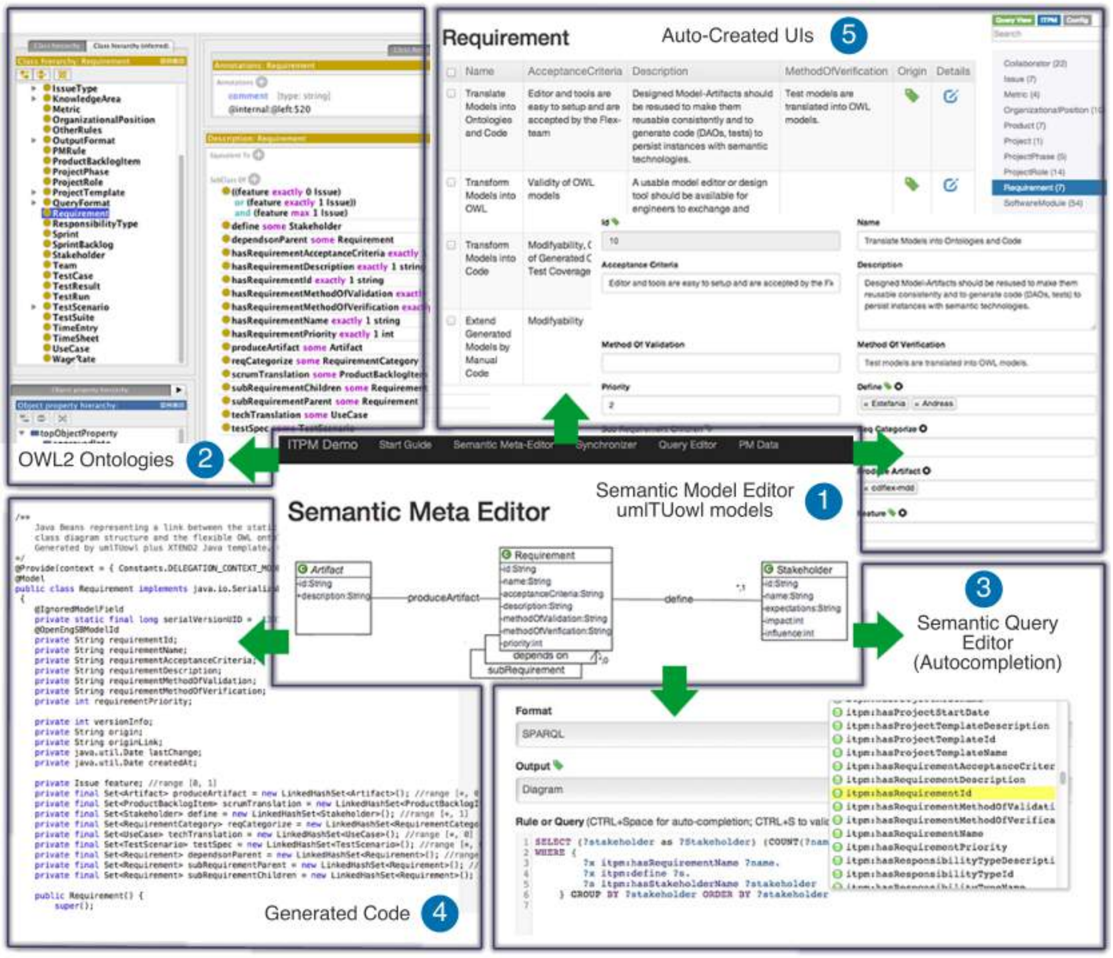
Figure 6.5 presents the Semantic Model Editor as an integral part of the *ITPM Dashboard*. The relations between the central ITPM models, and the derived software artefacts (ontologies, code, UI elements) are visualized.

1. The Semantic Model Editor UI is the central element of the MDSE approach. The visible outcome of the editor are a set of UML class diagrams.

2. Based on the UML class diagram notion, shareable and machine-interpretable OWL 2 ontologies are derived. For that reason, the existing umlTUowl approach is extended.

3. The created models are utilized to support users in formulating semantic queries. Based on the available concepts, the Semantic Query Editor provides auto-completion features for the implementation of SPARQL queries. The models serve as a dictionary, which can be looked-up by the project manager, or the knowledge engineer, and simplify the formulation of complex integration queries.

4. The umlTUowl meta-model is utilized to derive Java code (beans, DAOs, unit tests) for the access of semantic data stores. The prototype includes code generators for the OWLAPI and Jena (SDB and TDB).



	Name	AcceptanceCriteria	Description	MethodOfVerification	Origin	Details
	Translate Models into Ontologies and Code	Editor and tools are easy to setup and are accepted by the Flex-team	Designed Model-Artifacts should be reused to make them reusable consistently and to generate code (DAOs, tests) to persist instances with semantic technologies.	Test models are translated into OWL models.		
	Transform Models into OWL	Validity of OWL models	A usable model editor or design tool should be available for engineers to exchange and			
	Transform Models into Code	Modifyability, of Generated C Test Coverage				
	Extend Generated Models by Manual Code	Modifyability				

```
/**
    Java Beans representing a link between the stati
    class diagram structure and the flexible OWL ont
    Generated by umlTUowl plus XTEND2 Java template,
*/
@Provide(context = { Constants.DELEGATION_CONTEXT_MO
@Model
public class Requirement implements java.io.Seriali
 {
    @IgnoredModelField
    private static final long serialVersionUID =
    @OpenEngSBModelId
    private String requirementId;
    private String requirementName;
    private String requirementAcceptanceCriteria;
    private String requirementDescription;
    private String requirementMethodOfValidation;
    private String requirementMethodOfVerification;
    private int requirementPriority;

    private int versionInfo;
    private String origin;
    private String originLink;
    private java.util.Date lastChange;
    private java.util.Date createdAt;

    private Issue feature; //range [0, 1]
    private final Set<Artifact> produceArtifact = new LinkedHashSet<Artifact>(); //range [*, 0
    private final Set<ProductBacklogItem> scrumTranslation = new LinkedHashSet<ProductBacklogI
    private final Set<Stakeholder> define = new LinkedHashSet<Stakeholder>(); //range [*, 1]
    private final Set<RequirementCategory> reqCategorize = new LinkedHashSet<RequirementCatego
    private final Set<UseCase> techTranslation = new LinkedHashSet<UseCase>(); //range [*, 0]
    private final Set<TestScenario> testSpec = new LinkedHashSet<TestScenario>(); //range [*,
    private final Set<Requirement> dependsonParent = new LinkedHashSet<Requirement>(); //range
    private final Set<Requirement> subRequirementParent = new LinkedHashSet<Requirement>(); //
    private final Set<Requirement> subRequirementChildren = new LinkedHashSet<Requirement>();

    public Requirement() {
        super();
```

Format: SPARQL
Output: Diagram
Rule or Query (CTRL+Space for auto-completion; CTRL+S to valid

```
SELECT (?stakeholder as ?Stakeholder) (COUNT(?nam
WHERE {
        ?x itpm:hasRequirementName ?name.
        ?x itpm:define ?s.
        ?s itpm:hasStakeholderName ?stakeholder
    } GROUP BY ?stakeholder ORDER BY ?stakeholder
```

Figure 6.5: The Semantic Model Editor and derived artifacts (ontologies, user interface elements, code

.

5. The ontologies are leveraged to derive UI elements, such as tables, forms, and navigation items. The resulting UIs aid managers and engineers, to centrally search and modify data instances. By adjusting the PM models appropriately, the layout of the forms can be tweaked. For instance, the visualization of help texts, and the layout of input fields (text field, text area, checkbox, etc.) are supported.

6.3.2 Extending umlTUowl: Mapping between UML, OWL, Code, and UI

The supported UML class diagram elements, and the mapping between UML and OWL concepts, based on *umlTUowl*, are well-documented [48]. The proposed solution extends *umlTUowl* in two ways: First, the scope of the mapped UML and OWL concepts is extended. Second, umlTUowl is leveraged to derive and generate complementary software artifacts (code and UI elements).

Extending the Scope of the UML to OWL Mappings

The umlTUowl approach is extended to support a larger subset of UML to OWL mappings. Table 6.3 presents an updated version of the table presented by Grünwald and Moser [48]. The highlighted elements on the right-hand side represent the UML class diagram elements that are supported by the extended version of umlTUowl, and the Semantic Model Editor. "Multiplicity of attribute value" means that a class attribute can have several values (e.g. a pet can have several nicknames), rather than a single value. Data constraints for attributes are defined via annotations in the Semantic Model editor. These are saved either as *OWL Facets*[4], or as comments in the OWL ontology. The implementation of OWL facets for the expression of data constraints in form of regular expressions resulted in problems with the Pellet reasoner[5], i.e., the reasoner hang up. Hence the data constraints are stored as comments rather than as OWL facets, by default. "Redefinition of derived attributes" addresses the issue that a data type of an attribute can be overridden, if a subclass defines the same attribute with a different data type. Support for "attribute uniqueness" is implemented via the *hasKey* axiom[6] in OWL. The modeling of enumerations is integrated into the Semantic Model Editor, because (a) enumerations showed to keep the graphical models more compact, (b) enumerations frequently appeared while implementing PMS connectors (e.g. issue types, priorities, work item states). The mapping of enumerations into OWL and Java code was of minor concern. Thus, UML enumerations are translated into associations (object properties), and the enumeration values are represented as specifications of the abstract class. For the implementation of the prototype, this turned out to be an effective solution, with low implementation effort.

Mapping between UML, OWL, Code, and UI Elements

Based on a MDSE approach, umlTUowl is extended for the derivation of additional artifacts (cf. figure 6.5). A compatible set of relevant mappings is identified. The main objective is not to provide a complete set of mappings, but to show that OWL ontologies leverage the effective mapping between various software artifacts, are intuitively, and contribute to the reduction of redundant development work.

Table 6.4 outlines the mappings between the main four types of derived software artifacts (UML class diagram notation, OWL, Java code, UI elements), based on the high-level concepts of UML, and the umlTUowl-internal meta-model. The mapping between UML and OWL is already outlined in figure 6.3.2, and not discussed further. Subsequently, the mappings between UML, OWL, and Java code, and the derivation of UI elements, are outlined. A comprehensive overview on the mapping is provided in appendix A.2.

[4]W3C Recommendation for OWL2: `http://www.w3.org/TR/owl2-syntax/`
[5]Pellet Reasoner: `clarkparsia.com/pellet`
[6]W3C Specification of OWL 2 Keys: `http://www.w3.org/TR/owl2-syntax/#Keys`
[7]Uniqueness partially implemented implicitly (only for single elements)

Supported UML Class Diagram Elements	**Previously Unsupported Elements / Newly Supported Elements**
• classes	• multiplicity of attribute value
• abstract classes	• attribute values except primitives
• interfaces	• package elem. inside a diagram
• generalization	• data constraints
• multiple packages (diagrams)	• class operations
• attributes	• association classes
• visibility of attributes (partly)	• n-ary associations
• attrs. with primitive data types	• overlapping/disjoint classes
• attrs. with XML data types	• roles
• associations	• XOR annotation
• navigable associations	• redefinition of derived attributes
• multiplicity of associations	• subset annotation
• aggregations	• ordering + uniqueness attr. annotation.[7]
• compositions	• datatype meta annotation
• labelled endpoints (MS Visio)	• enumerations
• comments / note elements	• stereotypes

Table 6.3: Supported vs. not supported UML class diagram elements. The highlighted items are those elements that are supported by the extended version of *umlTUowl*.

Mapping between UML, OWL, and Java Code Each UML model (class diagram) consists of a set of sub-packages. The Semantic Model Editor enables the specification of a name-space, and an optional OpenEngSb module type, for each model (ITPM default model, connector, tool domain). Based on the namespace and the OpenEngSb module type, the path for the code generation within the developer's code repository is derived.

Classes are distinguished into interfaces, abstract classes, enum classes, and concrete classes (classes that can be instantiated). For each class, a Java bean is created. The bean consists of private variables that correspond to the model attributes (OWL data properties), and associations (OWL object properties). The main supported primitive data-types are string, integer, double, float, boolean, and date respectively datetime. For primitive attributes with multiplicity, an *ArrayList* is created. Depending on an association's cardinally, either a single variable, or an `ArrayList` is created (for multiplicities > 1). For both, attributes and associations, getters and setters are created. The variable names for associations are derived as follows: if the user specified a name, then this name is used. Otherwise, the allocation of the name follows a simple, configurable pattern (e.g., for compositions, the name is typically derived from the names of the linked classes, such as "hasRequirement"). Nested hierarchies (generalizations) are supported for any type of class. For instance, if class A is a child of concrete super-class B, and two interfaces C and D, then the code `A extends B, implements C, D` is derived.

DAOs and unit tests are only generated for concrete classes. Each DAO provides basic CRUD methods, and a set of additional methods to perform different types of full-text search (e.g., search for the content of specific fields). While accessing the semantic data stores, the developer can specify whether to retrieve the instances at the first level (e.g., a list of requirements), or to also read instances at deeper levels (e.g., also retrieve the entire data for the linked stakeholders).

For each DAO, basic data validation is included. By default, each string must contain at least one character, and each numeric field must be greater than zero. The validation mechanism can be relaxed, by extending the specific DAO, and reformulating the validation for the specific fields. When the derived code artifacts are re-generated, these manual extensions remain. The Semantic Model editor supports the formulation of more complicated data constraints, based on OWL Facets. The OWL facets are translated into Java constraints and are verified during the persistence of data instances. A typical constraint would include the validation of email addresses, based on a regular expression.

For each DAO, a set of JUnit tests is created. The tests verify the DAOs against a set of thresholds. For instance, if an association has cardinality 1-5 assigned, then tests for zero, one, five, and six linked entities are generated (the first and the last test represent negative tests). Another example is the verification of bidirectional associations: If a stakeholder is assigned to a requirement, and the requirement is persisted, then it must be ensured that the requirement is also back-linked from the stakeholder.

The OpenEngSb specific annotations (table 6.4) enable the efficient interplay of the MDSE and the OpenEngSb. The most relevant annotations will be detailed in section 6.4.4.

Derivation of UI Elements For all ITPM models, except tool-specific connector models, a menu tab is created within the ITPM dashboard (sub page: *PM Data*). For each (concrete) class, a search page is derived. This page consists of a search form, a table with pagination, and control elements for the deletion, adding, and modification of records. The content of the table is computed based on a simple heuristic, to avoid exceeding the maximum page width. For instance, if the table becomes too broad, then columns with a high rate of empty cells are removed from the view. Similarly, columns with a high diversity of content are favored to columns with a very low variety.

For each (concrete) class, a new item in the navigation bar is created. Compositions are not shown in the menu, because those can only exist in conjunction with another instance, and hence they are linked as sub-entries within the record form of the main entry. The navigation item links to the search table. Within the search table, records can be removed, edited, or added. For each record, basic meta information, viz., the origin of the record (e.g., " Jira Instance of Organization A"), the version of the record, its creation date and the last date of modification, are maintained and visualized. Generated forms enable the modification of records based on the central dashboard. Forms consist of input fields for (meta) attributes, and sub-forms that enable the user to modify and browse the content of classes that are linked via meta associations. The main field types are summarized below.

Fields based on Attributes By default, a text field is created for each model attribute. Based on the data type, some of the fields are optimized. For instance, a check-box is rendered for

UML Class Diagram (Sem. Model Editor)	OWL Concept	Generated Code	Derived UI Elements
Model	OWL Ontology	None (but used to derive code location).	New tab in main menu
Package	reflects in harmonized and merged OWL ont.	Java package	-
Class	OWL Class	Java bean; DAO; Unit test class	Navigation item linking to a table search, and a form
Attribute	Data property with OWL restriction (cardinality) and data range (XML or OWL data types); rdfs:subClassOf axiom links to class.	Private field incl. getter and setter	Input field (text field, checkbox, select menu)
Association	Object property with range and domain; rdfs:subClassOf axiom that links to the OWL class(es). owl:inverseOf axiom in case of unidirectional associations.	Max. cardinality > 1: A getter method that enables the access to list elements. Cardinality <= 1: Getter and setter for modifying a private variable.	A dynamic select field that enables the user to add, remove, or modify linked instances via a sub-form that can be fold-out within the main form.
Generalization	rdfs:subClassOf	Java Inheritance (class A extends class B; class A implements interface B)	A meta association points to a nested class hierarchy: the fold-out sub-form includes an additional dynamic field for the selection of the specific class type (only active when adding new instances).
Comment Annotation	rdfs:comment	Java comments and additional data constraints (regular expressions)	Help texts at form and field-level (collapsed popovers).
OpenEngSb Specific Annotation	rdfs:comment	The location for the source code (e.g., for connectors, domains), and the generated artifacts (beans, DAOs) are controlled.	For tool-specific connector models no UI elements are derived.

Table 6.4: Overview of the main meta elements and their mappings.

boolean values. If a meta attribute defines a data constraint, and that constraint explicitly states that the number of characters exceeds a specified length (e.g., 100 characters), then instead of an input field, a longer HTML text area is rendered. Analogously, although not implemented yet, the input of a date can be supported by a date picker, the input of numeric values could be restricted further, and so on.

Fields that link to an Enumeration Linking to an enumeration means that the value is part of a known solution space. In that case, a dynamic auto-completion field [9] supports the user and verifies the input.

Fields / Sub Forms based on Associations Data instances are linked to each other based on associations (OWL object properties). There are three main challenges regarding the representation of linked instances. First, linked data instances need to be displayed transparently. Because most associations permit multiple data instances to be linked, it must be ensured that (a) the instances are clearly identifiable, (b) each record is linked only once, and (c) only the relevant instances are available for selection.

Second, there must be a way to add non-existing records, and link them with the main record fluently. In other words, the user does not want to switch between forms, in order to create and link data records with each other. Additionally, in some cases (e.g., recursive self-references, compositions) this is neither possible.

Third, existing links need to be removable. This includes the physical removal of links, inverse links (i.e., semantic properties), and the removal of the linked data instance itself, in case of compositions.

The above-mentioned challenges are approached as follows. For each association, a dynamic select field (select2) is created. Based on the defined model cardinality, the selection of entries is restricted. For each selected data record, a tag with the name of the record is displayed. The naming of the tag is based on a heuristic; attributes whose name equals "name" are assumed to contain the name of the instance. If no such attribute exists, then the first attribute that contains "name" is selected. Otherwise, the first attribute of type string, or the id, is used. The creation of the name can be overwritten programmatically.

The user selects the specific tag, and clicks on a displayed "edit" icon, in order to modify any nested data instance. A (recursive) sub form, including all attribute fields of the sub-entity, is displayed. The user can edit the record in place. After the modifications are performed, the sub form is collapsed again.

Additionally, the user can add new records, or remove links to existing records, by removing the tag from the select form. The behavior for adding a new record is similar to the procedure of editing a record. The main difference is that an additional select menu is displayed, which lets the user select the specific sub-type of the new instance, based on the modeled class hierarchy. The record is not added physically, until the changes in the main record are submitted.

All forms and fields support the placement of help texts. In other words, all comments that are added to the semantic model do not only increase the re-usability across modelers, but they are also reused as help texts within the generated forms.

[9] Select2 UI Component: `http://ivaynberg.github.io/select2/`

6.4 A Repository for Semantic ITPM Models

The previous section described the technological aspects for the persistence of semantic models, respectively ontologies. The efficient reuse of models, for the derivation of code, and UI elements, has been made clear. However, the structure of the designed project management models, and the efficient population of the resulting PM ontologies, has been left out so far. This section describes the integration of PM models and data, based on the *Semantic ITPM Model Repository*, and MDSE.

The diversity of PM systems, tools, and standards, results in complex and elusive data integration models. The omnipresent necessity to utilize domain- and enterprise-specific PM processes and data, implies that the prescription of an optimal, universally applicable ITPM model, such as the RefModPM, is not expedient. Hence, the solution approach does not focus on the specification of a complete ITPM model, but suggests a framework, to enable the community-based exchange of small-scaled PM (sub-) models that are understandable, reusable, adaptable, extensible, and easy to integrate.

The remainder of this section is structured as follows. First, the implemented quality attributes for the semantic ITPM model repository are summarized (section 6.4.1). The goal is to make PM models shareable, reusable, and efficiently applicable. Afterwards, the structure / framework of the model repository, as implemented for the ITPM dashboard prototype, is described (section 6.4.2). This part includes a description of the ontology structure, the organization of central ITPM models based on the PMBOK knowledge areas, and the types of data (configurations, rules, ITPM models, connector models), the degree of their reusability, and their application. Next, the efficient integration of local data into the central semantic ITPM data store, based on the designed models, and MDSE is described (section 6.4.3). Finally, the support of the Semantic Model Editor, for the efficient integration of local data, based on MDSE, and the OpenEngSb, is exemplified in section 6.4.4.

6.4.1 Model Repository Quality Attributes

Subsequently, the most relevant quality attributes that are addressed in the design of the ITPM model repository, are summarized. The used software and data quality attributes refer to the ISO 25010 and ISO 25012 quality standards [12] [68], and the *Information Quality Evaluation Framework* of Rafique et al. [99].

Comprehensibility All models are representable and extensible, based on the common UML class diagram notion of the *Semantic Model Editor*. The representation of ontologies via UML syntax, enables the compact and efficient design of PM models. Based on the experience of the CDL-Flex with industrial partners (IT domain), UML class diagrams are well-accepted by practitioners, and promote the communication between management and technicians.

Reusability / Portability All models are shareable across organizations, divisions, and communities (PM, knowledge engineering), based on the semantics of the underlying OWL ontologies. The persistence of the class diagrams as OWL files solves the problem of pre-existing solutions in which either (a) the models are provided in UML class diagram notion, but cannot be reused electronically because of the lack of a common interpretable syntax (XML Metadata Interchange (XMI), cf. [48]) [6], or (b) the models are provided as pure OWL ontologies, but difficult to understand / maintain by practitioners (lack of industrial knowledge; no compact representation of concepts) [3] [117]. The ITPM repository stimulates the collection and elaboration of ITPM knowledge, in form of reusable models. Once a model is available (e.g., for a specific PMS connector, such as Jira), the model becomes explorable and reusable, by any organization or partner.

Transparency The models enable the communication of available PM software interface concepts to the management. API concepts, usually only subject to technicians, become transparently available in form of vivid graphical diagrams.

Technical Reusability Also technical modules are reusable by other organizations. For instance, some implemented PM software connectors are provided to the OpenEngSb project [10]. The designed PM models serve as the documentation for the implemented software. Similarly, the models are reusable for the derivation of code and artifacts, such as generic DAOs or UI elements, as presented in section 6.3. Finally, SPARQL queries and rules, although demanding minor modifications, can be reused across organizations, to implement escalation mechanisms, anti-patterns, or on-the-fly visualizations.

Scalability / Representational Adequacy Literature and industrial experiences show that ITPM models tend to get out of hand and result in very large, unmanageable diagrams. The Semantic Model Editor is optimized to efficiently support the modeling of consistent, small-scale models. In addition, a template-based strategy for the structuring of the model repository, based on the PMBOK knowledge areas, is provided. The procedure is elaborated in section 6.4.2.

Adaptability All models and common concepts are tailor-able to the organization's needs and environment. Either the original model can be modified, or a copy can be created and adapted.

Accessibility Unlike other modeling approaches, such as the EMF[11], the approach requires no setup of resource-intensive development environments and plugins that are only usable by developers. The prototype is installable in a few minutes, or can be hosted on a central server, so that related project roles can access a shared Web-based instance without additional installation effort.

Consistency A problem related to the lack of a commonly accepted UML exchange format (XMI) is that the consistency of diagrams is difficult to ensure. As a consequence, UML class diagrams are often considered as graphical representations, rather than reusable, semantic models. The application of OWL ontologies for the persistence of models ensures

[10]OpenEngSb Website: `http://www.openengsb.org`

[11]Eclipse Modeling Framwork: `www.eclipse.org/emf/`

the consistency of models in several ways. First, the application of a reasoner ensures the consistency of a model. Second, the application of the MDSE-based code generator, provides an additional verification step for the consistency of a model. For instance, if no data type is specified for a class attribute, then the code generation will result in a compiler error, or test failures. Third, ontologies enable knowledge engineers to specify additional constraints. For instance, the knowledge engineer could create a rule, which states that a team member can only work 24 hours per day. If the condition is violated, then the reasoner will report the inconsistency.

6.4.2 Structure of the Model Repository

The model repository comprises four types of ontologies:

1. The *data source configuration* ontology contains the records for the integrated PM software instances, and maintains the access and the mapping between local data stores and the central semantic store.
2. The *rule storage ontology* preserves defined queries over integrated PM data, and stores information about the visualization within the dashboard (e.g. table format, graph format, diagram).
3. The *ITPM core model* represents the central model. This ontology maintains all common concepts, including properties and links between the various concepts. The ITPM model is the foundation for the rest of the ontologies and the *ITPM dashboard*.
4. The *connector and tool-domain specific models* represent the fourth artifact. These models document accessible PM software products based on their APIs.

While the structure of (1) and (2) is fixed, (3) and (4) represent the extensible model repository, which includes the adaptable PM concepts for the deducting and visualization of aggregated PM data, and the specific connector models for the integration of versatile PM software products and PM knowledge.

Table 6.5 summarizes the main ontologies, including their core structures, the degree of reusability, and their application. The structure of the data source configuration ontology, and the rule storage ontology are fixed. These ontologies only need to be filled with instances to operate the *ITPM dashboard*. Data source configuration records are either project- or enterprise specific, while rules/queries are reusable by the project manager across inter-organizational projects. Based on the rule-based system, a project manager is able to build a personal repository / set of rules and queries that can be reused and successively improved throughout the execution of projects. In order to collect and structure queries, by default each rules needs to be assigned to one of the PMBOK knowledge areas. This categorization is extensible. For instance, domain-specific areas can be added.

The ITPM core model is the most comprehensive model. It is split into sub-packages, each representing one PMBOK knowledge area. To keep the ITPM model scalable, identified common concepts must be modeled as UML classes within the adequate knowledge area, including

Model	Structure	Context / Reusability	Need to Adapt	Application / Usage
Data Source Configuration Ontology	Single ontology	Project or enterprise-wide.	No need	Add data instances in ITPM Dashboard and monitor the integration based on the synchronization cockpit. Configure the mapping of multi-models.
Rule Storage Ontology	Single ontology; rules are structured according to the PMBOK knowledge areas	Typically each PM maintains a set of rules; these can be exchanged with other PMs or organizations.	No need	Add queries based on the Semantic Query Editor. KPIs, tables, graphs, etc. can be visualized in the PM Data sub-page, or integrated into the central ITPM dashboard.
ITPM Core Model	One model; Various packages (PMBoK knowledge areas plus industry-specific extensions)	Community-wide.	Low-med.	Adapt/extend the models according to specific needs. Map the connector-specific models to the related concepts of the ITPM Core models.
PM Connector Models	One package per model	Connector models are typically determined by their available APIs and hence can be reused and gradually refined across the community.	Very low	Support for the implementation of a connector based on API and business requirements. The models can be reused to generate code to smooth the transformation between the tool API and the common concepts.
Tool-Domain Specific Models	One package per model	Application specific (OpenEngSb). Alternative/extension to the ITPM Core Ontology.	Low-med.	Same as ITPM Core model, but the specific tool-domains are isolated and are not linked with concepts of other domains.

Table 6.5: Structure of the ITPM Model Repository.

all relevant attributes and associations. To keep the model representations compact, a defined class can be reused in any other sub-package, to express associations between classes that belong to different knowledge areas. For instance, *requirement* and *artifact* are defined in the *scope management* area. An *artifact* is assigned to one or several requirements, and consists of an id, a name, and a description. A *test case* belongs to the *quality management* area. Because test cases relate to an *artifact* as well, the artifact class needs to be modeled twice. However, because the artifact class belongs to *scope management*, and the related attributes (id, name, description) are already defined within this knowledge area, it is sufficient to re-create the *artifact* class without attributes in the *quality management* diagram. umlTUowl extensions automatically resolve the relations and the class associations, while deriving OWL ontologies and the code artifacts. The proposed approach helps to avoid redundant definitions, and provides a clear categorization of classes into knowledge areas.

Although almost any class can be mapped to one of the PMBOK knowledge areas, the PMBOK structure has been found do not always be adequate for the representation of organization-specific or PMS-related data structure patterns. Hence, additional sub-packages can be added to model specifics. For instance, according to the PMBOK, *issue management* belongs to *communication management*. However, based on the investigated software products, it has been found that the ITPM model is more comprehensible, if the enclosed issue management concepts are modeled in a separate package. By doing so, the representational adequacy of the model is improved, and the mappings between the issue management concepts (respectively tool domain concepts) become more traceable.

For each tool-connector, a separate model is created. These models are shareable between operators of the same PM software product. Because PM tools often have a fixed API, the identification and refinement of models is typically converging, and hence can be improved successively by various project managers, knowledge engineers, developers, or any other PM-related role.

6.4.3 Integration of ITPM Data based on Model-Driven Software Engineering

The ITPM core models represent abstract, virtual concepts. These concepts are derived from specific PM software tools, with the objective to provide the project manager with an abstracted, understandable view on relevant PM data. This view enables the project manager to:

- access unified, commonly understandable PM models and vocabulary. Irrelevant details of local PM systems are abstracted away.
- derive aggregated knowledge, such as reporting figures, or monitoring information (e.g., based on SPARQL queries or SWRL rules).
- synchronize / share data between local tools.
- create work-flows to automate manual synchronization scenarios.

The specific PM data instances reside within the local PMS, but are periodically synchronized (a) between tools of the same type (e.g., PM tracking software), (b) across heterogeneous tools, based on work-flows, (c) with the central PM data store.

To enable data exchange, mappings between the local and the common concepts must be implemented. The concepts of the core ITPM model (common concepts) must be designed thoughtfully. A trade-off between a very fine-granular model, which enables the accurate synchronization of concepts across local PM systems, but is expensive to implement, and a very coarse grained model, which lacks of specific data exchange, and possibly reporting features, but is cheap to implement, needs the taken.

The design of models, based on the CNIP, and the Ontology 101 engineering methodology, has already been described in section 6.2.2. This description is extended to pave the way for the implementation of PM-related models, and the integration of local PM data, based on the PMBOK knowledge areas, and the ITPM model repository.

Based on related work, and the studied use-case (chapter 5), the following modeling methodology is suggested.

1. The high-level PM data concepts in place need to be studied and identified (e.g., requirements, issues). The high-level concepts (classes) are refined step-wise, by adding attributes, and associations. For the modeling of the common concepts, PM literature, and organization-internal standards, should be used. For instance, for the common concepts of the ITPM dashboard prototype, "A Project Manager's Book of Forms. A Companion to the PMBOK Guide." was utilized. The PM book of forms is a great entry point for the implementation of a set of PMBOK-related data models. It contains a set of applicable PM forms that are directly transferable into UML class diagram notation (classes, attributes, associations). Hence, the application of industry standards, and the minimization of modeling overhead, is ensured [111].

2. The data concepts of the specific PM software sources are identified and modeled, each within a single connector model. The common model is compared with the local models, based on the Semantic Model Editor (dual-view feature).

3. All attributes and associations that are available in the local concept, but not present in the core ITPM model, are reviewed. Missing, but important concepts (attributes, associations) are added to the common model, if necessary.

4. At the end of the modeling process, the common and local models are homogenized. The mappings are either indicated by the names of the model concepts (classes, attributes, associations), or described via comments. For this purpose, the Semantic Model Editor supports the annotation of model elements at different levels of granularity.

5. A developer is charged to implement the designed models, based on the creation / extension of an OpenEngSb tool connector. The main tasks are:

    a) Generate the source code (beans, DAOs, tests), based on the Semantic Model Editor.

    b) Implement / supplement the mappings between the tool API, and the generated connector model (classes, attributes, associations, data values / enumerations). Depending on the tool API's data format, parts of the mappings can be automatized (e.g., JSON/XML: automated mapping between JSON and Java is possible). Based on the project manager's requirements, the mapping needs to be either bidirectional, or unidirectional (the latter implies lower implementation effort).

    c) Implement the mapping between the connector model and the PMBOK knowledge areas, respectively OpenEngSb tool domains.

d) Test the implementation and deploy the changes (ITPM dashboard update).
e) Refine the provided models on demand, and repeat the steps a - d.

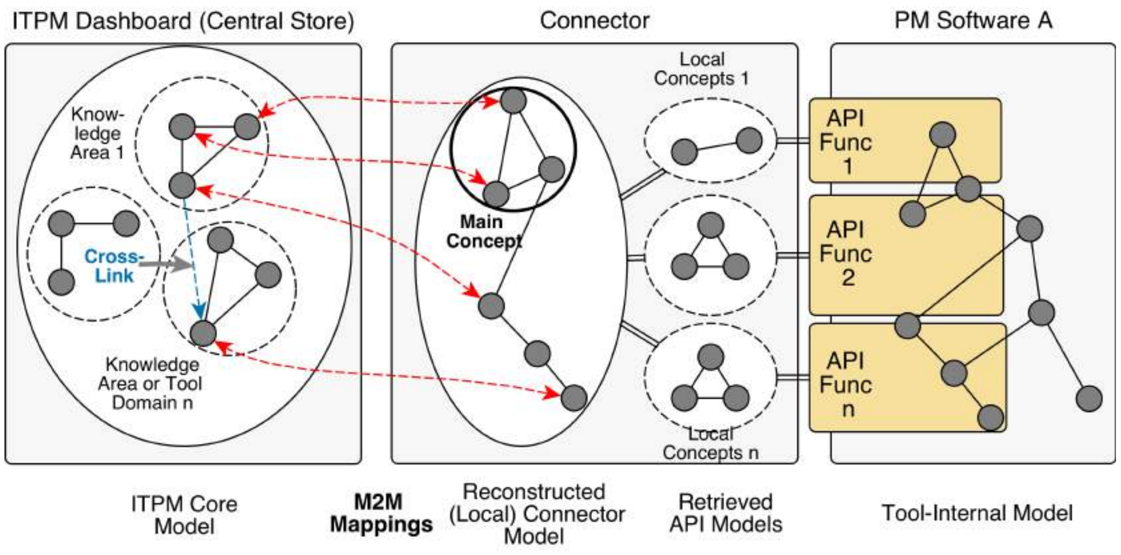


Figure 6.6: Mapping between PMS Models and the ITPM Model (Model-Based View).

Figure 6.6 visualizes the mapping between (local) software tools and the ITPM core model (common concepts), and takes a model-based perspective. The approached integration solution can be interpreted as a set of Model to Model (M2M) transformations. To detail the effectiveness of the MDSE - based integration approach, the four identified model types in figure 6.6 are summarized, subsequently.

Tool-Internal Model The tool-internal model represents the data structure on which a specific PM software product (in the figure the product is called “PM Software A”), or a PM knowledge source, builds upon. This model is typically neither accessible, nor disclosed. For instance, Jira issue utilizes a relational data storage, but the exact data tables and interrelations are hidden. Laymen may reason about the internal data model, based on UI elements, API access, or other documentation artifacts.

Retrieved API Models Most tools provide APIs to retrieve and/or modify (parts of) the internal data model. Some APIs are very comprehensive, other provide only basic read access. Some products provide a single API (e.g., REST), others provide complementary APIs of various technologies (e.g., Jira: management of project core data and users via SOAP; access to issues via REST API). To access specific API models, various function calls with different parameter values must be performed. Each distinct call results in a subset of the totally accessible data model. Some calls may provide overlapping results, others may lack of eligible data. Based on the API documentation, and experimenting with the API, the technician will reconstruct the tool-internal data model.

Reconstructed Connector Model (Local Concepts) The physical implementation of the reconstructed tool-internal model, although desirable (at least partly), is unreasonable for most tools: The implementation costs outweigh the improve in quality (maintainability, stability). Hence, the developer typically just needs to ensure that the concepts of the "reconstructed connector" are appropriately mapped with the retrieved API models, to ensure that the requirements for the mapping with the ITPM core model (common concepts) are satisfied.

However, there are also several cases in which the employment of an intermediate, tool-internal-like model is quite simple. If the API provides access to well-structured models, e.g., in XML or JSON format, such as REST APIs do, then based on the data structure, these models become automatically processable by JSON/XML to Java converters (e.g. FlexJson[12]). The requirement for the JSON/XML to Java conversion is the availability of appropriate Java beans. These beans are automatically generated via the MDSE approach. Once the Java artifacts are in place, the local concepts, based on the tool API can be transformed directly into the reconstructed connector model, including additional validation steps, based on the derived connector model code. Thus, the transformation becomes essentially a model-to-model (M2M) approach, in which the local connector model concepts are mapped to the concepts of the ITPM core model, and vice versa (in the prototype: Java to Java mapping).

ITPM Core Model The data instances of the ITPM Core model are those, accessible via the UI of the ITPM dashboard. The mappings between the local concepts of a connector, and the ITPM core model focus on a specific knowledge area, respectively sub-package. For instance, Jira issues belong to the *issue management* package[13]. However, a tool connector typically accesses also other knowledge areas. For instance, issues in Jira may belong to a certain *component*. The attributes of the local concept *component* is mapped to the common concept *artefact* which relates to *scope management*.

A main benefit of the solution approach is the maintenance of cross links (CL) between knowledge areas within the ontology (cf. figure 6.6). CL are maintained as object properties in the central OWL data store. This is an essential difference to the OpenEngSb *tool-domain* concept. Based on tool domains, all concepts of a connector are mapped to exactly one knowledge area, respectively tool domain. Hence, if various concepts need to be mapped between the local and the central representation, the tool domain needs to include redundant information about related concepts of other tool domains. For instance, in Jira, an *issue* (main concept) would either need to contain all data about the *issue reporter* (additional attributes), or an attribute with a key, to identify the related *collaborator* within a different domain. The retrieval of aggregated data based on tool domains is less efficient, because (a) the OWL object property (CL) cannot be queried directly, (b) the keys between the two different tool domains must be stored redundantly, and (c) matched manually (more complex, inconsistent SPARQL queries).

[12]FlexJson: `http://flexjson.sourceforge.net/`

[13]Strictly speaking, issues belong to the PMBOK's communication management knowledge area. An additional UML package was added to keep the model understandable, though.

Based on the suggested M2M approach, the integration can be automated to a large part, and the stability, testability, and maintainability of the implementation are improved. Especially the stability of the synchronization process is improved, because the validation mechanisms detect errors earlier, and help to trace error sources more accurately. Note that the mapping of the entire local model is neither feasible, nor sensible. The main reasons are that (a) not for all local models an equivalent common model exists. (b) the implementation efforts to reach data completeness are very high. (c) the synchronization of the entire local data model results in bloated interfaces, and inefficient workload of technical resources (performance). Hence it is sufficient to identify a *main concept*, for which the mapping of required attributes and associations is implemented. Auxiliary concepts can be added on demand.

To provide an example, in Jira, the main concept is an *issue*. The mapping between the local issue concept and the central one has been implemented for the ITPM prototype very completely. Auxiliary concepts are *project*, *user*, or *component*. These concepts are required in order to maintain issues. Hence the mapping of the required sub-concepts has been implemented too, but the integration focuses on the relevant attributes and associations.

6.4.4 Semantic Model Editor UI-Support for OpenEngSb Integration

The discussion in section 6.4.3 shows that the implementation of a M2M approach, based on an intermediate connector model, should be aspired, if feasible. Besides code generation capabilities, the *Semantic Model Editor* provides additional features to smooth the implementation of new connectors, based on the tool API they bind to. Subsequently, the main features, based on the UI-based annotation of model elements, are summarized.

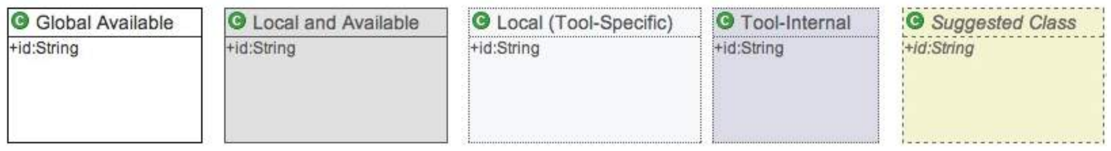


Figure 6.7: Annotations of Model Concepts for the Generation of Code

Annotation of Models As already outlined in section 6.3, the annotation types *connector*, and *tool domain* can be assigned to a model in order to control the location of the generated code. The annotated models are marked with a specific icon within the *Semantic Model Editor* repository.

Annotation of Packages, Classes, and Attributes For each package, class, and attribute a scope can be assigned. The related concepts are colorized within the editors UI (cf. 6.7). Additionally, and an internal annotation is added. The scope defines, how the code generator proceeds with the specific concepts, and provides background information for collaborative modeling:

Tool-Internal The modeled concept is part of a PMS / tool (cf. figure 6.6; tool-internal model, retrieved API models). The concept is either not accessible, or it is yet unclear whether, and how, the concept should be integrated. In addition, this annotation is used to inform about an inherent concept, which for any reason is not used within the tool connector implementation. No code is generated for tool-internal concepts.

Example: The Jenkins build server provides a REST API. However, because the interface is complex and not intuitive, it cannot be directly transformed into a connector. Hence, a model is created for the REST API (information purpose), but the mapping between the retrieved API models, and the local connector models is conducted manually.

Local (Tool-Specific) The code for the local connector concepts is generated, so that they can be leveraged to automatically transform API models into a (local) connector model. The utilization of the DAOs validation procedures is optional.

Example: By creating the connector models purposefully, the transformation between several models of Jira's REST API (JSON), and the generated Java beans are automatized. The generated Java-bean are automatically populated with data content from JSON. The generated DAO is used to verify that the integration is complete.

Local and Available Same as "local (tool-specific)", but the local connector model is persisted entirely in the central data store, based on the local DAOs. This state has not been used throughout the use-case evaluation. The intend is to maximize the data completeness of the integration for ITPM projects that need to integrate various instances of the same software product.

