The Semantic Model Editor: Efficient Data Modeling and Integration based on OWL Ontologies

Andreas Grünwald[1] Dietmar Winkler[2] Marta Sabou[2] Stefan Biffl[2]
Christian Doppler Laboratory for Software Engineering Integration for Flexible Automation Systems (CDL-Flex)
Vienna University of Technology, Institute of Software Technology and Interactive Systems
Favoritenstrasse 9/188, A-1040 Vienna, Austria
[1]a.gruenw@gmail.com, [2]<firstname.lastname>@tuwien.ac.at

ABSTRACT

Semantic Web and Linked Data are widely considered as effective and powerful technologies for integrating heterogeneous data models and data sources. However, there is still a gap between promising research results and prototypes and their practical acceptance in industry contexts. In context of our industry partners we observed a lack of tool-support that (a) enables efficient modeling of OWL ontologies and (b) supports querying and visualization of query results also for non-experts. The selection and application of existing semantic programming libraries and editors is challenging and hinders software engineers, who are familiar with modeling approaches such as UML, in applying semantic concepts in their solutions. In this paper we introduce the Semantic Model Editor (SMEd) to support engineers who are non-experts in semantic technologies in designing ontologies based on well-known UML class diagram notations. SMEd – a Web-based application – enables an efficient integration of heterogeneous data models, i.e., designing, populating, and querying of ontologies. First results of a pilot application at industry partners showed that SMEd was found useful in industry context, leveraged the derivation of reusable artifacts, and significantly accelerated development and configuration of data integration scenarios.

Categories and Subject Descriptors

H2.1 **[Information Systems]**: Database Management – *logical design, data models.*

General Terms

Management, Design, Human Factors, Standardization.

Keywords

Semantic Model Editor, OWL, Data Models, IT Project Management, Data Integration, Feasibility Study.

1. INTRODUCTION

The integration of data models from heterogeneous tools is a success-critical issue in automation systems development projects, where stakeholders, coming from different disciplines, have to collaborate and exchange data [3][4]. For instance, in the automation systems domain, e.g., constructing hydro power plants, among others, mechanical, electrical, and software engineers are involved. Individual engineers apply highly-specific tools and related data models. These tools and data models are loosely connected and include limited capabilities in data exchange [9]. Semantic technologies can help to overcome technical and semantic gaps to improve engineering processes for automation systems. More specifically, Semantic Web technologies are promising candidates for extracting and integrating knowledge and data across heterogeneous engineering tools and project partners. A typical use case is the transformation of project artifacts (e.g., requirements or engineering plans) between different models and data formats. Data models, respectively ontologies, constitute a central part of the CDL-Flex's integration framework[1] supported by the Open Engineering Service Bus (OpenEngSB)[2] [4]. These models are used to elaborate, specify, and implement mappings between local data formats of related tools [3] to enable seamless data exchange between these tools and support collaboration of their users.

Based on observations at our industry partner, i.e., a large-scale hydro power plant development and integration organization, we found that related Semantic Web technologies still suffer from a gap between research outcomes (theory, prototypes, and programming APIs) and their practical applicability and usability. We identify a set of shortcomings of the currently available tool implementations for Semantic Web technologies in the context of their applicability in automation systems engineering projects: (a) available Semantic Web tools and APIs (e.g., Protégé[3]) require explicit expert know-how in order to apply the tools and to create sophisticated data models, e.g., OWL ontologies. The need for knowledge-engineering expert know-how hinders the application of ontologies for engineers who are not familiar with semantic technologies, e.g., software engineers or domain experts in the automation systems domain; (b) practitioners are faced with a set of heterogeneous and independent tools that have to be combined to enable the applicability in a specific application domain. Thus, the rollout of such a bundled solution without explicit expert know-how is often carried out tediously and inefficiently; and (c) the available set of ontology and query editors and the quality of their features are limited. For instance, UML class diagrams are frequently used for the specification of tool data models. While ontologies are designed in Protégé, the implementation and integration of the models and the transformation into OWL ontologies must be conducted manually by experts, e.g., *knowledge engineers*. However, the manual implementation and

[1] CDL-Flex: Christian Doppler Laboratory for "Software Engineering Integration for Flexible Automation Systems", http://cdl.ifs.tuwien.ac.at

[2] Open Engineering Service Bus: http://www.openengsb.org

[3] Protégé: http://protege.stanford.edu

maintenance of related tool connectors is tedious and error-prone. As there are no common user interfaces (UI) available, queries for retrieving ontology instances must be maintained programmatically, leading to inefficient implementations of agile customer requirements. None of available solutions, e.g., Jena or Protégé, currently provide sufficient support for efficient and effective SPARQL query editing. Models are the central artifacts for efficient and effective integration of complex engineering tools, residing at different partners and teams. In addition, there is often a gap between domain expert's requirements of model representations and requirements for the implementation of these models on a technical level. Needs are not addressed sufficiently and result in inefficient integration processes and redundant process steps. Thus, we see a strong need to support domain experts in better applying semantic technologies by providing a Semantic Model Editor (SMEd) that enables (a) an efficient approach for creating and maintaining ontologies and (b) an efficient way for constructing and executing queries with respect to industry needs.

The remainder of this paper is structured as follows: Section 2 summarizes related work on data models, transformation, editors, and data integration approaches. Section 3 presents the research issues. Section 4 presents the Semantic Model Editor (SMEd) for modeling ontologies based on UML class diagram notation. We present the solution concept in Section 5 and the feasibility study of the prototype application in Section 6. Finally, Section 7 summarizes and presents future work.

2. RELATED WORK

This section summarizes related work on data model specification formats and Semantic Web technologies with focus on transformations, editors, and data model integration approaches.

