Reconciling Practice and Rigour in Ontology-based Heterogeneous Information Systems Construction

Carme Quer[1[0000-0002-9000-6371]], Xavier Franch[1[0000-0001-9733-8830]], Cristina Palomares[1[0000-0003-4722-5584]], Andreas Falkner[2], Alexander Felfernig[3[0000-0003-0108-3146]], Davide Fucci[4], Walid Maalej[4], Jennifer Nerlich[5], Mikko Raatikainen[6[0000-0002-2410-0722]], Gottfried Schenner[2], Martin Stettinger[3], Juha Tiihonen[6]

[1]Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
{cquer, franch, cpalomares}@essi.upc.edu
[2]Siemens AG Österreich, Vienna, Austria
{andreas.a.falkner,gottfried.schenner}@siemens.com
[3]Graz University of Technology, Graz, Austria
{felfernig,stettinger}@ist.tugraz.at
[4] University of Hamburg/HITeC
{fucci,maalej}@informatik.uni-hamburg.de
[5]Vogella GmbH
jennifer.nerlich@vogella.com
[6]University of Helsinki
{mikko.raatikainen,juha.tiihonen}@helsinki.fi

Abstract. Ontology integration addresses the problem of reconciling into one single semantic framework different knowledge chunks defined according to its own ontology. This field has been subject of analysis and many consolidated theoretical results are available. Still, in practice, ontology integration is difficult in heterogeneous information systems (HIS) that need to integrate assets already built and running which cannot be changed. Furthermore, in practice, the composed assets are usually not really defined according to an ontology but to a data model which is less rigorous but fit for the purpose of defining a data schema. In this paper, we propose a method for integrating assets participating in a HIS using a domain ontology, aimed at finding an optimal balance between semantic rigour and feasibility in terms of adoption in a real-world setting. The method proposes the use of data models describing the semantics of existing assets; their analysis in order to find commonalities and misalignments; the definition of the domain ontology, considering also other sources as standards, to express the main concepts in the HIS domain; the connection of the local models with this domain ontology; and its abstraction into a metamodel to facilitate further extensions. The method is an outcome of a collaborative software development project, OpenReq, aimed at delivering an ontology for requirements engineering (RE) designed to serve as baseline for the data model of an open platform offering methods and techniques to the RE community. The construction process of this ontology will be used to illustrate the method.

Keywords: ontology integration; heterogeneous information systems; domain ontology; requirements engineering ontology.

1 Introduction

Modern information systems are rarely monolithic, but instead they are heterogeneous, composed of different subsystems that altogether provide the required functionality. Quite often, these subsystems follow their own rules and manage their own data schemas, which need to fit together at different levels, from conceptual (e.g., to provide a consolidated vocabulary) to operational (e.g., to allow their interoperability). This is especially true in collaborative software development projects, where different organizations bring some existing assets that need to be combined into a holistic system.

The reconciliation of the different data schemas can be implemented through ontology integration. An ontology defines an explicit specification of a conceptualization [1]. Ontologies are tightly related to other conceptual modelling artifacts as modelling languages and metamodels [2]. Ontology-based data model integration addresses the problem of building a new ontology for heterogeneous information systems (HIS) composed of subsystems that need to interoperate [3]. Methods for ontology integration have been proposed for more than 20 years (see Section 2), but integration in real settings remains a challenging problem due to several reasons. Among them, there is the need to find an adequate trade-off between rigour in the integration and feasibility in terms of return on investment for the organizations involved.

The need for such practical method became evident during the OpenReq collaborative software development project in which the authors are participating (www.openreq.eu). The main goal of OpenReq is to develop, evaluate, and transfer highly innovative methods, algorithms, and tools for community-driven RE in large and distributed software-intensive projects. To this end, four universities and five companies from Europe collaborate in the deployment of a platform providing services to the community. The platform will be built upon a data schema derived from a domain ontology for RE which should reconcile a global perspective to satisfy the requests of the community and a local perspective to integrate the current assets, techniques and needs from all the project partners. The purpose of the ontology is thus supporting the development and integration of techniques while serving as a reference framework for the community.