Example: A specific project requires the integration of several parallel Jira instances, administered by different, autonomous project organizations.

Suggested Class This state does not directly relate to the models, as described in section 6.4.3. The state is used to suggest concepts that are not yet accessible.

Example: The project manager requests a new attribute, which is not implemented yet. For instance, a "due date" for the common issue concept should be added.

Global Available The default state for common concepts. Means that either the concept is available for querying within the ITPM core model (the content can be visualized within the ITPM dashboard), or that the scope has not been defined explicitly (for connector models).

6.5 Concluding Remarks

This chapter introduced the SIFPM for the systematic integration of PM data at various organizational levels. The main (design-science) outcomes are

- the CNIP, for the negotiation and integration of PM integration at the management level,
- the ITPM Dashboard, for the central management of PM data visualization, and integration, the visualization and monitoring of PM

- the Semantic Model Editor, for the maintenance of models, and the derivation of additional code and data artifacts,
- the Semantic Model Repository for ITPM models, which addresses (a) sharing and reusing PM models across projects and managers, (b) the efficient integration of local PM data, based on a MDSE approach, and the OpenEngSb.

The solution outcome is generally applicable for complex IT projects with heterogeneous project partners, but is especially useful for IT project management in consortia. More precisely, the focus is on new consortia, with a limited lifespan, respectively project duration (e.g., 0.25 - 2 years).

The visualization and configuration capabilities for the project manager has not been addressed so far. The configuration of the dashboard, including the specific description of model-mappings, work-flows, and SPARQL queries, based will be elaborated in chapter 7.

Due to the limited space of the work, the entire capabilities of the dashboard (and its submodules) cannot be demonstrated. Additional screenshots of the UI are however provided in appendix B. Likewise, an overview of the architectural software layers (module view), technical integration processes, MDSE mappings, and reference models is provided in appendix A and appendix D.

CHAPTER 7

Results and Evaluation

The systematic PM integration framework (SIFPM) is applied to the consortium UC, (introduced in chapter 5), to demonstrate its usefulness. The results of the application are evaluated.

The evaluation focuses on the demonstration of the *net benefits* for the project manager, based on the configuration of the ITPM dashboard. In addition, the efficiency for the integration of new PMS connectors is evaluated from a technical perspective (comparison of development efforts).

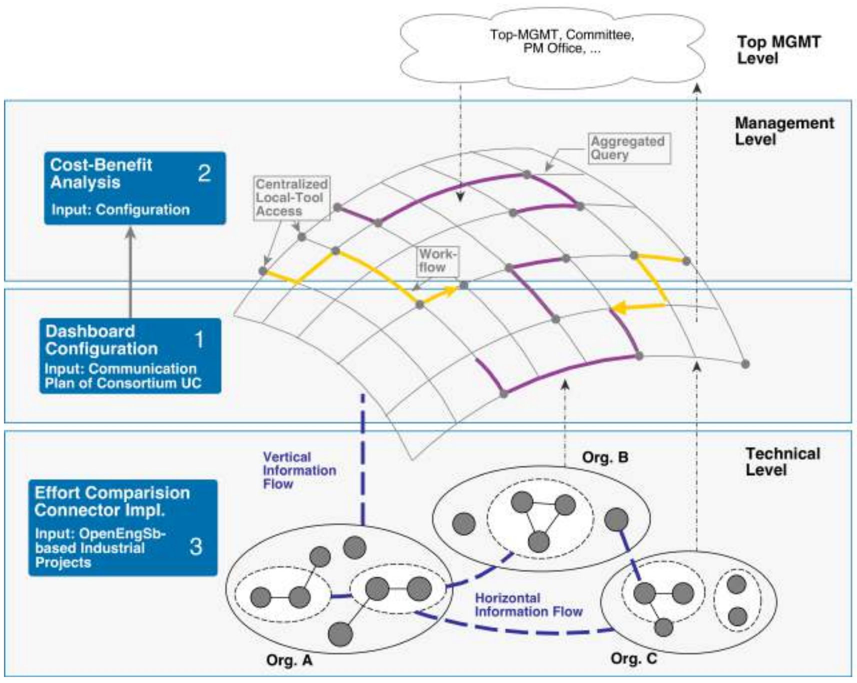

Top-MGMT, Committee, PM Office, ...
Top MGMT Level
Management Level
Cost-Benefit Analysis 2
Input: Configuration
Centralized Local-Tool Access
Aggregated Query
Work-flow
Dashboard Configuration 1
Input: Communication Plan of Consortium UC
Effort Comparision Connector Impl.
Input: OpenEngSb-based Industrial Projects 3
Vertical Information Flow
Org. B
Technical Level
Horizontal Information Flow
Org. A
Org. C

Figure 7.1: Overview on the evaluation steps and the addressed organizational levels.

Figure 7.1 summarizes the main steps of the evaluation and the involved organizational levels. Section 7.1 demonstrates the automation and visualization capabilities of the ITPM Dashboard. For that purpose, the communication plan of the consortium, introduced in chapter 5, is revised, and potential automation scenarios are identified and addressed. The fitness of the ITPM dashboard is demonstrated via its configuration for the consortium UC. The outcome of the dashboard configuration serves as the input for the *cost-benefit analysis* in chapter 7.2. Within the analysis, the installation and configuration costs of the ITPM dashboard are compared against the time-savings for the project manager. The assumption is that no technical implementation efforts are necessary. In other words, all PMS connectors are already available in the official OpenEngSb repository [1]. Section 7.3 evaluates the integration framework on the technical level. On that score, the implementation effort for the conventional OpenEngSb connector development process is compared against the proposed MDSE approach.

7.1 Dashboard Configuration

The configuration of the ITPM dashboard mainly involves (a) the adding of available/data sources to the synchronization cockpit, (b) the implementation of cross-domain work-flows, and (c) the construction of SPARQL queries to derive and visualize knowledge from integrated data sources (cf. figure 7.1).

The capabilities of the ITPM Dashboard are evaluated for the use-case, which has been presented in chapter 5. The following basic assumptions are made in order to make the evaluation of the dashboard configuration more compact:

1. The CNIP meeting[2] has already been performed by the managers of the consortium. A manager (respectively team leader), and a technician from each organization participated. Also, a functional manager (top-management) of organization A was present. In total, the meeting comprised eight participants.

2. The meeting participants discussed the main integration issues in a three-hour session. They followed the CNIP.

3. The project partners identified the most relevant PMS and tools of each organization. They added all relevant sources in the ITPM synchronization cockpit, including the access data for the tools' APIs. While the meeting, the the present technicians ensured that all APIs were accessible for the project manager in read and write mode, so that they are usable throughout the project.

4. Based the joint creation of a communication plan, the meeting participants identified seven problem areas with integration/automation potential (cf. figure 7.2). These seven areas imply the creation of work-flows and queries. The requirements for the technical integration of the work-flows and queries have been sketched, but the implementation of these artifacts has not been performed yet.

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

[2]Consortium Negotiation and Integration Meeting, cf. chapter 6.1

5. The meeting concluded as follows: The technicians confirmed that there is no need for the creation of new connectors, or the extension of existing connectors. The management agreed that the configuration expenses for the automation of the seven identified problem areas are justified, and potentially beneficial for further projects. The ITPM dashboard has been installed and configured on the project manager's laptop. In a next step, the project manager and a technician/knowledge engineer will meet to finalize the configuration, by implementing the missing work-flows and queries.

Subsequently, the configuration of the ITPM dashboard is evaluated, from the viewpoint of a project manager and a knowledge engineer, who follow-up the CNIP meeting in a joint session. The main input for the modeling session is the revised communication plan of the project consortium, and the pre-configured ITPM dashboard on the project manager's local computer. The revised communication plan is visualized in figure 7.2. The difference between the original version (figure 5.3) and the revised version are that the seven identified problem areas are highlighted in the latter. These problems are grouped into two groups: The first group addresses synchronization and integration scenarios on the horizontal level, i.e., the synchronization between different local representations, based on common concepts. The second group addresses the vertical integration of data, viz., the derivation of PM knowledge, based on the centrally integrated data, SPARQL queries, and ITPM dashboard visualizations.

7.1.1 Horizontal Integration and Synchronization

Synchronization of Requirements (Figure 7.2 - 1) The first scenario addresses the synchronization between the MS Excel requirements of the researchers, and the management requirements (Google Drive spreadsheets).

The mapping of the multi-model data sources entirely takes place by specifying the format and appearance of each column, based on the *data source configuration ontology*: no additional connector implementation effort results. The project manager and the knowledge engineer add the mappings for each of the two local requirement formats (cf. bottom left corner of figure 7.3), and the common concepts (top left corner). The configuration is performed via the automatically derived ITPM dashboard UI forms (1,2). The mappings are interpreted by the *MDD Connector* during runtime (cf. 6.4.2).

As figure 7.3 shows, the data formats of the spreadsheet files are similar, so they can be efficiently linked to the related common concepts of the PMBOK scope management knowledge area (cf. [111]):

1. Each spreadsheet row corresponds to a single requirement.
2. The primitive fields (e.g., *id*, *name*, *description*) are mapped to the corresponding attributes of the common *Requirement*.
3. The optional field *parentId* corresponds with the inverse association of *Requirement. subRequirement* (the association *subRequirement* of the common class *Requirement*).

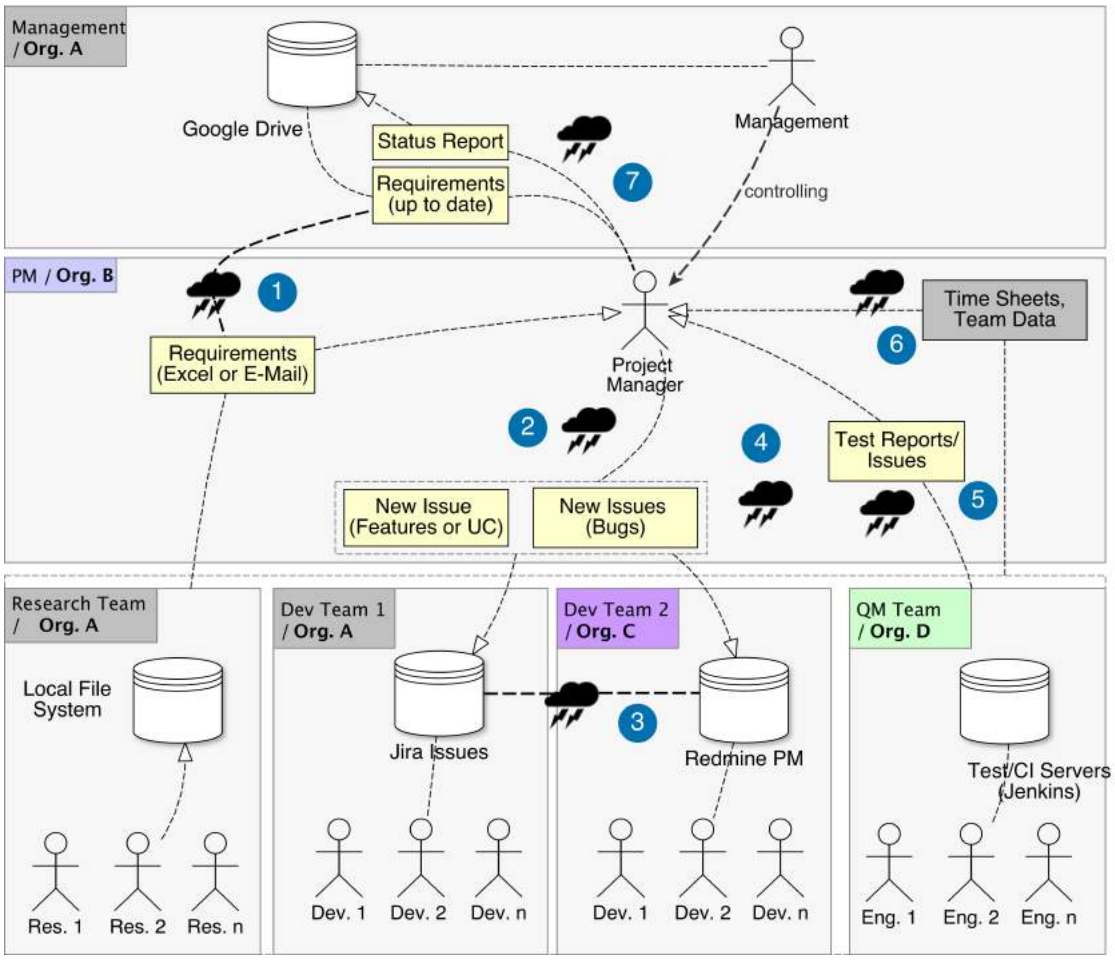


Figure 7.2: Communication plan with identified scenarios for optimization (scenarios 1-5: horizontal integration. scenarios 5-7: vertical integration (5 includes both).

4. The fields *interrelatedReqIds* contain multiple ids, separated via semicolon. For each of those id's, a self-referencing association *Requirement.depends_on* is created.

5. The field *artifactId* contains a cross-reference to *Artifact*. The id contains the name of the related work-package, which in the UC corresponds with the name of the software module (e.g., *itpm-dashboard*). For each new id, a new instance of *Artifact*, and a bidirectional reference (*produceArtifact*) to the related requirement, are created.

There are two main differences between the local requirement representations. First, the order of the fields in the local spreadsheets are different. For instance, the management is more concerned about the priorities and the stakeholders and hence wants to have these fields first. The order of the fields is already addressed during the mapping of the fields to the common concepts. The second difference is that researchers capture the internal acronyms for requirement

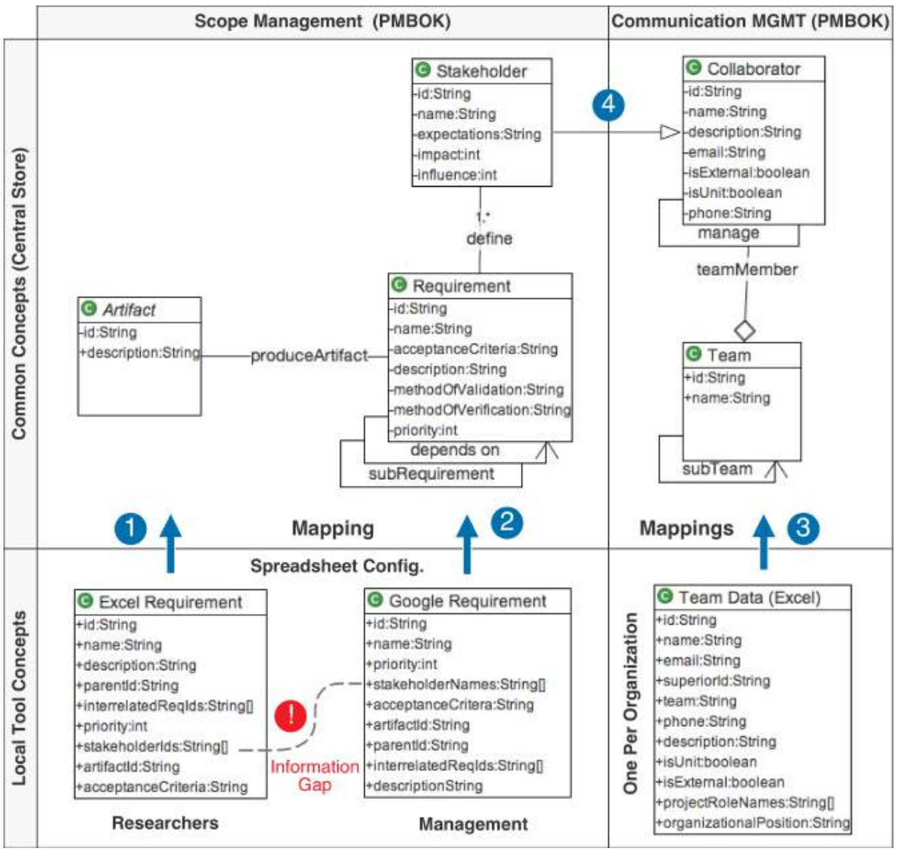


Figure 7.3: Mapping between the local requirement spreadsheet formats, and the common concepts (simplified version).

stakeholders (e.g., “agr” for “Andreas Grünwald”), while the management requires the complete stakeholder names (they do not understand the meaning of the acronyms) (cf. red exclamation mark in figure 7.3). This mismatch cannot be resolved without having additional information. Luckily, as indicated in the communication plan (figure 7.2, each consortium partner also provides an Excel-list with all project participants, including stakeholders, involved (consecutively referred to as *team data*). The team data includes hierarchical information and the internal acronyms for each contact, and can be integrated into the ITPM dashboard (3). The structure of the local team data format is summarized in the right-bottom corner of figure 7.3. The relation between stakeholders and collaborators is indicated in figure 7.3 (4). To resolve the mismatch within the data source configuration, the knowledge engineer (or the manager) maps the field *stakeholderIds* to *Stakeholder.id* (top-left corner), and marks that the field can contain multiple entries. Likewise, the field *stakeholderNames* is mapped to *Stakeholder.name*. Based on the collaborator records, which are loaded into the central semantic ITPM store, the id’s are associated with the existing stakeholders.

As soon as the configuration records are activated in the *synchronizer cockpit*, the local spreadsheet sources are polled, frequently. A spreadsheet is up-streamed/down-streamed as soon as the modification date of the file exceeds the date of the last poll request.

Derivation of Issues from Requirements (Figure 7.2 - 2) The second scenario addresses the high manual effort for the creation of issues (features) in the local software tracking systems of the two development teams (organization A and C). The project manager and the knowledge engineer elaborate three work-flows, which are visualized subsequently. The current prototype supports the runtime configuration of work-flows based on Drools.

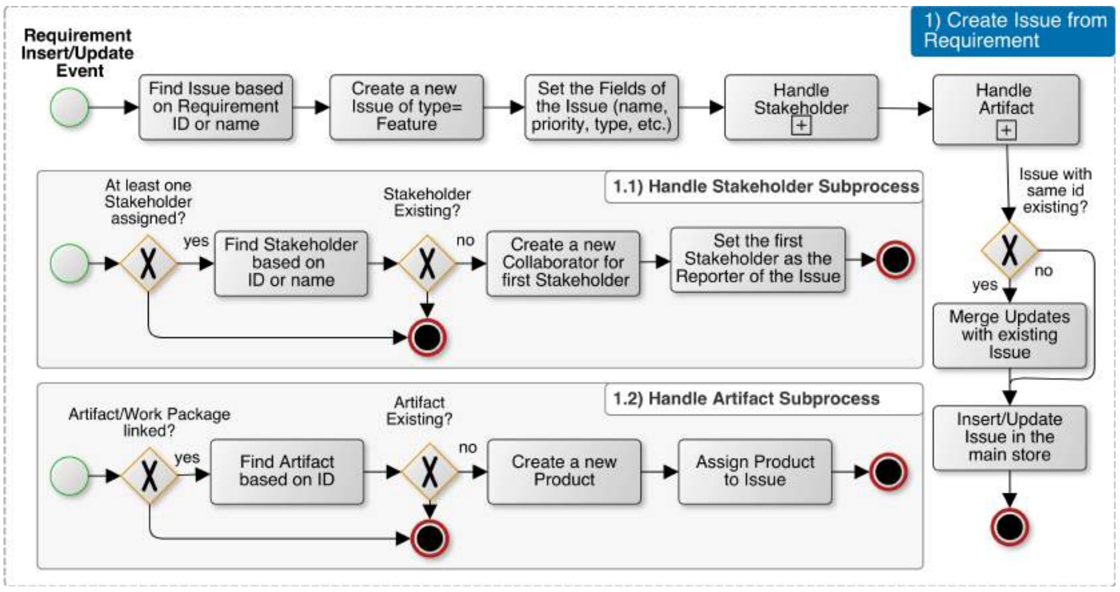

1) Create Issue from Requirement
Requirement Insert/Update Event
Find Issue based on Requirement ID or name
Create a new Issue of type= Feature
Set the Fields of the Issue (name, priority, type, etc.)
Handle Stakeholder
Handle Artifact
1.1) Handle Stakeholder Subprocess
At least one Stakeholder assigned?
yes
Find Stakeholder based on ID or name
Stakeholder Existing?
no
Create a new Collaborator for first Stakeholder
Set the first Stakeholder as the Reporter of the Issue
Issue with same id existing?
yes
no
Merge Updates with existing Issue
1.2) Handle Artifact Subprocess
Artifact/Work Package linked?
yes
Find Artifact based on ID
Artifact Existing?
no
Create a new Product
Assign Product to Issue
Insert/Update Issue in the main store

Figure 7.4: Work-flow for the automated creation/update of issues based on requirements.

The first work-flow (figure 7.4) is triggered by the OpenEngSb work-flow engine, whenever a requirement is inserted or updated. For each new requirement a corresponding issue is created in the central ITPM data store. If a correlated issue already exists, then the relevant charges are merged. The name, priority, and the description of the requirement are reused for the creation/modification of related issues. The work-flow also supports the automated creation and assignments of stakeholders and artifacts. The first stakeholder is assigned as the *reporter* of an issue. For each provided artifact (id), a new *product* is created in the central store. A *product* equals a deliverable, with a unique name (the id), and an optional description and version number.

Whenever a requirement is deleted, the corresponding issue must be removed as well (figure 7.5). As it is unlikely that requirements are removed after the requirements-specification phase, there is no need to verify the state of the issue before removal.

The third work-flow addresses the back-propagation of issue changes to the corresponding requirement (figure 7.6). It is reasonable to synchronize the priority and the description of the

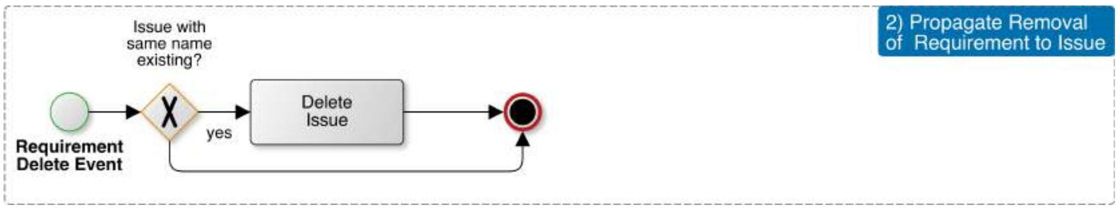

Issue with same name existing?
yes
Delete Issue
Requirement Delete Event
2) Propagate Removal of Requirement to Issue

Figure 7.5: Work-flow for the propagation of requirement deletions.

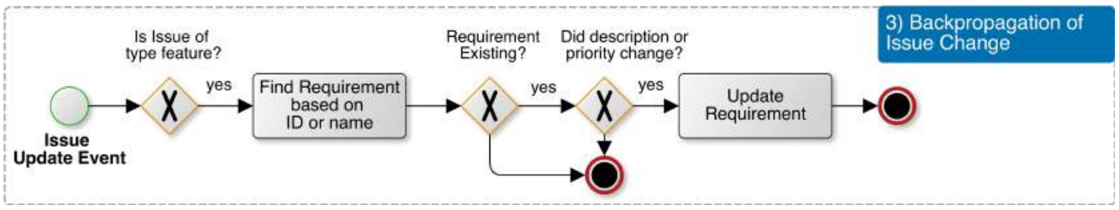

Is Issue of type feature?
yes
Find Requirement based on ID or name
Requirement Existing?
yes
Did description or priority change?
yes
Update Requirement
Issue Update Event
3) Backpropagation of Issue Change

Figure 7.6: Work-flow for the automated creation/update of issues based on requirements.

original requirement, to keep the artifacts inline. In addition, the work-flow enables the project manager, to edit requirements, based on the UI of a more powerful PMS (e.g., editing in Jira or Redmine).

The execution of the work-flows takes place on the level of the common concepts. The project manager does not need to care, how many and which specific PM tracking systems are in place. The implemented synchronization mechanism takes care of the distribution into the local PMS. As the design of the work-flows indicates, not only *issues* are synchronized, but also related concepts, such as artifacts, and stakeholders/collaborators. The synchronization mechanism even configures default projects and user permissions, on-the-fly.

Synchronization between PM Trackers / Project Configuration (Figure 7.2 - 3) The third identified optimization scenario addresses to the synchronization between Jira Issue and Redmine. There are two extreme cases, on how to manage the simultaneous development of software artifacts for two distributed teams: (1) The two development teams are managed completely isolated. This approach works as long as there are no overlapping features, and no integration takes place. (2) The two teams are managed as a single, virtual team. The advantage of this approach is that idle states and imbalanced workloads are reduced. In addition, the features are developed much more integrated, and the coordination effort of the project manager is reduced significantly. However, the *virtual team approach* requires an accurate synchronization of the development teams' state, i.e., a single "virtual" PM platform.

By automating the synchronization between Jira and Redmine a single, virtual platform can be simulated. Because sophisticated connectors for Jira and Redmine are available, no additional effort results for the project manager: As soon as the access data is configured within the

ITPM dashboard, the synchronization takes place, either continuously, or manually controlled, from time to time (e.g., every day at 6 pm).

Besides the automated exchange of central tracking concepts (issues, artifacts, comments), the provided issue connectors relief the project manager from the tedious configuration of the local projects. In particular, the synchronization mechanism automatically creates a default project during the first synchronization, assigns the project manager as the administrator, and copies user data on demand.

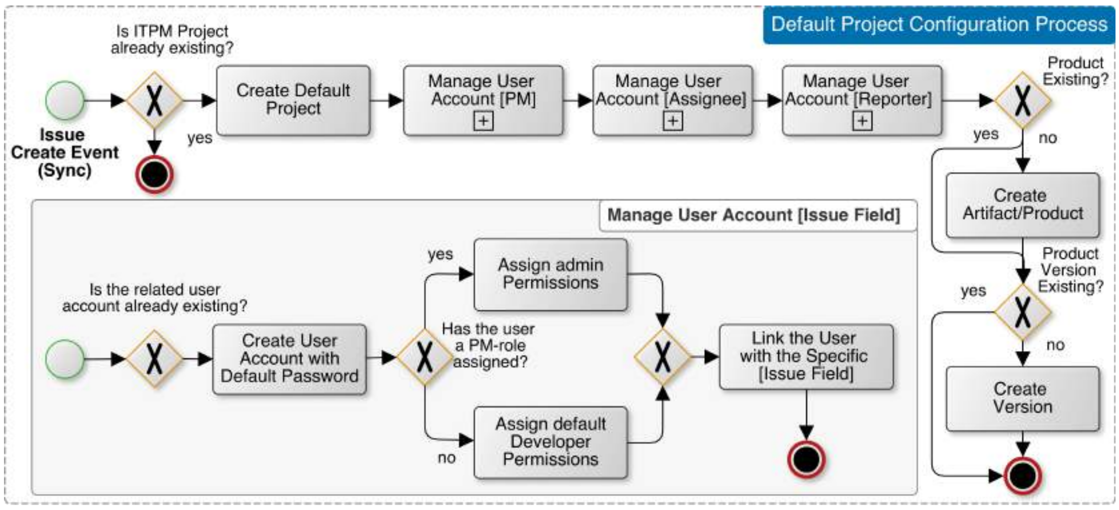

Default Project Configuration Process
Is ITPM Project already existing?
Issue Create Event (Sync)
yes
Create Default Project
Manage User Account [PM]
Manage User Account [Assignee]
Manage User Account [Reporter]
Product Existing?
yes
no
Create Artifact/Product
Product Version Existing?
yes
no
Create Version
Manage User Account [Issue Field]
Is the related user account already existing?
Create User Account with Default Password
Has the user a PM-role assigned?
yes
Assign admin Permissions
no
Assign default Developer Permissions
Link the User with the Specific [Issue Field]

Figure 7.7: The Default Configuration Process is triggered during the downstream of issues (part of the solution approach; no manual implementation effort necessary).

Figure 7.7 summarizes the intrinsic project configuration work-flow of the synchronization mechanism.

Figure 7.8 summarizes the mappings between the local and the common concepts of the available tool connectors (simplified version). The encircled numbers (1-4) indicate the mapping between classes. The numbers 5-7 indicate the resolution of mappings on the attribute level.

1. IssueType: All three views (the local view and the common view) contain the types *bug* and *feature*, so the mapping is straight-forward. Jira provides additional issue types for agile PM (e.g., *user story* or *epic*). All views support the creation of additional issue types.

2. Priority: The priorities of Jira are mapped to a scale between one and five (1: blocker; 5: trivial). Redmine knows only four priorities, so “Prio4” and “Prio5” are both mapped to the Redmine priority “Low”.

3. State: The common view maintains a simplified version of the issue state, which is easier to query, and sufficient for the project manager: An issue that is neither active nor completed corresponds to the value “New” in Redmine (the related class is called *Tracker*

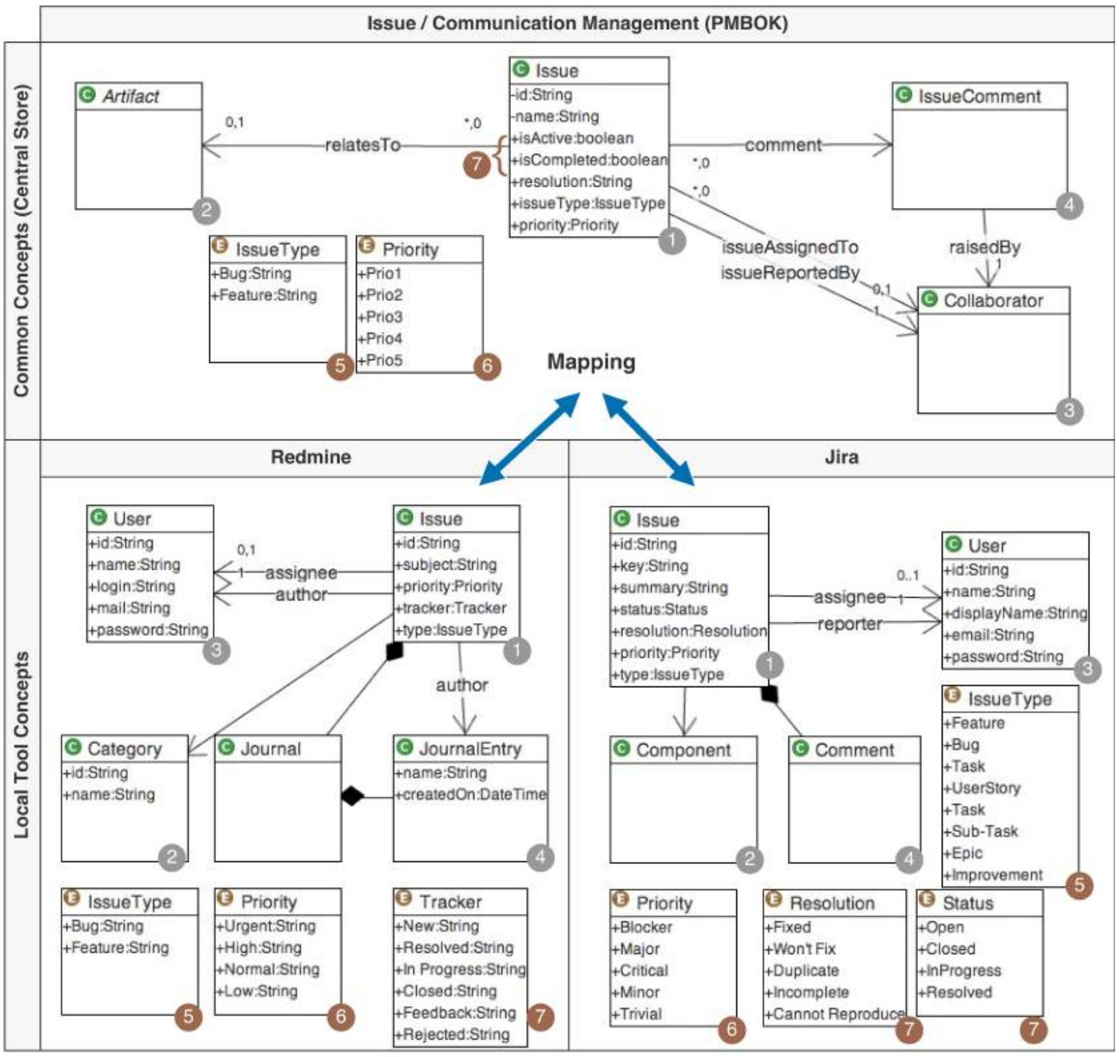


Figure 7.8: Mapping between the local and common PMS concepts (simplified version).

in Redmine). The status "active, but not completed" corresponds with Redmine's "In Progress". If an issue is completed, but not active any more, then it is "Closed".

The mapping between the common concepts and Jira is more complicated, because Jira maintains the status in two fields, viz., "Resolution", and "Status". The status of a Jira issue is updated by applying a "transition". Transitions base on simple work-flows and describe, which successor states are reachable from a current state. For instance, if an issue is in progress, it cannot be set to "open" anymore. The field "resolution" can only be set if a issue is closed or resolved.

The state "active, but not-completed" of the common concept is mapped to the Jira state "In Progress'. The state "not active, not completed" is mapped to "Open". "Active, not completed" is mapped to "resolved", and "not active, completed" is mapped to "Resolved". The resolution is set to "Fixed" on default.

Reporting of Bugs by the QM (Figure 7.2 - 4) The fourth inefficiency appears because the project manager must synchronize (create and assign) the identified bugs of the QM team (organization D) with the local PMS of the development teams.

The QM team receives the basic test instructions from the project manager. They can access the central source code repository (e.g., Bitbucket[3]) to monitor code updates, and to add unit and integration tests[4]. However, the QM has not access to the two different PMS of the organizations A and C, and they are not confident, to which team a identified bug should be assigned. As a result, the bug descriptions are forwarded to the project manager, who adds the issues (of type "bug") to one of the local issue trackers. The team leader can then assign the bug to a developer, or a available programmer self-assigns the bug.

The manual back-propagation of status updates to the QM team is tedious and inefficient as well. First, the project manager must frequently screen the PMS, and must forward forward bug updates to the QM. It is probable that the manager overlooks already resolved bugs, which can result in idle times and delays of the QM team. Second, some bugs can be blockers, i.e., the QM requests their resolution immediately. In that case it is likely that the QM managers poll the project manager oftentimes, because they want to know if the specific bug already has been resolved.

The central issue for the automation of scenario 4 has already been solved by entirely synchronizing the local PMS (figure 7.2 - 2). Because the project state is tracked in one virtual system, bugs can be added to any of the two tracking systems, and are automatically distributed to the responsible team/developer. In order to add and track bugs directly, the QM team needs access to one of the issue trackers. Fortunately organization C grants the QM team access to their Redmine instance. The administrator of organization C adds the user accounts manually, and the project manager provides the QM team members with the access data.

Because organization C cannot tell whether their project team is complete, or if there will join one or two additional QM engineers on short-notice, the project manager decides to create a simple work-flow for the creation of user accounts and default passwords, based on the team data of organization D (already integrated in scenario 1). The related work-flow is visualized in figure 7.9.

Note that figure 7.9 refers to the sub-process "Manage User Account", which is part of the default project configuration process (figure 7.7). The creation of a dummy issue is required, because the synchronization mechanism of the ITPM dashboard only down-streams instances,

[3]Bitbucket: `http://www.bitbucket.com`

[4]The repository is not included in the communication plan (figure 7.2), because it contains technical artifacts rather than directly PM-related data.

which are linked to the main concept of a connector (the main concept of PM/issue trackers is “issue”).

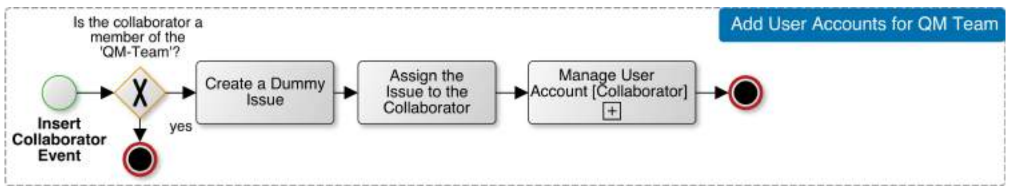

Add User Accounts for QM Team
Is the collaborator a member of the 'QM-Team'?
Insert Collaborator Event
yes
Create a Dummy Issue
Assign the Issue to the Collaborator
Manage User Account [Collaborator]

Figure 7.9: Work-flow for the creation of user accounts for the QM team (based on team data).

As a side-effect, the automated derivation of issues based on requirements (figure 7.6) improves the traceability between requirements, issues, software artifacts, and test results, which is beneficial for the QM team.

Automated Derivation of Test Reports (Figure 7.2 - 5) The implementation of automated test reporting requires two steps. First, the relevant QM data (test results, product quality metrics) must be extracted from the Jenkins build server API, and translated into concepts that are interpretable by the project manager. Second, based on the integrated data, queries and dashboard elements for the aggregated visualization of QM data can be created.

The mappings between the local and common QM concepts are summarized in figure 7.10. As the bottom half of figure 7.10 shows, the concepts of Jenkins are technically-oriented, and at a first glance, to not show a direct relation with the software artifacts and tests of the common model. Before describing the mapping in detail, a brief overview on the structure of the local and common concepts, is provided:

1. Local Jenkins Structure: For each main artifact in the source code repository, a Jenkins *build job* must be created by the QM team. The quality engineer links the job with the source code repository, and schedules the build process (e.g., execute the job whenever a new update appears in Git; execute the job every 2 hours). Once a job is started (either manually, or automatically), then the source code of the *main module*, and all *sub-modules* are compiled, and all linked tests are executed (e.g., based on a build automation tool such as Maven[5]). Each build results in a number of *test reports*, on the *test suite* level, and on the *test-case* level. Each report includes the number of succeeded, failed, and ignored tests. For each *sub-module*, a test report is created as well.

2. Common Concepts Structure: The common concepts capture the class structure more vividly. The design relates to the approach of the OSLC framework, and the structure of the JUnit test framework [6]. The main entry point is the *test Scenario*, which can be decomposed into *suites* and *test cases*. A test scenario can relate to a *unit test* scenario, and *integration test* scenario, etc. Opposed to the structure of Jenkins, the test reports,

[5]Maven: `http://maven.apache.org`
[6]JUnit: `http://junit.org`

i.e., test results, are available on all three levels. Thus, accumulated test results on the higher*test-scenario* level are available for the project manager.

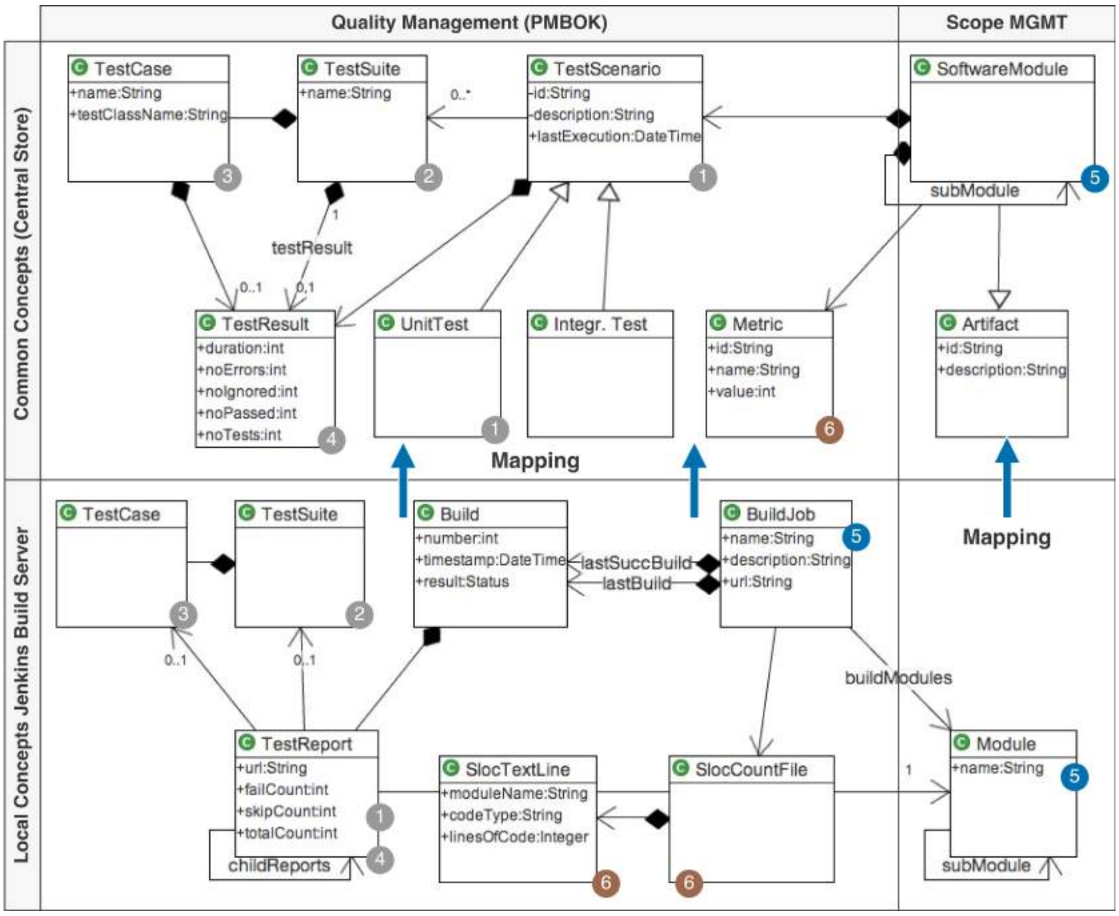


Figure 7.10: Mapping between the Concepts of the Jenkins Build Server and the Common Concepts of the QM and Scope MGMT areas (simplified version).