2.1 Data Models: UML vs. OWL

The Unified Modeling Language (UML) is wide-spread and established in industry and well supported by various model editors modeling editors [13]. Because of a graphical representation models are understandable, compact, and support communication also to non-technical experts, e.g., managers. The compact representation includes main concepts (classes), attributes, hierarchies, and associations. However, an important shortcoming of UML is the poor specification of the underlying XML Metadata Exchange (XMI) format. Although XMI has been designed to make UML class diagrams reusable, vendor-specific UML model editors currently fail to implement this standard [11]. Models and languages, e.g., the OMG's Ontology Definition Metamodel (ODM)[4] and OCL[5] specifications are available but to our knowledge not widely spread in industry. OWL ontologies, i.e., explicit and formal specifications of a conceptualization, aim at documenting knowledge in a machine-interpretable way [8]. OWL ontologies ensure consistent formalization of (knowledge) models and instances and provide a richer set of expressiveness than UML. However, OWL and related editors need additional expert know-how (knowledge engineer) for the specification of ontologies and the usage of specific modeling tools.

2.2 Transformation between UML and OWL

Related work shows that transformations between (a subset of) UML class diagram elements and OWL ontology concepts is intuitive and improves understandability, communicability, and efficiency of modeling approaches [10][11][19]. Grünwald *et al.* [11][13] provide an overview on ontology editors on the Open Source (OS) market. They conclude that although sufficiently treated in literature, no functional editors respectively transformation engines between UML and OWL exist. As a result, they introduce umlTUowl[6] enabling the transformation of UML class diagrams from a set of vendor-specific modeling tools (e.g., MS Visio, Visual Paradigm) into OWL 2 DL [11]. While umlTUowl is considerably requested by researchers (> 500 downloads) no central editor is available that supports (collaborative) modeling and maintenance of individual models. Thus, domain experts cannot edit / maintain existing UML class diagrams if a specific version of a vendor-specific modeling tool (e.g., MS Visio) is used on a specific local machine. We believe that this limitation is a hurdle for applications in industry. In addition, we see additional potential for efficient implementation of existing models, i.e., to reduce the manual implementation efforts and to increase the visualization quality of ontology editors, by incorporating additional software engineering features into such an editor.

2.3 Ontology Editors and Semantic Web Programming APIs

Table 1 presents different aspects of available Semantic Web frameworks and a comparison of the main features of three Semantic Web (programming) APIs, including related modeling tools. Protégé has been selected as a state-of-the-art approach of OS OWL ontology editors. Protégé includes a large set of plug-ins, such as the SPARQL query editor. The authors are aware of alternative ontology editors. However, existing ontology editors and IDE's, such as the TwoUse Toolkit [15], require a significant amount of expert know-how, setup costs and training, before they are applicable. Moreover, existing editors often apply additional specifications, annotations, and (UML) profiles, which make the modeling process more complex, and often restrain end-users to a specific ecosystem (e.g., Eclipse EMF/ECore[7]). This leads to inefficient design processes and incompact models with limited reusability and exchangeability capabilities. See [13] for more details of the analyzed tools.

Table 1. Sample Comparison of Semantic Web Programming Approaches.

API / Feature	Protégé / OWLAPI	Jena API	OpenEngSB / EKB
Ontology Editor	Yes	No	No
UI Query Editor	Yes	No	No
OWL Support	Yes	Yes	No
SPARQL Support	Yes	Yes	No
Scalable Data Storage	No	Yes	Yes
Code Generator	No	Yes	No

Main results of Table 1 include: (a) features of available Semantic Web technologies are heterogeneous. For instance, to implement scalable high-performance semantic data storages, several technologies have to be combined (e.g., Protégé for modeling and Jena for implementation); (b) specific technologies are expensive to select, rollout, and implement. The successful application of a technology is prone with risk and heavily relies on expert know-how; (c) lack of visualization capabilities and querying of ontologies. For instance, Jena does not provide an ontology editor

[4] OMG ODM: http://www.omg.org/spec/ODM/

[5] OMG OCL: http://www.omg.org/spec/OCL/

[6] umlTUowl: https://bitbucket.org/agruenwald/umltuowl

[7] EMF: www.eclipse.org/modeling/emf

at all. From industrial experience we know that a powerful query language, such as SPARQL, is essential to escape from the monotonic nature of OWL DL query and to realize more sophisticated queries. The Protégé ontology editor is complete and powerful, but not very compact. The representation of individuals is basic and there exists no way to continuously observe and monitor defined queries. Available SPARQL plug-ins are limited and do not provide syntax-highlighting, auto-completion, or mature error handling. Bizer *et al.* [5] identify weak ontology editor support as a main research challenge of Semantic Web.

A major disadvantage of Protégé and the underlying OWLAPI is that ontologies can only be persisted in serial files, which prevents scalable data access. Thus, we evaluated the OWLDB plug-in for Protégé aiming to support mappings between OWL constructs and data storages to enable efficient persistence of large ontologies. However, the plug-in was not installable for Protégé 4.0 [14].

In addition, available programming APIs are low-level. Hence, they require high learning efforts by (untrained) software engineers. Several code generators facilitate the complexity of semantic data storage access for simpler Create-Read-Update-Delete (CRUD) operations and anticipate expensive consistency checks, e.g., cardinality checks for data and object properties or functional dependencies, to reduce the reasoning overhead. From the perspective of a software architect, the abstract code generation layer should be accessible enabling the exchange of the underlying API. While RDFReactor [18] provides such a mature code generator for Jena and Sesame[8], it does not support the OWLAPI. To our best knowledge, currently there exists no code generator for OWLAPI that produces high-quality code, is flexibly adaptable, and enables an easy integration within an integration platform, e.g., the OpenEngSB architecture.

2.4 Data Integration for Project Management

In the context of IT project management (PM), Ahlemann introduce RefMod^{PM}, which is a reference model for the rollout of sophisticated enterprise PM systems [2]. RefMod^{PM} consists of 18 UML class diagrams, and 10 activity diagrams, which must be manually remodeled, adapted, and implemented, in order to implement them. For instance, Ahlemann demonstrates how the reference model can be adapted to integrate project data (i.e., phases, milestones, and activities) from a specific PM system. Abels *et al.* translate RefMod^{PM} into PROMONT, an equivalent OWL ontology [1]. The authors describe the integration of heterogeneous project partners and their local ontology representations on a conceptual level. However, they do not provide evidence on the implementation of the ontology but describe the need for an UI-supported UML/OWL transformation and integration framework. In addition, several papers utilize Protégé to implement simple data integration scenarios [7][8][16][17]. Interestingly, a multitude of the studied papers use Protégé to model and query the ontology. They present a compact overview of the related ontology in UML class diagram notation, which requires redundant modeling efforts.