In this context, the present work addresses the following research goal:

Research Goal. To propose a method based on domain ontologies to integrate the data models of assets participating in a HIS with optimal trade-off of semantic rigour and feasibility in terms of implementation effort and adoption in a real setting.

The rest of the paper is organized as follows. Section 2 provides the background. Section 3 presents the context of our research. Section 4 shows the method proposed to construct the domain ontology applied in the OpenReq case, which is developed in details in sections 5 to 10. Finally, Section 11 conducts some discussion and identifies future work.

2 Background

The application of ontology integration in the context of HIS was claimed by Sowa [4], who also identified two possible ways to proceed: replace the original ontologies by the new one, or use the new one as an intermediary between the HIS. This second option seems more appropriate when integrating models with little room for changes, as it is the case for the context that we are addressing in this paper.

Calvanese et al. [5] formally defined the semantics of integration in this scenario. It is characterised by a mapping between the new ontology (called global ontology) and local ontologies (which are used as a starting point for the integration). This mapping can be defined adopting either a global-centric approach or a local-centric approach. In the global-centric approach, every term in the global ontology has associated a view (i.e., a query) over the local ontologies, while in the local-centric approach every term in a local ontology is mapped onto a view over the global ontology.

In our work, we follow this idea reviewed also in De Giacomo et al. [6]. In this kind of integration the ontology is a formal and conceptual view and constitutes the component to which the clients of the integrated information systems use to interact with them. In our case, we consider also as clients the own information systems integrated that use the ontology to interact with the rest of the system. Thus, the ontology provides the semantic data integration of the heterogeneous information systems.

3 The Context: The OpenReq Project

The OpenReq collaborative development project will support (see Fig. 1): a) the automated identification of requirements from different knowledge sources (e.g., communities or natural language text documents); b) the personal recommendation of requirements as well as requirement-related aspects (such as quality tips or requirement metadata fields) and stakeholders; 3) the support of group decision making (e.g., in release planning) by providing a solution that fulfills all users preferences or indicates the conflicts that need to be solved to provide a solution; 4) the automated identification of (hidden) dependencies between requirements. OpenReq will provide an open source tool and a set of APIs that will integrate these innovative technologies applied to RE.

OpenReq is a classic example of a collaborative software development project that needs to address contradicting challenges for producing a HIS, as defined in the introduction:

- Different partners bring to the project their own assets in the form of implemented software components. The partners developed these components for their own purposes and they do not want to change their data model and (underlying) ontology.
- These assets need to be reconciled because the project aims at delivering a single, unifying platform. Furthermore, given that the platform shall be open to the RE community, it is utterly important that the resulting domain model is cohesive and general-enough.

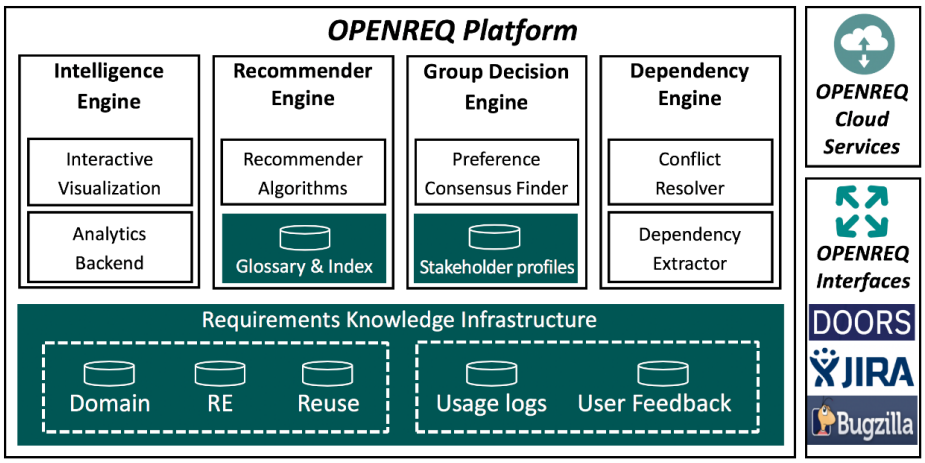


Fig. 1. The OpenReq approach to RE.