The mappings between the test outcomes (figure 7.10 - 1,2,3,4) of the local and the common levels are straightforward. The only difference is that the test results for the test-scenario level must be derived during the upstream procedure of the ITPM dashboard synchronization mechanism. More precisely, the test results of the test suites must be accumulated. A critical issue is, how the test results of Jenkins can be linked to meaningful artifacts that are known by the central ITPM data store. The answer is, that the mapping is based on the *artifact id*, which is linked within the requirement specifications. More precisely, the name of the build job must contain the artifact id in some way, to establish the connection between *artifacts* and software *modules*. Analogously, *sub-modules* can be linked to artifacts, if their name (typically derived from the package namespace in Java, or from Maven metadata) contains the artifact id. If more than a single artifact match, then the more accurate matching is selected, respectively a match based on a *build job* is preferred over a match on the level of the sub-module (figure 7.10 - 5).

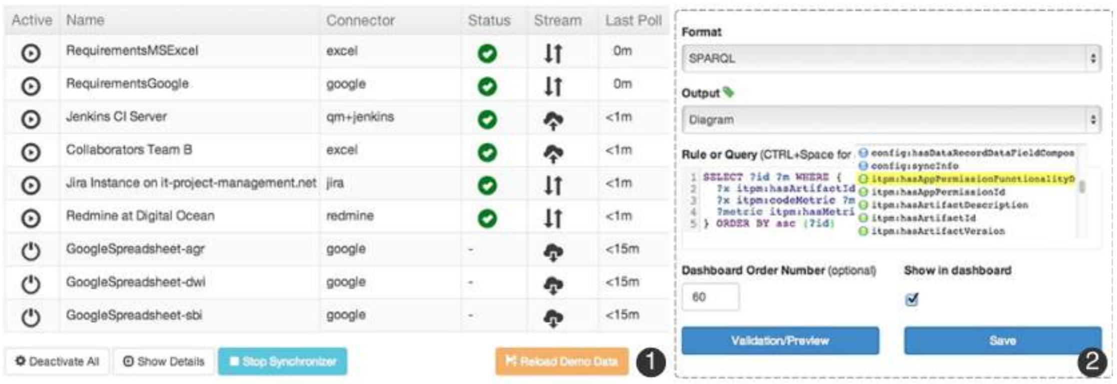
The classes *SlocCountFile*, *SlocTextLine*, and *Metric* (figure 7.10 - 6) have been left out so far. *SLOCCount* is a Jenkins plugins that summarizes the total lines of code for specific sub-modules and packages[7]. For each build job a sequential *SlocCountFile* is created. The file consists of text lines (*SlocTextLine*) that contain the module name, the type of analyzed code (e.g., Java or XML), and the associated lines (separated by tab stops). The file information is mapped to the common *SoftwareModule* concept. A *SoftwareModule* can have various metrics. The name of the metric specifies its type (in this case the name is *LOC*).

For the UC, the assumption is that the QM connector is already available off-the-shelf. Thus, the project manager and the knowledge engineer only need to check, if the access data for the Jenkins API has been correctly added to the synchronizer while the management meeting, and then can start immediately to stream the QM results.

Note that the connector only provides one-way synchronization (upstream), because from a project manager's point of view, there are no QM artifacts to downstream into Jenkins. Part 2 of the UC-based QM data integration is continued in section 7.1.2.

7.1.2 Vertical Data Integration and Visualization

The second part of the dashboard configuration addresses the derivation of project insights, based on SPARQL queries and predefined, customizable ITPM dashboard visualizations. As all distributed PMS and tools have been configured in section 7.1.1, the data is readily available in the central ITPM data store, and can be queried.



Active	Name	Connector	Status	Stream	Last Poll
▶	RequirementsMSExcel	excel	✓	⇅	0m
▶	RequirementsGoogle	google	✓	⇅	0m
▶	Jenkins CI Server	qm+jenkins	✓	↑	<1m
▶	Collaborators Team B	excel	✓	↑	<1m
▶	Jira Instance on it-project-management.net	jira	✓	⇅	<1m
▶	Redmine at Digital Ocean	redmine	✓	⇅	<1m
⏻	GoogleSpreadsheet-agr	google	-	↓	<15m
⏻	GoogleSpreadsheet-dwi	google	-	↓	<15m
⏻	GoogleSpreadsheet-sbi	google	-	↓	<15m

Format: SPARQL

Output: Diagram

Rule or Query (CTRL+Space for

```
SELECT ?id ?m WHERE {
  ?x itpm:hasArtifactId
  ?x itpm:codeMetric ?m
  ?metric itpm:hasMetri
} ORDER BY asc (?id)
```

```
config:hasDataRecordDataFieldCompos
config:syncInfo
itpm:hasAppPermissionFunctionalityD
itpm:hasAppPermissionId
itpm:hasArtifactDescription
itpm:hasArtifactId
itpm:hasArtifactVersion
```

Dashboard Order Number (optional): 60

Show in dashboard: ☑

Figure 7.11: The configured *Synchronization Cockpit* (1), and the *Semantic Query Editor* (2).

Figure 7.11 - 1 shows the *Synchronization Cockpit* that enables the project manager to monitor and control the ongoing data synchronization process. The configured cockpit contains

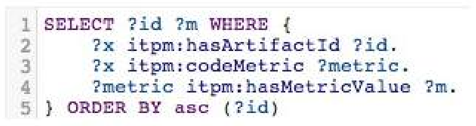
[7]SLOCCount: `https://wiki.jenkins-ci.org/display/JENKINS/SLOCCount+Plugin`

representatives for all data sources, discussed in section 7.1.1 (the last three rows link to collaborator timesheet's and are explained throughout the remainder of this section).

Figure 7.11 - 2 visualizes the main elements of the *Semantic Query Editor*, which is used for the design and visualization of queries within the ITPM dashboard. The project manager, with the help of a knowledge engineer, can utilize the *Semantic Model Editor* to look for the available data models, and leverage the *Query Editor* to auto-complete the query. The knowledge engineer utilizes the query editor, to validate and preview the designed query, and to specify the format (table, diagram, nested graph, KPI), and the location of the output within the ITPM dashboard.

Subsequently, a subset of the overall developed SPARQL queries, which provide a mid-ground of (a) presentability, and (b) usefulness, are evaluated in detail, based on the left scenarios (5, 6, 7) of the communication plan (figure 7.2).

Automated Derivation of Test Reports (Figure 7.2 - 5, part 2) As the local Jenkins installation of organization D is not accessible for the project manager, the QM team-leader must periodically report the main test results and metrics to the project manager (e.g., once a week). The creation of the test reports is a tedious manual activity: A QM engineer must collect the scattered information on the Jenkins server, and aggregate the data into diagrams, based on Excel, or Power Point. The project manager needs to distribute relevant results and states, to the top-management, and the local teams. Time pressure results, if the QM reports are tardily delivered. In addition, rework becomes necessary, if the reports are too detailed, lack of relevant information, or are not appropriately formatted.
To improve the reporting of QM data, the project manager and the knowledge engineer jointly implement a set of SPARQL queries for the integration of basic QM data and software metrics. Subsequently, two representative SPARQL queries are exemplified.



```
SELECT ?id ?m WHERE {
    ?x itpm:hasArtifactId ?id.
    ?x itpm:codeMetric ?metric.
    ?metric itpm:hasMetricValue ?m.
} ORDER BY asc (?id)
```

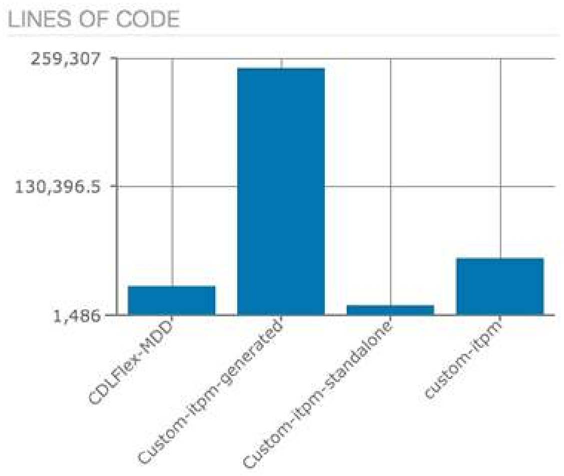
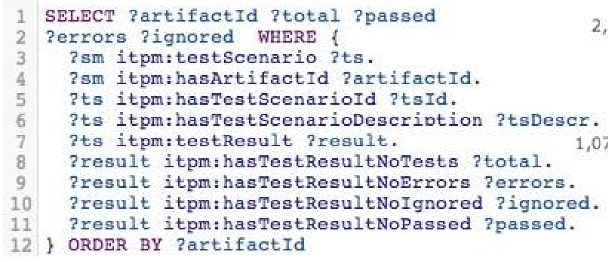


Figure 7.12: SPARQL query and ITPM dashboard visualization for the representation of the Lines of Code (LOC) per main software module.

Query 1: Lines of Code per Software Module The first query configuration renders the *total LOC per main software artifact* in a one-dimensional bar-chart (figure 7.12). The ITPM dashboard derives the appropriate chart format, based on the the available columns (`id`, `im`), and their data formats (text, numeric). The SPARQL query consists of three triples, which establish the link between class concepts and data values, via properties. In row 2, all entities within an existing `artifactId` (data property) are selected. As a result, `?x` provides a reference to instances of the type `Artifact`. Row 3 restricts the solution set to those instances that own a `codeMetric` object property, i.e., they link to an instance of the type `Metric`. Row 4 references the data property `metricValue` so that it is available in the solution set. Finally, row 5 sorts the result in ascending order of the id.

Query 2: Test Results Grouped by Software Module and Test Outcome The second query aggregates the *test results* in a two-dimensional bar-chart (figure 7.13). The number of passed, failed, and ignored test runs is derived on the scenario-level, and linked with the core software modules of the project. The precise number of test failures can be obtained by (a) hovering the bar charts of the diagram within the ITPM dashboard, or by (b) changing the output type of the query to "table".



```
SELECT ?artifactId ?total ?passed
?errors ?ignored  WHERE {
  ?sm itpm:testScenario ?ts.
  ?sm itpm:hasArtifactId ?artifactId.
  ?ts itpm:hasTestScenarioId ?tsId.
  ?ts itpm:hasTestScenarioDescription ?tsDescr.
  ?ts itpm:testResult ?result.
  ?result itpm:hasTestResultNoTests ?total.
  ?result itpm:hasTestResultNoErrors ?errors.
  ?result itpm:hasTestResultNoIgnored ?ignored.
  ?result itpm:hasTestResultNoPassed ?passed.
} ORDER BY ?artifactId
```

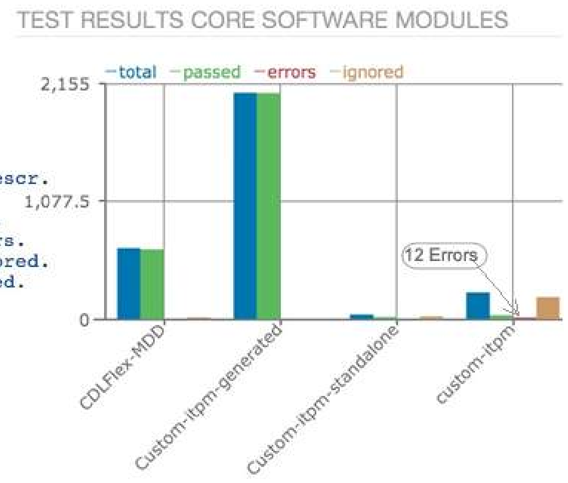


Figure 7.13: Visualization of test results, grouped by software module name and result type.

LOC are a popular and efficient measure to perform and communicate basic inferences about a project's size, its complexity, progress, and performance. Mc Connell provides several methods to derive accurate estimations for upcoming project activities (e.g., the efforts for construction or testing), based on LOC, respectively Function Points (FP)[8]. In addition, FP and LOC can be utilized for the estimation of defects. For instance, the implementation of a piece of code results in an average rate of 1.75 defects per FP (cf. [83, p.242]).

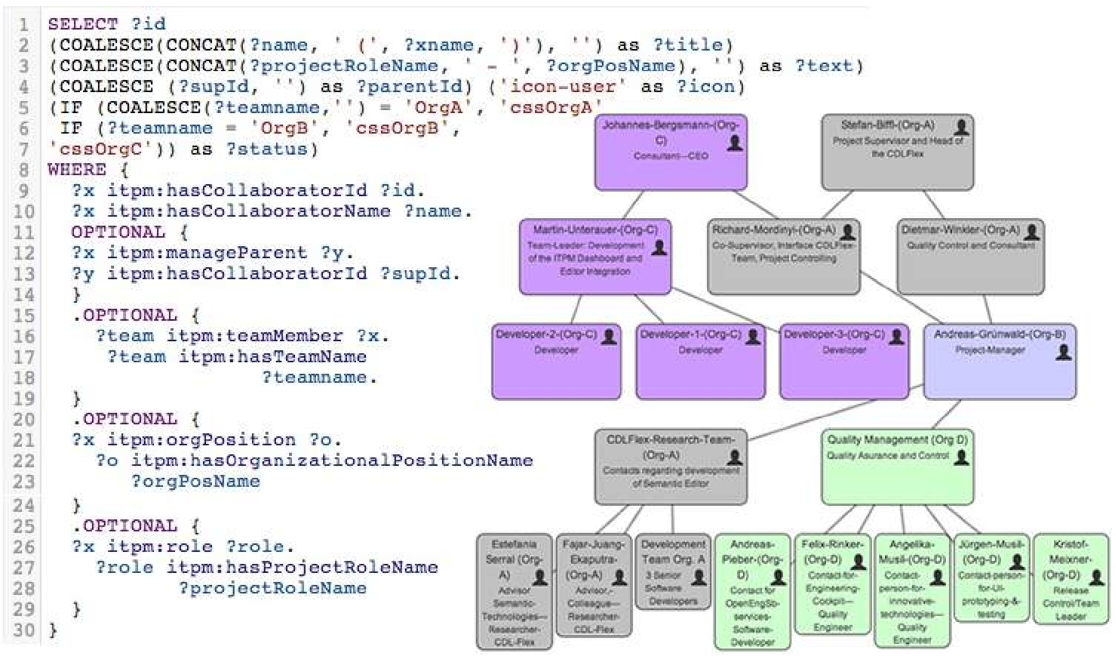
The visualization of test results provide quality insights regarding the stability of the software,

[8]LOC can be transformed into FP via conversion tables; e.g. one FP equals 55 Java LOC [83, p.202]

and its high-level test coverage. For instance, the diagram in figure 7.13 indicates that the model-driven modules (CDLFlex-MDD, Custom-itpm-generated) of the project are tested very intensively, while the stability of the UI (Custom-itpm-standalone), and the OpenEngSb integration, including the synchronization mechanism, and all connectors (custom-itpm), still can be improved.

Integration of Time Efforts (Figure 7.2 - 6) The monitoring of time exposures, and the efficient allocation of human resources is a crucial management task. A wide-spread method track efforts is the utilization of personal spreadsheets. These spreadsheets are forwarded to the team leaders, or the project manager, which review, accumulate, and analyze the efficiency of the resource allocation. Because the collection is a time-consuming, repetitive task, the project manager decides to automate the collection and analysis of the collaborators' time efforts.

Query 3: Visualization of Team Hierarchy The manager utilizes the existing team data spreadsheets and the ITPM dashboard, to derive a graphical representation of the team hierarchy. The hierarchy allows the manager to visibly allocate collaborators to teams and organizations, and to systematically integrate the personal time-sheets of the team members.



```
 1 SELECT ?id
 2 (COALESCE(CONCAT(?name, ' (', ?xname, ')'), '') as ?title)
 3 (COALESCE(CONCAT(?projectRoleName, ' - ', ?orgPosName), '') as ?text)
 4 (COALESCE (?supId, '') as ?parentId) ('icon-user' as ?icon)
 5 (IF (COALESCE(?teamname,'') = 'OrgA', 'cssOrgA'
 6  IF (?teamname = 'OrgB', 'cssOrgB',
 7 'cssOrgC')) as ?status)
 8 WHERE {
 9   ?x itpm:hasCollaboratorId ?id.
10   ?x itpm:hasCollaboratorName ?name.
11   OPTIONAL {
12   ?x itpm:manageParent ?y.
13   ?y itpm:hasCollaboratorId ?supId.
14   }
15   .OPTIONAL {
16     ?team itpm:teamMember ?x.
17      ?team itpm:hasTeamName
18                  ?teamname.
19   }
20   .OPTIONAL {
21   ?x itpm:orgPosition ?o.
22     ?o itpm:hasOrganizationalPositionName
23        ?orgPosName
24   }
25   .OPTIONAL {
26   ?x itpm:role ?role.
27     ?role itpm:hasProjectRoleName
28            ?projectRoleName
29   }
30 }
```

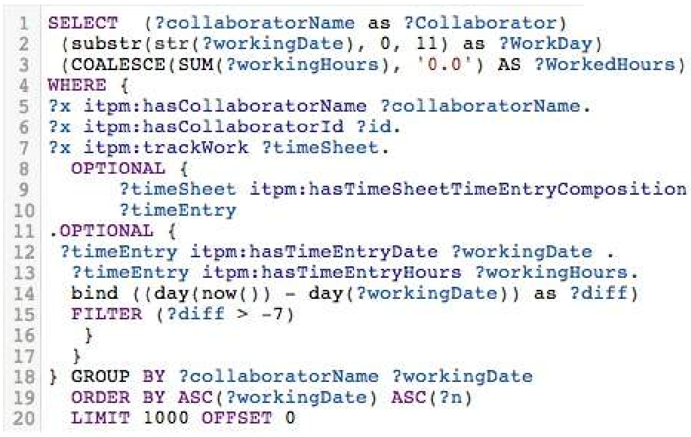
Johannes-Bergmann-(Org-C)
Consultant—CEO
Stefan-Biffl-(Org-A)
Project Supervisor and Head of the CDLFlex
Martin-Unterauer-(Org-C)
Team-Leader: Development of the ITPM Dashboard and Editor Integration
Richard-Mordinyi-(Org-A)
Co-Supervisor, Interface CDLFlex-Team, Project Controlling
Dietmar-Winkler-(Org-A)
Quality Control and Consultant
Developer-2-(Org-C)
Developer
Developer-1-(Org-C)
Developer
Developer-3-(Org-C)
Developer
Andreas-Grünwald-(Org-B)
Project-Manager
CDLFlex-Research-Team-(Org-A)
Contacts regarding development of Semantic Editor
Quality Management (Org D)
Quality Asurance and Control
Estefania Serral (Org-A)
Advisor Semantic-Technologies—Researcher-CDL-Flex
Fajar-Juang-Ekaputra-(Org-A)
Advisor,-Colleague—Researcher-CDL-Flex
Development Team Org. A
3 Senior Software Developers
Andreas-Pieber-(Org-D)
Contact for OpenEngSb-services-Software-Developer
Felix-Rinker-(Org-D)
Contact-for-Engineering-Cockpit—Quality Engineer
Angelika-Musil-(Org-D)
Contact-person-for-innovative-technologies—Quality Engineer
Jürgen-Musil-(Org-D)
Contact-person-for-UI-prototyping-&-testing
Kristof-Meixner-(Org-D)
Release Control/Team Leader

Figure 7.14: Graphical representation of the team hierarchy within the ITPM dashboard.

The derivation of the team hierarchy is visualized in figure 7.14. The example demonstrates the efficiency of SPARQL to query graph hierarchies. Line 12 establishes the link between collaborators (`?x`) and their superiors (`?y`). The optional statements (line 15, line 20, line 25) collect information about the team membership, and the roles of the collaborator within the organization and the team. The lines 2-7 format the output and map it to the predefined ITPM graph structure. More precisely, the field names `?title`, `?text`, `?icon`, and `?status` are keywords that enable the knowledge engineer, to format the appearance of the graph (icons, color, shapes, labels), based on common practices of the Twitter Bootstrap framework (cf. chapter 2.3).

Query 4: Time Exposures per Day and Collaborator for the Past Week For each team member (or sub-team), a data source configuration is added to the ITPM dashboard (*Synchronizer Cockpit*). For the UC, all team members track their expenses on a provided spreadsheet template (as the case at the CDL-Flex). The spreadsheet template consists of the following columns (the format of the CDL-Flex-internal time-sheets is applied): week of year, month, optional issue number, optional artifact ID, optional UC name, task description, effort in hours.

Because all collaborators utilize the same spreadsheet format, the mapping between the spreadsheet columns and the common concepts only needs to be performed once. For the remainder, the configuration can be copied; only the source location must be modified. It is up to the collaborators, to track their efforts in MS Excel, or online (Google spreadsheets). The integration of both formats is supported.



```
1  SELECT  (?collaboratorName as ?Collaborator)
2   (substr(str(?workingDate), 0, 11) as ?WorkDay)
3   (COALESCE(SUM(?workingHours), '0.0') AS ?WorkedHours)
4  WHERE {
5  ?x itpm:hasCollaboratorName ?collaboratorName.
6  ?x itpm:hasCollaboratorId ?id.
7  ?x itpm:trackWork ?timeSheet.
8    OPTIONAL {
9        ?timeSheet itpm:hasTimeSheetTimeEntryComposition
10       ?timeEntry
11  .OPTIONAL {
12   ?timeEntry itpm:hasTimeEntryDate ?workingDate .
13    ?timeEntry itpm:hasTimeEntryHours ?workingHours.
14    bind ((day(now()) - day(?workingDate)) as ?diff)
15    FILTER (?diff > -7)
16     }
17    }
18 } GROUP BY ?collaboratorName ?workingDate
19    ORDER BY ASC(?workingDate) ASC(?n)
20    LIMIT 1000 OFFSET 0
```

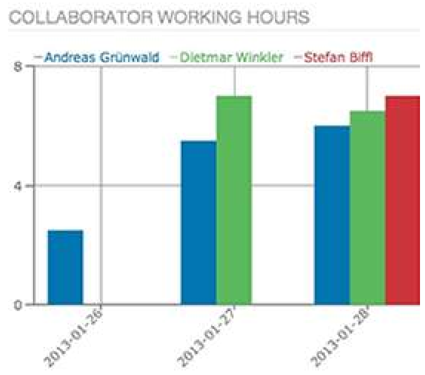


Figure 7.15: SPARQL for the aggregation of time exposures per day and collaborator (for the past seven days).

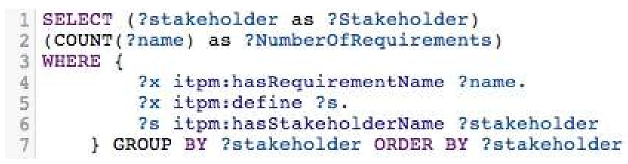
Figure 7.15 demonstrates the utilization of SPARQL, to retrieve the grouped time efforts, for each collaborator, throughout the past seven days. The core ITPM ontology maintains links between collaborators, time-sheets, and contained time entries, which SPARQL exploits to extract the dates and working hours from the time entries, and to link them to the related collaborators. The result is grouped by collaborators and dates (line 18). The lines 14 and 15 restrict the result set to time entries of the past seven days. Both, time sheets and time entries are optional (a time-sheet might not be available for every collaborator), which must be considered in the SPARQL query (lines 8 and 11).

The automated collection of time exposures does not only relieve the project manager from tremendous time-tracking efforts, but also improves (a) monitoring activities and (b) traceability. For (a), the project manager immediately detects, if a team member does not provide accurate tracking information, or the workload across collaborators is imbalanced. Regarding (b), the project manager can perform additional queries to link between issues/requirements and/or tests, for instance to derive, (1) how many time has been spent for the development of a software artifact in total, or (2) how many time has been spent by specific collaborators on an issue/requirement/artifact.

The integrated time efforts provide the basis for additional analysis and reporting activities. For instance, *wage rates* can be efficiently integrated and linked to project roles and collaborators. The wage rates can be either project-specific, or industry-based (cf., Payscale[9]).

Data Visualizations for Reporting and Controlling (Figure 7.2 - 7) The management of organization A demands accurate status reporting, every two weeks. Hence the project manager must derive a high-level summary of the project state, and send it to the executives.

To reduce the reporting effort, the project manager utilizes the ITPM dashboard for the creation of accurate reporting figures, such as KPIs, and high-level diagrams. These figures can easily be copied into a status report (e.g., MS Word), or are directly accessible by executives, via the ITPM dashboard's URL. Subsequently, a subset of queries is provided, which (a) help to reduce manual reporting efforts, and (b) aid the project manager herself in executing the project.



```
SELECT (?stakeholder as ?Stakeholder)
(COUNT(?name) as ?NumberOfRequirements)
WHERE {
        ?x itpm:hasRequirementName ?name.
        ?x itpm:define ?s.
        ?s itpm:hasStakeholderName ?stakeholder
    } GROUP BY ?stakeholder ORDER BY ?stakeholder
```

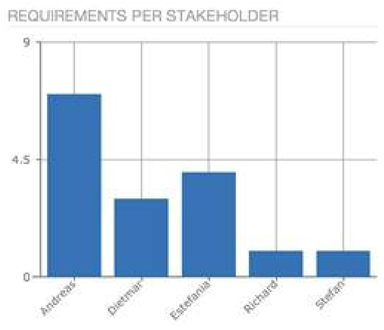


Figure 7.16: SPARQL query to visualize the number of requirements per stakeholder (diagram provides only a subset of the solution).

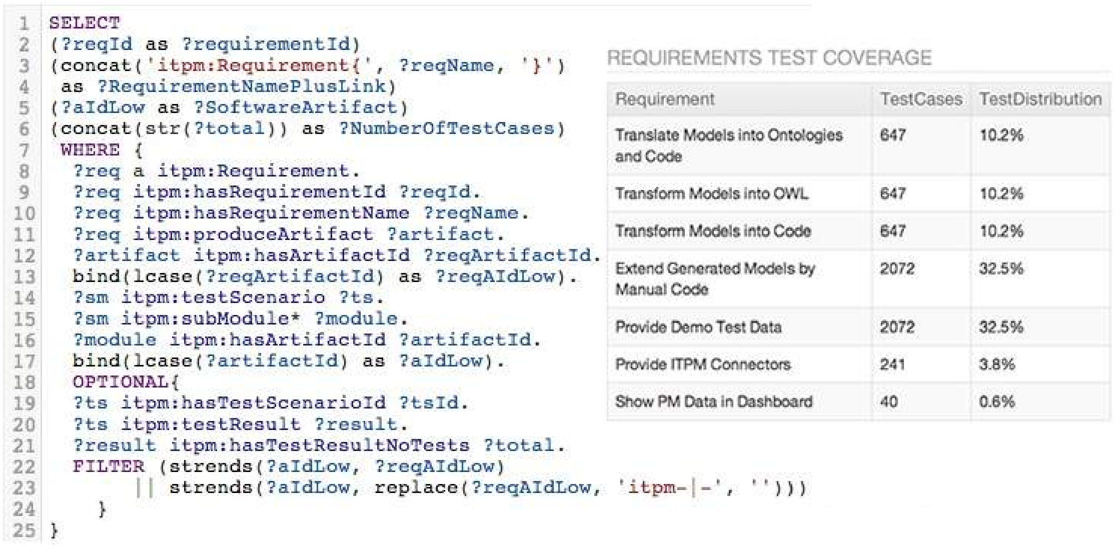
Query 5: Number of Requirements per Stakeholder The first reporting query addresses the problem that the formats of the local requirement spreadsheets, makes the inference of the number of requirements per stakeholder difficult. The problem is that the assigned stakeholder ids for each requirement are concatenated (e.g., "agr, dwi"). While this issue may seem trivial to the reader, it is in fact not easy to resolve for laymen, with limited technical background. In

[9]Payscale.com: `http://www.payscale.com`

addition, the problem is complicated, if non-structured requirement specifications are provided (e.g., MS Word, or Jira Confluence[10]).

Figure 7.16 states the SPARQL query for grouping the total number of defined requirements per stakeholder. The resulting diagram is appropriate for reporting to the executives, but also provides feedback about the stakeholders' influence and impact.

Query 6: Test Distribution across Requirements The second reporting query demonstrates, how requirement specifications can be traced to test results, which are located at the Jenkins build server (figure 7.17). The query returns the requirement id (line 2), the requirement name, including a link to the auto-generated form within the ITPM dashboard UI (line 3), the id of the related artifact (line 5), and the number of test cases (line 6).



```
 1 SELECT
 2 (?reqId as ?requirementId)
 3 (concat('itpm:Requirement{', ?reqName, '}')
 4  as ?RequirementNamePlusLink)
 5 (?aIdLow as ?SoftwareArtifact)
 6 (concat(str(?total)) as ?NumberOfTestCases)
 7  WHERE {
 8   ?req a itpm:Requirement.
 9   ?req itpm:hasRequirementId ?reqId.
10   ?req itpm:hasRequirementName ?reqName.
11   ?req itpm:produceArtifact ?artifact.
12   ?artifact itpm:hasArtifactId ?reqArtifactId.
13   bind(lcase(?reqArtifactId) as ?reqAIdLow).
14   ?sm itpm:testScenario ?ts.
15   ?sm itpm:subModule* ?module.
16   ?module itpm:hasArtifactId ?artifactId.
17   bind(lcase(?artifactId) as ?aIdLow).
18   OPTIONAL{
19   ?ts itpm:hasTestScenarioId ?tsId.
20   ?ts itpm:testResult ?result.
21   ?result itpm:hasTestResultNoTests ?total.
22   FILTER (strends(?aIdLow, ?reqAIdLow)
23        || strends(?aIdLow, replace(?reqAIdLow, 'itpm-|-', '')))
24     }
25 }
```

REQUIREMENTS TEST COVERAGE

Requirement	TestCases	TestDistribution
Translate Models into Ontologies and Code	647	10.2%
Transform Models into OWL	647	10.2%
Transform Models into Code	647	10.2%
Extend Generated Models by Manual Code	2072	32.5%
Provide Demo Test Data	2072	32.5%
Provide ITPM Connectors	241	3.8%
Show PM Data in Dashboard	40	0.6%

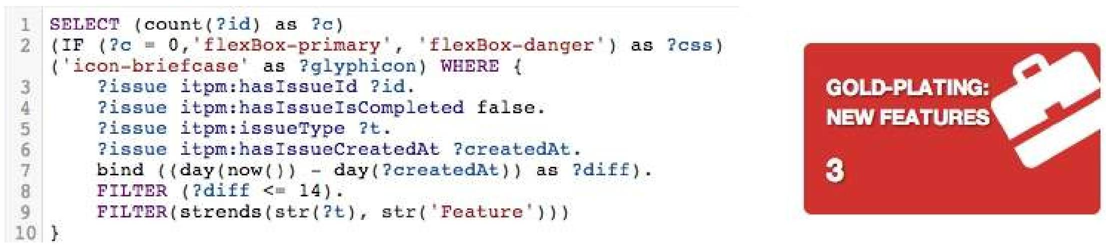
Figure 7.17: SPARQL query to determine the test coverage of requirements. The right-hand side visualizes the result, plus the distribution of tests.

The requirements are linked to the software modules via the *artifactId* (lines 12, 16). The information about the main software modules and their sub-modules is derived from Jenkins. Because the names of the modules in Jenkins mainly reside from Java package names, the artifact id must be normalized. For instance, all IDs are converted into lower-case (lines 13, 17), the prefix "itpm" is cleared, and all minuses are removed from the *artifactId* (line 23). In addition, two modules are considered to be the same, if the end with the same *artifactId* (lines 22, 23). Line 15 traverses the nested software module hierarchy, returning all sub-modules. For instance, the software module *cdlflex-mdd* consists of the packages *cdlflex-mdd*, *umltuowl*, *sembase*, and so on. The ids of all modules and sub-modules are mapped with the existing test scenarios (lines 14, 15). The test result is obtained at the level of the scenario (lines 20, 21).

[10]Jira Confluence: `http://www.atlassian.com/Confluence`

Based on the number of test cases per requirement, the distribution is derived. The distribution is visualized in the table of figure 7.17, but is excluded from the presented SPARQL query, to keep the query compact. The traceability between requirements and tests aids the management in detecting business needs that are not properly tested. The table in figure 7.17 reveals that some requirements have the same number of test cases assigned. The reason is that these requirements link to software artifacts that are part of the same sub-module hierarchy.

Query 7: Anti-Pattern Detection Another effective reporting method is the provision of summary tables or KPIs, which aggregate specific states or progresses within a single, meaningful figure.



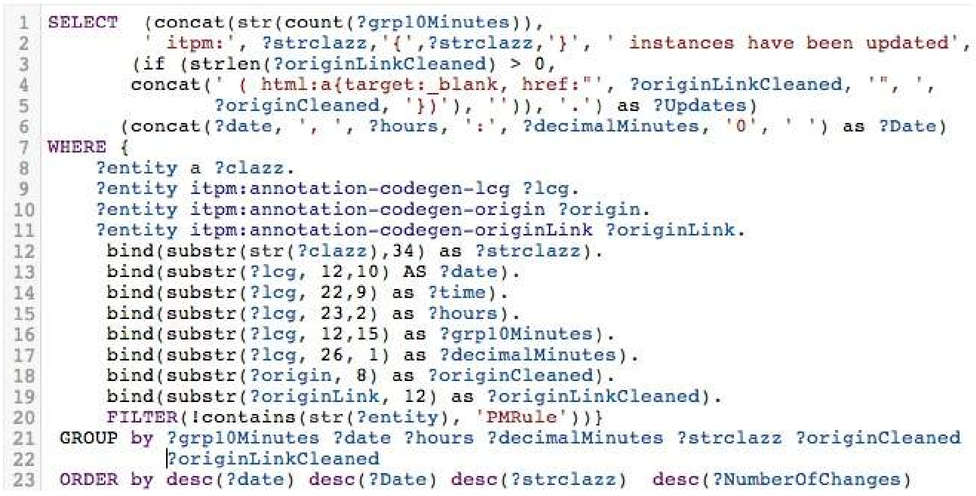
```
SELECT (count(?id) as ?c)
(IF (?c = 0,'flexBox-primary', 'flexBox-danger') as ?css)
('icon-briefcase' as ?glyphicon) WHERE {
    ?issue itpm:hasIssueId ?id.
    ?issue itpm:hasIssueIsCompleted false.
    ?issue itpm:issueType ?t.
    ?issue itpm:hasIssueCreatedAt ?createdAt.
    bind ((day(now()) - day(?createdAt)) as ?diff).
    FILTER (?diff <= 14).
    FILTER(strends(str(?t), str('Feature')))
}
```

Figure 7.18: Anti-Pattern Detection: Adding (unnecessary) features closely before a deadline.

Figure 7.18 demonstrates the implementation of an anti-pattern to avoid gold-plating. The SPARQL query accumulates all new *features* that have been created in a time-frame of 14 days from now. Unwanted, (not critical) business features that are added by local developers, close before the deadline, are detected. The result is visualized in a KPI. The color of the KPI automatically changes from red to green, if the anti-pattern is not violated.

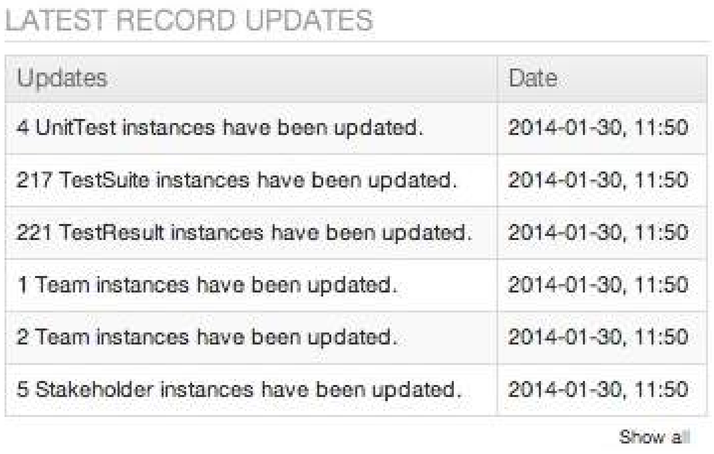
The presented anti-pattern can either be activated/deactivated manually, by the project manager, or it is extensible as follows. The creation date is compared against the milestones of the project (attribute “end date” in the project plan). The anti-pattern is automatically activated x days before a milestone (e.g., x=14).

Query 8: Monitoring of PM Artifact Updates Finally, figure 7.19 summarizes the updates of artifacts in any local system. The results are grouped by the main class of artifact (e.g., requirement, unit test, issue) into ten-minute slots (cf. figure 7.20). The resulting table provides a link to directly access the artifact in the local PMS instance. For instance, if an issue has been created in Jira, then the link directly forwards the manager to the specific issue URL in Jira. If no link to the local tool is available, then the user is forwarded to the auto-generated tables of the ITPM dashboard. The table relieves the project manager from tremendous data collection activities.



```
SELECT  (concat(str(count(?grp10Minutes)),
         ' itpm:', ?strclazz,'{',?strclazz,'}', ' instances have been updated',
        (if (strlen(?originLinkCleaned) > 0,
        concat(' ( html:a{target:_blank, href:"', ?originLinkCleaned, '", ',
               ?originCleaned, '})'), '')), '.') as ?Updates)
       (concat(?date, ', ', ?hours, ':', ?decimalMinutes, '0', ' ') as ?Date)
WHERE {
    ?entity a ?clazz.
    ?entity itpm:annotation-codegen-lcg ?lcg.
    ?entity itpm:annotation-codegen-origin ?origin.
    ?entity itpm:annotation-codegen-originLink ?originLink.
     bind(substr(str(?clazz),34) as ?strclazz).
     bind(substr(?lcg, 12,10) AS ?date).
     bind(substr(?lcg, 22,9) as ?time).
     bind(substr(?lcg, 23,2) as ?hours).
     bind(substr(?lcg, 12,15) as ?grp10Minutes).
     bind(substr(?lcg, 26, 1) as ?decimalMinutes).
     bind(substr(?origin, 8) as ?originCleaned).
     bind(substr(?originLink, 12) as ?originLinkCleaned).
     FILTER(!contains(str(?entity), 'PMRule'))}
 GROUP by ?grp10Minutes ?date ?hours ?decimalMinutes ?strclazz ?originCleaned
          ?originLinkCleaned
 ORDER by desc(?date) desc(?Date) desc(?strclazz)  desc(?NumberOfChanges)
```

Figure 7.19: SPARQL query for the visualization of the last updates of any integrated artifact.



LATEST RECORD UPDATES

Updates	Date
4 UnitTest instances have been updated.	2014-01-30, 11:50
217 TestSuite instances have been updated.	2014-01-30, 11:50
221 TestResult instances have been updated.	2014-01-30, 11:50
1 Team instances have been updated.	2014-01-30, 11:50
2 Team instances have been updated.	2014-01-30, 11:50
5 Stakeholder instances have been updated.	2014-01-30, 11:50

Show all

Figure 7.20: Result for the SPARQL query in figure 7.19 (excerpt).

7.1.3 Remarks on the Dashboard Configuration

Section 7.1 demonstrates the powerfulness of the ITPM dashboard for (a) the synchronization of data across local tools, based on data models and work-flows, (b) the integration of data into a central view, (c) the centralized visualization of data, based on SPARQL queries, and (d) the efficient configuration of distributed PM data sources, and dashboard visualizations (diagrams, graphs, tables, KPIs).

A major part of section 7.1.1 is dedicated to the description of mappings, between local PM tools, and common concepts. The mappings address semantic integration issues, and visualize that, even for a small number of tools, data integration tends to become complex, quickly.

All presented UML class diagrams are modeled with the Semantic Model Editor. The workflows are implemented as Drools rules. Both are reusable and included in the available prototype.

In total, a set of 25 queries/rules has been created for the UC, based on the ITPM dashboard (only a subset of the queries has been presented). Most of the queries belong to scope management, QM, time management, or integration management (PMBOK knowledge areas). The combination of several rules (e.g., combining open issues with the team management view) leverages the automation of sophisticated escalation mechanisms. For instance, if an issue has been assigned for longer than two weeks, but is still unresolved, then the superior of the developer can be identified based on SPARQL, and reported. However, sophisticated SPARQL queries have been found to embrace between 25 and 50 lines, and hence are not discussed here. Nevertheless, some of the visualization results are provided in appendix B.

Section 7.1 indicates that the configuration effort for the ITPM dashboard is manageable, and significantly reduces the coordination effort for the project manager. The dashboard configuration, and the time efforts, spent by the related project members, serve as the input for the cost-benefit analysis, and the comparison of the technical integration processes, in sections 7.2 and 7.3.

7.2 Cost-Benefit Analysis

The results of the dashboard configuration are analyzed numerically, to expose the usefulness of the integration framework in quantitative numbers.

In a first step, the manual integration efforts are determined *without* any automated data integration (neither the CNIP is performed, nor the ITPM dashboard is in place). For this purpose, the automatable scenarios, identified in section 7.1, were played through, and analyzed, thoroughly. In a second step, the configuration and setup costs of the ITPM dashboard, are summarized. The result is compared against the costs of the purely manual integration effort. The gap between the dashboard configuration costs and the manual integration efforts correspond with the net benefits for the project manager, when applying the systematic PM integration framework (SIFPM).

A small subset of 30 requirements, with similar structure as those of industrial CDL-Flex projects, is chosen, to observe and measure the UC integration scenarios in an appropriate and manageable way. Measurements that were not be obtainable directly, such as the number of reported (requirement) change requests and software bugs, were derived and interpolated thoroughly, based on the CDL-Flex's expert know-how, and the correlation of these artifacts with the project complexity (e.g., project size, number of requirements, number of collaborators). In addition, best-practice estimation techniques, as suggested by McConnell, were applied [83].

7.2.1 Analyzing the Manual Integration Approach

For the manual integration approach, the consortium project members do not consider, how local data is integrated, or accessed. The project manager is faced with (synchronization) conflicts the first time they appear, and handles data synchronization and permission issues in a reactive way.