3. RESEARCH ISSUES

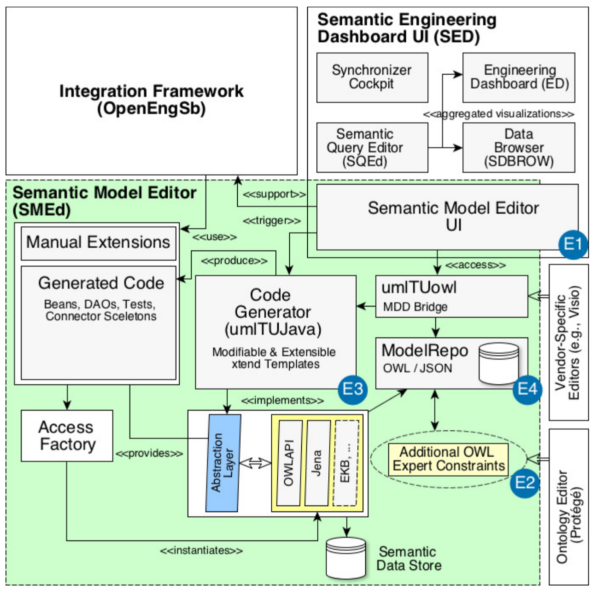
Based on related work and strong needs for an approach that supports model definition and maintenance for non-experts (e.g., domain experts), we derive the following research issues:

RI.1. How should a semantic model editor be designed to support requirements coming from software engineers and domain experts to define and maintain ontologies? Established ontology editors require expert know-how to define and maintain ontologies, which hinders a widespread application in industry contexts, where only software engineers and domain experts are available.

RI.2. How can we support non-experts to efficiently and effectively browse and visualize ontologies? Currently software engineers spend a considerable implementation effort on translating management requirements into ontologies and code. The creation of UI elements, even for prototypes, is time-consuming, and the qualities of the outcomes vary significantly.

4. SOLUTION APPROACH

We present the *Semantic Model Editor (SMEd)*, a web-based editor for the design of OWL ontologies based on UML class diagram notation. The interfaces of the SMEd are embedded within a *Semantic Engineering Dashboard (SED)*, enabling efficient integration of models and synchronization and visualization of heterogeneous data from different sources.



Integration Framework (OpenEngSb)
Semantic Engineering Dashboard UI (SED)
Synchronizer Cockpit
Engineering Dashboard (ED)
<<aggregated visualizations>>
Semantic Query Editor (SQEd)
Data Browser (SDBROW)
Semantic Model Editor (SMEd)
<<support>>
<<trigger>>
Semantic Model Editor UI
E1
Manual Extensions
<<use>>
<<produce>>
<<access>>
Generated Code
Beans, DAOs, Tests, Connector Sceletons
Code Generator (umlTUJava)
Modifiable & Extensible xtend Templates
E3
umlTUowl
MDD Bridge
ModelRepo
OWL / JSON
E4
Vendor-Specific Editors (e.g., Visio)
<<implements>>
Access Factory
<<provides>>
Abstraction Layer
OWLAPI
Jena
EKB, ...
Additional OWL Expert Constraints
E2
Ontology Editor (Protégé)
<<instantiates>>
Semantic Data Store

Figure 1. Architectural Overview, including main extensions of umlTUowl (E1 – E4).

Figure 1 presents the architecture of the prototype: The SMEd provides the main User Interface (UI) for modeling OWL ontologies and incorporates features for code generation (umlTUJava), data editing, visualization, and maintaining of models (ModelRepo). See Section 4.1 for details on the extensions (E1-E4) of umlTUowl [11], the foundation for SMEd. Based on the output of SMEd, additional UI components for the visualization of data have been implemented, e.g., the Semantic Query Editor (SQEd) and the Semantic Data Browser (SDBROW). SQEd enables the configuration of graphical components within the central engineering dashboard (ED). Note that SMEd is designed to support the efficient integration of engineering tools via the OpenEngSB framework (top-left of Figure 1). Model-driven software engineering (MDSE) can help to enable efficient, transparent and flexible development of tool connectors based on generated code artifacts, and model-to-model (M2M) transformations, respectively workflows [6]. Note that integration and synchronization mechanisms [3][4] of the SED are out of scope of this paper; see details in [12].

[8] Sesame: http://www.openrdf.org

4.1 The Semantic Model Editor (SMEd)

To capture requirements and candidate extensions for umlTUowl [11], we conducted workshops in industry and academia. Based on feature requests collected from these key experts, four main extensions of the umlTUowl transformation approach have been identified (Figure 1; E1 – E4).

Web-Based UI Editor (E1). umlTUowl [11] supports a subset of UML editors (e.g., Visual Paradigm, MS Visio, Argo UML). Because vendor-specific XMI data formats are incompatible to each other, a web-based graphical UI editor, based on UML 2.0, is needed to overcome this heterogeneity. Figure 2 presents the UI prototype of the SMEd, including an illustrative data model.

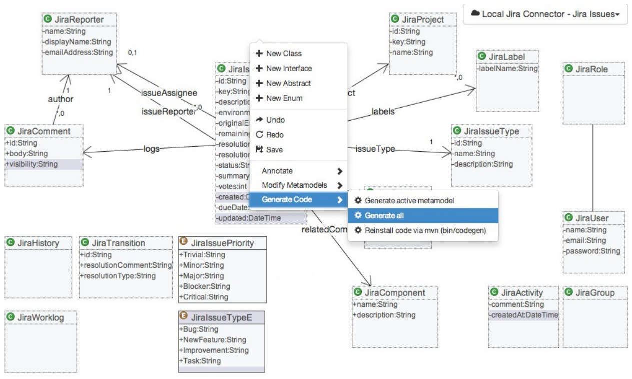


Figure 2. UI-Prototype of SMEd with a Sample Data Model.