4 The Method

In this section, we briefly present the method proposed for ontology-based integration in the context stated above, emerging from our experience in the OpenReq project. The method is composed of five steps (see Fig. 2) briefly enumerated below and developed in detail in the next sections.

- **Step I**: *Creation of the Baseline*. Obtain (if they do not exist yet) the local data model of each of the existing assets to be reconciled.
- **Step II**: *Analysis of the Baseline*. Local data models are aligned to understand which are the core concepts and identify possible contradictions and variants.
- **Step III**: *Definition of the Domain Ontology*. From the previous analysis and considering conveniently standards and other reference models for the domain, the domain ontology is defined around the core concepts, integrating the different variants and solving all detected contradictions.
- **Step IV**: *Mapping among the Local Data Models and the Domain Ontology*. In order to support the semantic alignment of the existing assets, the mapping among the local data models and the domain ontology is defined.
- **Step V**: *Definition of the Metamodel*. With the purpose to better structure the domain ontology and to support easier maintainability, a metamodel is built abstracting the concepts appearing in the domain ontology.

5 Step I: Creation of the Baseline

In this first step, the goal is to represent in the same modelling paradigm the ontologies that the different partners participating in a collaborative development project are currently using as semantic framework for their assets. This way, we avoid two of the

main types of ontology mismatches that could have make the integration process harder: paradigm heterogeneity and language heterogeneity [7].

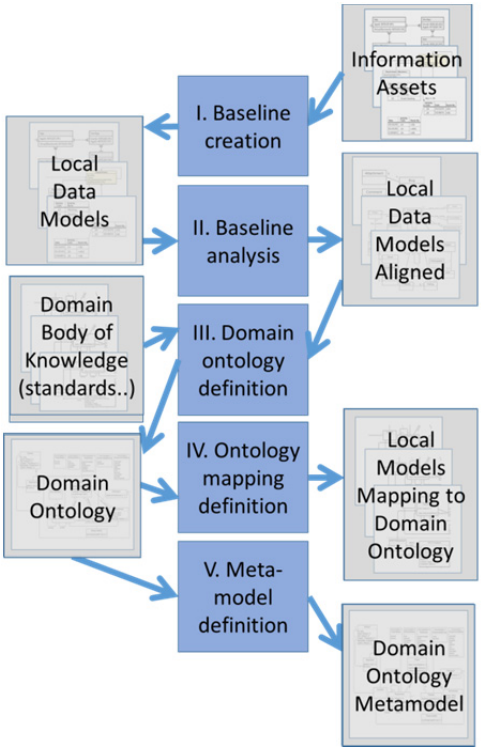


Fig. 2. The OpenReq approach to the definition of a domain ontology for RE.

6 Step I: Creation of the Baseline

In this first step, the goal is to represent in the same modelling paradigm the ontologies that the different partners participating in a collaborative development project are currently using as semantic framework for their assets. This way, we avoid two of the main types of ontology mismatches that could have make the integration process harder: paradigm heterogeneity and language heterogeneity [7].

As already stated, it will not be usual to have fully-fledged ontologies defining the assets to be integrated into the HIS. Therefore, we propose to use data models to describe such assets, as simplified representation of the (underlying) ontologies. More precisely, we propose UML class diagrams plus a vocabulary of terms for each input model, since the idea is to use UML for the domain ontology representation. The use of UML class diagrams in ontology representation is quite usual and well-established [8] and has two advantages. On the one hand, class diagrams for the assets may already exist or otherwise, they are easy to build from a database schema, which is a technical artefact that can be assumed to exist. On the other hand, class diagrams

(particularly, in UML) are a widespread notation that usually will not require any kind of training. Working with UML instead of other more accurate formalisms has one drawback, namely the limitations in reasoning capabilities. If such capabilities were required, we could still apply this same method, using other ontology representation language, e.g., based on Description Logics.