Modular PM Actions	**PM Effort / Minutes**		
	Best Case	**Worst Case**	**Average**
0.1 Navigate to Data Source	1	10	5.5
0.2 Copy / Transform Requirement	2	15	8.5
0.3 Read Email and/or Extract Information	5	20	12.5
0.4 Locate Item within Data Source	1	10	5.5
0.5 Resolve Sync. Conflict (Top-MGMT)	20	100	60
0.6 Find all items of a specific type/status	5	10	7.5
0.7 Update Status of Feature	1	5	3
0.8 Create Report for Specific PM item	10	30	20
0.9 Resolve Sync. Conflict (Local Teams)	10	20	15
0.10 Request Report	5	20	12.5
0.11 Proofread/Forward Report	5	60	32.5

Table 7.1: Modular building blocks for the analysis of manual PM actions.

Table 7.1 outlines the modular PM activities that have been decomposed from the UC-scenarios in section 7.1. These activities are recurringly performed by the project manager, to access and exchange PM data between local systems, and consortium members. The presented time efforts rest on (a) recordings throughout the evaluation of the prototype, and (b) empirical values from industrial CDL-Flex projects.

The activities in table 7.1 represent the building blocks for higher-level activities, which are used to recompose the manual integration activities of the UC scenarios, and to ensure that all manual integration activities are covered. Table 7.2 visualizes a sub-part of the comprehensive composition structure, to demonstrate, how the manual integration efforts are accumulated.

For instance, “Transform requirements for MGMT” (activity 1.1) consists of the modular tasks “Navigate to Data Source” (activity 0.1) (executed once), and “Copy/Transform Requirement” (0.2) (executed $n = 30$ times, because the UC has been evaluated with 30 requirements). The transformation of requirements into Google, based on activity 1.1 is executed once throughout the entire project (cf. table 7.2, column “Appearances”). Hence the effort per appearance in hours is $(1 * 5.5 + 30 * 8.5)/60 = 4.3$ (the numbers 5.5 and 8.5 relate to the effort of the modular PM activities, in minutes; cf. table 7.1). The effort for the total project (last column of table 7.2), is calculated by multiplying the effort (per appearance) with the number of appearances throughout the project. Activity 1.1 appears only, viz., during the requirements specification phase (cf. first column of table 7.2).

P	Management Activity / Sub-Activity	Appear-ances (App.)	Effort per App. (Hours)	Effort Total Project (Hours)
Requirement Spec.	1.1 Transf. Requirements for MGMT	1	4.3	4.3
	0.1 Navigate to Data Source			
	0.2 Copy / Transform Requirement (n=30 items)			
	1.2 Change Single Requirement	n/3=10	0.5	5.1
	0.3 Read Email and/or Extract Information			
	0.1 Navigate to Data Source			
	0.4 Locate Item within Data Source			
	0.2 Copy / Transform Requirement			
	1.3 Resolve Sync. Conflict with Top-MGMT	5	1	5.3
	0.5 Resolve Sync. Conflict (Top-MGMT)			
Design & Implementation	2.1 Transform Requirements into Features	1	11.1	11.1
	0.1 Navigate to Data Source (2 times; Jira, Redmine)			
	2.1.1 Configure Project Core Data (2 times; Jira, Redmine)			
	2.1.2 Transform Requirement into Feature (n items)			
	2.1.3 Create Missing User Account (for Devs; 12 in total)			
	2.1.4 Create Missing Software Module (12 in total)			
	2.2 Create/Update (Req. Change)	n/5=6	0.85	5.1
	1.2 Change Single Requirement			
	0.1 Navigate to Data Source			
	0.4 Locate Item within Data Source			
	2.2.1 Create Feature			
	2.1.3 Create Missing User Account (for QM; 6 in total)			
	2.1.4 Create Missing Software Module (for QM; 6 in total)			
Test	4.1 Propagate Bugs from QM to DEV Teams	n*3/4=23	0.43	9.6
	0.3 Read Email and/or Extract Information			
	0.1 Navigate to Data Source			
	2.2.1 Create Feature			

Table 7.2: (De-)composition of manual PM integration activities (example).

The change of a single requirement (activity 1.2) is less efficient than the creation/update of all requirements at once, because the item / change needs to be extracted (e.g., based on an email, or via a face-to-face discussion with a researcher) (activity 0.3), and the specific requirement must be located within the requirements document (activity 0.4). In addition, the documents must be looked up on the local file systems every single time, while for activity 1.1 the search process for all 30 requirements takes place only once. One of the findings throughout the prototype evaluation was that user interactions resulted in higher efforts than expected. For instance, in the UC, the requirement documents were located within a complex file structure and difficult to locate. The access to Google spreadsheets involved the seeking of the access data (e.g., email inbox), and the provision of user permissions by the corresponding administrator.

The number of requirement changes, features and bugs is derived from the overall number of requirements. For instance, the number of requirement change requests throughout the initial project phases (requirement specification, design of the software architecture) has been assumed with $n/3$. The number of features is defined with $n * 3$ (for each requirement, three finer-grained backlog-items are created by the development team, on the average). The number of bug reports (activity 4.1) derives from the number of features, and is $n * 3/4$ (for each fourth feature a bug, or serveral bugs at once, are reported by the QM team).

Changes in later phases are more expensive. If a requirement is changed during the design or implementation phase (activity 2.2), then the requirement sources must be synchronized (1.2), and the related features in the PMS of the development teams must be updated. The number of requirement change requests throughout the implementation phase has been assumed with $n/5$ (changes in later project phases are less likely than in the initial phases). The creation (or update) of a feature requires the administration of user accounts, and software modules, from time to time (activities 2.1.3, 2.1.4). Without a systematic integration process, the administrator of the PMS needs to be contacted, if a required user turns out to be missing.

The manual integration efforts of the entire composition have been accumulated, and are associated to the seven integration scenarios of the UC. The final result is presented in section 7.2.3.

7.2.2 Subsuming Costs of the Systematic ITPM Integration Framework

The evaluation of the dashboard configuration in section 7.1 already outlined the main manual efforts for the integration of data sources, work-flows, and data visualizations. In addition, installation costs, and the execution of the CNIP meeting must be taken into consideration.

Role	**Hourly Rate Estimations (Cheapest=1)**	**Normalized Hourly Rate (PM=1)**
Developer (DEV) / Quality Engineer (QE)	1.00	0.67
Knowledge Engineer (KE)	1.00	0.67
Researcher (RES)	1.50	1.00
Project Manager (PM)	1.50	1.00
Executive (MGT)	2.00	1.33

Table 7.3: Hourly rate conversion table for project roles.

Table 7.3 weights the hourly rates for the heterogeneous project roles, which are involved in the use-case. The used conversion rates are in accordance with industrial rates, used for project cost estimations (obtained from Payscale, and rounded). For instance, the wage rate of a project manager (PM) is 1.5 times the rate of a developer (DEV). The normalized rate is calculated as $1/1.5 = 0.67$ (the wage rate of a DEV is 0.67 times the rate of a PM). The mapping is required to (a) accumulate the time efforts of the different collaborators that participate in the systematic integration approach, and (b) to compare the outcome with the non-integrated manual approach.

ITPM Consortium Integration Activities and Involved Roles	**Duration of Activity (Hours)**	**People Involved**	**People Involved Weighted**	**Total Hours Weighted**
Preparation of CNIP Meeting (PM)	2.0	1	1	2.0
CNIP Meeting (3 MGT, 3 DEV, PM, KE)	3.0	8	7.67	23.0
Setup of the ITPM Dashboard (PM, KE)	2.5	2	1.67	4.2
Data Source Configuration (PM, KE)	10.0	2	1.67	16.7
Workflow-Configuration (PM, KE)	25.0	2	1.67	41.7
Dashboard Config. Visualizations (PM, KE)	5.0	2	1.67	8.3
Additional Efforts (PM)	2.5	1	1.00	2.7
Integration without Connector Implementation				**98.5**

Table 7.4: Manual configuration effort for the systematic dashboard integration (weighted).

Table 7.4 outlines the accumulated efforts for the most relevant systematic integration activities. The CNIP management meeting takes three hours, and involves eight participants (three executives, three developers, the project manager, and a knowledge engineer). The value of each participant is weighted, based on table 7.3. For instance, the weighted value for the meeting is $3 * 1.33 + 3 * 0.67 + 1 * 1 + 1 * 0.67 = 7.67$. The total effort for each activity results from the multiplication of the activity's duration with the weighted number of involved people (e.g. $3.0 * 7.67 = 23.0$).

For all configuration activities a knowledge engineer is scheduled to assist the project manager. The reason is that the manager is most probably not familiar with SPARQL and several other technical aspects. The project manager discusses the visualization needs and requirements with the engineer, who implements and fine-tunes SPARQL queries, work-flows, and possibly simple integration tests. Although a pre-built ITPM dashboard prototype is downloadedable and can be installed within a few minutes, an additional buffer is incorporated, to account for the case that the knowledge engineer prefers to build the prototype from the source code. The position "Additional Efforts" addresses the project manager's effort for regularly accessing the *ITPM Dashboard* and its sub-features. This effort is scheduled, to compensate the calculated navigation costs of the non-integrated approach.

7.2.3 Results of the Analysis

The setup and configuration costs of the ITPM dashboard-based approach, respectively the SIFPM, are compared against the benefits of reducing inevitable manual PM integration efforts. For this purpose, the determined efforts for both approaches are mapped to the structure of the UC.

The final result is visualized in table 7.5. The comparison shows that the configuration costs for the systematic integration framework clearly trade-off, resulting in a reduction of 45% of a project manager's costs, for the particular UC. In addition, the integrated approach results in higher data quality (i.e., accuracy, accessibility, traceability, portability) and increases the reporting and tracking activities of the incorporated (development) teams [99].

Activity / UC Scenario	Dashboard Integration Effort	Manual Integration Effort
CNIP Management Meeting	25.0	-
Installation of the ITPM Dashboard	4.2	-
Synchronization of Requirements (RES / MGT) (1)	6.5	14.8
Derivation of Issues from Requirements (2)	38.3	44.8
Synchronization between PM Trackers / Project Configuration (3)	1.5	7.0
Reporting of Bugs by the QM (4)	3.3	27.8
Automated Derivation of Test Reports (5)	2.1	12.9
Integration of Time Efforts (6)	10.1	31.8
Creation of Data Visualizations for Reporting and Controlling (7)	4.7	39.7
Additional Efforts (UI Access & Navigation)	2.7	-
Total Efforts (no Connector Implementation)	**98.5**	**178.7**
	55%	**100%**

Table 7.5: Effort comparison for the ITPM dashboard-based approach and the manual approach.

7.3 Comparison of Development Efforts for PM Connectors

Section 7.2.3 evaluated the UC under the assumption that all connectors are available at first hand. This is not unrealistic, as the CDL-Flex is eager to develop and provide further, commonly used PM connectors, for the OpenEngSb project. In addition, related projects, such as Taskadapter, provide reusable PM connectors that reduce the integration effort for the OpenEngSb environment (e.g., Redmine Connector). Finally, the UC-evaluation showed that a considerable part of the data sources rests on reusable, highly flexible spreadsheet connectors that require no implementation effort.

Type of Connector Implementation	Duration of Activity (Hours)	First Domain Connector ?	Involved Developers (DEVs)	Involved DEVs. Weighted	Total Hours Weighted
Jira Issue	35	yes	1	0,67	23.3
Redmine (PM/Issue Tracking)	20		1	0.67	13.3
Jenkins QM	20	yes	1	0.67	13.3
Excel (Multi-Model)	60	yes	1	0.67	40.0
Google Spreadsheet (Multi-M.)	20		1	0.67	13.3
Scrumwise (not part of UC)	8		1	0.67	5.3
Average Impl. Effort	**27.17**		**1**	**0.67**	**18.1**
OpenEngSb Training Effort	40		**1**	0.67	26.5

Table 7.6: Implementation efforts for OpenEngSb connectors for the prototype implementation.

Table 7.6 summarizes the implementation efforts for connectors, that were implemented throughout the research, and while working on industrial CDL-Flex projects. The table reveals that the implementation for the first connector within a new domain is significantly higher than for the remaining connectors. Indeed, several artifacts (models, unit and integration tests) are reusable for the implementation of further connectors, and learning effects appear.

The main implication of table 7.6 is that the average implementation of an OpenEngSb PM connector, based on the MDSE approach, takes 27 hours. This value is biased, because the implementation efforts depend on the skills of the developer, her experience with the OpenEngSb, and the type and application of the connector. Nevertheless, it is a sound reference value is for further analysis and estimation.

7.3.1 Evaluation Results for Different Numbers of Connectors and Requirements

To ensure the stability of the UC evaluation, the designed activity analysis model, described in section 7.2.1, is utilized, to perform a basic simulation of a project manager's net benefits, depending on the number of available connectors, and the number of missing connectors, which must be implemented by the consortium project team.

The design of the decomposable cost-benefit analysis model enables basic simulation of manual integration costs, based on a number of input parameters. The available input parameters, and their configuration for the UC evaluation in section 7.2.1, are: number of requirements (30), project complexity factor (1), number of collaborators (22), project duration (20 weeks), duration of the analysis phases (8 weeks), duration of the implementation/testing phases (10 weeks), (sub-) issues per requirement (3).

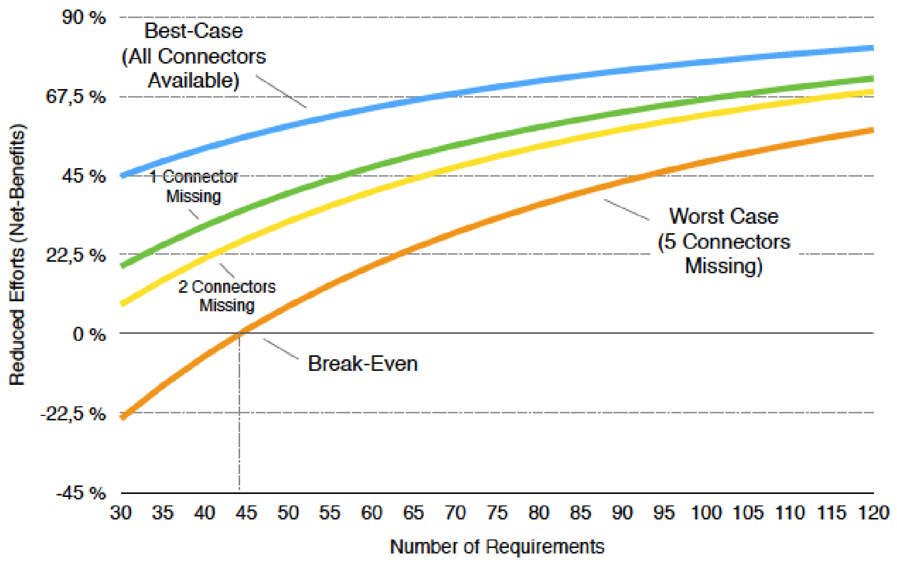


Figure 7.21: Net benefits resulting from the application of the ITPM dashboard.

Figure 7.21 summarizes the simulation results. The number of requirements, and the available connectors are adjusted, while all other input parameters of the model remain the same. The figure illustrates that, even if one or two connectors are missing, the application of the systematic integration framework still is beneficial for the project manager. In the worst-case scenario, in which none of the five OpenEngSb connectors is available, the application of the ITPM dashboard is not advisable (-22.5% of additional costs). However, it is very likely that the investment in the systematic integration approach trades off for projects with more than 45 requirements.

The simulation outcome clearly visualizes the advantages of the SIFPM, already for a very small number of requirements. The net benefit for the project manager increases with the number of requirements and PM artifacts. Although the simulation results are simplified, because several environmental factors, such as increasing project complexity, and longer project duration are not considered, the simulation model is stable, because these factors tend to increase the benefits of an integrated approach even more [83].

7.3.2 Process-Based Effort Comparison for the Development of Connectors

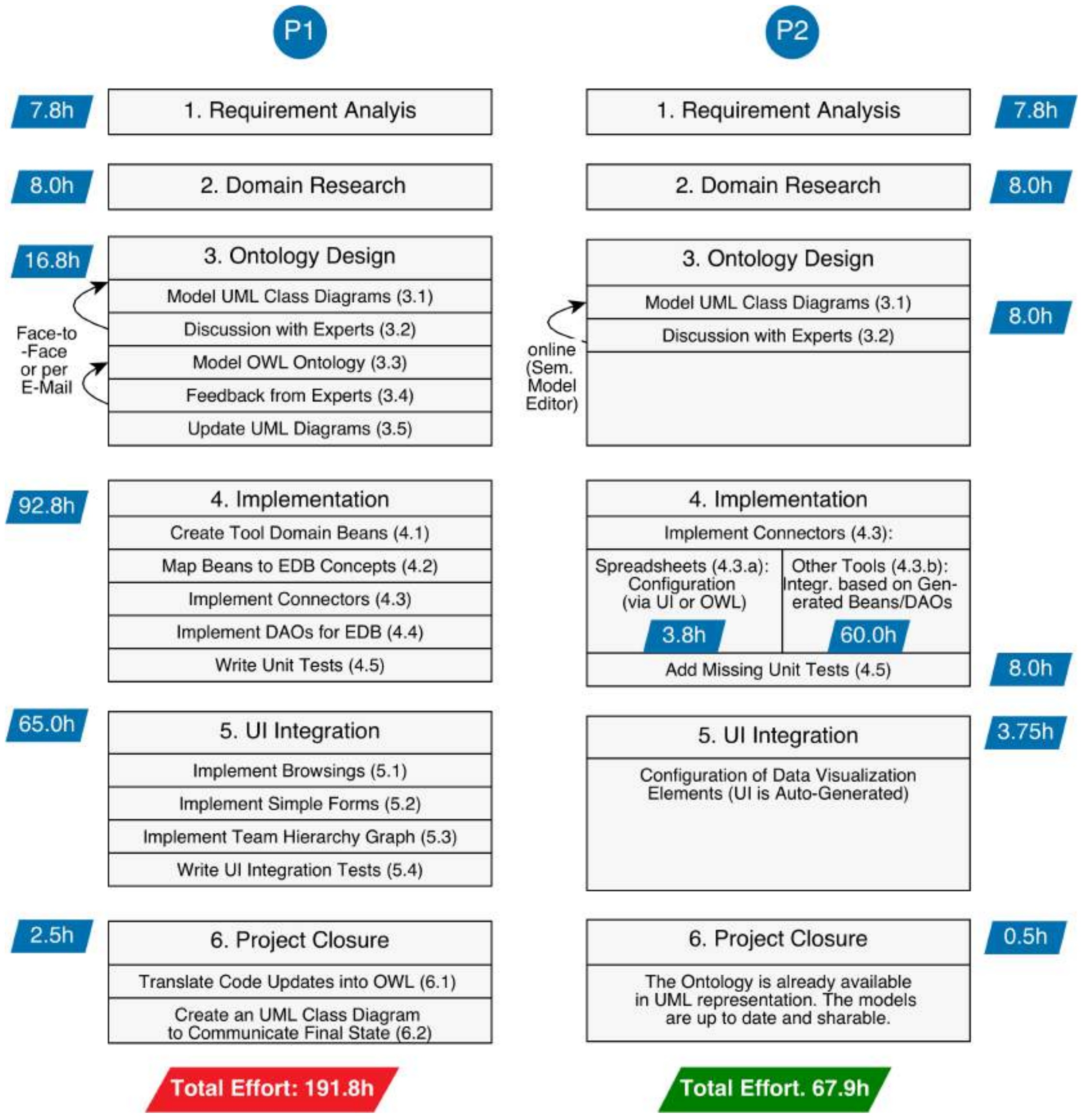
So far, the analysis of the net benefits focused on the viewpoint of the project manager. This section compares the MDSE-based solution approach against the (conventional) OpenEngSb connector implementation and integration approach, based on the time records of an industrial project.

The findings of the process comparison are summarized in figure 7.22. The process P1 resembles the traditional process. The process P2 depicts the efficient MDSE-based integration approach, based on the Semantic Model Editor, and the code-generation.

Conventional OpenEngSb Tool Integration Process

Process P1 has been extracted from the historical records of an industrial project. The main objective of the project was to (a) research and define three new project tool domains (projects, collaborators, team data), (b) to develop three related spreadsheet connectors for the integration of a customer's demo data, and (c) implement basic UI elements for the visualization and modification of the underlying data. The development has been conducted by an external developer (contract on work base), and comprised 191.8 hours.

P1 started with the analysis of the customers (and the CDL-Flex's project manager's) requirements by the (external) developer (1). Step 2 comprised a domain research, which involved the identification and verification of existing solution approaches (ontologies). In step 3 the ontology was developed. To facilitate the discussion between the involved project collaborators (the project manager, a researcher, two experienced CDL-Flex developers, the external developer), several representations of the ontology needed to be created, and maintained. The resulting models (UML, OWL) were discussed and updated several times, based on discussions with the project manager, and feedback from the researchers. For instance, first the ontology needed to be modeled in UML (3.1). A screenshot of the resulting class diagram was taken and forwarded to the project manager, who analyzed the result, and discussed it with the researcher, and the developers (3.2). When the model seemed to be mature enough, an OWL ontology was derived (3.3). This ontology again, was refined successively, to match with the degree of granularity of pre-existing ontologies at the CDL-Flex (3.4). To communicate the progress to the project manager, the corresponding UML class diagrams were updated several times (3.5).



P1
P2
7.8h
1. Requirement Analyis
1. Requirement Analysis
7.8h
8.0h
2. Domain Research
2. Domain Research
8.0h
16.8h
3. Ontology Design
Model UML Class Diagrams (3.1)
Discussion with Experts (3.2)
Model OWL Ontology (3.3)
Feedback from Experts (3.4)
Update UML Diagrams (3.5)
Face-to -Face or per E-Mail
3. Ontology Design
Model UML Class Diagrams (3.1)
Discussion with Experts (3.2)
online (Sem. Model Editor)
8.0h
92.8h
4. Implementation
Create Tool Domain Beans (4.1)
Map Beans to EDB Concepts (4.2)
Implement Connectors (4.3)
Implement DAOs for EDB (4.4)
Write Unit Tests (4.5)
4. Implementation
Implement Connectors (4.3):
Spreadsheets (4.3.a): Configuration (via UI or OWL)
3.8h
Other Tools (4.3.b): Integr. based on Generated Beans/DAOs
60.0h
Add Missing Unit Tests (4.5)
8.0h
65.0h
5. UI Integration
Implement Browsings (5.1)
Implement Simple Forms (5.2)
Implement Team Hierarchy Graph (5.3)
Write UI Integration Tests (5.4)
5. UI Integration
Configuration of Data Visualization Elements (UI is Auto-Generated)
3.75h
2.5h
6. Project Closure
Translate Code Updates into OWL (6.1)
Create an UML Class Diagram to Communicate Final State (6.2)
6. Project Closure
The Ontology is already available in UML representation. The models are up to date and sharable.
0.5h
Total Effort: 191.8h
Total Effort. 67.9h

Figure 7.22: Comparison of the traditional (P1) and the MDSE-based (2) integration process.

The implementation of the connectors (4) required the translation of the modeled ontology into corresponding Java classes/beans (4.1). The beans were extended with (Java) annotations to map the semantic meaning of specific fields to the internal format of the EDB, in which the OpenEngSb instances reside. For instance, a bean's id is annotated explicitly (4.2). After the beans were created, the corresponding domain interfaces and OpenEngSb connectors have been created (4.3). The connectors included basic CRUD methods, to modify and retrieve data from Excel spreadsheets, and to persist them into the EDB (4.4). Finally, a dozens of unit tests for the Excel connectors, and the DAOs were created.

In the UI Integration phase (5), the connectors were integrated into the pre-existing OpenEngSb framework, based on Wicket. The main implementation steps involved the creation of simple tables, including fulltext search (5.1), the implementation of forms to modify the integrated data instances (5.2), and the graphical representation of the team hierarchy based on a Java Script tree (5.3). The functionality of the UI integration was ensured by executing a handful of jUnit integration tests. Finally, the deviations between the implemented Java beans, and the model artifacts (UML classes, OWL ontology) needed to be reverse-engineered and passed to the project manager (phase 6).

MDSE-Based Tool Integration Process

The MDSE-based counterpart of process P1 is visualized on the right-hand side of figure 7.22 (process P2). Compared to P1, the main process steps remain the same. However, a number of (sub-)activities become obsolete; only the sub-activities (3.1, 3.2, 4.3, 4.5) remain. The main time savings of the MDSE result from the collaborative modeling approach (3.1, 3.2), and the efficient re-utilization of the central design models for the derivation of code and UI elements (4.3, 5). In addition, the ITPM dashboard enables the configuration of elaborate tables, diagrams, and graphs (break-down structures). Subsequently, the main activities of P2 are summarized, step by step.

The effort in process step (3) is halved, because the Semantic Model Editor facilitates the collaborative development of OWL ontologies based on UML class diagram representation. As a result, (a) the inefficient communication of expert feedback via email or face-to-face discussions is avoided (3.2), (b) the tedious manual synchronization between UML diagrams and OWL ontologies disappears, and (c) the model quality (granularity, credibility, accuracy, consistency) increases.

A big part of the routine work in step (4) becomes obsolete, because (a) Java beans, (b) DAOs, and (c) numerous unit tests for the verification of the existing models and DAOs, are generated (4.3, 4.5). The *MDDConnector*, which is part of the ITPM dashboard prototype, enables the configuration and mapping of spreadsheet formats via OWL, or the dashboard's UI. Hence, the integration effort for spreadsheets is minimal (4.3.a).

To compare the processes objectively, case (4.3.b) is considered as the counterpart of (4.3.a), in which no automated configuration is possible. Instead, in (4.3.b) all connectors must be implemented from scratch. The efforts of (4.3.a) and (4.3.b) are averaged for the process comparison.

The MDSE-based approach also reduces the time effort for the integration of new connectors significantly: The Semantic Model Editor facilitates the analysis, and modeling of tool APIs, and facilitates the generation of code artifacts for local tool connectors (cf., chapter 6.4.4). For instance, if the data structure of a REST API is modeled accurate in every detail, then the generated bean instances become mappable with the content of the local data models (e.g., JSON), automatically. The generated DAOs provide interfaces to validate these beans, and allow their further processing.

The flexibly configurable ITPM dashboard makes the implementation of UI elements obsolete. All required browsings (5.1) and forms (5.2) can be automatically derived from the maintained models within the *Semantic Model Editor*. In addition, sophisticated visualization mechanisms enable the representation and formatting of data in work-breakdown structures. Hence the costly implementation of (5.3) becomes obsolete, and is replaced by a simple SPARQL query configuration. On a side node, the outcome of the configured break-down structure was of higher quality than the manually developed ad-hoc visualization.

Finally, the reverse-engineering from code into models (step 6) is not required any more, because in the MDSE-based approach all code artifacts are synchronized with the central ITPM models/ontologies.

The comparison of the total efforts reveals that the MDSE-based approach accelerates the development process for the specific industrial project, by almost 125 hours. In other words, the total effort is cut by 65%. Interestingly, all the UI requirements were completely covered by the ITPM dashboard (e.g., creation of work-breakdown structure), and the quality of the interface elements was even higher, than for the conventional approach.

CHAPTER 8

Discussion and Limitations

The outcomes of the design-science based research approach, including the UC results, are critically reflected, based on the designed artifacts.

RI-1 Efficient Integration of PM Software and (Abstract) Data Concepts in Temporary IT Project (Management View)	**RI-2 Efficient Utilization of Data Models for PM Data Integration (Implementation View)**
RI-1.1 Negotiation and Integration of PM Data at the Management Level	RI-2.1 Efficient Technical Implementation of PM Data Integration Scenarios
RI-1.2 PM Data Integration / Mapping Between Local PM Systems	RI-2.2 UML Class Diagrams for Efficient Communication and Integration
RI-1.3 An Integrated PM View to Improve Monitoring and Reporting Activities	RI-2.3 Maintaining Shareable and Adaptable ITPM Reference Models

Table 8.1: Research Issues and Sub-Issues

Table 8.1 summarizes the research issues defined in chapter 3. Throughout the discussion, the sub-issues are associated to the corresponding design artifacts. Threats to validity, and the limitations of the research are stressed throughout the discussion, and complemented at the end of the chapter.

The Consortium Negotiation and Integration Process (CNIP) The CNIP addresses the negotiation and integration of local PMSs and data sources at the management level (R1.1, R1.2). The process incorporates proven methods and process elements that stem from Boehm's Win-Win process, DWH integration, change management, and the PMBOK (e.g., the creation of a communication plan).

Boehm et al. apply the Win-Win approach for the effective negotiation of optimal stakeholder (project) requirements. In contrast, in the CNIP not PM requirements, but the identification, access and integration of valuable PM data across various partners, is negotiated. However, because the Win-Win approach is theoretically funded, it is analogously applicable for the efficient negotiation of integration requirements.

The features that are incorporated in large DWH integration projects, are packed into an off-the-shelf applicable, smaller-scale process, to enable a cost-efficient execution of the process for temporary project organizations. The output of the process are typically UML class diagrams (RI-2.2) that serve as the input for the technical implementation (2.1), and document the agreed integration model. However, the CNIP also addresses the resolution of inefficient integration scenarios, based on a set of alternative solutions, such as the homogenization of the project members' internal project processes, or the utilization of a joint PMS. Such solutions can be more efficient than the exchange of data via the intermediate *ITPM Dashboard*, and might be right under the project managers nose.

Two issues of the CNIP are not explicitly addressed throughout the description of the process. The first issue relates to the case, in which a new consortium partner joins a running project. In the second case, a consortium already exists, and a new project, including one (or more) additional consortium partner(s), is initiated. The first issue is not discussed further, because of the commonly known rule-of-thumb that *adding manpower to a late software project makes it even later* [23]. For the second issue, it is recommended to re-execute the CNIP. The project manager, who is in charge to setup the CNIP meeting, should decide, based on the consortium environment, whether to (a) invite all consortium members again, (b) only include critical consortium members and/or representatives, or (c) to negotiate only with the new consortium member(s).

The mapping between the components and the UI of the *ITPM Dashboard* is discussed in detail. The CNIP is theoretically funded. For instance, the modeling of a common data model (e.g., in UML) is also a common practice for the design of DWH. Moreover, the modeling of (knowledge) models is supported by the *Ontology 101* engineering methodology. The CNIP rests on industrial experiences of the CDL-Flex. The process is elaborated at the conceptual level, and exemplified exemplified during the configuration of the UC environment (cf. 7).

ITPM Dashboard and Sub-Modules The *ITPM Dashboard* is the main outcome of the research process, and facilitates the centralized integration, configuration and monitoring of local data sources, and aggregated data visualizations.

Semantic Model Editor The *Semantic Model Editor* is a valuable by-product that emerged throughout the design process, and provides a simple graphical ontology editor, based on UML class diagram notation (RI-2.2). The main benefits of the editor, compared with state-of-the-art editors, such as Protégé, are (a) the compact and simple representation of OWL concepts via UML class diagrams, (b) the facilitation of collaborative modelling via a Web-Based UI, (c) the automated derivation of high-quality UI elements (tables, forms), (d) the tight incorporation of (customizable) code generation capabilities for Java, and (e) the effective creation of configurable data visualization, in conjunction with the *Semantic Query Editor*. In addition, the *Semantic Model Editor* tightly integrates code generation features for the OpenEngSb, both together forming a valuable OS community contribution (RI-2.1).

The graphical interface of the *Semantic Model Editor* only supports a limited set of OWL, which results from the trade-off between simplicity and completeness. However the auto-created OWL ontologies are designed, so that knowledge engineers can extend ontologies (e.g., add expert rules, additional restrictions) via other editors. These rules enable the leveraging of additional reasoning capabilities (not discussed here), and are maintained isolated, so that they remain after model updates.

Model-Driven Code Generator (MDDCodeGen) The MDDCodeGen is incorporated within the *Semantic Model Editor*, and facilitates the generation of beans, DAOs, and unit tests, for the abstracted CRUD access of semantic data stores (OWLAPI, Jena) (R-2.1, R-2.2). For this end, the existing *umlTUowl* meta-model has been utilized and extended (chapter 6.3). The code generation approach is similar to RDFReactor, which is the state-of-the-art code generator for semantic technologies. The main advantages of the MDDCodeGen, compared with the RDFReactor, are that (a) MDDCodeGen is optimized for the integration within the OpenEngSb environment (e.g., the beans are semantically annotated), (b) MDDCodeGen supports the OWLAPI, which the RDFReactor does not, and (c) the created unit tests provide an additional process step for verifying the stability of the data models, and related DAOs. While the RDFReactor is based on *Velocity* templates, the MDDCodeGen utilizes *Xtend* for the effective extension and debugging of code templates in Eclipse (cf. [45] [46] on more details about the code generation approach).

Semantic Query Editor / Central ITPM Dashboard The *Semantic Query Editor* facilitates the creation of aggregated queries and rules, for the visualization of aggregated PM data within the *Central ITPM dashboard* (R-1.3). The current prototype has been tested with SPARQL queries, but the support for additional languages, such as OWL DL, is incorporated into the architecture. The *Semantic Query Editor*, in conjunction with the *Semantic Model Editor*, is a valuable research outcome, for several reasons. First, there is a demand in ontology and query editors. For instance, Jena does not provide a SPARQL editor at all. For Protégé, only a very simple SPARQL plugin exists. The plugin does not support syntax highlighting or auto-completion, which the *Semantic Query Editor* does. Second, the query outcomes can be visualized, and are flexibly configurable, so that the results become permanently observable. The visualizations (graphs, diagrams, tables, work-breakdown structures, etc.) can be organized and edited on-the-fly, and are extensible, based on the Twitter Bootstrap CSS framework.

With regard to PM, the *Semantic Query Editor* enables project managers to define queries and/or rules, that are structured, based on the knowledge areas of the PMBOK. Chapter 7.1.2 demonstrates the application of the query editor for the creation of SPARQL rules, and the aggregation and visualization of related data. The definition of rules/queries via SPARQL enables the project manager to efficiently detect escalations, or anti-patterns, such as unresolved issues closely before a deadline. One extension in this regard, is to enable the shipping of notifications (e.g., via email) to the project manager, for instance if specific deviations appear. The organization of rules, based on knowledge areas, supports the project manager in creating a query repository, which she can fill with “her, personal PM knowledge”. These knowledge queries/rules are reusable for future projects, and shareable with other managers.

Central ITPM Dashboard and Data Access The central ITPM dashboard enables the visualization and monitoring of aggregated data (R-1.3). Its main features for PM monitoring and reporting have already been summarized above. The main strength of the dashboard is the dynamic configurable layout, which enables the modification and experimentation with simple data aggregations, and visualizations, on-the-fly. Compared with the current visualization capabilities of the OpenEngSb, for which, for each new tool domain, a UI must be implemented, the ITPM dashboard represents an essential relief from manual implementation efforts, demonstrated in chapter 7.3.

An additional feature of the (overall) *ITPM Dashboard* is the *PM Data view* (cf. 6.3), which facilitates the central browsing and modification of integrated data, based on auto-generated UI elements. Because the UC test data (e.g., requirements, issues) frequently exceeded the visualization boundaries, a simple algorithm has been implemented that reduces the data columns, based on a set of simple heuristics (first remove all empty columns, then remove all columns with more than 90% of empty cells, afterwards remove boolean attributes, and so on). Throughout the evaluation of the UC, these heuristics led to compact, meaningful representations of the integrated data. The auto-generated tables improve the traceability of PM data, because for each record the origin, the dates for creation and modification, and the origin of the record are persisted. Based on the table-view, the project manager is directly forwarded to the local PMS, and (if supported by that PMS) to the specific sub-page of the linked record (for instance, for Jira and Redmine, the specific issue page is visualized directly). Based on these functionalities, the project manager can either (a) edit integrated data based on the common representations, directly within the auto-derived and powerful dashboard UI, or (b) leverage the full capabilities of a (preferred) local PMS (e.g., modify local attributes, or utilize features that are not included in the ITPM dashboard).

The auto-derived UI forms, whose mapping with corresponding elements is elaborated extensively in chapter 6.3, resulted in (for the author) surprisingly good outcomes: The structures of OWL ontologies naturally map with the concepts of UI forms, and enable the user-oriented visualization of attributes, and nested associations. The models of the *Semantic Model Editor* are leveraged to (a) derive the appearance of the fields (e.g., text fields, check-boxes, multi-choice auto-completion menus), (b) provide help-text support based on deposited model comments, (c) perform basic validations (e.g., check the cardinalities of attributes and associations), and (d) to create, link and edit associated data instances, based on embedded sub-forms. The meta-data within the model is also effectively utilized to deal with compositional associations: Because in UML, a composition implies that the linked element cannot exist without its parent element, the related data instances are automatically deleted when they are unlinked. These features have been evaluated prototypically (manual UI tests). One limitation of the current editor is that the visualization of nested entries is only supported for one level. For instance, a stakeholder of a requirement is editable directly within the form, but the associated project data (of the requirement of the stakeholder) cannot be modified in the same view (the user has to switch to the stakeholder view). The reasons for this limitation are (a) the data visualization is mixed-up for a larger number of nested forms, and (b) the resolution of nested self-references (e.g., a requirement links to a stakeholder, which links back to the same requirement) would require additional implementation effort.

Synchronization Cockpit Based on the *Synchronization Cockpit*, the project manager is in possession to monitor, and actively control the entire synchronization process, including the synchronization of specific sources (upstream, downstream, activate/deactivate sources).

One important feature of the cockpit and the underlying algorithm, is the reporting of synchronization failures (e.g., missing or offline data sources, unresolvable data dependencies). In addition, interdependencies between sources are automatically resolved, as far as possible. For instance, during the UC evaluation, team data was upstreamed from two different sources. Spreadsheet A contained a link to a (superior) team member of record B. While synchronizing, the dependency was detected, and resolved, as soon as the data of spreadsheet B was available in the central semantic data store (the link between the two records was then explicitly retained in the central store).

The *Synchronization Cockpit* facilitates the project manager to either run the synchronization process permanently in background, or once in a while (e.g., once a day), depending on the resource-intensity of the local data access. Depending on the type of data, changes are either detected on file-level (e.g., Excel), or on record-level (e.g., Redmine). Nevertheless, the synchronization always takes place on record-level. E.g., provisions are in place that detect whether the local representation of a record and its central representation vary, and whether a synchronization is necessary or not.

The synchronization process itself was not focus of the research, but has been implemented as a means to an end, to demonstrate the implementation of the mappings between the local and the common data concepts. However, because the OpenEngSb currently includes no algorithm for the synchronization of data (a) within same, and across different data domains, based on work-flows, (b) for the synchronization of data of the same tool instance (e.g., two instances of Jira Issue), the process has been contributed to the OpenEngSb, so that it can be developed further. In its current state, the main shortcomings of the process are: (a) The process does not incorporate the synchronization of local time systems (e.g. the time at Google Drive, and a local system are not synchronized). Second, concurrent access by multiple (local) team members is addressed only limited. For instance, if one team (member) updates a feature in local PMS A, but simultaneously a collaborator of another team commits the deletion of the replicated feature via the local PMS B, then the resolution of the conflict is not clearly addressed. However, there are provisions to resolve simultaneous data modifications, based on the timestamp of the last update. Nevertheless, these synchronization issues requires additional investigation.

Model Integration and Re-utilization The integration and re-utilization of models is demonstrated in several ways. First, the models are the central artifact of the CNIP (1.1). Second, a mapping between UML class diagram elements and additional model annotations, based on the *Semantic Model Editor* is provided (2.2). To the best knowledge of the author, and related work [48], literature does not provide such a comprehensive, consistent mapping between (a) UML class diagrams, (b) ontologies, (c) object-oriented (Java) code, and (d) UI elements. However, related work addresses mappings between partial artifacts: For the mapping between (a) and (b), *umlTUowl* is extended. For instance, the support for enumerations, attribute multiplicity, and simple data constraints (mapped to OWL Facets) are added. Useful, not-implemented features include the annotation for attribute uniqueness, and ordering (in the solution approach, the first attribute, or attributes named "id" are assumed to be unique identifiers). The mapping between (b) and (c) incorporates the approaches of the OpenEngSb, Jastor, and RDFReactor. The mapping between (a) and (d), respectively (a), (b), (c), and (d), is designed with the intention to simplify the re-usability of the *Semantic Model Editor*'s outcomes for project managers. The SWB approach could not be reused in this respect, because the specific mapping solutions, were not available. UI generation approaches that solely rest on UML class diagrams (or relational database schemes) were excluded from the research process.

Third, the gap between the integration of tool-internal models, and local as well as common concepts, is reduced (RI-2.1). This issue is discussed, based on the OpenEngSb architecture. Chapter 6.4.3 outlines that the utilization of data models reduces the manual integration effort for developers, and simplifies the mapping between different representations of the same, conceptual model (tool-internal view, local model, common model, association with PMBOK knowledge areas and tool domains). The main benefits of models in this respect are (a) the automation of data integration (e.g., automated JSON/XML) transformations, (b) the role of data models for the documentation and communication of APIs, local concepts, etc., (c) the reusability of models and their extensibility/adaptablity, and (d) the utilization of the models for code generation.

Repository for Semantic ITPM Models The main idea of the semantic repository is to efficiently reuse and share formalized PM knowledge (models, queries/rules, data source configurations) and implementations (PM connectors) (RI-2.3). The solution approach briefly evaluates the feasibility of a shared repository, and the reusability of specific artifacts. The main research contribution in this respect are, (a) the structured organization of (abstract) ITPM models and specific PMS models, in an extensible and reusable fashion, (b) the representation of models in both, compact UML class diagram notation, and machine-interpretable, OWL, and (c) the conceptual discussion, on how to reuse/share ITPM models, rules, (d) and PM connectors.

Specific attention has been paid to address the criticism of practitioners regarding the application of normative, best-practice models. Nevertheless, the solution approach also supports normative models, such as the RefModPM, and the related PROMONT ontology. Currently, the exchange of models is limited to data models (i.e., UML class diagrams). The exchange of work-flows is not supported, which is a limitation, compared to RefModPM and PROMONT.