The editor enables collaborative design of OWL ontologies overcoming limitations regarding model incompatibility issues and required expert know-how on Semantic Web technologies. As ontologies are visualized in UML class diagram notation, the design of compact, small-scale, and understandable models is facilitated. The underlying harmonizing algorithm of umlTUowl and the OWL 2 DL reasoning mechanisms ensure (a) pruning of redundant information (sub-packages and redundant classes are merged) and (b) the persistence of consistent models. The UI incorporates basic modeling features, such as drag & drop, undo/redo, and saving of intermediate changes via a model history. All model elements (e.g., packages, classes, attributes, and associations) can be annotated. Comments, respectively annotation are used to clarify the meaning of specific elements, or to communicate issues to involved stakeholders (including managers and technicians). As some UML elements cannot be expressed in OWL without adding additional semantics, this information must be preserved as annotations. For instance, coordinates of UML classes, class types, or interfaces are stored as OWL annotations by applying specific prefixes.

Adapted UML to OWL Transformations (E2). The previous umlTUowl transformation approach [11] supports a selected set of UML elements. Mappings comprise UML packages, classes, abstract classes, interfaces, generalizations, attributes (including visibility and primitive XML-built in data types), association (including navigability and cardinality), aggregations, compositions, and comments. To improve the representation of data models with respect to (a) compactness and (b) intuitiveness, the coverage of UML to OWL mappings has been extended. Main extensions focus on (a) *enumerations* (represented as (sub-)class hierarchies in OWL), (b) *multiplicity of attribute values* (resolved via min/max constraints for OWL data properties), (c) *data constraints* (treated as internal comments within the umlTUowl meta model and translated into OWL facets), and (d) *uniqueness of attributes*. However, several UML diagram elements are still not supported by SMEd, as they are of low practical relevance for current industry applications, e.g., class operations, association classes, n-ary associations, overlapping / disjoint classes, XOR and subset annotations, and stereotypes. As OWL is more expressive than UML, the SMEd restricts knowledge engineers (i.e., Semantic Web experts) to express more sophisticated relations, such as for instance disjoint or equivalence class statements, advanced property characteristics (e.g., functionality, transitivity, symmetry, and reflexivity), property relations (inverse property or property chains), additional data restrictions (i.e., data ranges), and combined OWL DL statements. The SMEd utilizes a second OWL layer, i.e., an OWL ontology on top of each ontology, respectively model (see Figure 1; E2). This ontology imports the (base) ontology and can hold additional expert statements and constraints. Based on this strategy trivial parts of a (OWL) model can be separated from more complex statements; irrelevant complexity is hidden and the full expressiveness of OWL available for knowledge engineers.

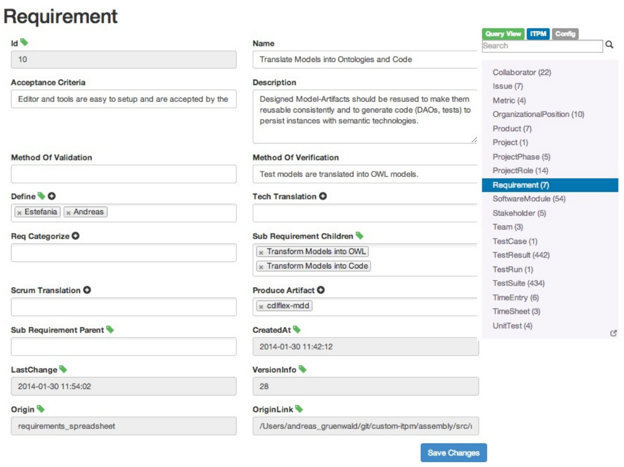
Table 2. Mapping Examples between Derived Artifacts.

UML	OWL	Java Code	UI
UML Class	OWL Class	Bean + DAO (if not abstract)	Table + Form (if not abstract)
UML Attribute	OWL Data Property	Property with specific data type + getters / setters	Table Column / HTML Field (e.g., checkbox, text area)
UML Generaliza-tion	OWL SubClassOf Property	Extends or implements	(sub-) type of linked classes can be selected in dynamic a menu.
UML Association	OWL Object Property	List element; getter; Cardinality constraints	Nested form, dynamic menu to link between classes.
Comment	OWL Annotation	Comment	Help texts at form/field level.
Constraint	OWL Facet	Java condition & add. unit tests	Error messaging at runtime.

Code Generator / Generic Data Access and UI (E3). To reduce the development effort for the integration of engineering tools, the existing SMEd models are utilized to derive additional (code) artifacts. See Grünwald *et al.* [12] for details on the code generation approach based on UML models. Table 2 presents a subset of the implemented mappings between the UML elements, related OWL concepts, Java code, and Web-UI elements. The models / ontologies are used to (a) derive an abstraction layer for accessing various OWL libraries (Figure 1; umlTUJava) and (b) for the creation of generic Web-UIs, such as tables and forms (Figure 1; SDBROW). The execution of generated code is fully integrated in SMEd.

umlTUJava Code Generator (E3-1). Each UML and OWL class results in a generated Java bean with getters and setters. For non-abstract classes data access objects (DAO) are derived (including related unit tests). Each DAO implements an abstract interface that provides CRUD methods to persist and validate related beans in a semantic data storage. The abstract DAO interface is used as an abstraction layer for the underlying Semantic Web Programming API. In the current SMEd prototype version, code generation for the OWLAPI, and the Jena API (SDB, TDB) are implemented. By utilizing an Access Factory (see Figure 1), the specific data storage implementation can be injected at runtime. For instance, the OWLAPI is more useful for the development of early prototypes, while the Jena API provides better performance in production environments. The abstraction layer hides the complexity of the low-level Semantic API and improves testability of the system. For each supported storage API, an adaptable Xtend template for the creation of DAO must be implemented. The Xtend template engine provides convenient auto completion, syntax highlighting, and debugging features, and an intuitive syntax. The interfaces of the generated code enable developers to extend specific DAOs and beans with manual code. For instance, by extending a specific DAO, a verification and validation mechanism can be applied. Analogously, beans can be extended with additional methods. The access factory instantiates the correct class instances automatically. Manual code is separated from generated code and is preserved when the derived code is regenerated. For details see Grünwald *et al.* [12].