The OpenReq case. In OpenReq, the baseline is composed of five local data models (and associated vocabulary) that we identify hereafter with the acronym of the partner[1]:

- The UPC data model. Quite general, it is a subpart of the PABRE system conceptual model for requirements reuse [9].
- The TUGRAZ data model. In addition to some general-purpose concepts for requirements, this model also includes concepts related to release planning (i.e., distributing the requirements into releases).
- The HITEC data model. It is based on the concept of user feedback (as source of requirements), expressed as comments and ratings in an app stores [10] and social media [11], but also as usage data obtained, for example, from handheld devices.
- The SIEMENS data model. This model is specific for requests for proposal (RFPs) in an industrial context such as rail automation.
- The VOGELLA data model. It comprises very few classes that correspond to the Bugzilla system used by the Eclipse project to maintain issues.

In all the cases, the partners represented such data models with UML class diagrams, which were leveraged in this first step by means of a vocabulary including all the relevant concepts introduced in the diagram. For the sake of illustration, just to understand the big diversity that we may find in such collaborative development projects, Fig. 3 shows the class diagrams of two partners, UPC and VOGELLA.

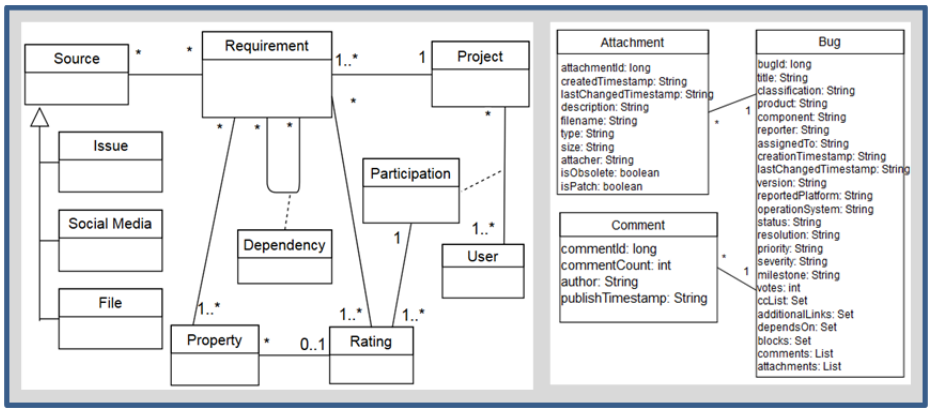


Fig. 3. Fig. 3 Class diagrams included in the baseline: UPC (up) and VOGELLA (down) (UPC data model does not include attributes due to space reasons).

[1] We refer to authors' affiliations in the first page

7 Step II: Analysis of the Baseline

The main purpose of this step is to analyse the local models and vocabularies that compose the baseline for their later alignment. In a collaborative software project, we may expect different interests from all the partners, different contexts, scopes, etc. In addition, every domain may have its own additional challenges, e.g. heterogeneity of data sources. Therefore, it is utterly important to profoundly understand this variety to find alignments, overlapping, contradictions and differences, i.e., data heterogeneity.

In general, we may expect several data heterogeneity causes to emerge [12]: schematic (e.g., same concept with a different name), semantic (e.g., same name for different concepts) and intensional (e.g., fundamental differences in the domain).

The OpenReq case. In OpenReq, the central concept around which all models revolve is that of requirement. This concept is represented explicitly in three models, and remarkably in two of them (UPC and TUGRAZ) its notion is quite similar. However, since the attributes are slightly different, it cannot be said that the concept is exactly the same. In the third case (i.e., SIEMENS) the concept of requirement is explicitly defined in the ontology, but in fact it is wider: in addition to requirements, pieces of text that are candidate to become requirements, and pieces of text that have been assessed and finally discarded as requirements, also are included in this concept. In the other models, the requirement concept as such does not exist. Instead, two related concepts appear, namely bug (VOGELLA) and users' feedback (HITEC). Both concepts are a potential source of requirements. Fig. 4 summarizes these classes.