In its current state, the ontological artifacts of the repository (models, rules, queries, configurations) are efficiently reusable by a single project manager, and/or a set of familiar project managers, who work in similar organization contexts, and all apply the *ITPM dashboard*. As the results are all persisted in OWL, the ontologies potentially also can be shared across the community. However, as the queries are tied to the organization-specific application of tools and system, the exchange of these artifacts will most likely require their redesign; an issue that is not addressed throughout this thesis.

Reflection of UC Results The UC configuration exemplifies the application of the MDSE approach based on, (a) a set of mappings between local and common data models (UML class diagram notation), (b) a set of work-flows for the automation of manual synchronization activities across various PMSs, knowledge areas, and tool domains, and (c) a set of SPARQL queries for the automated derivation of aggregated PM knowledge, and the visualization of escalations and anti-patterns (RI-1.1).

The specific mappings between local and common concepts include the following (semantic) mapping issues: (a) mapping between different representations of the same entities, and their IDs (e.g. internal stakeholder acronym vs. full stakeholder name), (b) mapping between same concepts with different designations (e.g., Jira: "feature type"; Redmine: "tracker"), (c) mapping of same attributes with different data ranges (e.g. priority), (d) mapping between a bunch of attributes/concepts (e.g. the representation of a status in Jira and within the core ITPM data model). One result of the UC configuration is that some semantic mappings cannot be resolved in an objective way, unambiguously. For instance, the mapping of different priority scales (e.g., Jira: five categories; Redmine: four categories) will always remain subjective.

The mappings between the tools are effectively implemented in Java. The evaluation and application of an efficient mapping language between models is not addressed within the thesis, an is left for further research. Two possible candiates are ATL, and the OpenEngSb-internal transformation editor, for the graphical representation of model mappings.

The implementation of SPARQL queries based on the *Semantic Query Editor* is convenient; sophisticated data visualizations (e.g., the representation of a work-breakdown structure) are efficiently configurable and adaptable, even by the project manager. However, the design of SPARQL also reveals that more complex, nested queries (e.g., represent all team members in a hierarchy. Highlight all superiors, who manage a team member with one or more open issues) become very comprehensive (> 30 lines), and are difficult to interpret and maintain. In this respect, the upcoming SPARQL 1.2 version will provide simplifications, including more powerful path expressions[1]).

One criticism taken-up regarding the UC, was that the synchronization of data between "spreadsheets" seems not to be challenging. Expert experience at the CDL-Flex show that the complex-

[1]SPARQL Extensions: `http://www.w3.org/wiki/SPARQL/Extensions`

ity of data integration problems is often underestimated by practitioners; a fallacy which has been also experience throughout the conduction of this thesis. Related work and expert discussions show that spreadsheets are a widely used by project managers. The main challenge that is addressed within the thesis is not the “trivial” synchronizing between a set of files, but the mapping and resolution of semantic ambiguities on the record-level. In this regard, the structure of the spreadsheets (e.g., for requirements specification, time sheets) is similar to the structure of state-of-the-art PM software, so the UC can be seamlessly transformed into an environment with more sophisticated software products (e.g., IBM Doors, Harvest Time Tracking[2], MS Project).

Additional Limitations and Threats to Validity The current prototype only addresses the management of a single, isolated project, and only for the role of the project manager. A related limitation is that the generated DAOs are not optimized to efficiently retrieve data instances that are bound to a specific project.

The *Semantic Model Editor* currently utilizes an internal meta-model for the transformation between UML class diagrams and OWL ontologies. This meta-model origins from *umlTUowl* and facilitates the efficient transformation between Java, JSON, and the Web interface (via AJAX). Although conceptually complete, the reverse-engineering of OWL ontologies for the representation within the *Semantic Model Editor* still must be implemented.

The limitation of the mapping between local and common concepts, based on Java, has already been discussed. Although the Drools-based work-flows can be configured during run-time, they lack of a graphical representation, and hence are difficult to interpret by project managers. Another limitation is that the design of SPARQL queries is difficult to perform by project managers, and requires the support of a knowledge engineer.

Finally, the evaluation of the solution approach is framed to the presented UC. Although several measures have been taken to ensure the stability of the cost-benefit analysis (e.g., the efforts for manual integration have been assigned scarcely; the benefits for the local teams have been neglected; the time efforts have not been raised for a higher number of requirements, based on an increased complexity factor [83]), the outcome can still vary, for instance based on the APIs of the applied tools, the number of executed CNIP meetings, and the skills of the project manager. To provide an example, some PMSs implement basic import features, based on spreadsheets. Although the capabilities of these features are limited, and the import must be manually performed over and over again, a skilled project manager may leverage these functionality to reduce the manual effort.

[2]Harvest: `http://www.getharvest.com`

CHAPTER 9

Conclusion and Future Work

In multi-organizational environments, such as project consortia, conflicts and integration challenges appear at various organizational levels, and complicate the efficient integration of data. The data integration challenges are intensified, if the involved project organizations are very heterogeneous, thus having ambivalent objectives.

This work proposes a systematic integration framework for knowledge-supported project management in IT consortia. The solution approach, based on design-science, addresses integration issues at different organizational levels, which are rarely addressed in PM and integration literature, in the context of IT consortia:

- The *Consortium Negotiation and Integration Process* provides an off-the-shelf, easy-to-setup process for the negotiation of a common view on data integration, at the management level. This process addresses issues such as (a) building commitment and awareness for data integration, (b) identifying relevant data sources for integration and identify/generate solutions, and (c) enforcing the implementation of the data integration.

- The *ITPM Dashboard* is a downloadable, Web-based prototype[1] that facilitates the CNIP based on a number of sub-modules. The main features for project managers are (a) a central, configurable view on PM data, including monitoring, navigation, and reporting features (e.g., the detection of project anti-patterns), and (b) the central configuration and control of the synchronization of local PM data and knowledge. One main advantage of the *ITPM Dashboard*, opposed to PM suites, is that it does not not approach to be an all-in-one solution: All project teams sill work with their preferred local PM systems. The dashboard centralizes all configured data sources, and accelerates the access and query of local data sources. Nevertheless, it can be completely operated in background, enabling the project manager to use her preferred local PMS.

[1]ITPM Dashboard: `https://bitbucket.org/agruenwald/custom-itpm-builds`

- At the technical level, the data integration is based on the OpenEngSb, which is actively developed at the CDL-Flex.
- Data models represent the link between the various levels of integration. For this purpose, MDSE and Semantic Web technologies (OWL, SPARQL) are leveraged to (a) create, document, and pass negotiated data models between the management level, and the development, (b) utilize data models for the creation of code and UI artifacts to make the technical integration more efficient, and (c) enrich models with semantic technologies in order to make them reusable and shareable.

  The main design artifact regarding models is the *Semantic Model Editor*, which facilitates the creation, organization, and utilization of OWL ontologies based on UML class diagram notation. The editor facilitates features for code generation, collaborative modelling, and is integrated with the *Semantic Query Editor*, which enables the creation of data visualizations (tables, diagrams, KPIs, work-breakdown graphs), based on SPARQL. The *Semantic Model Editor* represents an innovative semantic modeling tool that actually could be used as a standalone knowledge-engineering tool (several usage requests by researchers already have been received).

The prototype and the systematic integration framework are applied to a consortium UC for the development of an innovative software product. The configuration of the prototype for the UC showed that the approach is efficiently usable to identify the project members' data sources, to model the data structures, and to map and integrate ambivalent semantics, based on a common view.

The systematic integration framework closes the gap between heavy-weight DWH integration solutions, and lean, but unsystematic ad-hoc integration. The evaluation results, based on the UC, indicate that the manual integration efforts of the project manager are significantly reduced: If all connectors are available, then the effort is reduced by 45%, for 30 requirements. Even if all PMS connectors must be implemented from scratch, the integration effort for a project with 65 requirements, is reduced by almost 25%.

The comparison of the MDSE-based process for implementing a new OpenEngSb with the conventional approach shows that the effort is reduced by 65%, based on the available results of an industrial project. The main savings results from (a) the reduced communication and transformation effort of ontologies, (b) the reduced implementation effort based on the derivation of code, tests, and UI elements, (c) the configurable UI dashboard, which makes the manual implementation of prototypical UI visualizations, obsolete.

The main conclusion is that the systematic integration framework is a comprehensive PM integration framework that addresses several aspects of IT consortia projects that are currently not addressed in literature.

The framework is backed by a sophisticated Web-based ITPM dashboard, and evaluation results from a UC that has been carefully constructed based on workshop-results and expert feedback. In addition, related work (e.g., PMS, knowledge-based PM, DWH, EII, *Semantic Project Management*, MDSE, *Agile Web Dashboards*) has been researched to identify and incorporate PM data integration analogies.

The research outcome is valuable for managers in IT project consortia, but also for researchers / knowledge engineers who experiment with semantic technologies. For the later, the literature review revealed a strong need in efficient data integration and visualization tools (related work often relies on very basic PM data, because the setup of a data base, respectively the integration of semantic data causes a big effort).

Future Work Four main fields are suggested, to be addressed in future work.

Experimenting with Additional Data / Use-Cases This work invested plenty of effort in the identification of requirements for PM data integration, and the development of related design artifacts. However, the evaluation of the framework is based on a single UC. The first proposal for future work hence addresses the evaluation of the framework for further use-cases, respectively real-world scenarios. For instance, three consortia could be identified, for which the framework can be applied and refined in a pilot study (to provide a reference, Boehm et. al evaluated their WinWin Spiral model for 15 teams [19]).

Visualization and Configuration of Data A current shortcoming of the approach is that the configuration of the dashboard requires the support of a knowledge engineer. Future research should address the application of UI elements that simplify the configuration and derivation of data/knowledge visualizations for project managers. One suggestion would be to investigate the reuse of technologies used in DWH, such as Online Analytical Processing (OLAP). For instance, in data warehousing, *data cubes* are commonly used to facilitate the visualization of data on different abstraction levels. Managers can combine different data attributes, and *drill-down*, or *roll up*, the resulting cubes. These techniques could be reused for the ITPM dashboard, e.g., to show trends, or the number of issues per day/week/month/quarter.

Another visualization issue that addresses the configuration of work-flows and mappings. Currently, work-flows must be implemented based on Drools, which is unsuitable for project managers. Currently, a related master-thesis at the CDL-Flex is in progress, which examines the visualization of work-flows based on Activiti[2]. The result of this thesis could be incorporated into the ITPM dashboard. Additionally, the visualization / Web-based configuration of transformations between models should be researched.

Model Versioning Model versioning addresses the evolution of the data models, and the traceability of changes over time for both, models and data. The incorporation of model versioning is

[2]Activiti: `http://activiti.org`

relevant, because it could help (a) to automatically trail PM rules/queries based on changes in the related models, and (b) to ensure data consistency, if models change over time (e.g., throughout the project).

A related issue is the capturing of historical data in a space-efficient way (space-complexitity). Capturing not only the current project state, but retaining historical data, can help the project manager to derive additional information about the project progress (e.g., how many issues were open two weeks ago, how many issues have been solved meanwhile).

Extending of the Scope of the PM Dashboard The scope of the ITPM dashboard can be extended in several ways. First, knowledge areas that were not addressed in the UC can be researched (e.g., risk management is an interesting candidate). Second, the integration of additional PMS should be forced, to extend the repository of available PM connectors within the OpenEngSb repository. Third, the ITPM dashboard can be extended to provide multiple (customized) views for different project roles. For instance, one master thesis at the CDL-Flex currently evaluates/extends the ITPM dashboard for the construction of a QM-centric view.

Bibliography

[1] *Proceedings of the 24rd International Conference on Software Engineering & Knowledge Engineering (SEKE'2012), San Francisco Bay, USA, July 1-3, 2012*. Knowledge Systems Institute Graduate School, 2012.

[2] Kovair omnibus for integrated alm. an overview. Technical report, Kovair, 2014.

[3] Sven Abels, Frederik Ahlemann, Axel Hahn, Kevin Hausmann, and Jan Strickmann. Promont–a project management ontology as a reference for virtual project organizations. In *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, pages 813–823. Springer, 2006.

[4] F Ahlemann. Project management systems: Typology, state-of-the-art and forecast. In *SOVNET, Russian Project Management Association (ed.): Proceedings of the 17th World Congress on Project Management. Moscow*. Citeseer, 2003.

[5] F. Ahlemann and K. Backhaus. *Project Management Software Systems: Requirements, Selection Process and Products ; a Study by the Research Center for Information Systems in Project and Innovation Networks ; [unbiased Comparison of 34 Products, Concise Assessments, Independent Decision Support]*. Oxygon-Verlag, 2006.

[6] Frederik Ahlemann. Towards a conceptual reference model for project management information systems. *International Journal of Project Management*, 27(1):19–30, 2009.

[7] Adel M Aladwani. Change management strategies for successful erp implementation. *Business Process management journal*, 7(3):266–275, 2001.

[8] Manoli Albert, Jordi Cabot, Cristina Gómez, and Vicente Pelechano. Automatic generation of basic behavior schemas from uml class diagrams. *Software & Systems Modeling*, 9(1):47–67, 2010.

[9] Grigoris Antoniou and Frank Van Harmelen. *A semantic web primer*. MIT press, 2004.

[10] Eric Armengaud, Matthias Biehl, Quentin Bourrouilh, Michael Breunig, Stefan Farfeleder, Christian Hein, Markus Oertel, Alfred Wallner, and Markus Zoier. Integrated tool chain for improving traceability during the development of automotive systems. *ERTS^2 2012–EMBEDDED REAL TIME SOFTWARE AND SYSTEMS*, 2012.

[11] Roger Atkinson. Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria. *International journal of project management*, 17(6):337–342, 1999.

[12] Motoei Azuma. Iso/iec cd 25010: Software engineering – software product quality requirements and evaluation (square) – software and quality in use models. Technical report, International Organization for Standardization, 2008.

[13] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[14] S. Biffl, R. Mordinyi, and T. Moser. Automated derivation of configurations for the integration of software (+) engineering environments. In *1st International Workshop on Automated Configuration and Tailoring of Applications, ACoTA*, 2010.

[15] S. Biffl and A. Schatten. A platform for service-oriented integration of software engineering environments. In *Proceeding of the 2009 conference on New Trends in Software Methodologies, Tools and Techniques: Proceedings of the Eighth SoMeT_09*, pages 75–92. IOS Press, 2009.

[16] Stefan Biffl, Wikan Danar Sunindyo, and Thomas Moser. A project monitoring cockpit based on integrating data sources in open source software development. In *SEKE*, pages 620–627, 2010.

[17] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *International journal on semantic web and information systems*, 5(3):1–22, 2009.

[18] Barry Boehm, Prasanta Bose, Ellis Horowitz, and M-J Lee. Software requirements as negotiated win conditions. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*, pages 74–83. IEEE, 1994.

[19] Barry Boehm, Alexander Egyed, Julie Kwan, Daniel Port, Archita Shah, and Ray Madachy. Using the winwin spiral model: a case study. *Computer*, 31(7):33–44, 1998.

[20] Barry Boehm, Paul Grunbacher, and Robert O Briggs. Developing groupware for requirements negotiation: lessons learned. *Software, IEEE*, 18(3):46–55, 2001.

[21] Barry Boehm and Hasan Kitapci. The winwin approach: using a requirements negotiation tool for rationale capture and use. In *Rationale management in software engineering*, pages 173–190. Springer, 2006.

[22] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182, 2012.

[23] Frederick P Brooks Jr. *The Mythical Man-Month, Anniversary Edition: Essays on Software Engineering*. Pearson Education, 1995.

[24] A. Brown and G. Wilson. The architecture of open source applications. *Lulu. com*, 2011.

[25] William J Brown, Raphael C Malveau, Hays W McCormick III, and Thomas J Mowbray. Refactoring software, architectures, and projects in crisis, 1998.

[26] William J Brown, Hays W McCormick, and Scott W Thomas. *Anti-Patterns Project Management*. John Wiley & Sons, Inc., 2000.

[27] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *KR*, pages 2–13, 1998.

[28] P. Clements, F. Bachmann, L. Bass, D. Garlan, P. Merson, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting software architectures: views and beyond*. Addison-Wesley Professional, 2010.

[29] Spikes Cavell Research Company. Bull report, 1998.

[30] T De Marco and T Lister. Peopleware: Productive projects and teams, dorset house, publishing co. *Inc. New York, NY, USA*, 1987.

[31] Prem Devanbu, Ron Brachman, and Peter G Selfridge. Lassie: a knowledge-based software information system. *Communications of the ACM*, 34(5):34–49, 1991.

[32] B. Dippelreiter. Semprom - semantic based project management. dissertation., 2012.

[33] Birgit Dippelreiter and Michael Püttler. Scenarios for evaluating a semantic project management approach. *Scientific Journal of Riga Technical University. Computer Sciences*, 43(1):65–71, 2011.

[34] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of data integration*. Elsevier, 2012.

[35] Hai Dong, Farookh Khadeer Hussain, and Elizabeth Chang. Application of protégé and sparql in the field of project knowledge management. In *Systems and Networks Communications, 2007. ICSNC 2007. Second International Conference on*, pages 74–74. IEEE, 2007.

[36] John Favaro. Value based management and agile methods. In *Extreme Programming and Agile Processes in Software Engineering*, pages 16–25. Springer, 2003.

[37] Mariano Fernández-López and Asunción Gómez-Pérez. Overview and analysis of methodologies for building ontologies. *The Knowledge Engineering Review*, 17(02):129–156, 2002.

[38] Bent Flyvbjerg and Alexander Budzier. Why your it project may be riskier than you think. *Harv Bus Rev*, 89(9):23–25, 2011.

[39] Guy G Gable. Integrating case study and survey research methods: an example in information systems. *European Journal of Information Systems*, 3(2):112–126, 1994.

[40] Stanislaw Gasik. Knowledge oriented project management. *21st IPMA World Congress*, 2007.

[41] Stanislaw Gasik. A model of project knowledge management. *Project Management Journal*, 42(3):23–44, 2011.

[42] I Gorton, I Hawryszkiewycz, and K Ragoonaden. Collaborative tools and processes to support software engineering shift work. *BT technology journal*, 15(3):189–198, 1997.

[43] James Grenning. Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods: Renaissance Software Consulting*, 3, 2002.

[44] Thomas R Gruber et al. Toward principles for the design of ontologies used for knowledge sharing. *International journal of human computer studies*, 43(5):907–928, 1995.

[45] Andreas Gruenwald, Dietmar Winkler, Estefania Serral, and Stefan Biffl. Semantic technologies for more-efficient model-driven development. Technical report, Christian Doppler Laboratory for Software Engineering Integration and Flexible Automation Systems, 2013.

[46] Andreas Gruenwald, Dietmar Winkler, Estefania Serral, and Stefan Biffl. Semantic technologies to accelerate model-driven development. 2013.

[47] A. Grünwald and T. Moser. umltuowl - a both generic and vendor-specific approach for uml to owl transformation. In *SEKE* [1], pages 730 – 736.

[48] Andreas Grünwald and Thomas Moser. umltuowl - a both generic and vendor-specific approach for uml to owl transformation. In *Proc. of SEKE*. Knowledge Systems Institute Graduate School, 2012.

[49] Joseph Gulla. Seven reasons it projects fail. *IBM Systems Magazine*, 2012.

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

[51] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data integration: the teenage years. In *Proceedings of the 32nd international conference on Very large data bases*, pages 9–16. VLDB Endowment, 2006.

[52] Alon Y Halevy, Naveen Ashish, Dina Bitton, Michael Carey, Denise Draper, Jeff Pollock, Arnon Rosenthal, and Vishal Sikka. Enterprise information integration: successes, challenges and controversies. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 778–787. ACM, 2005.

[53] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: concepts and techniques*. Morgan kaufmann, 2006.

[54] Paul Harris. Selecting project management software. 2003.

[55] Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. Executing sparql queries over the web of linked data. In *The Semantic Web-ISWC 2009*, pages 293–309. Springer, 2009.

[56] Matthias Heindl and Stefan Biffl. A case study on value-based requirements tracing. In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 60–69. ACM, 2005.

[57] S Heinonen, Jukka Kääriäinen, and Juha Takalo. Challenges in collaboration: tool chain enables transparency beyond partner borders. In *Enterprise Interoperability II*, pages 529–540. Springer, 2007.

[58] Brian Helbrough. Computer assisted collaboration—the fourth dimension of project management? *International Journal of Project Management*, 13(5):329–333, 1995.

[59] Jörg Henss, Joachim Kleb, Stephan Grimm, and IITB Fraunhofer. A protégé 4 backend for native owl persistence. In *2009 International Protégé Conference, Amsterdam, Netherlands*, 2009.

[60] Alan R Hevner. The three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):87, 2007.

[61] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[62] Jeff Hiatt and Timothy J. Creasey. *Change Management: The People Side of Change*. Prosci Learning Center, 2003.

[63] Linz Hines, Scott Baldwin, Mark Giles, and Juan Peralta. Implementing agile development in a waterfall project. Technical report, 2009.

[64] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F Patel-Schneider, and Sebastian Rudolph. Owl 2 web ontology language primer. *W3C*, 27, 2009.

[65] Pascal Hitzler, Markus Krotzsch, and Sebastian Rudolph. *Foundations of semantic web technologies*. CRC Press, 2011.

[66] Matthew Horridge and Sean Bechhofer. The owl api: A java api for owl ontologies. *Semantic Web*, 2(1):11–21, 2011.

[67] Rico Huijbers, Funs Lemmens, Bram Senders, Sjoerd Simons, Bert Spaan, Paul van Tilburg, and Koen Vossen. Software project management: methodologies & techniques. *Department of Mathematics & Computer Science*, page p20, 2004.

[68] ISO/IEC. Iso/iec 25012. software engineering — software product quality requirements and evaluation (square) — data quality model. Technical report, International Organization for Standardization, 2008.

[69] Joseki Jena and Pellet Fuseki. Semantic web frameworks, 2004.

[70] Anant Jhingran. Enterprise information mashups: integrating information, simply. In *Proceedings of the 32nd international conference on Very large data bases*, pages 3–4. VLDB Endowment, 2006.

[71] J. Johnson. Chaos summary. *The Standish Group Report*, 2009.

[72] Claudio Jossen and Klaus R Dittrich. The process of metadata modeling in industrial data warehouse environments. In *BTW Workshops*, pages 16–27. Citeseer, 2007.

[73] Aditya Kalyanpur, Daniel Jiménez Pastor, Steve Battle, and Julian A Padget. Automatic mapping of owl ontologies into java. In *SEKE*, volume 4, pages 98–103. Citeseer, 2004.

[74] Andrew Kannenberg and H Saiedian. Why software requirements traceability remains a challenge. *CrossTalk The Journal of Defense Software Engineering*, 2009.

[75] Gregory E Kersten. *NEGOTIATIONS AND E-NEGOTIATIONS. People, Models, and Systems*. Springer, 2010.

[76] Harold Kerzner. *Project management: a systems approach to planning, scheduling, and controlling*. Wiley, 2013.

[77] Ralph Kimball. *The data warehouse lifecycle toolkit: expert methods for designing, developing, and deploying data warehouses*. John Wiley & Sons, 1998.

[78] Henrik Kniberg. *Kanban and Scrum-making the most of both*. Lulu. com, 2010.

[79] KPMG. Global it project management survey, 2005.

[80] Ingolf H Kruger, Diwaker Gupta, Reena Mathew, Praveen Moorthy, Walter Phillips, Sabine Rittmann, and Jaswinder Ahluwalia. Towards a process and tool-chain for service-oriented automotive software engineering. 2004.

[81] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM, 2002.

[82] Andriy Lesyuk. *Mastering Redmine*. Packt Publishing Ltd, 2013.

[83] Steve McConnell. *Software Estimation: Demystifying the Black Art: Demystifying the Black Art*. Microsoft press, 2009.

[84] Fred R McFadden. Data warehouse for eis: some issues and impacts. In *System Sciences, 1996., Proceedings of the Twenty-Ninth Hawaii International Conference on,*, volume 2, pages 120–129. IEEE, 1996.

[85] Shahriar Mohammadi and Ali Khalili. A semantic web service-oriented model for project management. In *Computer and Information Technology Workshops, 2008. CIT Workshops 2008. IEEE 8th International Conference on*, pages 667–672. IEEE, 2008.

[86] Philip Moncrief. *Project Management. The Secrets of Success.* Xlibris, 2004.

[87] R. Mordinyi, T. Moser, and S. Biffl. Test-case generation for the validation of integrated automation systems engineering environments.

[88] Peter W.G. Morris. *Reconstructing Project Management.* Wiley-Blackwell.

[89] Thomas Moser and Stefan Biffl. Semantic tool interoperability for engineering manufacturing systems. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1–8. IEEE, 2010.

[90] Héctor Muñoz-Avila, Kalyan Gupta, David W Aha, and Dana S Nau. Knowledge based project planning. *Knowledge management and organizational memories*, page 125, 2002.

[91] N Noy and Deborah L McGuinness. Ontology development 101. *Knowledge Systems Laboratory, Stanford University*, 2001.

[92] Bijan Parsia and Evren Sirin. Pellet: An owl dl reasoner. In *Third International Semantic Web Conference-Poster*, volume 18, 2004.

[93] Andreas Pieber. Flexible engineering environment integration for (software+) development teams. Master's thesis, Vienna Technical University, 2010.

[94] PMI. *A Guide to the Project Management Body of Knowledge. (PMBOK Guide).* Project Management Institute, fourth edition, 2008.

[95] Axel Polleres. From sparql to rules (and back). In *Proceedings of the 16th international conference on World Wide Web*, pages 787–796. ACM, 2007.

[96] Mark Pruett. *Yahoo! pipes.* O'Reilly, 2007.

[97] PWC. Insights and trends: Current portfolio, programme, and project management practices, 2012.

[98] PWC. Insights and trends: Current programme and project management practices, 2012.

[99] Irfan Rafique, Philip Lew, Maissom Qanber Abbasi, and Zhang Li. Information quality evaluation framework: Extending iso 25012 data quality model. In *Proceedings of World Academy of Science, Engineering and Technology*, number 65. World Academy of Science, Engineering and Technology, 2012.

[100] Timothy Redmond. An open source database backend for the owl api and protege 4. In *OWLED*, volume 614, 2010.

[101] Nicholas C Romano Jr, Fang Chen, and Jay F Nunamaker Jr. Collaborative project management software. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 233–242. IEEE, 2002.

[102] Fran J Ruiz-Bertol, Daniel Rodríguez, and Javier Dolado. Applying rules to an ontology for project management. In *Proceedings of the 16th Spanish conference on software engineering and databases (JISBD 2011), A Coruña, Galicia, Spain, September*, pages 5–7, 2011.

[103] K. Schwalbe. *Information technology project management*. Course Technology Ptr, 2010.

[104] Dimitrios Settas and Ioannis Stamelos. Using ontologies to represent software project management antipatterns. In *SEKE*, pages 604–609. Citeseer, 2007.

[105] Shari Shang and Peter B Seddon. Assessing and managing the benefits of enterprise systems: the business manager's perspective. *Information Systems Journal*, 12(4):271–299, 2002.

[106] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[107] Rob Shearer, Boris Motik, and Ian Horrocks. Hermit: A highly-efficient owl reasoner. In *OWLED*, volume 432, 2008.

[108] Aaron Shenhar and Dov Dvir. *Reinventing project management: the diamond approach to successful growth and innovation*. Harvard Business Press, 2007.

[109] Evren Sirin and Bijan Parsia. Sparql-dl: Sparql query for owl-dl. In *OWLED*, volume 258, 2007.

[110] Michele Sliger. Agile projects in the waterfall enterprise. *Better Software*, 2006.

[111] Cynthia Snyder Stackpole. *A Project Manager's Book of Forms. A Companion to the PMBOK Guide - Fourth Edition*. John Wiley and Sons, 2009.

[112] Javier Solis, Hasdai Pacheo, Karen Najera, and Hugo Estrada. A mde framework for semi-automatic development of web applications. In *Proc on MDE and SE*, pages 241–246. SciTePress, 2013.

[113] Daniel B. Stang. Magic quadrant for it project and portfolio management, 2010.

[114] Wikan Danar Sunindyo, Thomas Moser, Dietmar Winkler, and Stefan Biffl. Foundations for event-based process analysis in heterogeneous software engineering environments. In *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*, pages 313–322. IEEE, 2010.

[115] Wikan Danar Sunindyo, Thomas Moser, Dietmar Winkler, Richard Mordinyi, and Stefan Biffl. Workflow validation framework in distributed engineering environments. In *On the Move to Meaningful Internet Systems: OTM 2011 Workshops*, pages 236–245. Springer, 2011.

[116] Kalpana Sureshchandra and Jagadish Shrinivasavadhani. Moving from waterfall to agile. *AGILE Conference*, 0:97–101, 2008.

[117] Piotr Szwed, Jan Werewka, and Grzegorz Rogus. Integration of classical and agile project management methodologies based on ontological models. *Production engineering in making*, 2010.

[118] Piotr Szwed, Jan Werewka, and Grzegorz Rogus. Ontology based alignment of classic and agile project managment for an it enterprise. 2010.

[119] Henri Theil. *Economics and information theory*, volume 7. North-Holland Amsterdam, 1967.

[120] Hal R Varian and Joseph V Farrell. *The economics of information technology: An introduction*. Cambridge University Press, 2004.

[121] Max Voelkel. Writing the semantic web with java. In *CDH Seminar'05 Galway, Ireland*, 2005.

[122] Max Völkel and York Sure. Rdfreactor-from ontologies to programmatic data access. In *Poster Proceedings of the Fourth International Semantic Web Conference*, page 55, 2005.

[123] M.N. Wicks. Tool integration in software engineering environments, 2004.

[124] MN Wicks. Tool integration in software engineering environments. Technical report, Citeseer, 2005.

[125] MN Wicks. Tool integration within software engineering environments: An annotated bibliography. *Heriot-Watt University, Tech. Rep*, 2006.

[126] OSLC Core Specification Workgroup. Oslc core specification version 2.0. *Open Services for Lifecycle Collaboration, Tech. Rep*, 2010.

[127] Jin Yu, Boualem Benatallah, Fabio Casati, and Florian Daniel. Understanding mashup development. *Internet Computing, IEEE*, 12(5):44–52, 2008.

[128] Michael H Zack. Managing codified knowledge. *Sloan management review*, 40(4):45–58, 1999.

[129] Yuxiao Zhao, Uwe Assmann, and Kristian Sandahl. Owl and ocl for semantic integration. 2004.

APPENDIX A

Additional Design Outcomes

A.1 Technical Integration Processes

A.1.1 Workflow Implementation Process

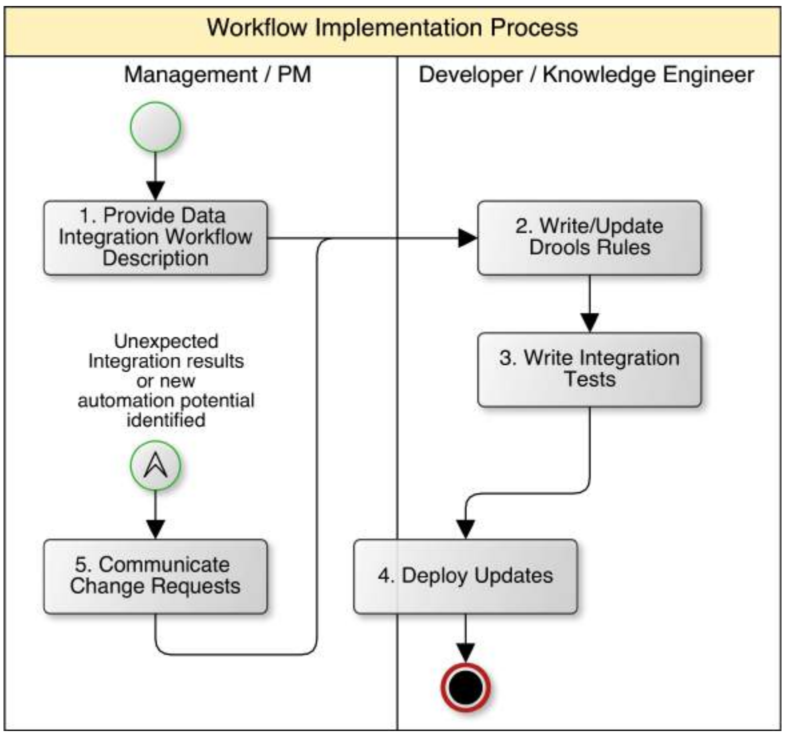

Workflow Implementation Process
Management / PM
Developer / Knowledge Engineer
1. Provide Data Integration Workflow Description
2. Write/Update Drools Rules
3. Write Integration Tests
Unexpected Integration results or new automation potential identified
5. Communicate Change Requests
4. Deploy Updates

Figure A.1: Implementation of work-flows based on the OpenEngSb integration framework.

A.1.2 Query Definition Process

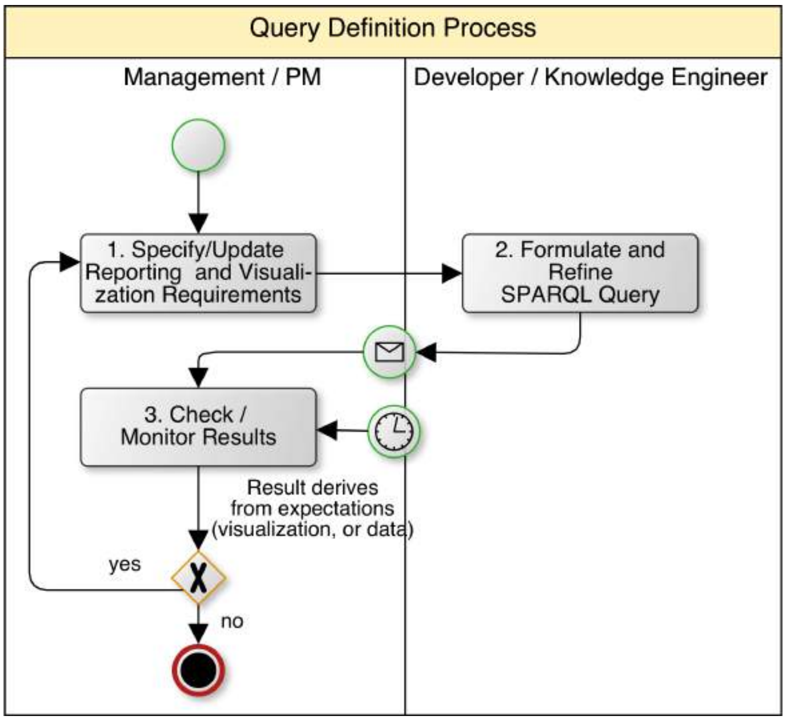

Query Definition Process
Management / PM
Developer / Knowledge Engineer
1. Specify/Update Reporting and Visualization Requirements
2. Formulate and Refine SPARQL Query
3. Check / Monitor Results
Result derives from expectations (visualization, or data)
yes
no

Figure A.2: Creation of dashboard visualizations based on SPARQL queries.

A.1.3 Model-Driven ITPM Data Integration Process

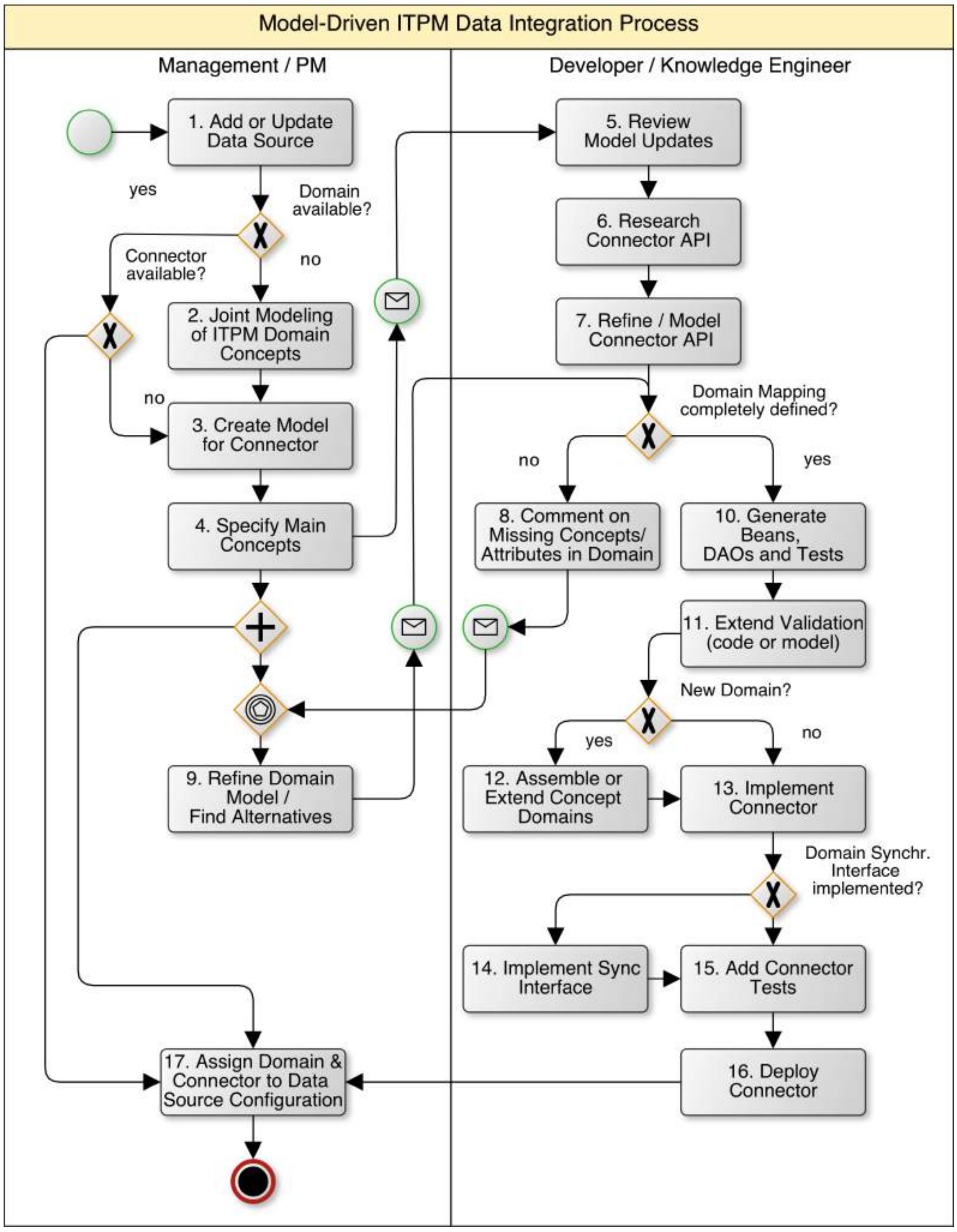

Model-Driven ITPM Data Integration Process
Management / PM
Developer / Knowledge Engineer
1. Add or Update Data Source
yes
Domain available?
Connector available?
no
2. Joint Modeling of ITPM Domain Concepts
no
3. Create Model for Connector
4. Specify Main Concepts
9. Refine Domain Model / Find Alternatives
17. Assign Domain & Connector to Data Source Configuration
5. Review Model Updates
6. Research Connector API
7. Refine / Model Connector API
Domain Mapping completely defined?
no
yes
8. Comment on Missing Concepts/ Attributes in Domain
10. Generate Beans, DAOs and Tests
11. Extend Validation (code or model)
New Domain?
yes
no
12. Assemble or Extend Concept Domains
13. Implement Connector
Domain Synchr. Interface implemented?
14. Implement Sync Interface
15. Add Connector Tests
16. Deploy Connector

Figure A.3: Conduction of the data integration process for a single PM tool. The process is based on the OpenEngSb framework (connectors, tool domains, domain concepts).

A.2 Complete Mapping between UML, OWL, Code, and UI Elements

Table A.1: Complete Mapping of model concepts and derived artifacts, based on MDSE.