Generic User Interface (E3-2). To support industry partner domain experts (non-experts in ontologies and knowledge management) the user interface supports browsing and editing of OWL data instances. As existing editors, such as Protégé, do not provide acceptable editors, we present a set of mappings between umlTUowl meta-models, respectively OWL ontologies, and HTML elements that automate the generation of a Web-Based UI. See Figure 3 for an example of a generated UI. Basically, each (non-abstract) class is transformed into a (searchable) table and a related form, integrated in the navigation bar of the data browser (SDBROW). Each table contains (direct and inferred) data properties of a class. Further, table views provide links to add, edit, and remove records. Generated forms enable the modification of records. They consist of input fields (data properties) and sub-forms (object properties) that enable users to modify and browse the content of linked individuals. Based on the data type and the cardinality of the related data properties the derived input fields may vary. For instance, boolean properties are transformed into checkboxes. If an explicit constraint (OWL Facet) is defined, e.g., stating that the number of characters can exceed a specific length (e.g., 100 chars) then `html:textarea` is rendered instead of a `html:input` field. Enumerations are rendered as dynamic menus with a predefined set of selectable values. Nested sub-forms are used to support editing, adding, and removing linked entries. If an object property links to a class with a nested class hierarchy, an additional select menu is rendered.



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.
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/ı

Save Changes

Query View ITPM Config

Search

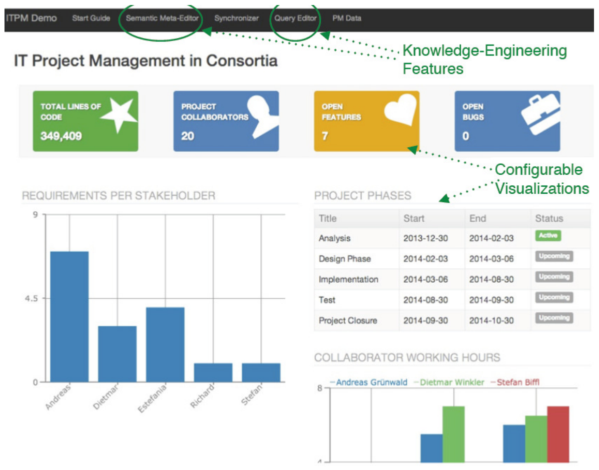
- Collaborator (22)
- Issue (7)
- Metric (4)
- OrganizationalPosition (10)
- Product (7)
- Project (1)
- ProjectPhase (5)
- ProjectRole (14)
- Requirement (7)
- SoftwareModule (54)
- Stakeholder (5)
- Team (3)
- TestCase (1)
- TestResult (442)
- TestRun (1)
- TestSuite (434)
- TimeEntry (6)
- TimeSheet (3)
- UnitTest (4)

Figure 3. Requirements Editing (Sample Generated Form).

The **Model Repository (ModelRepo - E4)** enables a centralized maintenance of multiple ontologies, respectively models. A major benefit of the ModelRepo is the capability to add and modify models (also generated with external and vendor-specific editors) with SMEd. Each model is stored in OWL and in an internal (JSON) file to simplify the mapping between JSON and Java, and to process AJAX requests more efficiently.

4.2 Integration in SED

As described in Section 4.1, the SMEd and related outputs are integrated within the SED in order to facilitate efficient modeling, integration, and monitoring of ontologies as well as engineering tools (e.g., for requirements management). Figure 4 presents a configuration prototype of the central engineering dashboard (ED) based on configurable SPARQL queries. All elements can be edited in-place via the SQEd interface. The SQEd provides settings to control positioning and visibility of elements within the ED. Elements that are disabled for the central ED are available within a sub-view of the SDBROW. The Synchronizer Cockpit (see Figure 1) aims at binding existing tool connector implementations to available tools and software instances and to provide settings for automated data synchronization within the OpenEngSB framework. Once a tool instance is configured, and the streaming is activated, all views and instances within the SED are refreshed automatically. Changes, either performed within the local tools, or within the central UI, are automatically propagated to all other tool instances, and remain traceable.



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 4. Engineering Dashboard (ED) Configuration Prototype for IT-Project Management.

Semantic Query Editor (SQEd). The Semantic Query Editor (SQEd) leverages the configuration of aggregated data visualizations and reports. The editor supports the visualization of tables, diagrams (e.g., bar charts), graphs and hierarchies, and key performance indicators (KPIs). Further, the SQEd supports the integration of various query languages. For the prototype, SPARQL has been implemented. The SPARQL editor incorporates simple syntax highlighting and auto-completion, based on JavaScript[9] and available SMEd models. For instance, as soon as an entered prefix matches with the namespace of an existing OWL ontology, the available OWL classes, data properties, and object properties are displayed, and can be filtered. The result of SPARQL queries can be previewed in table format.

5. USE CASE: IT-PM DATA INTEGRATION

To demonstrate the feasibility of the SMEd and SED prototypes, we focus on a use case to support IT project management in distributed and heterogeneous engineering environments. The basic requirements have been derived from our industry partner, a large-scale hydro power plant development and integration organization. Four locally distributed and heterogeneous project

[9] CodeMirror: http://codemiror.js

partners join their forces in a project consortium in order to implement an innovative and complex product, i.e., a hydro power plant (see Figure 5 for the organization hierarchy and the communication flow in the project consortium). Note that a hydro power plant includes a significant number of software components that control individual parts of the system. Organization A represents the key partner, responsible for controlling activities, and the definition of requirements. Organization B is responsible for project management. Both, organization A and organization C provide development teams. Organization D focuses on quality management tasks and activities. Main characteristics and challenges of the project are: (a) the project manager has to deal with a large amount of project data, e.g., a large set of requirements; (b) consortium members apply different types of PM software systems, processes, and data formats; (c) individual project partners collaborate in such a project consortium for the first time. As expected, the exchange and access of/to data throughout the project is challenging. Initially, the teams cannot access each other's PM systems because of individual local data formats; and (d) time and resources for the rollout of an integration solution are sparse.