For the sake of illustration, Table 1 exemplifies the main causes of data heterogeneity at the level of attributes for the requirement concept considering three of the local data models. Note that it may be the case that more than one heterogeneity cause occurs for a given attribute. An extended version of this table including all the models and all the causes could be considered the outcome of this step.

Table 1. Examples of the main causes of data heterogeneity at the level of attributes in three of the local data models with respect to the notion of requirement.

SIEMENS	TUGRAZ	UPC	Data heterogeneity
id: long	---	ID: Integer	*Semantic*: different scale for the same attribute *Intensional*: attribute is not considered in all the models
text: Text	description: String	Description: String	*Schematic*: different data type and name for the same attribute
type: {DEF, Prose, Not Classified}	---	---	*Intensional*: attribute is not considered in all the models
---	status: Enum	---	*Intensional*: attribute is not considered in all the models
---	creationDate: DateTime	CreatedAt: DateTime	*Schematic*: different name for the same attribute *Intensional*: attribute is not considered in all the models
---	priority: Float	Priority: Integer	*Schematic*: different name for the same attribute *Semantic*: different scale for the same attribute *Intensional*: attribute is not considered in all the models

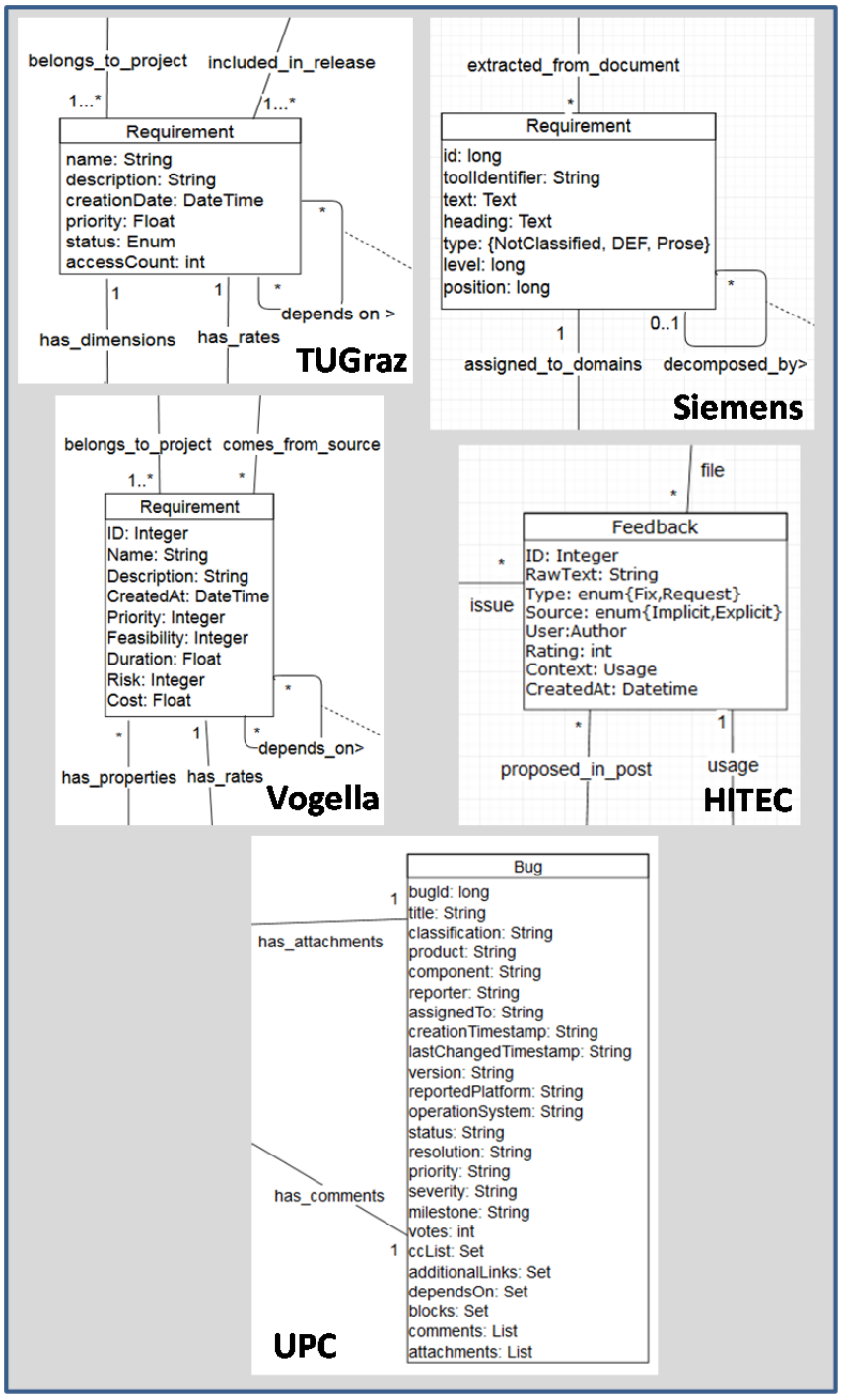


Fig. 4. The concept of requirement and related concepts in the 5 baseline data models.

8 Step III: Definition of the Domain Ontology (M1)

In the third step, the analysis made in the previous step is consolidated into a domain ontology, again considering the needs outlined in Section 3: it is required to satisfy the needs of the different project participants (i.e., asset providers), while opening the space to accommodate new, sometimes unforeseen evolutionary paths. The domain ontology will be represented through a UML class diagram plus associated glossary to define all the classes, attributes and associations appearing therein.