UML Class Diagram (Sem. Model Editor)	**OWL Concept**	**Generated Code**	**Derived UI Elements**
Meta Class			
Abstract Class	OWL Class; rdfs:comment	Abstract Java Class	-
Interface	OWL Class; rdfs:comment	Java Interface	-
Concrete Class	OWL Class	Java Class (serializable; OpenEngSb @Model annotation)	Menu item; table-based search; form
Enum Class	OWL Class; for each enum value a subclass is created. Object property between class that uses enum attribute (domain), and OWL enum superclass (range)	One abstract Java Class (enumeration class name); for each enumeration constant a subclass; to pass a value to the class that uses the enumeration as an attribute, an instance of the subclass with default constructor needs to be passed (setter). The DAO facilitates the management of the value.	a dynamic single-select autocompletion menu which enables the user to select one of the enum values.
Meta Attribute			
Primitive attribute without data type	Data Property; type=any	compiler error	runtime exception
Numeric (double, int) and string attributes	Data Property with OWL restriction (exactly 1) and data range (XML;OWL)	private field; getter and setter	text field
Attribute of type Boolean	Data Property with OWL restriction and data range Boolean	private field; getter and setter	checkbox
Attribute of type DateTime	Data Property with OWL restriction and data range DateTime	private field of type Date; getter and setter	text field (extension: date/time picker)
First Attribute, or attributes named Id	OWL hasKey axiom	same as other attributes plus @OpenEngSbModelId Annotation	input field is disabled for updates
Primitive data type with multiplicity	Data Property with optional OWL (has max) restriction and data range (XML;OWL)	private final field (generic list); getter to access the elements of that list;	not implemented
Meta Association			

Continued on next page

Table A.1 – *Continued from previous page*

UML Class Diagram (Sem. Model Editor)	**OWL Concept**	**Generated Code**	**Derived UI Elements**
Unidirectional association	object property with range and domain; subClassOf property that links to the OWL Class.	private final field (generic list); depending on multiplicity either getter to access the elements of that list; or if cardinality <= 1: setter and getter to access element directly	A dynamic select menu (select2) to edit the linked instances; A subform within the main form to add or edit concrete classes.
Bidirectional association	two object properties with range and domain, linked to the related class via a subClassOf property. The two object properties are connected via an OWL inverseOf property.	same as unidirectional association; but setters/getters are available in both classes.	same as unidirectional association
Aggregation	same as association, except that a comment is added. Additionally, if the association is unnamed, the naming is different (typically: hasClassAOfClassB)	same as unidirectional association; naming of getter might be different by default (analogeously to OWL object property)	same as association
Unidirectional aggregation	analogeous to aggregation and unidirectional association (object property + subClassOf property + comment)	same as bidirectional association; naming of getter might be different by default (analogeously to OWL object property)	same as association
Composition	same as aggregation	same as aggregation	same as aggregation; however classes that are parts of a composition are not shown in the main navigation menu; they are directly maintained within sub-forms; when a link is removed, also the data instance will be deleted.
Unidirectional composition	same as unidirectional aggregation	same as unidirectional aggregation	same as composition
Association with max . multiplicity <= 1	additional constraint in the domain range (min, max, or exactly)	private field with getter and setter	Same as association; but only one item can be selected in the dynamic field.
Association with max. multiplicity > 1	optional constraint in the domain range (min; max)	private field (list) and a getter to access and manipulate the list elements.	Same as association; the dynamic select field allows the user to link a maximum of x records.
Meta Comment / Annotation			
Comment on class	rdfs:comment	Java comment at class level	help text at the beginning of the form
Comment on Attribute	rdfs:comment	Java comment at getter/setter	help text next to the field (popover)

Continued on next page

Table A.1 – *Continued from previous page*

UML Class Diagram (Sem. Model Editor)	**OWL Concept**	**Generated Code**	**Derived UI Elements**
Comment on Association	rdfs:comment	Java comment at getter/setter	help text next to the field (popover)
Internal Annotation	rdfs:comment	-	-
Attribute with a constraint stating that the size can be >= 200 characters (regex: pattern[.{0,200}])	Data Property with OWL restriction and data range String. Depending on reasoner capabilities whether rdfs:comment, or OWL Facet (pattern)	Additional condition in the validation method of the related Java DAO.	text area
OpenEngSb Extensions			
Connector Annotation	rdfs:comment	Processing as any model, but the generated code will be moved to the specific dev. Location within the OpenEngSb connector.	will be ignored because tool-specific.
Domain Annotation	rdfs:comment	Processing as any model, but the generated code will be moved to the specific dev. Location within the OpenEngSb tool domain.	A tab in the navigation menu of the ITPM Dashboard (sub-page PM Data) will be created.
Concept "suggested as available"	rdfs:comment	no code is generated	no UI elements are derived
Concept "globally available"	-	concept is processed as usual (default state)	UI elements are created as usual
Concept "local (tool-specific)"	rdfs:comment	only Java beans and DAOs (for validation) are created.	no UI elements are derived
Concept local and available.	rdfs:comment	Java beans, DAOs, and tests are generated for a specific connector;	no UI elements are (currently) derived.
Concept "tool-internal" (info purpose)	rdfs:comment	no code is generated	no UI elements are derived

A.3 Software Architecture of the ITPM Dashboard

A.3.1 Module View

The architecture consists of six layers (cf. figure A.4). The lower layers (1, 2, 3) include the generic code generation and data access for semantic technologies based on the model-driven approach. These layers allow the generic exchange of the underlying semantic technology, and do not relate to any PM - specific domain knowledge. The upper layers (3, 4, 5, 6) build upon the model-driven code generation approach. These layers access the underlying semantic data stores via the generated code (layer 3), and contain code and logic to process, integrate, and visualize IT project management data. The solution has been tighly integrated with the OpenEngSb.

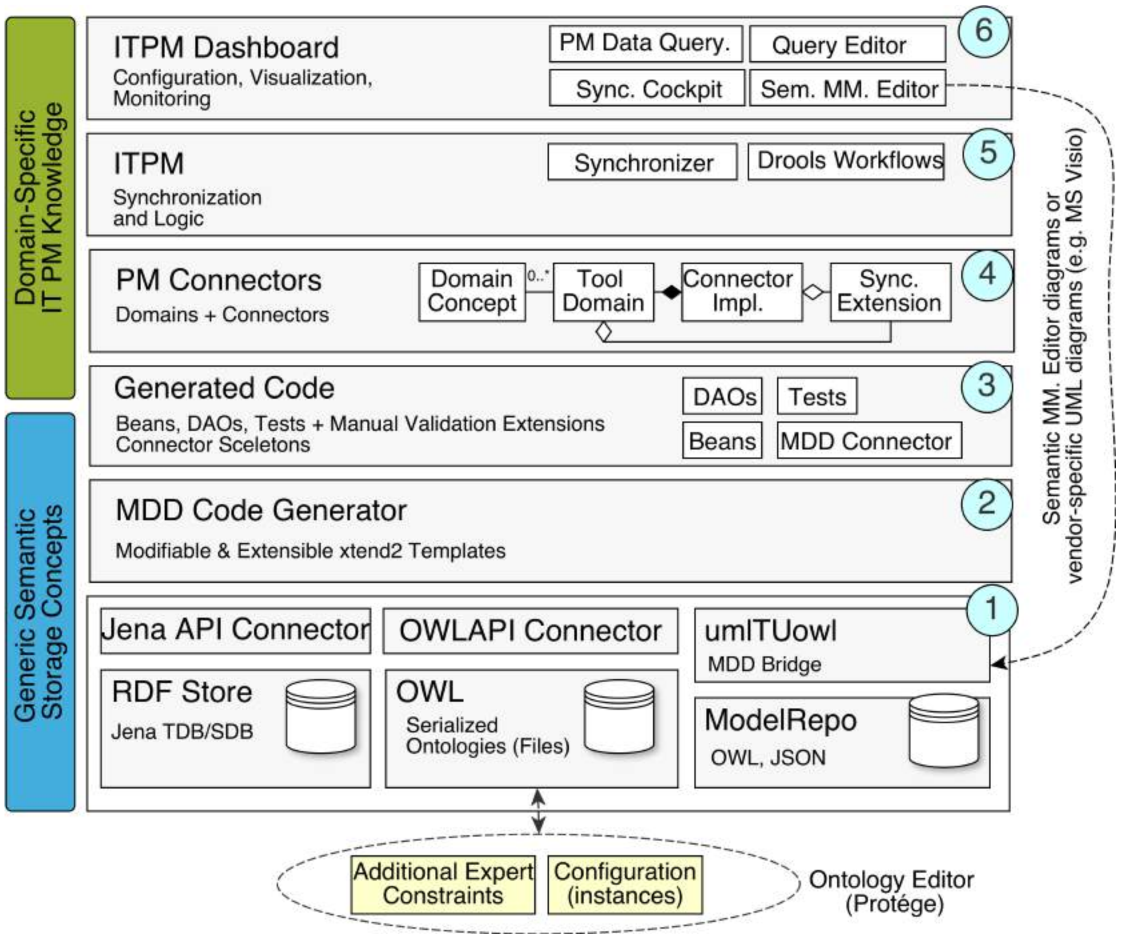
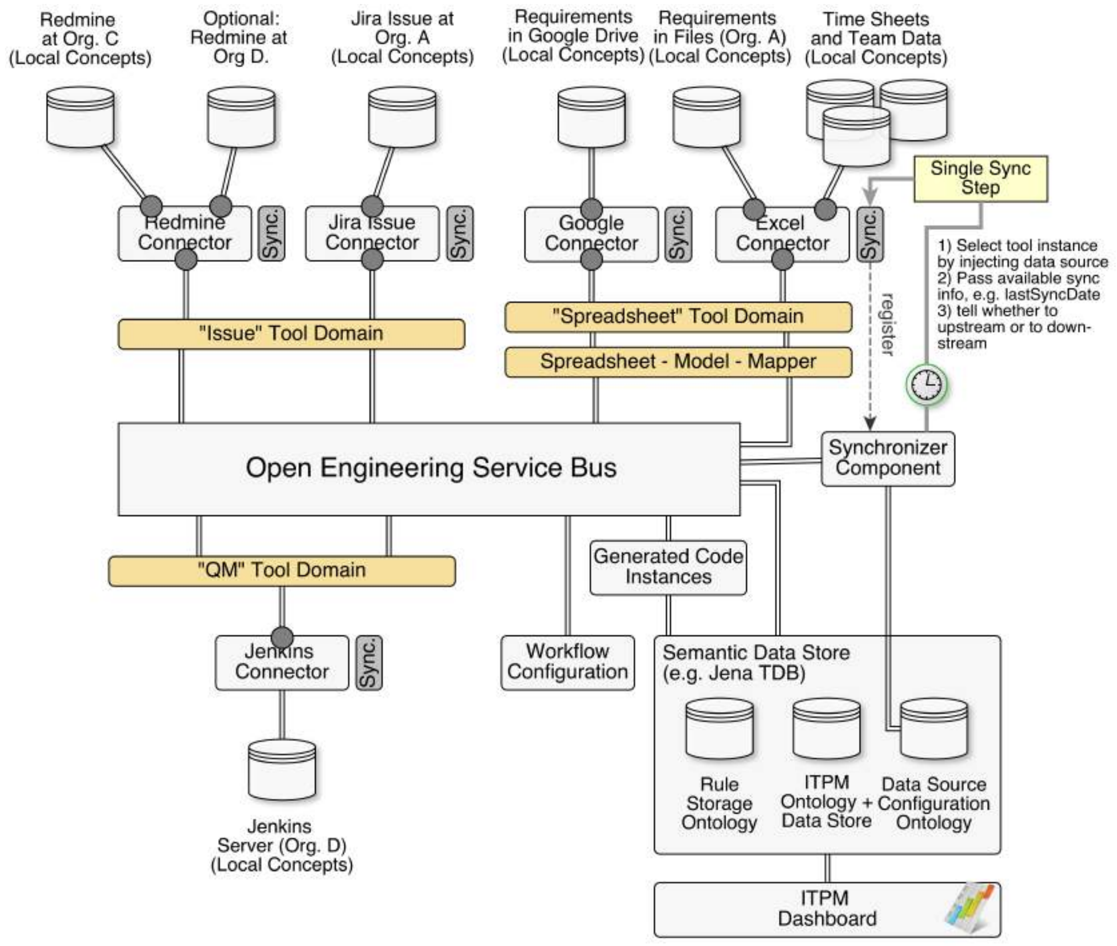

Domain-Specific IT PM Knowledge
Generic Semantic Storage Concepts
ITPM Dashboard
Configuration, Visualization, Monitoring
PM Data Query.
Query Editor
Sync. Cockpit
Sem. MM. Editor
6
ITPM
Synchronization and Logic
Synchronizer
Drools Workflows
5
PM Connectors
Domains + Connectors
Domain Concept
0..*
Tool Domain
Connector Impl.
Sync. Extension
4
Generated Code
Beans, DAOs, Tests + Manual Validation Extensions Connector Sceletons
DAOs
Tests
Beans
MDD Connector
3
MDD Code Generator
Modifiable & Extensible xtend2 Templates
2
Jena API Connector
OWLAPI Connector
umlTUowl
MDD Bridge
1
RDF Store
Jena TDB/SDB
OWL
Serialized Ontologies (Files)
ModelRepo
OWL, JSON
Semantic MM. Editor diagrams or vendor-specific UML diagrams (e.g. MS Visio)
Additional Expert Constraints
Configuration (instances)
Ontology Editor (Protége)

Figure A.4: The Architectural Software Layers

A.3.2 Component View

Figure A.5 summarizes the configuration structure for the UC and the OpenEngSb environment. The figure demonstrates the systematic integration approach of the local PMS and data sources, and the main (work-flow) configuration and synchronization components.

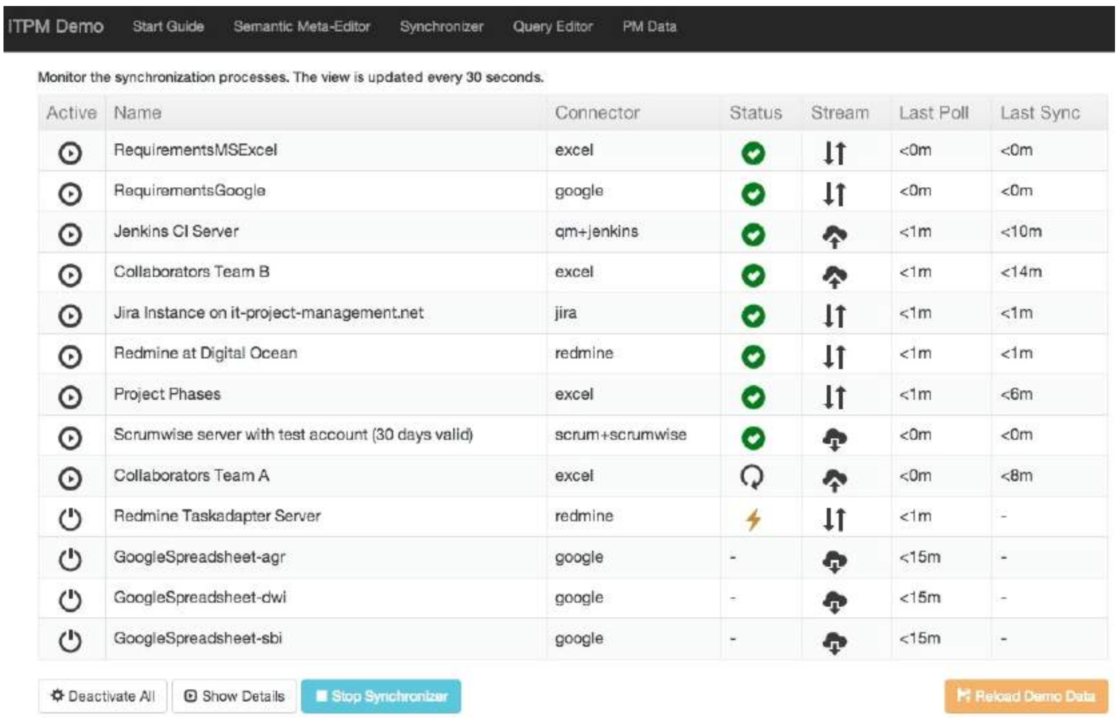


Redmine at Org. C (Local Concepts)
Optional: Redmine at Org D.
Jira Issue at Org. A (Local Concepts)
Requirements in Google Drive (Local Concepts)
Requirements in Files (Org. A) (Local Concepts)
Time Sheets and Team Data (Local Concepts)
Single Sync Step
Redmine Connector
Sync.
Jira Issue Connector
Google Connector
Excel Connector
1) Select tool instance by injecting data source
2) Pass available sync info, e.g. lastSyncDate
3) tell whether to upstream or to down-stream
register
"Issue" Tool Domain
"Spreadsheet" Tool Domain
Spreadsheet - Model - Mapper
Open Engineering Service Bus
Synchronizer Component
"QM" Tool Domain
Generated Code Instances
Jenkins Connector
Workflow Configuration
Semantic Data Store (e.g. Jena TDB)
Rule Storage Ontology
ITPM Ontology + Data Store
Data Source Configuration Ontology
Jenkins Server (Org. D) (Local Concepts)
ITPM Dashboard

Figure A.5: The architectural component view of the ITPM Dashboard / OpenEngSb for the use-case.

APPENDIX B

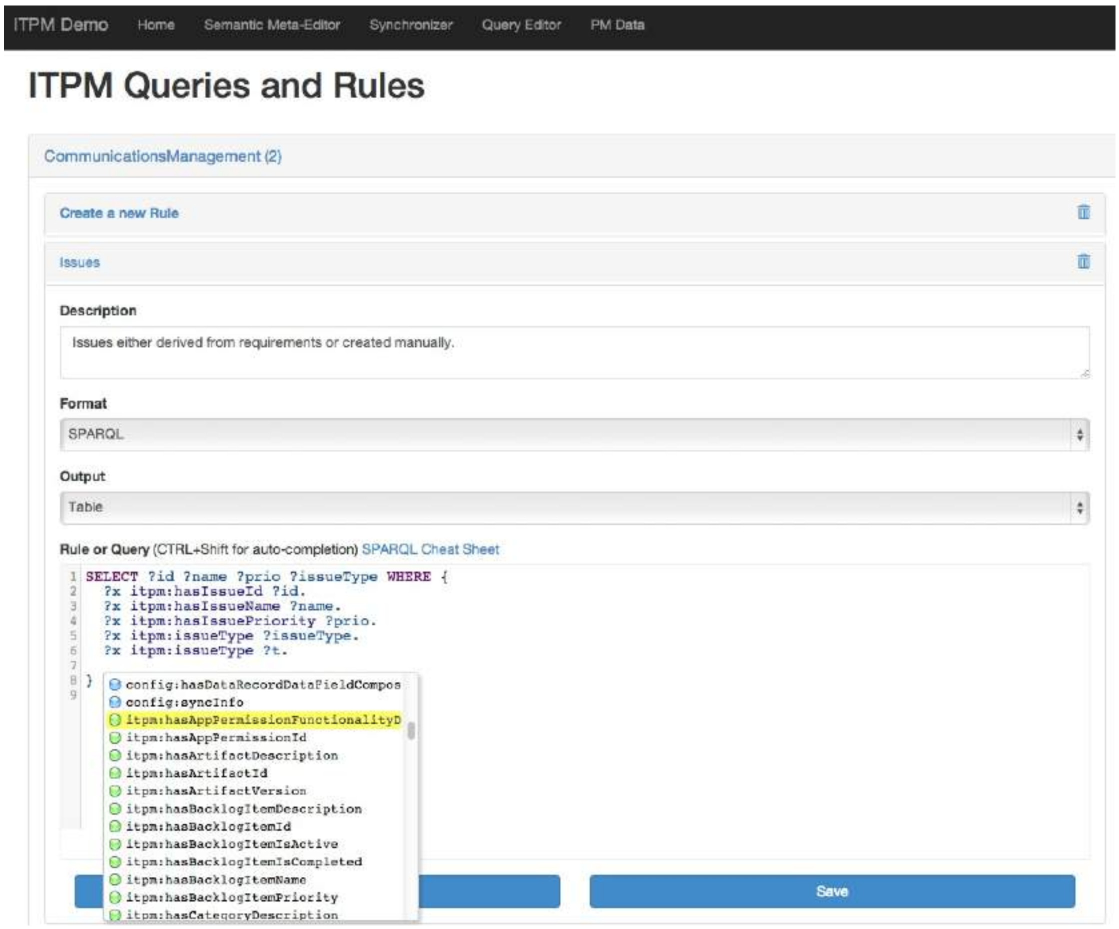
ITPM Dashboard UI



Monitor the synchronization processes. The view is updated every 30 seconds.

Active	Name	Connector	Status	Stream	Last Poll	Last Sync
▶	RequirementsMSExcel	excel	✓	⇅	<0m	<0m
▶	RequirementsGoogle	google	✓	⇅	<0m	<0m
▶	Jenkins CI Server	qm+jenkins	✓	↑	<1m	<10m
▶	Collaborators Team B	excel	✓	↑	<1m	<14m
▶	Jira Instance on it-project-management.net	jira	✓	⇅	<1m	<1m
▶	Redmine at Digital Ocean	redmine	✓	⇅	<1m	<1m
▶	Project Phases	excel	✓	⇅	<1m	<6m
▶	Scrumwise server with test account (30 days valid)	scrum+scrumwise	✓	↓	<0m	<0m
▶	Collaborators Team A	excel	↻	↑	<0m	<8m
⏻	Redmine Taskadapter Server	redmine	⚡	⇅	<1m	-
⏻	GoogleSpreadsheet-agr	google	-	↓	<15m	-
⏻	GoogleSpreadsheet-dwi	google	-	↓	<15m	-
⏻	GoogleSpreadsheet-sbi	google	-	↓	<15m	-

Figure B.1: Synchronizer Cockpit while active synchronization.



ITPM Queries and Rules

CommunicationsManagement (2)

Create a new Rule

Issues

Description

Issues either derived from requirements or created manually.

Format

SPARQL

Output

Table

Rule or Query (CTRL+Shift for auto-completion) SPARQL Cheat Sheet

```
SELECT ?id ?name ?prio ?issueType WHERE {
  ?x itpm:hasIssueId ?id.
  ?x itpm:hasIssueName ?name.
  ?x itpm:hasIssuePriority ?prio.
  ?x itpm:issueType ?issueType.
  ?x itpm:issueType ?t.

}
```

```
config:hasDataRecordDataFieldCompos
config:syncInfo
itpm:hasAppPermissionFunctionalityD
itpm:hasAppPermissionId
itpm:hasArtifactDescription
itpm:hasArtifactId
itpm:hasArtifactVersion
itpm:hasBacklogItemDescription
itpm:hasBacklogItemId
itpm:hasBacklogItemIsActive
itpm:hasBacklogItemIsCompleted
itpm:hasBacklogItemName
itpm:hasBacklogItemPriority
itpm:hasCategoryDescription
```

Figure B.2: The Semantic Query Editor. The editor supports syntax-highlighting and auto-completion for the provided ITPM Models.

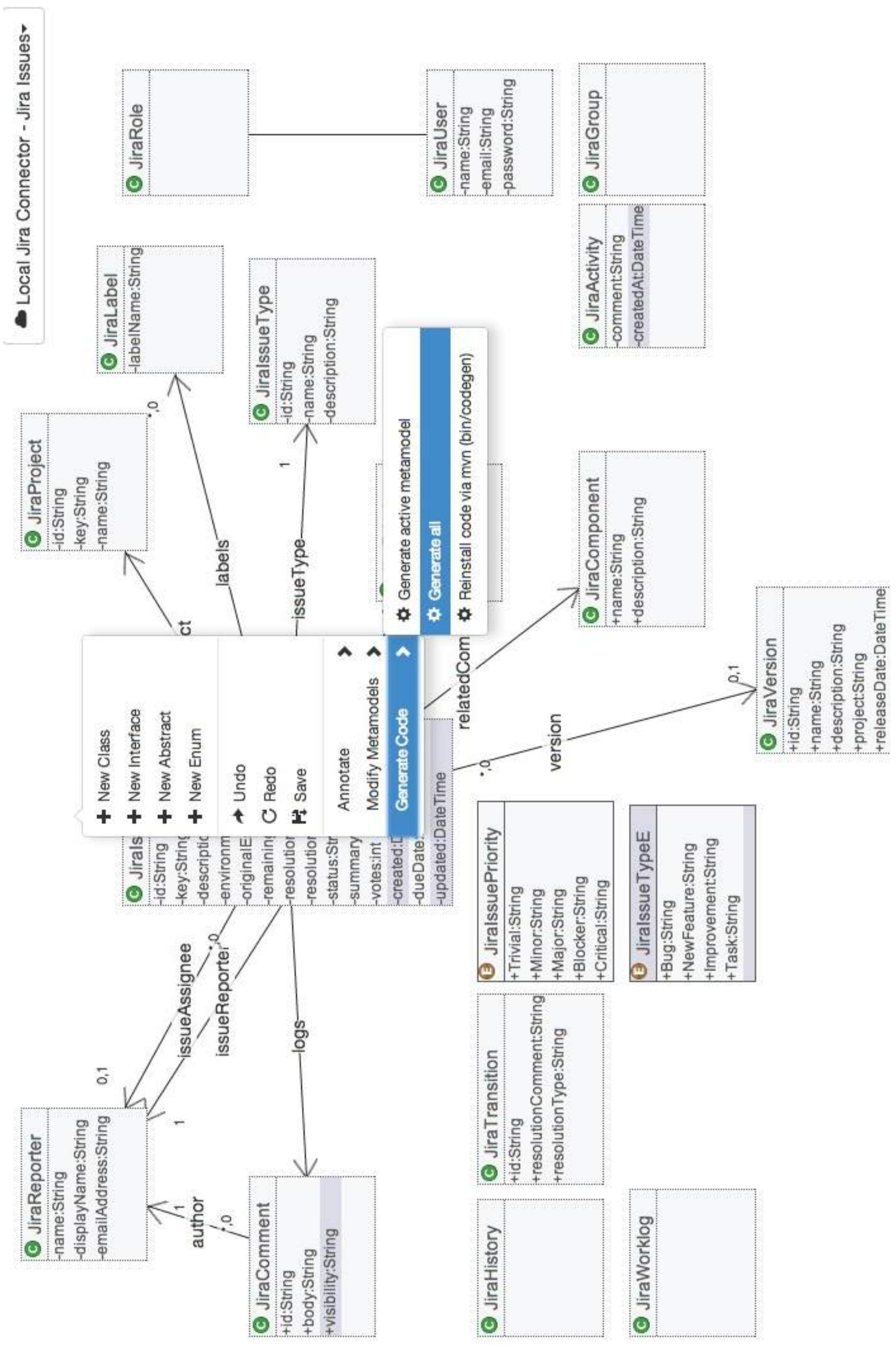
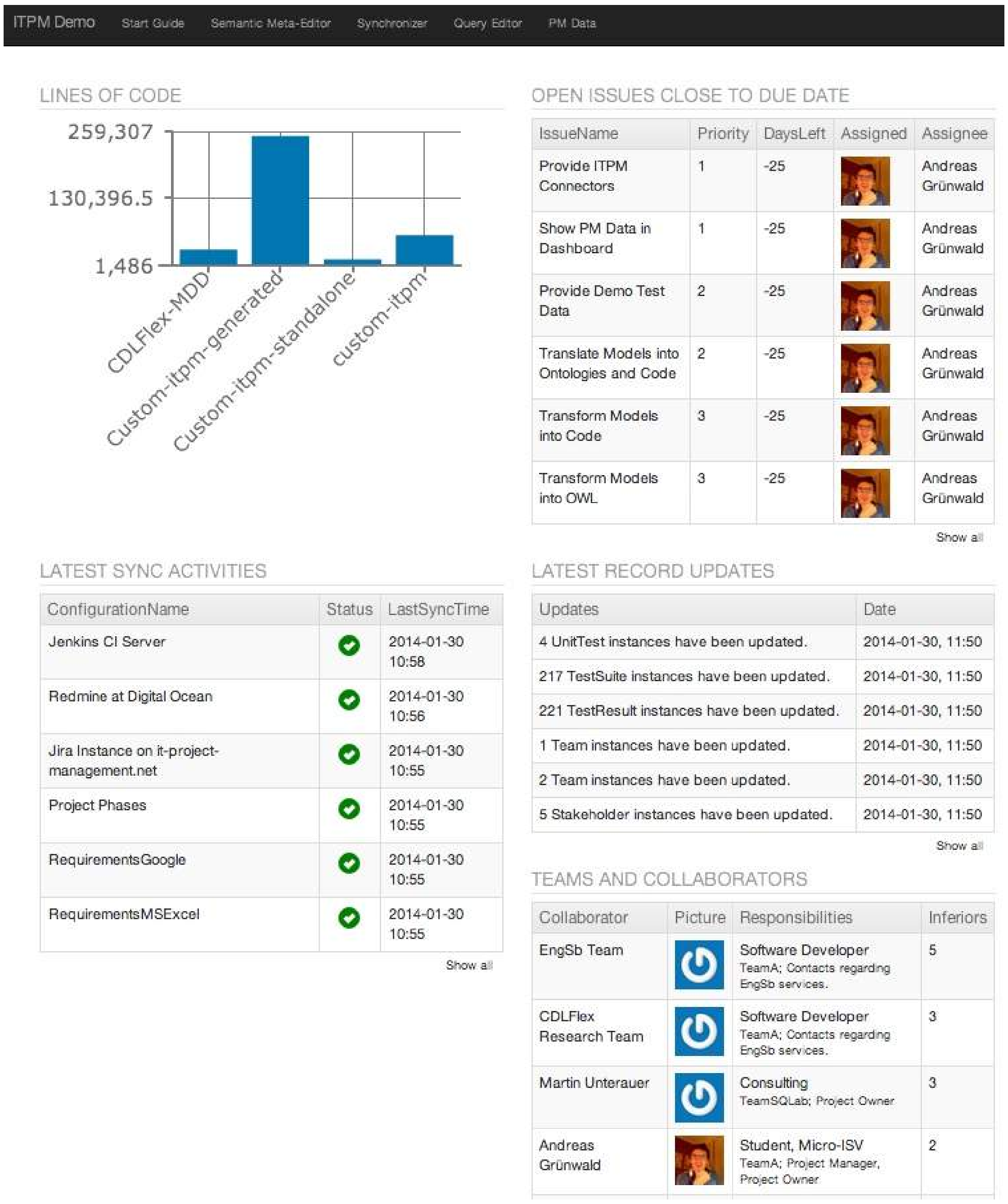


Figure B.3: The Semantic Model Editor. UML-alike modelling of ontologies, and on-the-fly source code generation.



OPEN ISSUES CLOSE TO DUE DATE

IssueName	Priority	DaysLeft	Assigned	Assignee
Provide ITPM Connectors	1	-25		Andreas Grünwald
Show PM Data in Dashboard	1	-25		Andreas Grünwald
Provide Demo Test Data	2	-25		Andreas Grünwald
Translate Models into Ontologies and Code	2	-25		Andreas Grünwald
Transform Models into Code	3	-25		Andreas Grünwald
Transform Models into OWL	3	-25		Andreas Grünwald

Show all

LATEST SYNC ACTIVITIES

ConfigurationName	Status	LastSyncTime
Jenkins CI Server	✓	2014-01-30 10:58
Redmine at Digital Ocean	✓	2014-01-30 10:56
Jira Instance on it-project-management.net	✓	2014-01-30 10:55
Project Phases	✓	2014-01-30 10:55
RequirementsGoogle	✓	2014-01-30 10:55
RequirementsMSExcel	✓	2014-01-30 10:55

Show all

LATEST RECORD UPDATES

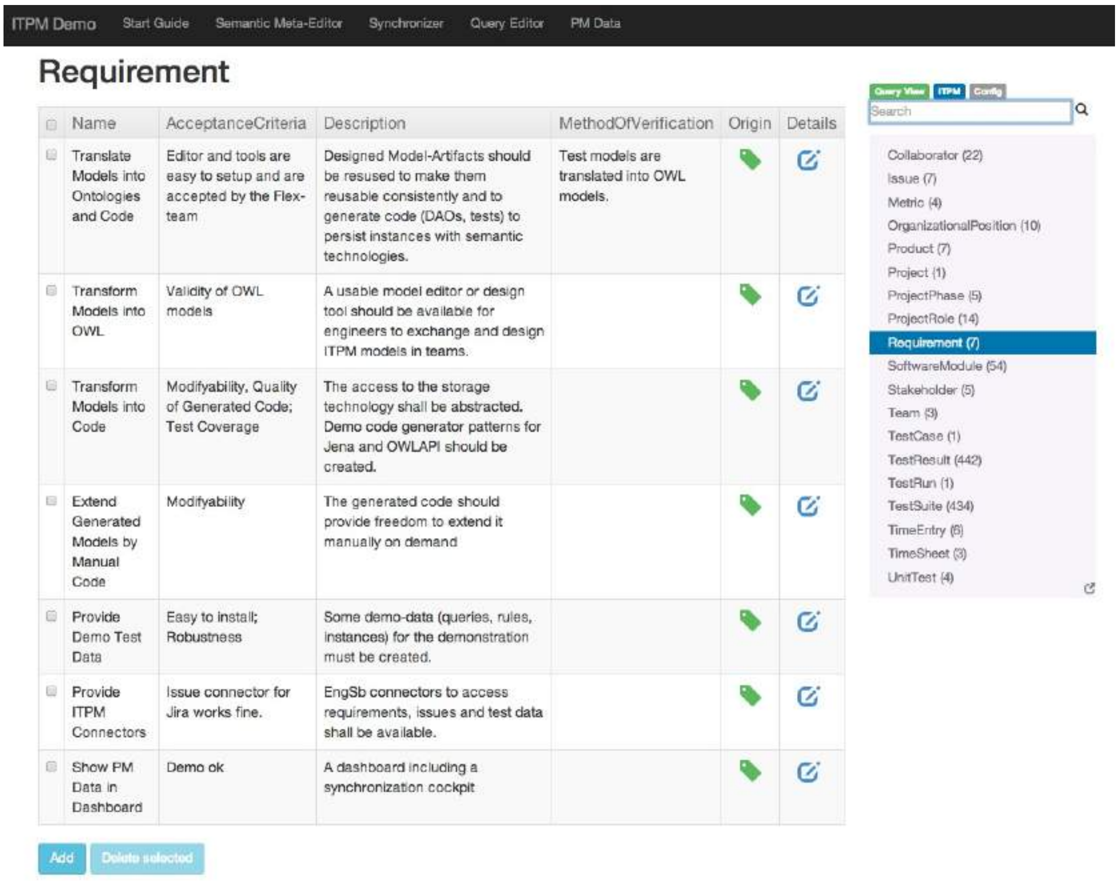
Updates	Date
4 UnitTest instances have been updated.	2014-01-30, 11:50
217 TestSuite instances have been updated.	2014-01-30, 11:50
221 TestResult instances have been updated.	2014-01-30, 11:50
1 Team instances have been updated.	2014-01-30, 11:50
2 Team instances have been updated.	2014-01-30, 11:50
5 Stakeholder instances have been updated.	2014-01-30, 11:50

Show all

TEAMS AND COLLABORATORS

Collaborator	Picture	Responsibilities	Inferiors
EngSb Team		Software Developer TeamA; Contacts regarding EngSb services.	5
CDLFlex Research Team		Software Developer TeamA; Contacts regarding EngSb services.	3
Martin Unterauer		Consulting TeamSQLab; Project Owner	3
Andreas Grünwald		Student, Micro-ISV TeamA; Project Manager, Project Owner	2

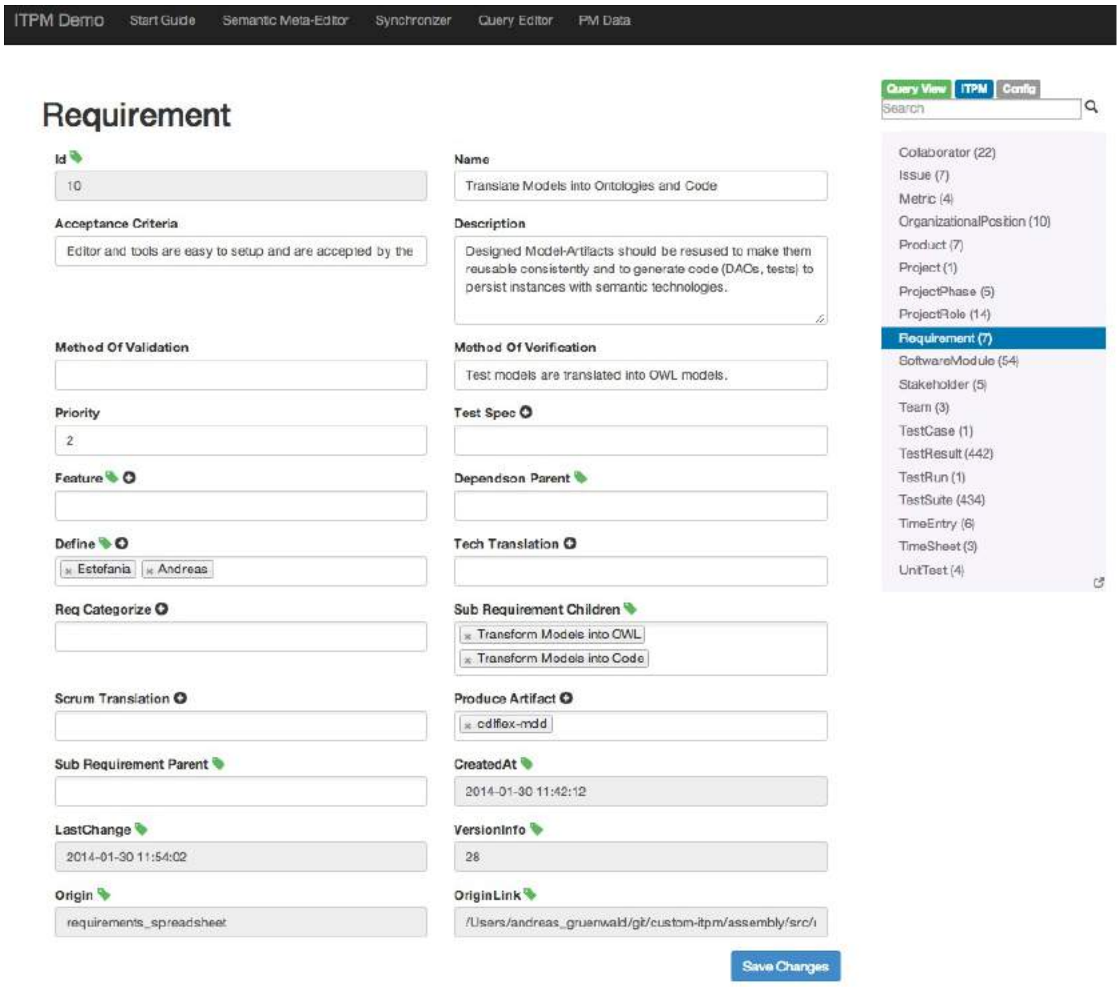
Figure B.4: Excerpt of the Central ITPM Dashboard Configuration: On the top-left corner the lines of code for the main software modules are visualized. Below, the last synchronization activities and stati are summarized. On the right-hand side, the open issues that are close (or exceed) the deadline, are enumerated. The responsible assignee is visualized (the picture is integrated from Gravatar.com). In the center of the right side, the last synchronization updates are summarized, grouped by their type based on ten-minute sequences. On the right bottom, for the project manager's convenience, the team members, their roles, and the number of direct inferiors are summarized. A click on any image will forward the project manager to the automatically generated overview of all collaborators, including their detailed contact information (e.g., emails). All of the dashboard elements are based on configurable SPARQL queries.



Requirement

	Name	AcceptanceCriteria	Description	MethodOfVerification	Origin	Details
☐	Translate Models into Ontologies and Code	Editor and tools are easy to setup and are accepted by the Flex-team	Designed Model-Artifacts should be resused to make them reusable consistently and to generate code (DAOs, tests) to persist instances with semantic technologies.	Test models are translated into OWL models.		
☐	Transform Models into OWL	Validity of OWL models	A usable model editor or design tool should be available for engineers to exchange and design ITPM models in teams.			
☐	Transform Models into Code	Modifyability, Quality of Generated Code; Test Coverage	The access to the storage technology shall be abstracted. Demo code generator patterns for Jena and OWLAPI should be created.			
☐	Extend Generated Models by Manual Code	Modifyability	The generated code should provide freedom to extend it manually on demand			
☐	Provide Demo Test Data	Easy to install; Robustness	Some demo-data (queries, rules, instances) for the demonstration must be created.			
☐	Provide ITPM Connectors	Issue connector for Jira works fine.	EngSb connectors to access requirements, issues and test data shall be available.			
☐	Show PM Data in Dashboard	Demo ok	A dashboard including a synchronization cockpit			

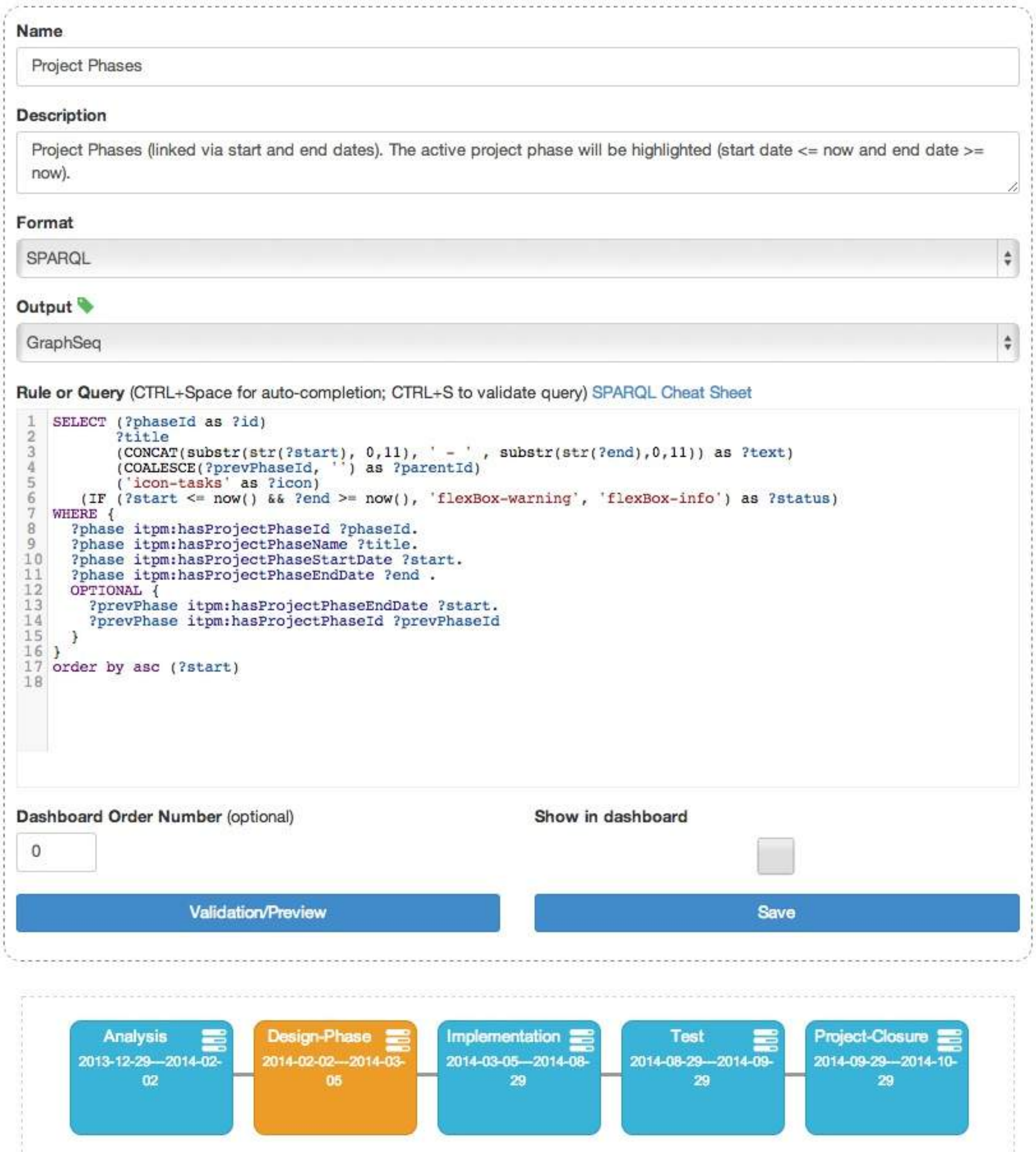
Figure B.5: Overview on Integrated Requirements Data. The view is automatically derived based on the ITPM models, and the MDSE-approach.