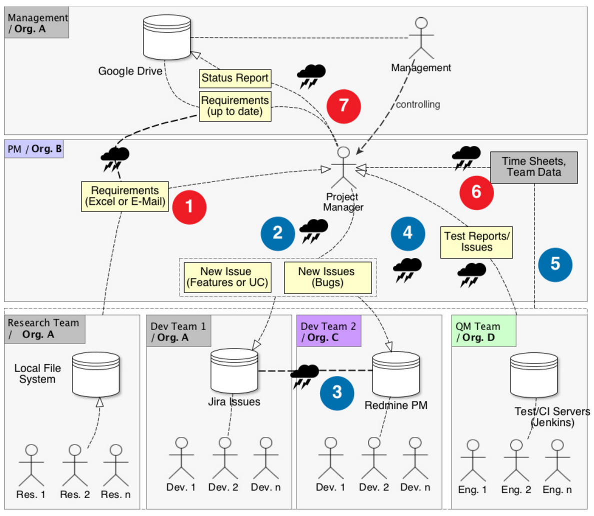


Figure 5. Communication Plan including the Identified Integration Scenarios.

Based on the identified communication plan (Figure 5), the project manager identified seven scenarios, which require high manual integration efforts, are error-prone, or which can only be carried out inefficiently manually. Automating these scenarios include (a) *Integrating various tool data within the same domain*, i.e., Scenario 1: Synchronization of requirements across different formats and systems and Scenario 3: Synchronization of PM issue tracking tools; (b) *Exchange of data across different tool domains*, i.e., Scenario 2a: Transformation of requirement into issues, Scenario 2b: Propagation of changes into the local issue tracking tools, and Scenario 4: Creation of user accounts for the QM team, and (c) *Generation of reporting data and the monitoring of project data*, i.e., Scenario 5: Creation of test reports and charts, Scenario 6: Integration and monitoring of time efforts, and Scenario 7: Derivation of management figures and KPIs (e.g., number of requirements per stakeholder, number of open features before an upcoming deadline, or tracing of updated artifacts).

Because of space limitations, in this paper we focus on conceptual evaluation of Scenario 1 (i.e., synchronization of different artifacts, i.e., requirements), Scenario 6 (i.e., integration and querying of heterogeneous data sets, e.g., time sheets), and Scenario 7 (i.e., derivation of management figures and KPIs).

6. RESULTS AND EVALUATION

To demonstrate the feasibility of the prototype, we evaluated SMEd and SED in context of the project consortium use case (see Section 5) with focus on three different important scenarios. The first part of the evaluation demonstrates the feasibility of SMEd and SED in the context of a large-scale (project management) integration scenario. Based on management requirements, we demonstrate the features of SED for the (continuous) integration of heterogeneous data models and the derivation of aggregated data visualizations. The second part of the evaluation addresses the technical integration process, i.e., the negotiation and design of common data models, and the tool integration approach. The prototype implementation is available online.[10]

6.1 Evaluation of the Management Utility

Due to page limitations, we do not include all seven integration scenarios in detail but we focus on three representative scenarios to demonstrate the capabilities of the prototype.

Scenario 1: Mapping of Requirements between Similar Tools. To enable the synchronization between tools within the same domain, the following steps are performed.

(a) Model Negotiation and Design: (1) In a joint session, key stakeholders (i.e., the management) and technicians meet to negotiate and define the global model (i.e., common concepts). The model negotiation is based on the SMEd. (2) The meeting participants sketch the main concepts (e.g., classes) of the local tool models, which should be considered within the SMEd. (3) The local models are refined. Conflicts that hamper synchronization between the global and the local concepts are resolved. In case of severe conflicts, the project manager or other key stakeholders are consulted. Conflicts can be communicated directly within the SMEd (via model annotations and comments). The global model must fulfill the following criteria: (a) the data structures of the model should be intuitive and understandable to managers; (b) the global model must reflect all elements (i.e., classes, attributes, and associations) of the local tools to be integrated; (c) participants of the modeling session verify whether the model is capable of deriving required aggregated information and knowledge based on the designed models (e.g., via SPARQL queries).

(b) Technical Integration of the Models: (4) Technical integration is based on the agreed models (see previous steps). Connectors code (e.g., Beans and DAOs) is generated directly via the SMEd. A technician extends the generated code (DAOs) and refines the validation of specific model elements, e.g., attributes. Based on the generated beans, mappings between local and global models take place in Java (essentially M2M). The connector methods are implemented to enable the interaction with the tool-specific APIs, workflow engines, and synchronization mechanisms; (5) Implementation of (additional) unit and/or integration tests for verification and validation purposes; (6) Deployment of the connector to the integration platform and binding of specific tool instances to the synchronizer cockpit of the SED. Figure 6 presents the negotiated models that serve as the starting point for the tool integration, including code generation, implementation of Model-to-Model (M2M) mappings, and

[10] Prototype: https://bitbucket.org/agruenwald/custom-itpm-builds/downloads

connector implementation. According to the use case, introduced in Section 5, the requirements must be integrated from *MS Excel* (Figure 6; 1) and *Google Spreadsheet* format (Figure 6; 2). Each spreadsheet row contains a single requirement. The structure of the requirements is similar to those of typical industrial software, e.g., IBM Doors. Mapping of fields and associations is straightforward. For instance, the main fields of the local requirement classes (*ID*, *name*, *description*, *acceptanceCriteria*, and *priority*) can be mapped directly. Interrelations with other concepts via their IDs (e.g., *parentId*) are transformed into explicit associations (e.g., *subRequirement*). The data browser (SDBROW) enables the convenient browsing and editing of these associations, and the SQEd can be used to derive additional reporting features (e.g., complexity of requirement based on the number of (in)direct dependencies). The only hurdle is to resolve the mismatch between the formats of assigned stakeholder IDs (*stakeholderIDs* vs. *stakeholderNames*). As each consortium partner maintains a spreadsheet (Figure 6; 3), containing a list of team members (e.g., *id name* and *email)*, their *team* assignment, and the hierarchy (*superiorId*), these data can be leveraged to close the information gap between the different concepts by utilizing the imported *Collaborator* data (Figure 6; 4).