The main task in this process is to solve the heterogeneity causes identified in Step II. Schematic causes are the easiest to solve, while intensional causes are the most difficult and require a strong decision based on the purposes of the domain ontology. The use of bodies of knowledge pertaining to the domain, e.g. in the form of standards, can help to make decisions in this process.

The OpenReq case. From the analysis above, it is clear that the `Requirement` class is central to the domain ontology. We present in detail the consolidation of this concept from the analysis carried out in Step II (for the rest of the ontology, due to space limitations, not all details are reported):

- The most fundamental intensional heterogeneity is agreeing on the concept of requirement itself. We decided to define requirement according to the IEEE standard glossary of software engineering terminology [13]: "A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document". This means that all definitions need to fit this referential framework. As an additional advantage, adhering to a well-known standard paves the way for dissemination and evolution.
- The rest of intensional heterogeneities refer to attributes that are not included in all the local models (*modulo* other heterogeneities). The selection of the attributes to include is based on expert judgement and, in a project of this nature, requires the consensus from all partners, who evaluate them in terms of the impact on their assets and their goal for evolution.
- Similarly, schematic and semantic heterogeneities are solved by expert criteria. In general, they are not fundamental for the final result.

As for the rest of information conveyed in the domain ontology, requirements are structured into a hierarchy by means of a `decomposition` association. Two other relationships, namely `conflict` and `synergy`, are mentioned in several local data models, therefore we introduce two associations with the same name.
Since the ultimate concept of OpenReq is the planning of requirements into releases, we introduce the `Release` class. Releases exist in the scope of `Projects`. Requirements bound to a release are bound to the project that defined such release, made explicit with a derived association, `belongs-to`. `TeamMembers` are members of a `Project` in which they participate (i.e., `Participant`) playing a given role. `TeamMembers` may have requirements assigned.