Requirement

Id	10	Name	Translate Models into Ontologies and Code
Acceptance Criteria	Editor and tools are easy to setup and are accepted by the	Description	Designed Model-Artifacts should be resused to make them reusable consistently and to generate code (DAOs, tests) to persist instances with semantic technologies.
Method Of Validation		Method Of Verification	Test models are translated into OWL models.
Priority	2	Test Spec	
Feature		Dependson Parent	
Define	Estefania, Andreas	Tech Translation	
Req Categorize		Sub Requirement Children	Transform Models into OWL, Transform Models into Code
Scrum Translation		Produce Artifact	cdlflex-mdd
Sub Requirement Parent		CreatedAt	2014-01-30 11:42:12
LastChange	2014-01-30 11:54:02	VersionInfo	28
Origin	requirements_spreadsheet	OriginLink	/Users/andreas_gruenwald/git/custom-itpm/assembly/src/i

Figure B.6: All ITPM model instances can be edited on-the-fly. The forms are derived based on the ITPM models, and the MDSE-approach. The linkage of nested entities is supported (in that case: sub-requirements, stakeholders, and artifacts).



Name

Project Phases

Description

Project Phases (linked via start and end dates). The active project phase will be highlighted (start date <= now and end date >= now).

Format

SPARQL

Output

GraphSeq

Rule or Query (CTRL+Space for auto-completion; CTRL+S to validate query) SPARQL Cheat Sheet

```
SELECT (?phaseId as ?id)
       ?title
       (CONCAT(substr(str(?start), 0,11), ' - ' , substr(str(?end),0,11)) as ?text)
       (COALESCE(?prevPhaseId, '') as ?parentId)
       ('icon-tasks' as ?icon)
   (IF (?start <= now() && ?end >= now(), 'flexBox-warning', 'flexBox-info') as ?status)
WHERE {
  ?phase itpm:hasProjectPhaseId ?phaseId.
  ?phase itpm:hasProjectPhaseName ?title.
  ?phase itpm:hasProjectPhaseStartDate ?start.
  ?phase itpm:hasProjectPhaseEndDate ?end .
  OPTIONAL {
    ?prevPhase itpm:hasProjectPhaseEndDate ?start.
    ?prevPhase itpm:hasProjectPhaseId ?prevPhaseId
  }
}
order by asc (?start)

```

Dashboard Order Number (optional)

0

Show in dashboard

Figure B.7: Visualization of the Project Phases (extracted from Excel) based on a simple SPARQL Query and the ITPM Dashboard's Graph Representation Capabilities.

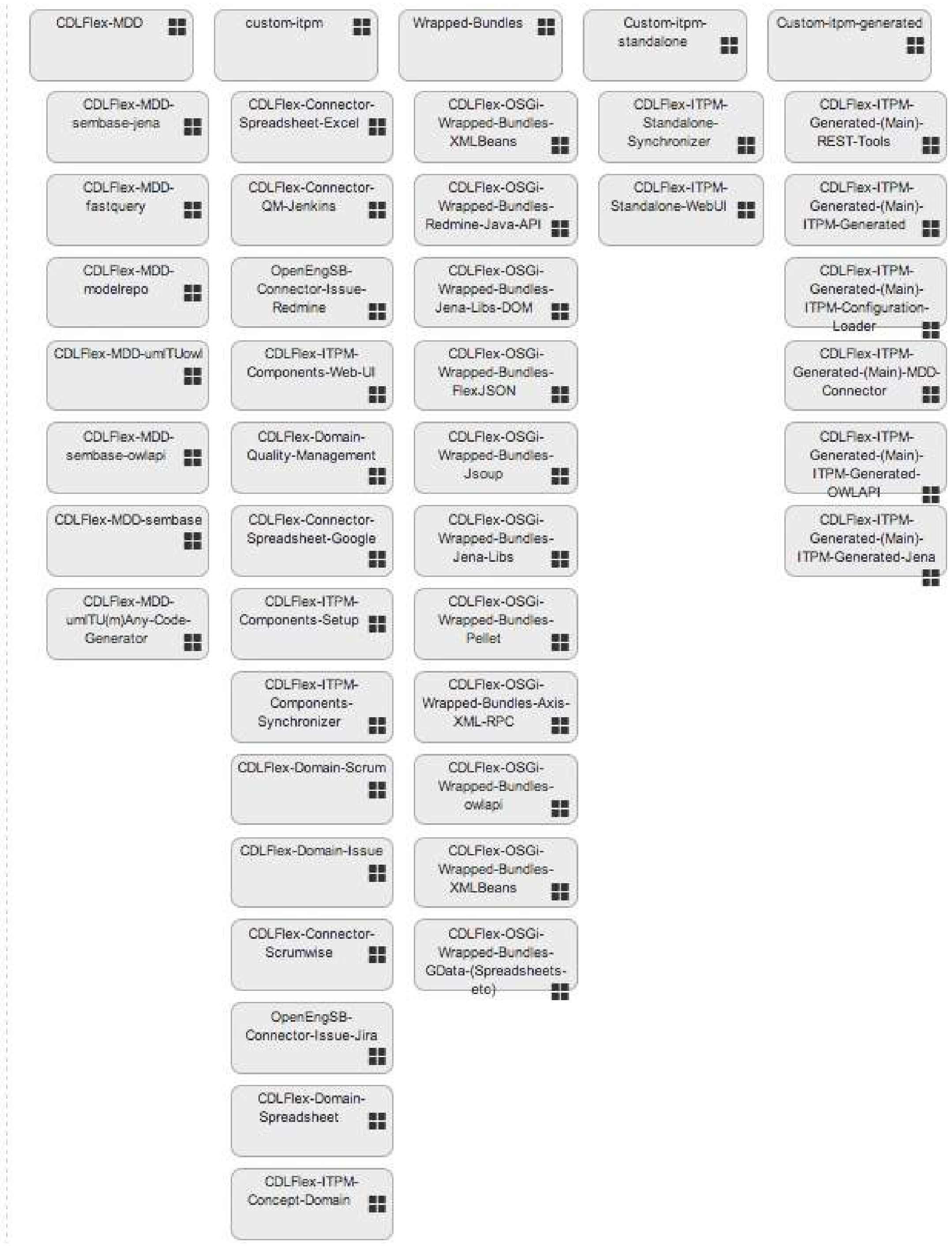

CDLFlex-MDD
CDLFlex-MDD-sembase-jena
CDLFlex-MDD-fastquery
CDLFlex-MDD-modelrepo
CDLFlex-MDD-umlTUowl
CDLFlex-MDD-sembase-owlapi
CDLFlex-MDD-sembase
CDLFlex-MDD-umlTU(m)Any-Code-Generator
custom-itpm
CDLFlex-Connector-Spreadsheet-Excel
CDLFlex-Connector-QM-Jenkins
OpenEngSB-Connector-Issue-Redmine
CDLFlex-ITPM-Components-Web-UI
CDLFlex-Domain-Quality-Management
CDLFlex-Connector-Spreadsheet-Google
CDLFlex-ITPM-Components-Setup
CDLFlex-ITPM-Components-Synchronizer
CDLFlex-Domain-Scrum
CDLFlex-Domain-Issue
CDLFlex-Connector-Scrumwise
OpenEngSB-Connector-Issue-Jira
CDLFlex-Domain-Spreadsheet
CDLFlex-ITPM-Concept-Domain
Wrapped-Bundles
CDLFlex-OSGi-Wrapped-Bundles-XMLBeans
CDLFlex-OSGi-Wrapped-Bundles-Redmine-Java-API
CDLFlex-OSGi-Wrapped-Bundles-Jena-Libs-DOM
CDLFlex-OSGi-Wrapped-Bundles-FlexJSON
CDLFlex-OSGi-Wrapped-Bundles-Jsoup
CDLFlex-OSGi-Wrapped-Bundles-Jena-Libs
CDLFlex-OSGi-Wrapped-Bundles-Pellet
CDLFlex-OSGi-Wrapped-Bundles-Axis-XML-RPC
CDLFlex-OSGi-Wrapped-Bundles-owlapi
CDLFlex-OSGi-Wrapped-Bundles-XMLBeans
CDLFlex-OSGi-Wrapped-Bundles-GData-(Spreadsheets-etc)
Custom-itpm-standalone
CDLFlex-ITPM-Standalone-Synchronizer
CDLFlex-ITPM-Standalone-WebUI
Custom-itpm-generated
CDLFlex-ITPM-Generated-(Main)-REST-Tools
CDLFlex-ITPM-Generated-(Main)-ITPM-Generated
CDLFlex-ITPM-Generated-(Main)-ITPM-Configuration-Loader
CDLFlex-ITPM-Generated-(Main)-MDD-Connector
CDLFlex-ITPM-Generated-(Main)-ITPM-Generated-OWLAPI
CDLFlex-ITPM-Generated-(Main)-ITPM-Generated-Jena

Figure B.8: The main software packages and their submodules, extracted from the build server, are visualized based on a SPARQL query and the ITPM Dashboards's graph visualization capabilities.

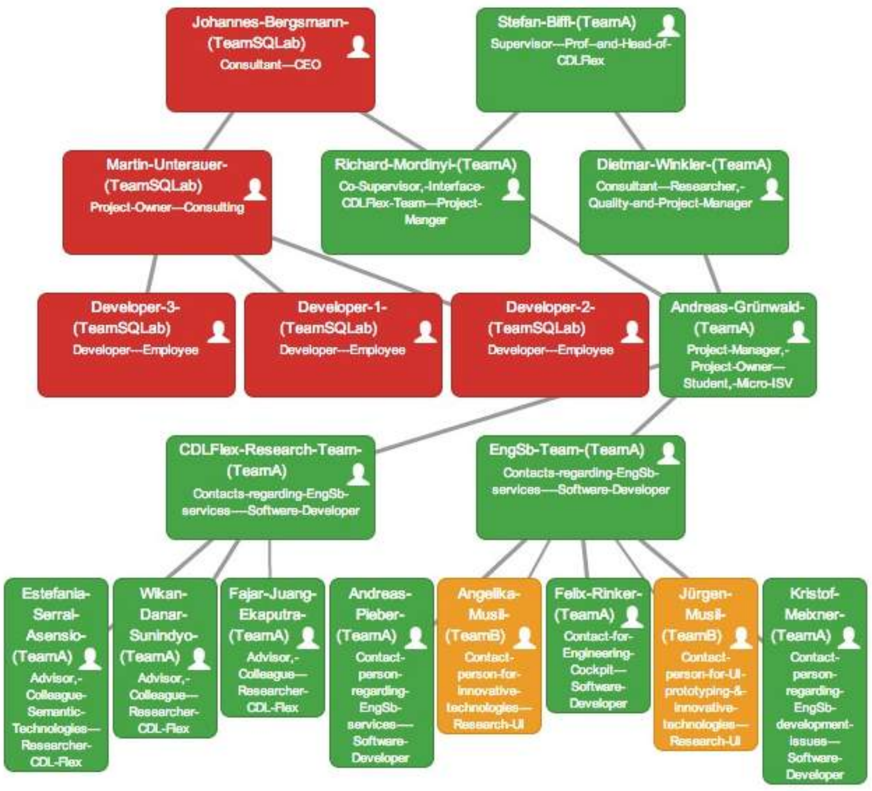
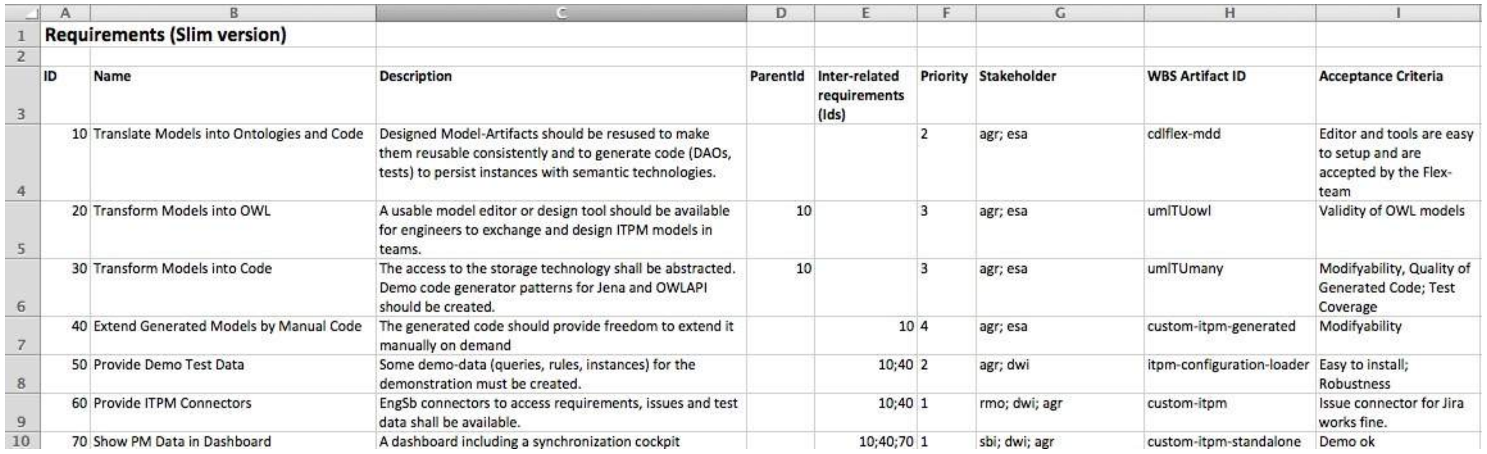

Johannes-Bergsmann-(TeamSQLab) Consultant—CEO
Stefan-Biffl-(TeamA) Supervisor---Prof--and-Head-of-CDLFlex
Martin-Unterauer-(TeamSQLab) Project-Owner—Consulting
Richard-Mordinyi-(TeamA) Co-Supervisor,-Interface-CDLFlex-Team—Project-Manger
Dietmar-Winkler-(TeamA) Consultant—Researcher,-Quality-and-Project-Manager
Developer-3-(TeamSQLab) Developer---Employee
Developer-1-(TeamSQLab) Developer---Employee
Developer-2-(TeamSQLab) Developer---Employee
Andreas-Grünwald-(TeamA) Project-Manager,-Project-Owner—Student,-Micro-ISV
CDLFlex-Research-Team-(TeamA) Contacts-regarding-EngSb-services----Software-Developer
EngSb-Team-(TeamA) Contacts-regarding-EngSb-services----Software-Developer
Estefania-Serral-Asensio-(TeamA) Advisor,-Colleague-Semantic-Technologies—Researcher-CDL-Flex
Wikan-Danar-Sunindyo-(TeamA) Advisor,-Colleague—Researcher-CDL-Flex
Fajar-Juang-Ekaputra-(TeamA) Advisor,-Colleague—Researcher-CDL-Flex
Andreas-Pieber-(TeamA) Contact-person-regarding-EngSb-services----Software-Developer
Angelika-Musil-(TeamB) Contact-person-for-innovative-technologies—Research-UI
Felix-Rinker-(TeamA) Contact-for-Engineering-Cockpit—Software-Developer
Jürgen-Musil-(TeamB) Contact-person-for-UI-prototyping-&-innovative-technologies—Research-UI
Kristof-Meixner-(TeamA) Contact-person-regarding-EngSb-development-issues—Software-Developer

Figure B.9: The Team Hierarchy, extracted from several external spreadsheets, visualized as a graph, based on the ITPM Dashboard's visualization capabilities. The different colors denote the team memberships.

APPENDIX C

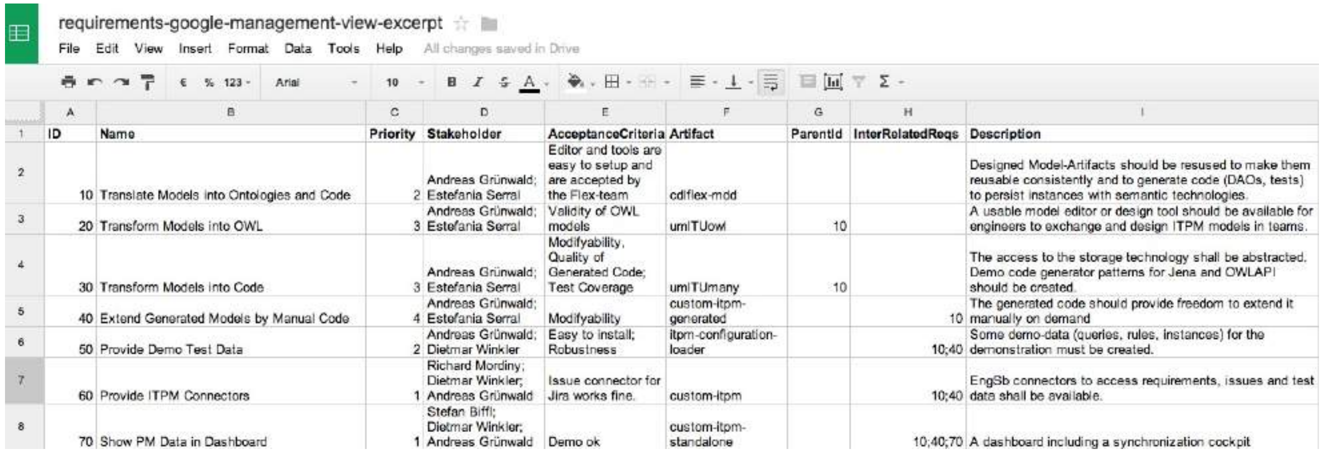
Local PM Tools and Demo Data



Requirements (Slim version)

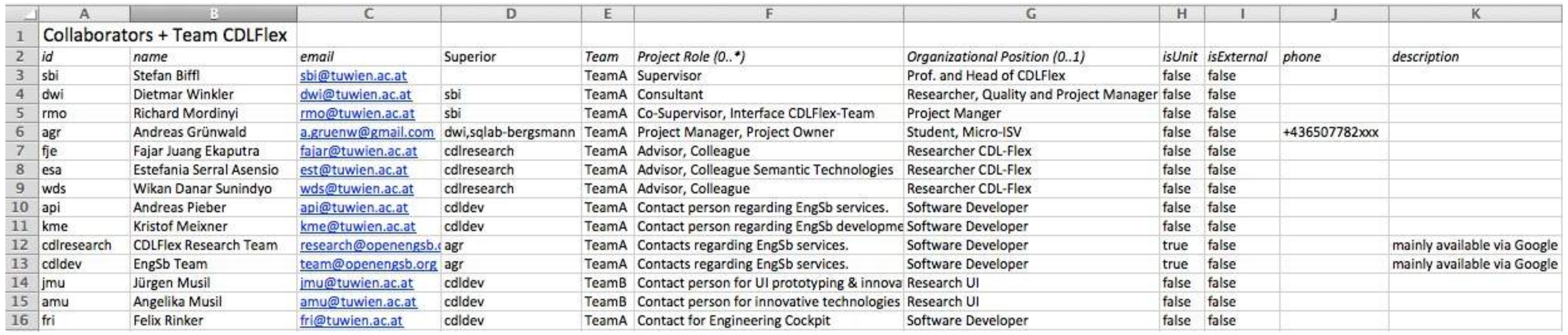
ID	Name	Description	ParentId	Inter-related requirements (Ids)	Priority	Stakeholder	WBS Artifact ID	Acceptance Criteria
10	Translate Models into Ontologies and Code	Designed Model-Artifacts should be resused to make them reusable consistently and to generate code (DAOs, tests) to persist instances with semantic technologies.			2	agr; esa	cdlflex-mdd	Editor and tools are easy to setup and are accepted by the Flex-team
20	Transform Models into OWL	A usable model editor or design tool should be available for engineers to exchange and design ITPM models in teams.	10		3	agr; esa	umlTUowl	Validity of OWL models
30	Transform Models into Code	The access to the storage technology shall be abstracted. Demo code generator patterns for Jena and OWLAPI should be created.	10		3	agr; esa	umlTUmany	Modifyability, Quality of Generated Code; Test Coverage
40	Extend Generated Models by Manual Code	The generated code should provide freedom to extend it manually on demand		10	4	agr; esa	custom-itpm-generated	Modifyability
50	Provide Demo Test Data	Some demo-data (queries, rules, instances) for the demonstration must be created.		10;40	2	agr; dwi	itpm-configuration-loader	Easy to install; Robustness
60	Provide ITPM Connectors	EngSb connectors to access requirements, issues and test data shall be available.		10;40	1	rmo; dwi; agr	custom-itpm	Issue connector for Jira works fine.
70	Show PM Data in Dashboard	A dashboard including a synchronization cockpit		10;40;70	1	sbi; dwi; agr	custom-itpm-standalone	Demo ok

Figure C.1: Excerpt of requirements test data, based on Microsoft Excel.


requirements-google-management-view-excerpt
File Edit View Insert Format Data Tools Help All changes saved in Drive

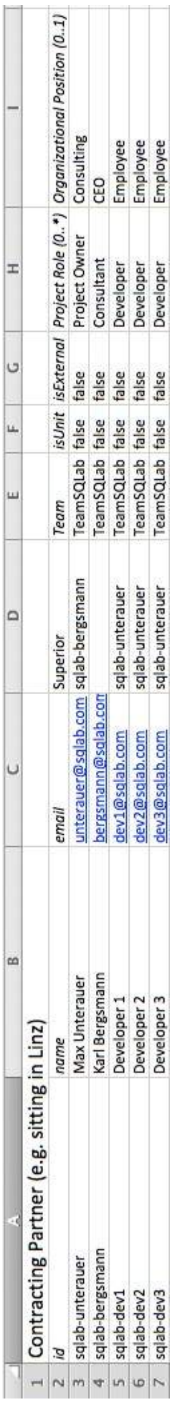
ID	Name	Priority	Stakeholder	AcceptanceCriteria	Artifact	ParentId	InterRelatedReqs	Description
10	Translate Models into Ontologies and Code	2	Andreas Grünwald; Estefania Serral	Editor and tools are easy to setup and are accepted by the Flex-team	cdiflex-mdd			Designed Model-Artifacts should be resused to make them reusable consistently and to generate code (DAOs, tests) to persist instances with semantic technologies.
20	Transform Models into OWL	3	Andreas Grünwald; Estefania Serral	Validity of OWL models	umlTUowl	10		A usable model editor or design tool should be available for engineers to exchange and design ITPM models in teams.
30	Transform Models into Code	3	Andreas Grünwald; Estefania Serral	Modifyability, Quality of Generated Code; Test Coverage	umlTUmany	10		The access to the storage technology shall be abstracted. Demo code generator patterns for Jena and OWLAPI should be created.
40	Extend Generated Models by Manual Code	4	Andreas Grünwald; Estefania Serral	Modifyability	custom-itpm-generated		10	The generated code should provide freedom to extend it manually on demand
50	Provide Demo Test Data	2	Andreas Grünwald; Dietmar Winkler	Easy to install; Robustness	itpm-configuration-loader		10;40	Some demo-data (queries, rules, instances) for the demonstration must be created.
60	Provide ITPM Connectors	1	Richard Mordiny; Dietmar Winkler; Andreas Grünwald	Issue connector for Jira works fine.	custom-itpm		10;40	EngSb connectors to access requirements, issues and test data shall be available.
70	Show PM Data in Dashboard	1	Stefan Biffl; Dietmar Winkler; Andreas Grünwald	Demo ok	custom-itpm-standalone		10;40;70	A dashboard including a synchronization cockpit

Figure C.2: Excerpt of requirements test data in Google Spreadsheet. The requirements are the same as in figure C.1, but the format and the structure is adapted to meet the requirements of the management.



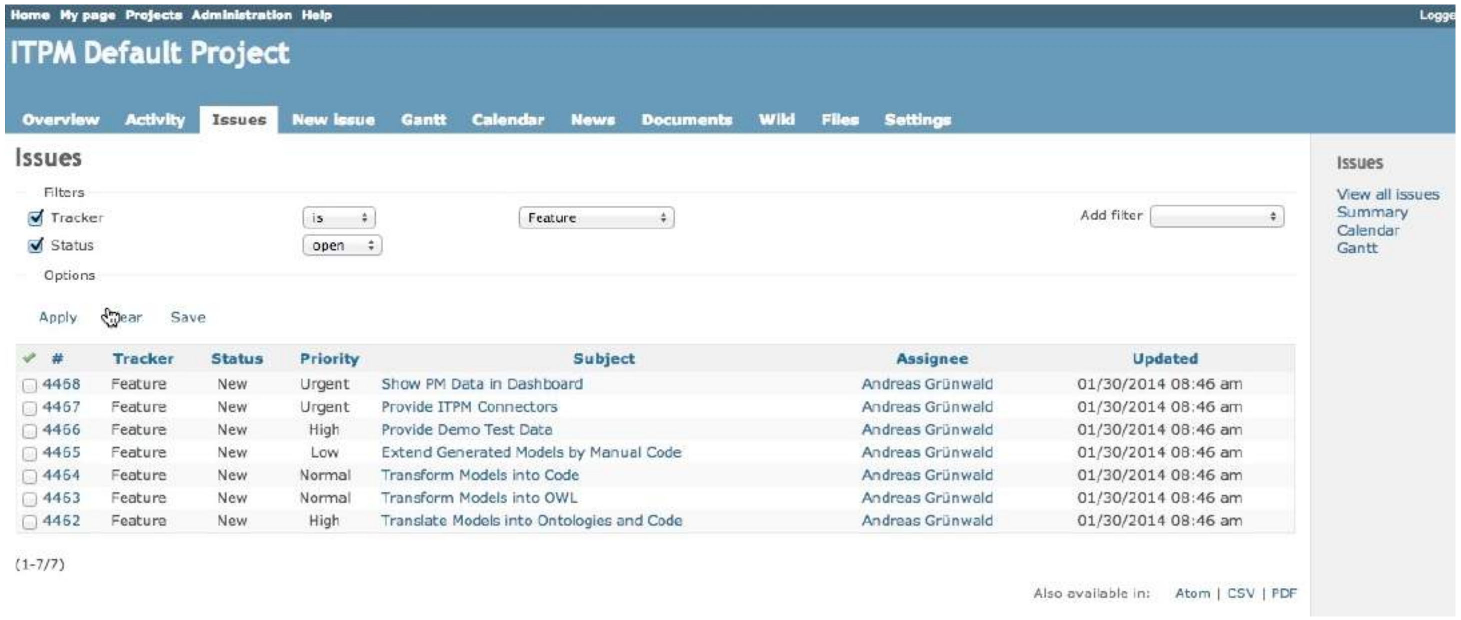
	A	B	C	D	E	F	G	H	I	J	K
1	Collaborators + Team CDLFlex										
2	*id*	*name*	*email*	Superior	*Team*	*Project Role (0..*)*	*Organizational Position (0..1)*	*isUnit*	*isExternal*	*phone*	*description*
3	sbi	Stefan Biffl	sbi@tuwien.ac.at		TeamA	Supervisor	Prof. and Head of CDLFlex	false	false		
4	dwi	Dietmar Winkler	dwi@tuwien.ac.at	sbi	TeamA	Consultant	Researcher, Quality and Project Manager	false	false		
5	rmo	Richard Mordinyi	rmo@tuwien.ac.at	sbi	TeamA	Co-Supervisor, Interface CDLFlex-Team	Project Manger	false	false		
6	agr	Andreas Grünwald	a.gruenw@gmail.com	dwi,sqlab-bergsmann	TeamA	Project Manager, Project Owner	Student, Micro-ISV	false	false	+436507782xxx	
7	fje	Fajar Juang Ekaputra	fajar@tuwien.ac.at	cdlresearch	TeamA	Advisor, Colleague	Researcher CDL-Flex	false	false		
8	esa	Estefania Serral Asensio	est@tuwien.ac.at	cdlresearch	TeamA	Advisor, Colleague Semantic Technologies	Researcher CDL-Flex	false	false		
9	wds	Wikan Danar Sunindyo	wds@tuwien.ac.at	cdlresearch	TeamA	Advisor, Colleague	Researcher CDL-Flex	false	false		
10	api	Andreas Pieber	api@tuwien.ac.at	cdldev	TeamA	Contact person regarding EngSb services.	Software Developer	false	false		
11	kme	Kristof Meixner	kme@tuwien.ac.at	cdldev	TeamA	Contact person regarding EngSb developme	Software Developer	false	false		
12	cdlresearch	CDLFlex Research Team	research@openengsb.	agr	TeamA	Contacts regarding EngSb services.	Software Developer	true	false		mainly available via Google
13	cdldev	EngSb Team	team@openengsb.org	agr	TeamA	Contacts regarding EngSb services.	Software Developer	true	false		mainly available via Google
14	jmu	Jürgen Musil	jmu@tuwien.ac.at	cdldev	TeamB	Contact person for UI prototyping & innova	Research UI	false	false		
15	amu	Angelika Musil	amu@tuwien.ac.at	cdldev	TeamB	Contact person for innovative technologies	Research UI	false	false		
16	fri	Felix Rinker	fri@tuwien.ac.at	cdldev	TeamA	Contact for Engineering Cockpit	Software Developer	false	false		

Figure C.3: Excerpt of project team data and hierarchy (MS Excel).



	A	B	C	D	E	F	G	H	I
1	Contracting Partner (e.g. sitting in Linz)								
2	*id*	*name*	*email*	Superior	*Team*	*isUnit*	*isExternal*	*Project Role (0..*)*	*Organizational Position (0..1)*
3	sqlab-unterauer	Max Unterauer	unterauer@sqlab.com	sqlab-bergsmann	TeamSQLab	false	false	Project Owner	Consulting
4	sqlab-bergsmann	Karl Bergsmann	bergsmann@sqlab.com		TeamSQLab	false	false	Consultant	CEO
5	sqlab-dev1	Developer 1	dev1@sqlab.com	sqlab-unterauer	TeamSQLab	false	false	Developer	Employee
6	sqlab-dev2	Developer 2	dev2@sqlab.com	sqlab-unterauer	TeamSQLab	false	false	Developer	Employee
7	sqlab-dev3	Developer 3	dev3@sqlab.com	sqlab-unterauer	TeamSQLab	false	false	Developer	Employee

Figure C.4: Excerpt of a contract partner's team structure (MS Excel). Note that figure C.3 references a manager of the contract partner (sqlab-bergsmann). The ITPM data integration framework is capable to resolve this dependency.


Home My page Projects Administration Help
Logge
ITPM Default Project
Overview Activity Issues New issue Gantt Calendar News Documents Wiki Files Settings
Issues
Filters
Options
Apply Clear Save
Issues
View all issues
Summary
Calendar
Gantt

Filter	Operator	Value
✓ Tracker	is	Feature
✓ Status	open	

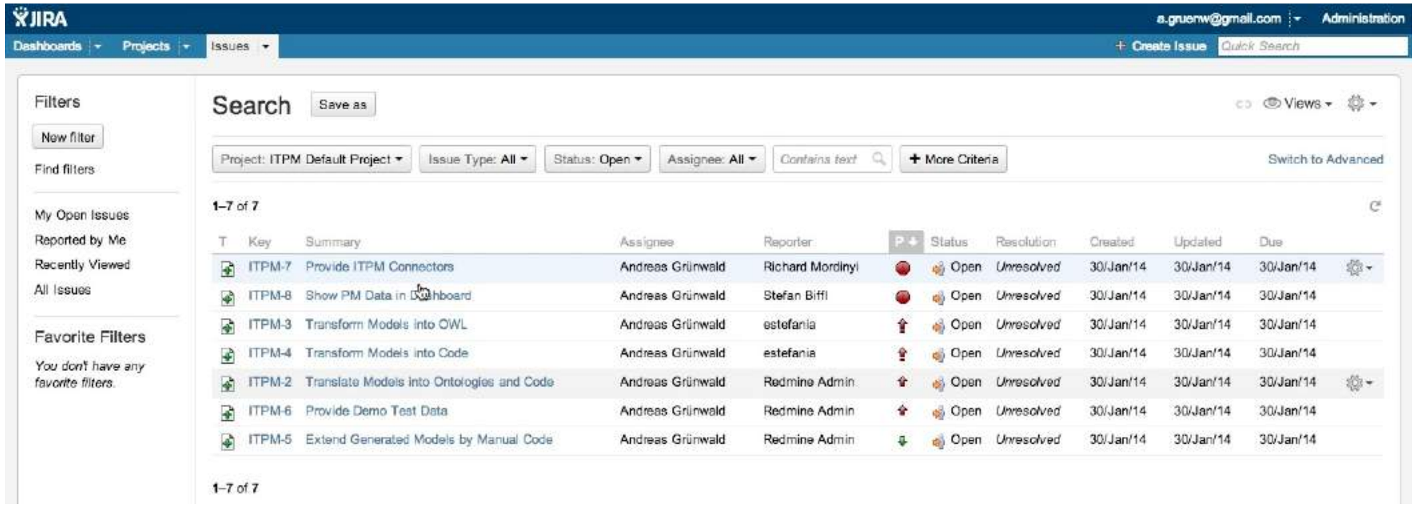
Add filter

#	Tracker	Status	Priority	Subject	Assignee	Updated
4468	Feature	New	Urgent	Show PM Data in Dashboard	Andreas Grünwald	01/30/2014 08:46 am
4467	Feature	New	Urgent	Provide ITPM Connectors	Andreas Grünwald	01/30/2014 08:46 am
4466	Feature	New	High	Provide Demo Test Data	Andreas Grünwald	01/30/2014 08:46 am
4465	Feature	New	Low	Extend Generated Models by Manual Code	Andreas Grünwald	01/30/2014 08:46 am
4464	Feature	New	Normal	Transform Models into Code	Andreas Grünwald	01/30/2014 08:46 am
4463	Feature	New	Normal	Transform Models into OWL	Andreas Grünwald	01/30/2014 08:46 am
4462	Feature	New	High	Translate Models into Ontologies and Code	Andreas Grünwald	01/30/2014 08:46 am

(1-7/7)

Also available in: Atom | CSV | PDF

Figure C.5: Redmine Project Management overview. Note that the issues correspond with the requirements of the figures C.1 and C.2 (they have been derived based on a work-flow).

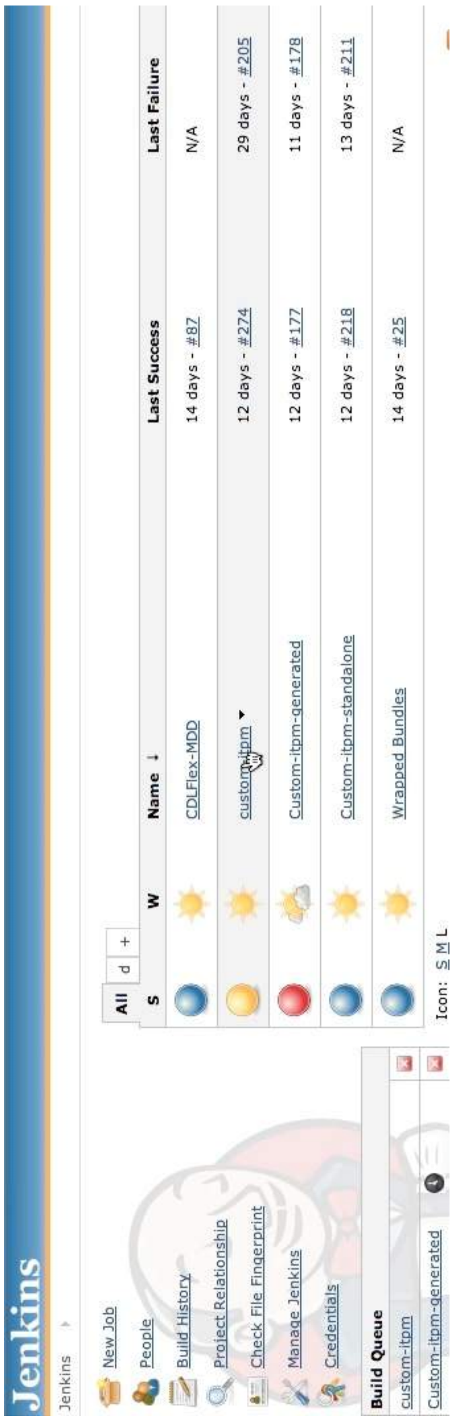


1–7 of 7

T	Key	Summary	Assignee	Reporter	P	Status	Resolution	Created	Updated	Due
	ITPM-7	Provide ITPM Connectors	Andreas Grünwald	Richard Mordinyi	●	Open	Unresolved	30/Jan/14	30/Jan/14	30/Jan/14
	ITPM-8	Show PM Data in Dashboard	Andreas Grünwald	Stefan Biffl	●	Open	Unresolved	30/Jan/14	30/Jan/14	30/Jan/14
	ITPM-3	Transform Models into OWL	Andreas Grünwald	estefania	↑	Open	Unresolved	30/Jan/14	30/Jan/14	30/Jan/14
	ITPM-4	Transform Models into Code	Andreas Grünwald	estefania	↑	Open	Unresolved	30/Jan/14	30/Jan/14	30/Jan/14
	ITPM-2	Translate Models into Ontologies and Code	Andreas Grünwald	Redmine Admin	↑	Open	Unresolved	30/Jan/14	30/Jan/14	30/Jan/14
	ITPM-6	Provide Demo Test Data	Andreas Grünwald	Redmine Admin	↑	Open	Unresolved	30/Jan/14	30/Jan/14	30/Jan/14
	ITPM-5	Extend Generated Models by Manual Code	Andreas Grünwald	Redmine Admin	↓	Open	Unresolved	30/Jan/14	30/Jan/14	30/Jan/14

1–7 of 7

Figure C.6: Jira Issues Overview. Note that the issues correspond with Redmine (figure C.5), and the requirements in Excel and Google Drive (figures C.1 and C.2).


Jenkins

Jenkins ▸

- New Job
- People
- Build History
- Project Relationship
- Check File Fingerprint
- Manage Jenkins
- Credentials

Build Queue

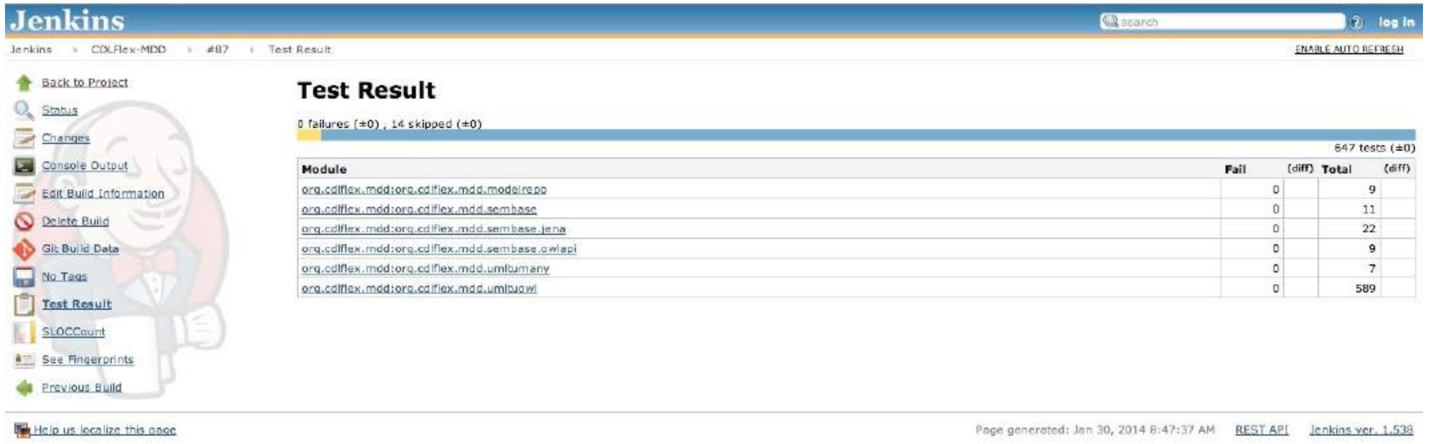
- custom-itpm
- Custom-itpm-generated

All | d | +

S	W	Name ↓	Last Success	Last Failure
🔵	☀	CDLFlex-MDD	14 days - #87	N/A
🟡	☀	custom-itpm ▾	12 days - #274	29 days - #205
🔴	☁	Custom-itpm-generated	12 days - #177	11 days - #178
🔵	☀	Custom-itpm-standalone	12 days - #218	13 days - #211
🔵	☀	Wrapped Bundles	14 days - #25	N/A

Icon: S M L

Figure C.7: Build results in Jenkins.