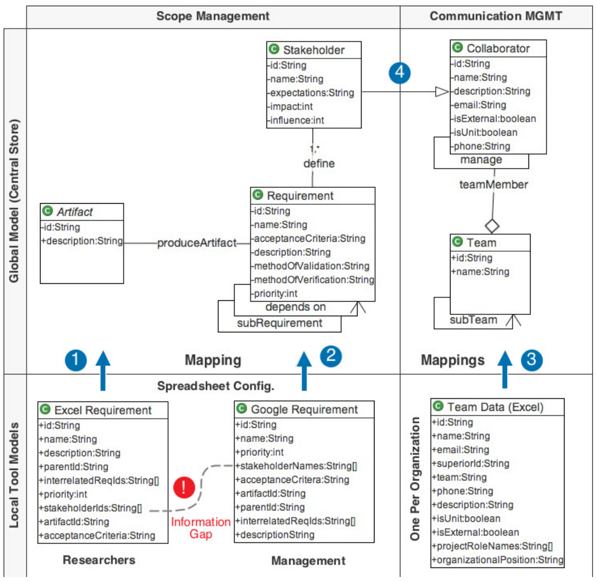
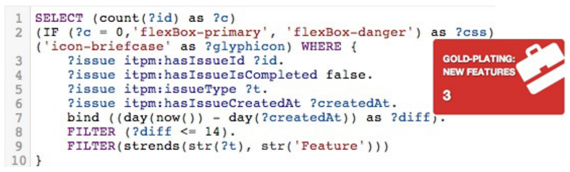


Figure 6. Mapping between Local and Global Requirement Concepts based on the output of the SMEd.

Scenarios 6 & 7: Derivation of Aggregated Data Visualization. Integrated and aggregated data can help engineers and managers to receive up-to-date information derived from a heterogeneous set of distributed data. The SQEd is utilized to formulate a SPARQL query, to enable project managers in monitoring the number of added features throughout the past 14 days.



```
1  SELECT (count(?id) as ?c)
2  (IF (?c = 0,'flexBox-primary', 'flexBox-danger') as ?css)
   ('icon-briefcase' as ?glyphicon) WHERE {
3      ?issue itpm:hasIssueId ?id.
4      ?issue itpm:hasIssueIsCompleted false.
5      ?issue itpm:issueType ?t.
6      ?issue itpm:hasIssueCreatedAt ?createdAt.
7      bind ((day(now()) - day(?createdAt)) as ?diff).
8      FILTER (?diff <= 14).
9      FILTER(strends(str(?t), str('Feature')))
10 }
```

Figure 7. SPARQL Query (Semantic Query Editor) and Resulting Dashboard KPI.

Figure 7 presents the SPARQL query using the SQEd's syntax highlighting format. The resulting KPI (Figure 7, right side) is displayed in the central engineering dashboard (ED). Each output format has a specified set of reserved parameter names facilitating the customized formatting of the output. In the example, presented in Figure 7, the output format `KPI' is chosen. *?c* holds the value of the KPI (e.g., 3). *?css* holds the name of a CSS class that is used for formatting, and *?glyphicon* can be used to change the icon. As the SED utilizes the Bootstrap CSS framework, any predefined Bootstrap CSS class and any icon of the embedded Glyphicon library is available. Line 2 of the SPARQL query states that the KPI should be colored in red (*flexBox-danger*), if the number of created features in the past 14 days exceeds 0. Otherwise the KPI is colored turquoise (*flexBox-primary*). The icon called 'icon-briefcase' is used as output.

6.2 (Technical) Modeling and Integration

The second part of the evaluation focuses on analyzing the capability of the SMEd for modeling support and the technical integration of tool connectors and related UIs. As detailed historical data for the implementation of three industrial connectors already exist, we compare the historical implementation effort with the new MDSE-based approach, which utilizes the SMEd / SED. The latter was performed in the context of the presented consortium use case, in order to make efforts comparable. The integration includes three types of data: (a) requirements, including information about stakeholders and related work packages; (b) Collaborator data, including names, ids, and contact information; and (c) team data, including team hierarchies and team affiliation (see Figure 5; Scenario 7).

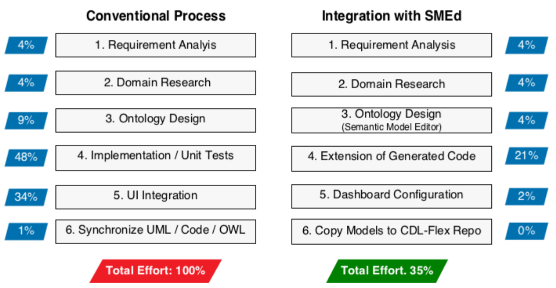


Figure 8. Comparison of a Traditional Integration Process (100%) vs. SMEd-Based Approach (effort reduced to 35%).

Figure 8 compares the main process steps and the share of design and development effort of the traditional integration approach with the SMEd-based approach. The performed process steps in the **traditional approach** (see Figure 8, left side) include: (Step 1) Collection of requirements for tool connectors and UI interfaces; (Step 2) Identification of available ontologies and data formats for each tool connector (i.e., requirements, collaborators, and teams); (Step 3) Design of related OWL ontologies, including discussions with experts and managers. Because OWL ontologies are not communicable to the manager and also hard to discuss with the domain experts, the ontology was initially modeled in UML. Changes and updates, either in the OWL ontology or within the UML class diagram, had to be executed and synchronized manually in order to assure accurate communication and discussion of the design artifacts. Synchronization of redundant models (i.e., OWL and UML) and formats was tedious and inefficient; (Step 4) Implementation of tool connectors in order to retrieve the data from different sources sources (e.g., spreadsheet solutions). This step involved the transformation of the OWL ontologies into code, and the mapping of the beans based on the notations of the integration platform, i.e., the OpenEngSB; (Step 5) Elicitation and implementation of user interface requirements;

and (Step 6) Manual synchronization of code and OWL. The final ontology was transformed into an UML class diagram for documentation and communication purposes. Based on the **SMEd-based approach** the following improvements were implemented (Figure 8, right side): (Step 3) The ontology and model design was cut in half, because there was no need to duplicate the effort for creating/updating UML diagrams and OWL ontologies. Based on the SMEd approach, the OWL ontology can be derived from the UML class diagram annotation. In addition, communication and negotiation of data models (presented within the web interface) turned out to be more efficient; (Step 4) The effort for the tool implementation was cut significantly, because Beans, DAOs, and Unit Tests could be derived automatically; (Step 5) No costly UI implementations were necessary because basic views (i.e., table and form) are automatically derived from the semantic models. In addition, the team graph visualization can be configured directly within the SQEd; and (Step 6) because all design and implementation artifacts are consistent, no adjustments are required at the end of the project. Thus, the implementation effort, based on the MDSE-based approach, was reduced by 65%.