Test Result

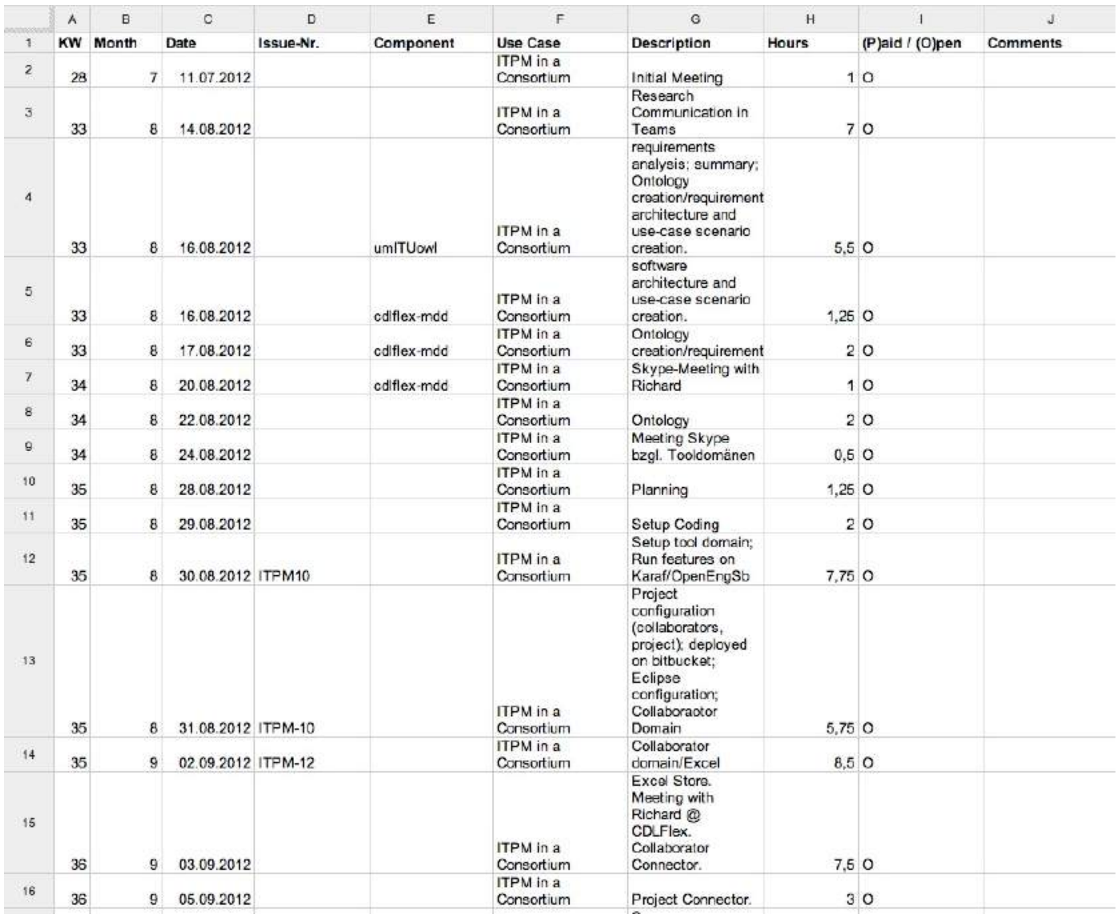
0 failures (±0) , 14 skipped (±0)

647 tests (±0)

Module	Fail	(diff)	Total	(diff)
org.cdlflex.mdd:org.cdlflex.mdd.modelrepo	0		9	
org.cdlflex.mdd:org.cdlflex.mdd.sembase	0		11	
org.cdlflex.mdd:org.cdlflex.mdd.sembase.jena	0		22	
org.cdlflex.mdd:org.cdlflex.mdd.sembase.owlapi	0		9	
org.cdlflex.mdd:org.cdlflex.mdd.umltumany	0		7	
org.cdlflex.mdd:org.cdlflex.mdd.umltuowl	0		589	

Page generated: Jan 30, 2014 8:47:37 AM

Figure C.8: Test results in Jenkins. The UI is simple, and the origin of the test, based on a requirement (traceability) is diffcult to determine for the project manager.



	A	B	C	D	E	F	G	H	I	J
1	KW	Month	Date	Issue-Nr.	Component	Use Case	Description	Hours	(P)aid / (O)pen	Comments
2	28	7	11.07.2012			ITPM in a Consortium	Initial Meeting	1	O	
3	33	8	14.08.2012			ITPM in a Consortium	Research Communication in Teams	7	O	
4	33	8	16.08.2012		umlTUowl	ITPM in a Consortium	requirements analysis; summary; Ontology creation/requirement architecture and use-case scenario creation.	5,5	O	
5	33	8	16.08.2012		cdlflex-mdd	ITPM in a Consortium	software architecture and use-case scenario creation.	1,25	O	
6	33	8	17.08.2012		cdlflex-mdd	ITPM in a Consortium	Ontology creation/requirement	2	O	
7	34	8	20.08.2012		cdlflex-mdd	ITPM in a Consortium	Skype-Meeting with Richard	1	O	
8	34	8	22.08.2012			ITPM in a Consortium	Ontology	2	O	
9	34	8	24.08.2012			ITPM in a Consortium	Meeting Skype bzgl. Tooldomänen	0,5	O	
10	35	8	28.08.2012			ITPM in a Consortium	Planning	1,25	O	
11	35	8	29.08.2012			ITPM in a Consortium	Setup Coding	2	O	
12	35	8	30.08.2012	ITPM10		ITPM in a Consortium	Setup tool domain; Run features on Karaf/OpenEngSb	7,75	O	
13	35	8	31.08.2012	ITPM-10		ITPM in a Consortium	Project configuration (collaborators, project); deployed on bitbucket; Eclipse configuration; Collaboraotor Domain	5,75	O	
14	35	9	02.09.2012	ITPM-12		ITPM in a Consortium	Collaborator domain/Excel	8,5	O	
15	36	9	03.09.2012			ITPM in a Consortium	Excel Store. Meeting with Richard @ CDLFlex. Collaborator Connector.	7,5	O	
16	36	9	05.09.2012			ITPM in a Consortium	Project Connector.	3	O	

Figure C.9: Timesheet for a single collaborator, according to the CDL-Flex format.

APPENDIX D

Data Models

The appendix D includes sub-models (packages) of the core ITPM ontology,and three local tool model representatives. These ontologies have been modeled with the Semantic Model Editor. In addition, the models for maintaining data source configurations, and queries/rules are provided. The full versions of the ontologies are provided within the downloadable prototype[1].

D.1 ITPM Core Ontologies

[1]ITPM Prototype: `https://bitbucket.org/agruenwald/custom-itpm-builds/downloads`

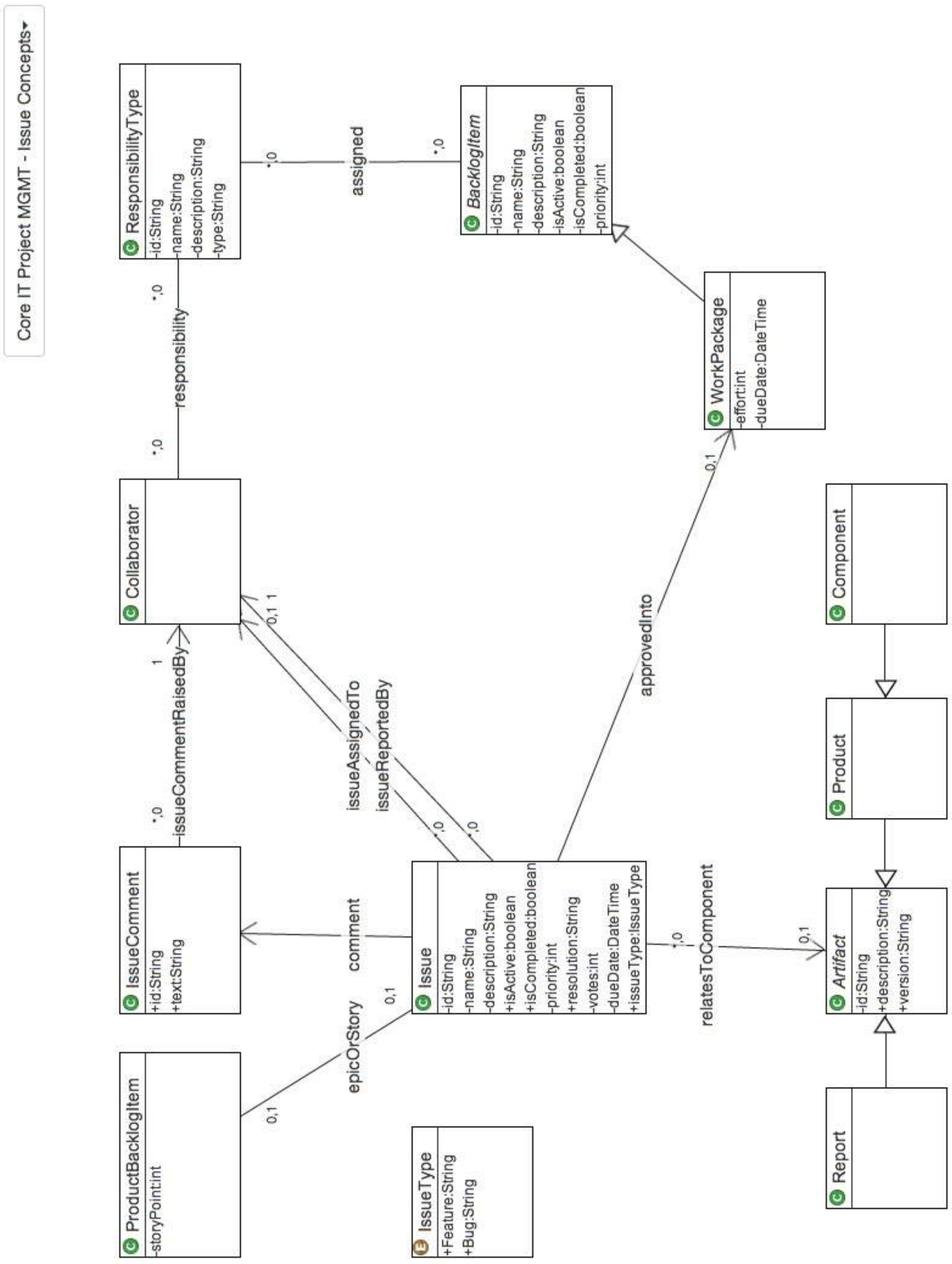


Figure D.1: The main part of the common issue concepts, which are part of the core ITPM ontology.

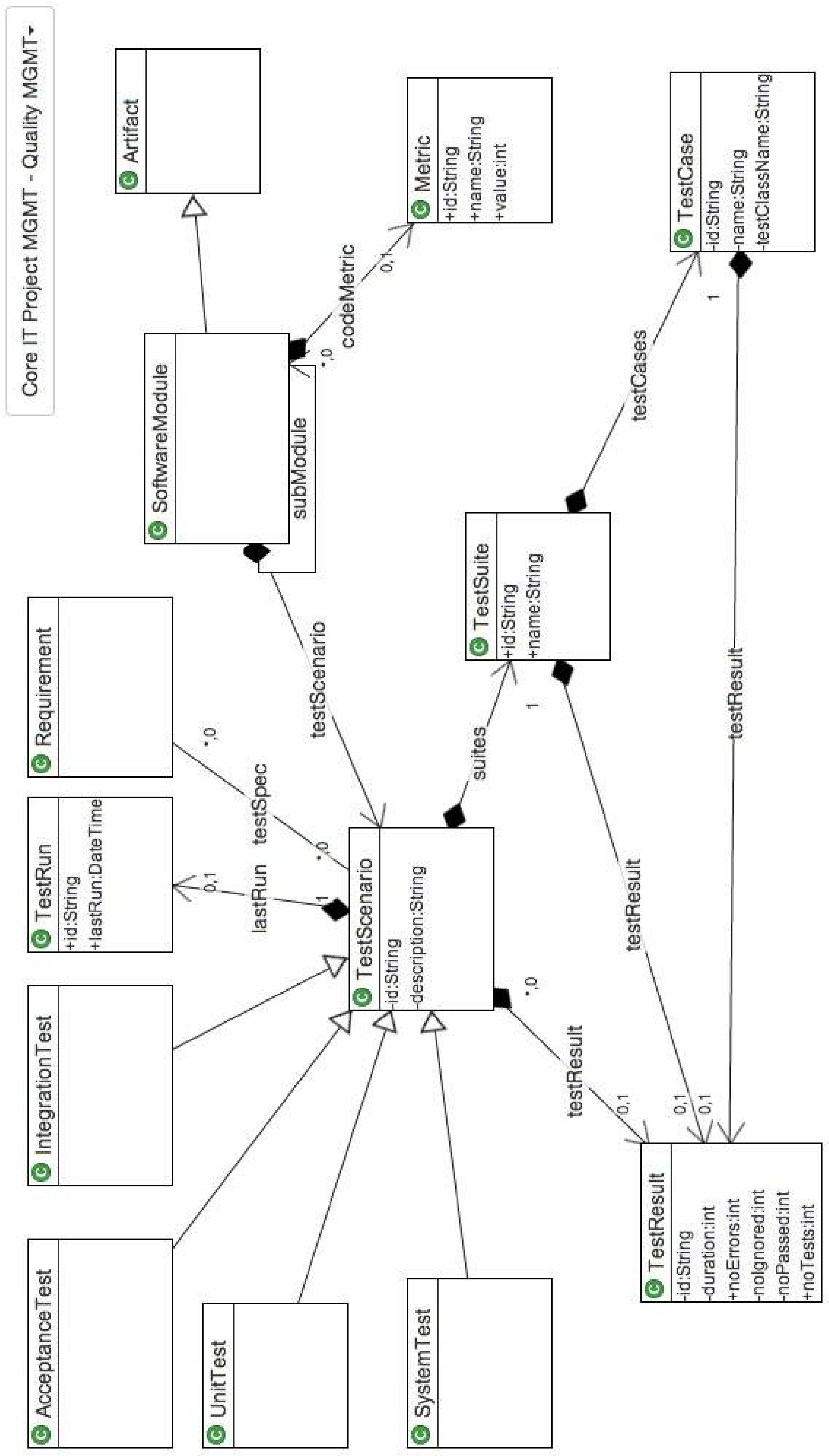


Figure D.2: The main part of the common QM concepts, which are part of the core ITPM ontology.

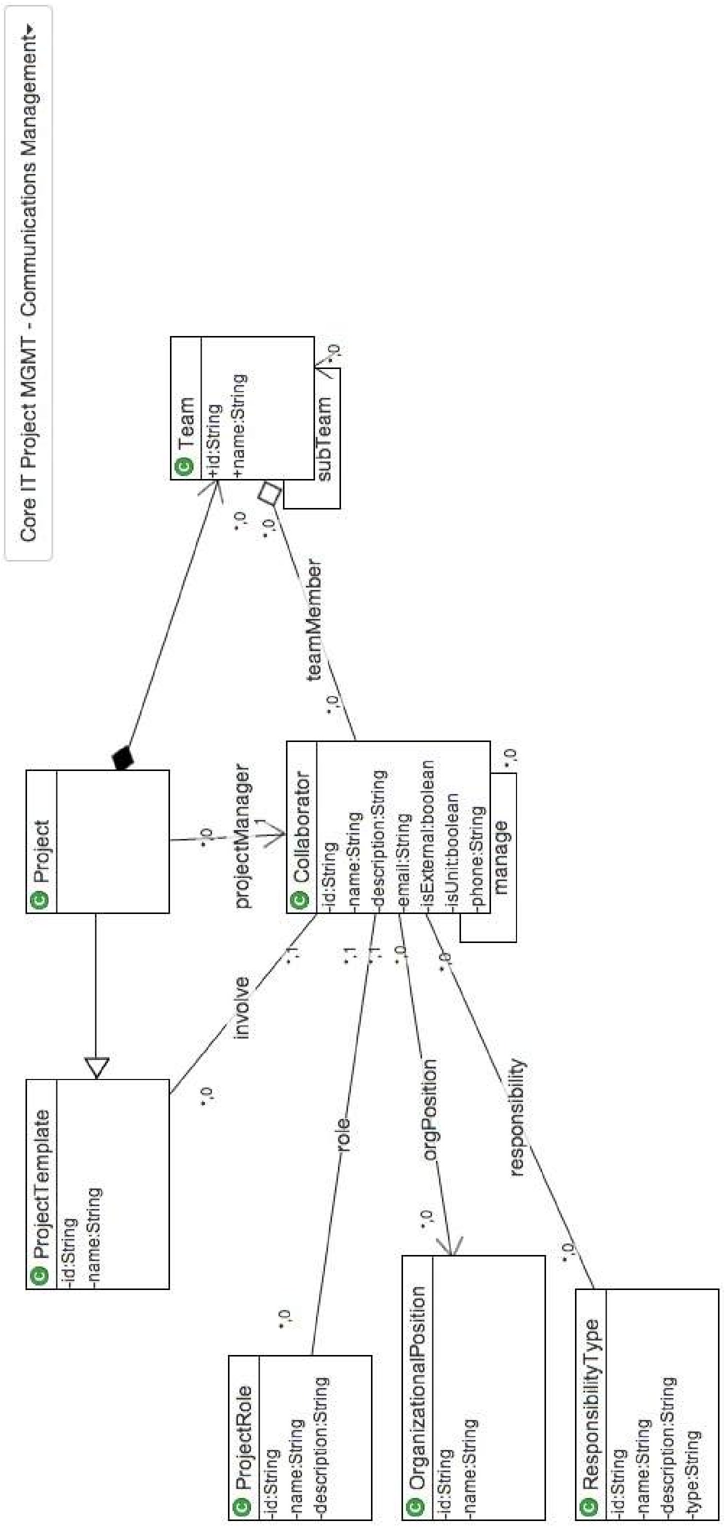


Figure D.3: Excerpt of the main elements of the Communication Management Knowledge Area of the core ITPM ontology as used for the prototype.

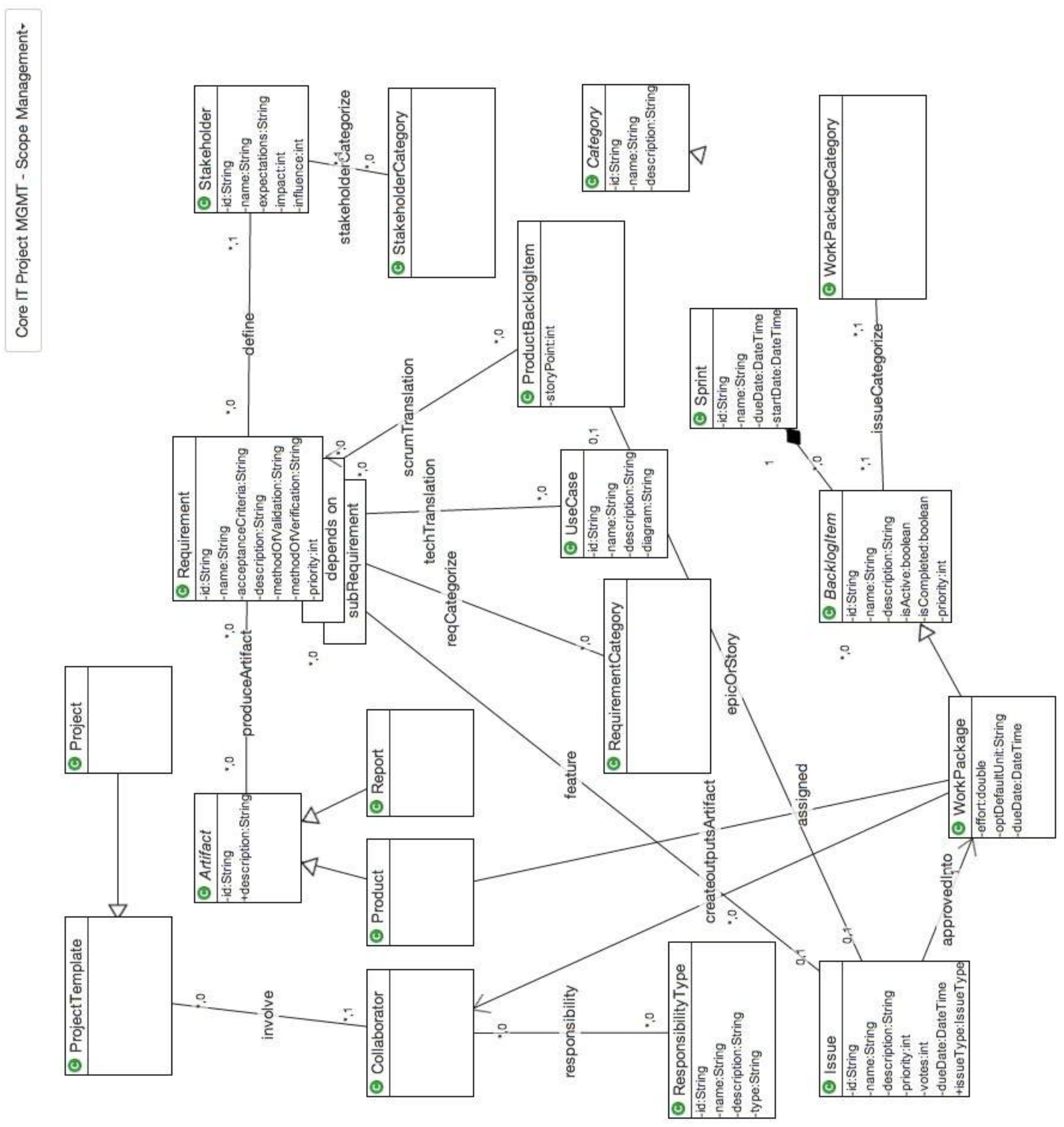


Figure D.4: Scope Management Knowledge Area of the core ITPM ontology (customized for the prototype).

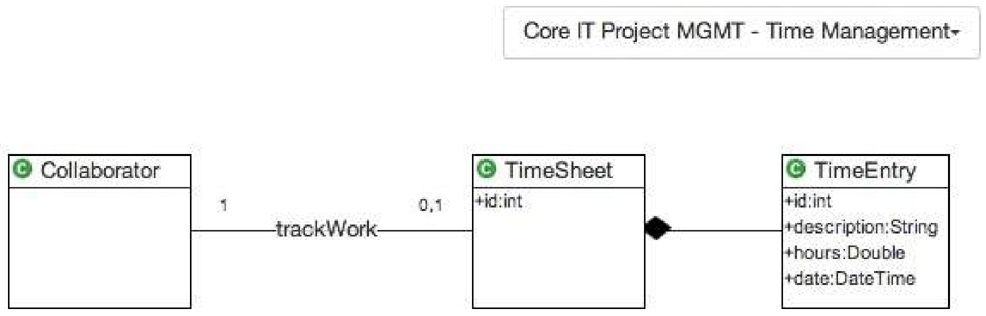


Figure D.5: Time Management Knowledge Area of the core ITPM ontology as used for the prototype.

D.2 Local Tool Models

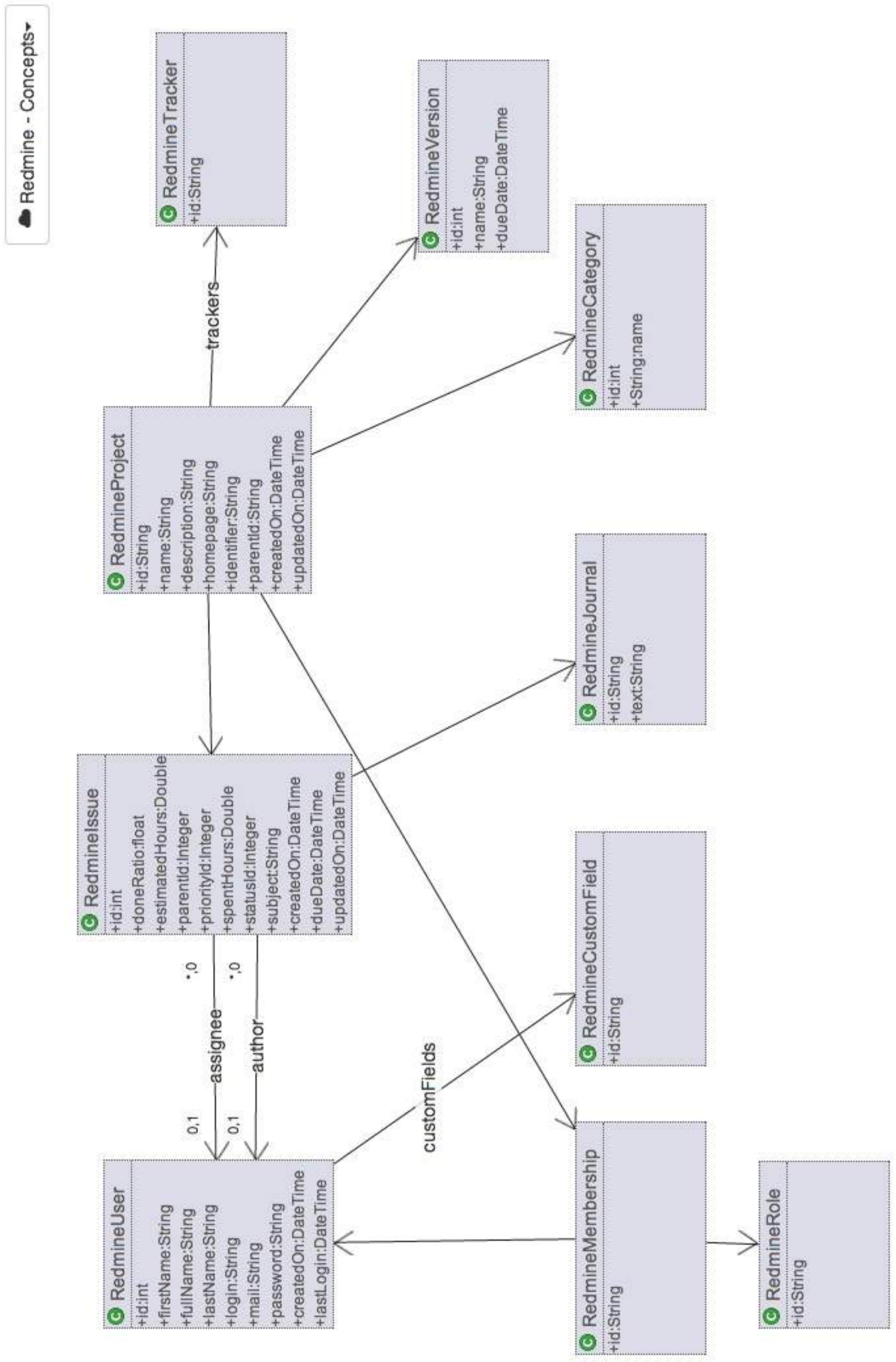


Figure D.6: Local Redmine Data Model. The dark-grey classes indicate that the model is marked as “information-only”. For this classes no Java code is generated, because the available Taskadapter Redmine API for Java already covers the related classes.

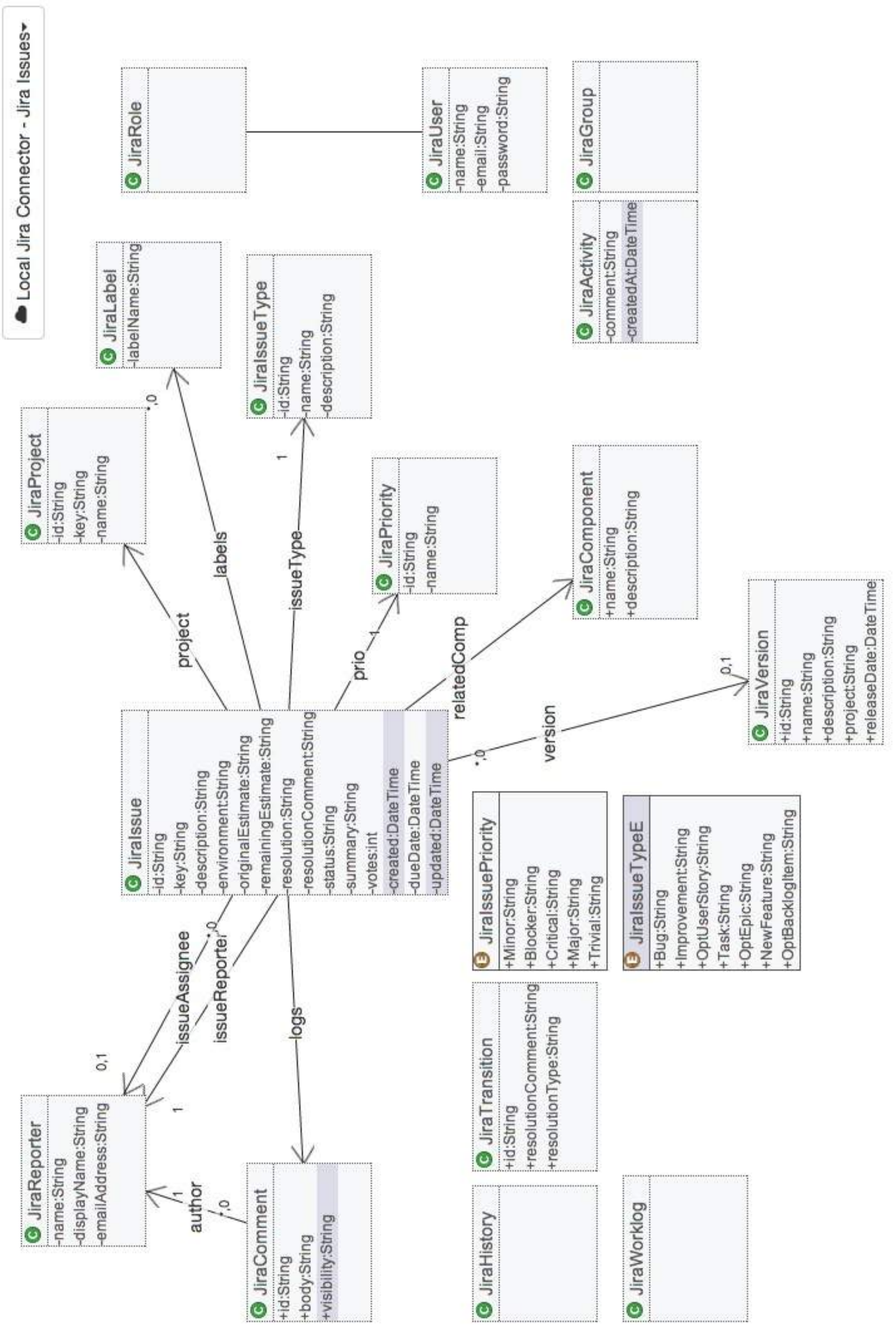


Figure D.7: Local Jira Issue data model. The dark-grey attributes are marked as "information-only" hence they are ignored by the code generator.

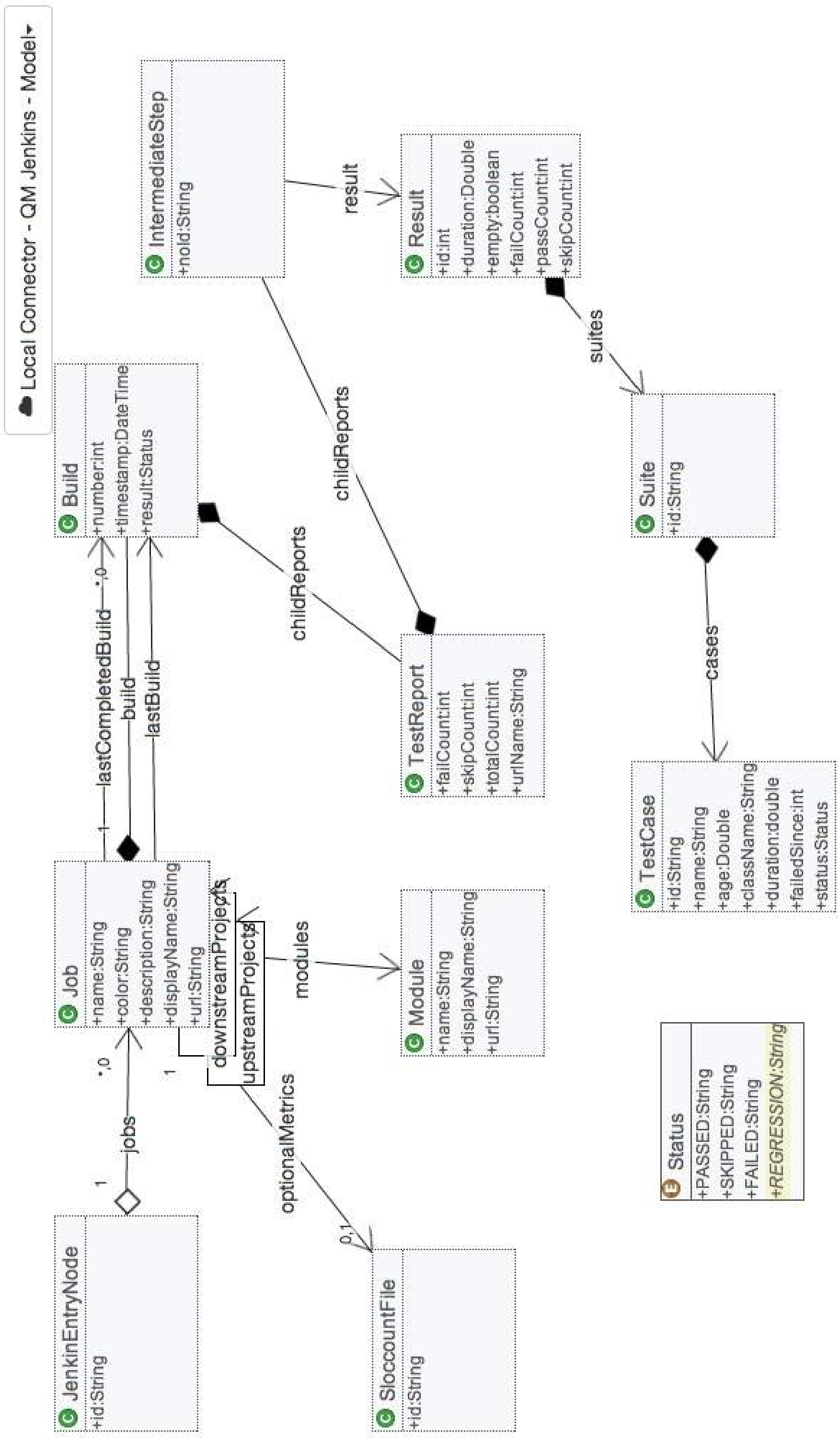


Figure D.8: The data model of the Jenkins Built Server API. The attribute “Status.Regression” (yellow) exemplifies a suggestion that is not implemented yet.

D.3 Models for Maintaining Configurations

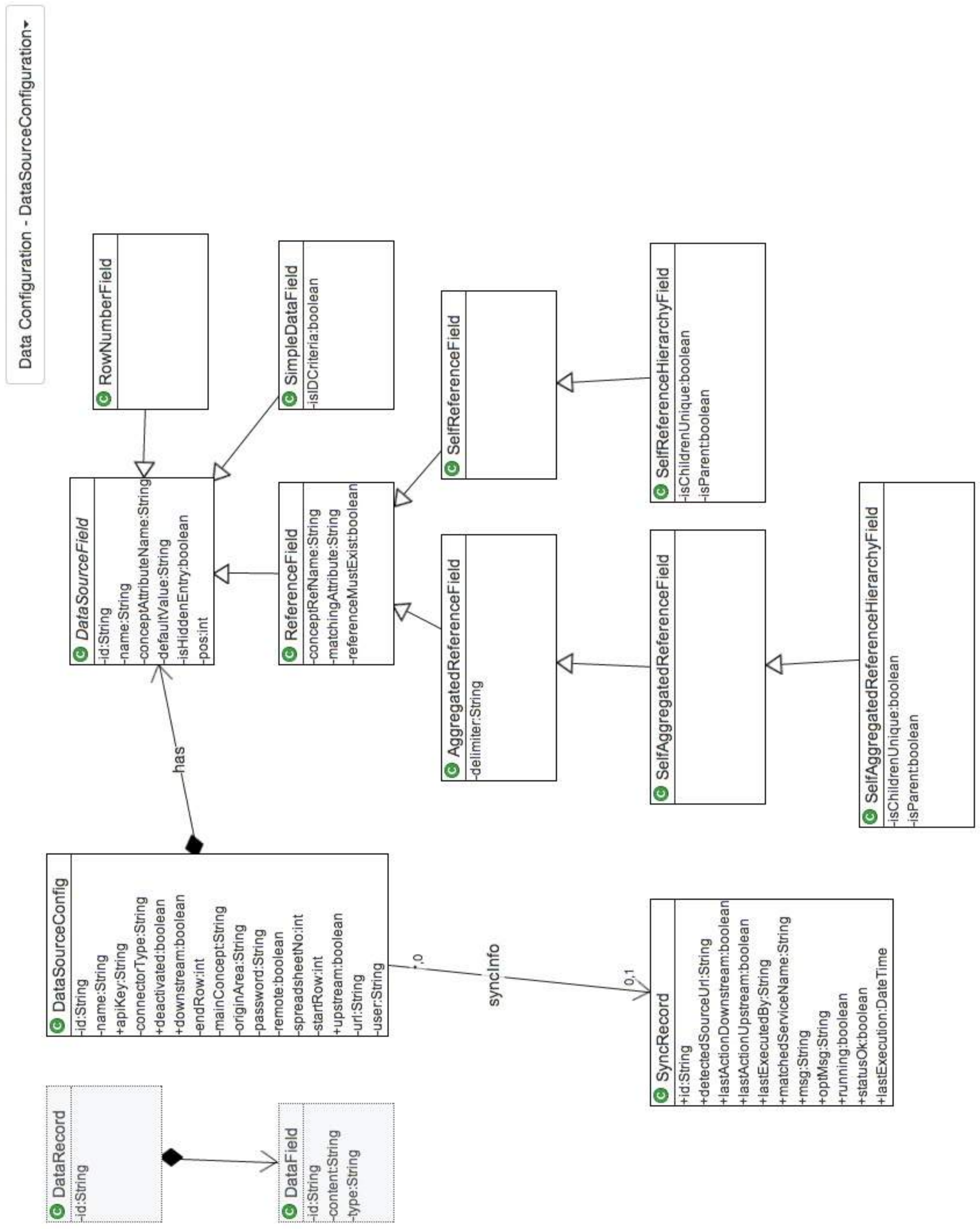


Figure D.9: *Data Source Configuration Ontology*, which is used to configure the synchronization of local PMS and spreadsheets formats.

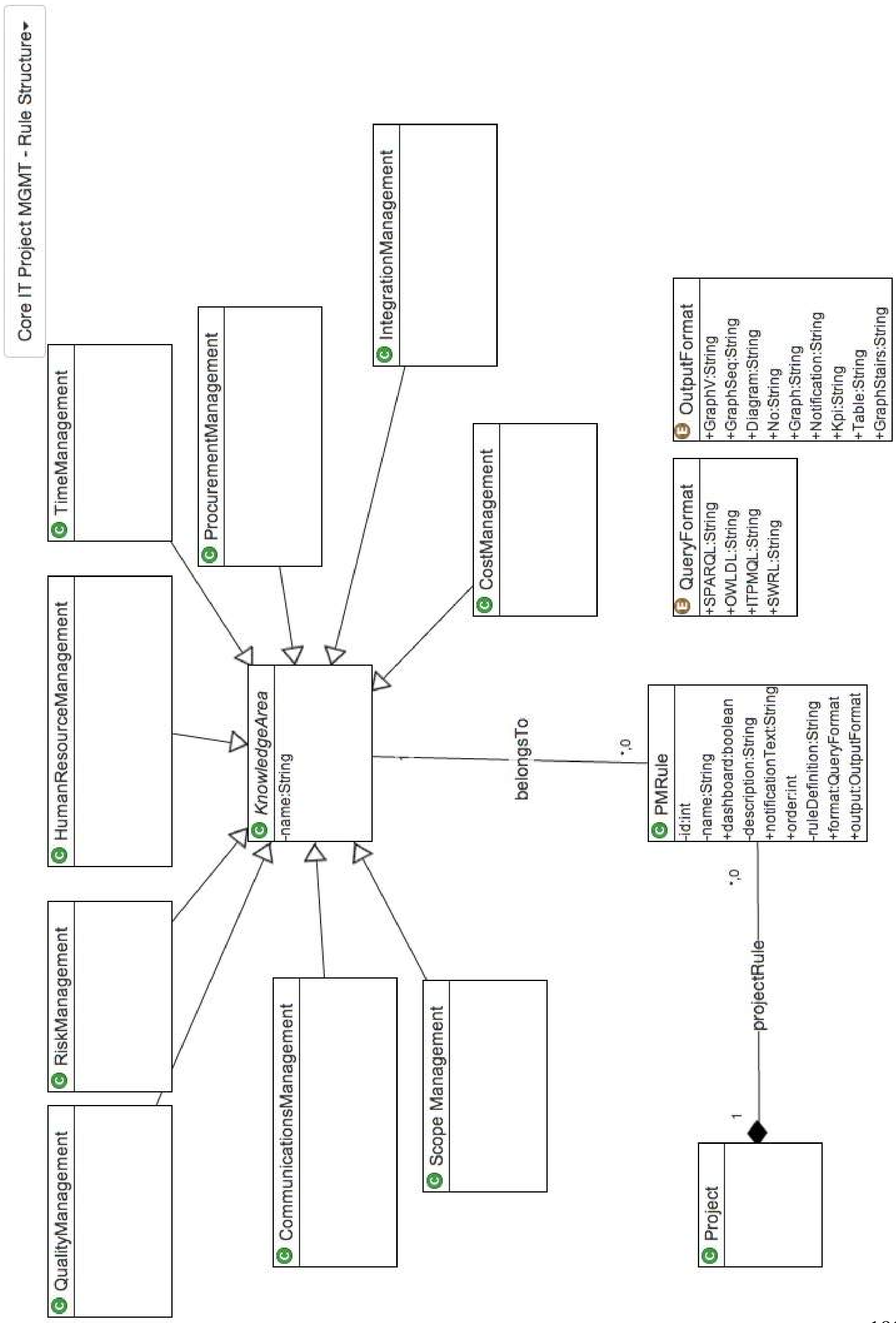


Figure D.10: The *Rule Storage Ontology* maintains the rules and queries that are displayed in the central ITPM dashboard.