7. CONCLUSION

In this paper we introduced the Semantic Model Editor (SMEd), which has been integrated into the Semantic Engineering Dashboard (SED). The SMEd supports the design of OWL ontologies especially applicable for non-experts in Semantic Web technologies. The application of SMEd and SED reduces the setup costs and supports the integration of engineering tools. The results of the prototype evaluation in a real-world scenario showed that SMEd supports non-experts in semantic technologies in better developing and maintaining ontologies and queries. In addition, the SMEd reduces the effort for implementing redundant artifacts significantly. By reutilizing models, (a) code (e.g., Beans, DAOs, and Tests) and UI views can be derived and (b) Semantic Web tools, e.g., a Semantic Query Editor (SQEd), can be implemented.

We conclude that the suggested mappings between UML models, OWL ontologies, code, and UIs are quite intuitive and useful. The maintenance of a single model version in favor of several redundant artifacts (a) reduces redundant development work significantly, (b) improves the consistency, accuracy, and timeliness of design artifacts and code, (c) leads to more accurate models, because more time can be spent on design activities rather than on manually evaluating designed models (i.e., reasoning and code generation mechanisms support model verification and validation), and (d) supports the communication because compact models are easier to understand (also high-level stakeholders and respectively non-technicians can interpret the UML models, at least with the support of a domain expert).

Future work will include the extension of the SED and evaluation in additional real-world use cases. One major research challenge is to provide a user-friendly graphical editor to enable the graphical mapping between model elements.

8. ACKNOWLEDGEMENT

This work has been supported by the Christian Doppler Forschungsgesellschaft, the Federal Ministry of Economy, Family and Youth and the National Foundation for Research, Technology and Development, Austria.

9. REFERENCES

[1] Abels S., Ahlemann F., Hahn A., Hausmann K., and Strickmann J.: "Promont – a Project Management Ontology as a Reference for Virtual Project Organizations", In: *Proc. of OTM Workshops*, pp.813–823, 2006.

[2] Ahlemann F.: "Towards a conceptual reference model for project management information systems". In: *Int. Journal of Project Management*, 27(1), pp.19–30, 2009.

[3] Biffl S., Schatten A., and Zoitl A.: "Integration of Heterogeneous Engineering Environments for the Automation Systems Lifecycle" In: *Proc. of INDIN*, pp.576-581, 2009.

[4] Biffl S. and Schatten A.: "A Platform for Service-Oriented Integration of Software Engineering Environments". In: *Proc. of SoMeT*, pp.75-92, 2009.

[5] Bizer C., Heath T., and Berners-Lee T.: "Linked Data - The Story So Far", In: *J. Sem. Web and IS*, 5(3), pp.1–22, 2009.

[6] Brambilla M., Cabot J., and Wimmer M.: "Model-Driven Software Engineering in Practice" In: *Lectures on SE*, 2012.

[7] Dippelreiter B.: "SEMPROM - Semantic Based Project Management", *Dissertation,* TU Vienna, Austria, 2012.

[8] Dong H., Hussain F.K., and Chang E.: "Application of Protégé and SPARQL in the Field of Project Knowledge Management", In: *Proc. of ICSNC*, pp.74–74, 2007.

[9] Fay A., Biffl S., Winkler D., Drath R., and Barth M.: "A Method to Evaluate the Openness of Automation Tools for Increased Interoperability", In: *Proc. of IECON*, 2013.

[10] Gasevic D., Djuric D., Devedzic V., and Damjanovi V. "Converting UML to OWL ontologies", In: Proc of the Int. *WWW Conference,* 2004.

[11] Grünwald A. and Moser T.: "umlTUowl - A Both Generic and Vendor-Specific Approach for UML to OWL Transformation", In: *Proc. of SEKE*, pp.730-736, 2012.

[12] Grünwald A., Winkler D., Serral E., and Biffl S.: "Semantic Technologies to Accelerate Model-Driven Development", In: *Proc. of Euromicro SEAA*, WIP, 2013.

[13] Grünwald A.: "Evaluation of UML to OWL Approaches and Implementation of a Transformation Tool for Visual Paradigm and MS Visio", Technical Report, IFS-CDL 14-42, TU Vienna, July 2014, Online available at http://qse.ifs.tuwien.ac.at/publication/IFS-CDL-14-42.pdf.

[14] Henss J., Kleb J., Grimm S., and Fraunhofer I.: "A Protégé 4 Backend for Native Owl Persistence, In: *Int. Protégé Conference*, 2009.

[15] Parreiras F.S., Staab, S., and Winter A.: "TwoUse: Integrating UML models and OWL ontologies", In: *Technical Report, 16/2007, University of Koblenz*, 2007.

[16] Ruiz-Bertol FJ., Rodríguez D., and Dolado J.: "Applying Rules to an Ontology for Project Management", In: *Proc. of JISBD*, pp.5-7, 2011.

[17] Settas D. and Stamelos I.: "Using Ontologies to represent Software Project Management Antipatterns", In: *Proc. of SEKE*, pp.604–609, 2007.

[18] Völkel M. and Sure Y.: "Rdfreactor - from ontologies to programmatic data access". In: *Poster Proc. of Semantic Web Conference,* pp. 55, 2005.

[19] Xu Z., Ni Y., Lin L., and Gu H. "A Semantics-Preserving Approach for Extracting OWL Ontologies from UML Class Diagrams", In: *Database Theory and Application*, 2009.