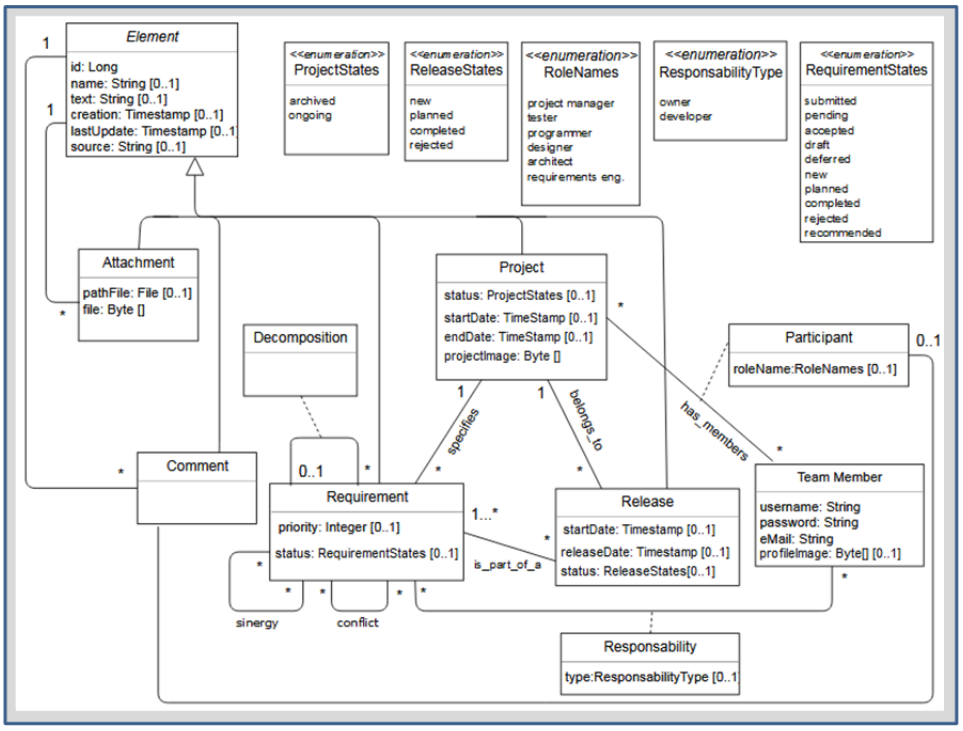


Fig. 5. The OpenReq reference model for requirements engineering.

Last, for practical reasons, we introduce one abstract class, `Element`. It is the most generic in the model and encapsulates attributes that are shared by virtually all the classes: identifier, name, description, creation date, last update date and their source. Also the associations to `Comment` and `Attachment` are related to `Element`, to allow for adding these explanatory elements to all type of elements. This generalization serves to exemplify the need of adding non-graphical integrity constraints to the model, e.g., an attachment cannot be attached to another attachment.

Fig. 5 shows the OpenReq domain ontology for requirements engineering. The domain ontology includes also the vocabulary, not reported due to space limitations.

9 Step IV: Mapping among the Local Data Models and the Domain Ontology (M1)

As mentioned in Section 2.2, mappings are a central element in the integration of HIS through ontologies [5].

The concrete definition of the mapping ultimately depends on the purpose for building the domain ontology. In some contexts, the ontology has the purpose of mediating among the local data models, e.g., interconnecting software components or data bases, or providing a single access point to a distributed data base, in which case

an implementation in the form of, for instance, SPARQL queries is required [14]. In other cases, the domain ontology is conceived as a semantic framework to provide an intermediate layer to the consumers and providers of a heterogeneous information system, facilitating the development of services on top of this layer. In this situation, a more conceptual implementation of the mapping is convenient.

We propose to implement the mapping at the level of the UML class diagrams used to represent the global ontology and local data models. In particular, we bind concepts in the local models to the domain ontology through specialization. At the end, all those concepts in the local data models which are related to the domain ontology will have their correspondence to the domain ontology, which could eventually be expressed through OCL expressions if needed. In some cases, they will be subclasses of a class in the domain ontology, in other cases there will be necessary to create a new class in the local data model that will be subclass of the domain ontology. Elements in the local ontologies not clearly related to the domain will remain independent of the domain ontology.

The OpenReq case. For the sake of space, we illustrate this step with two representative situations. First, we focus on how the different local data models connect with the `Requirement` class introduced in the domain ontology. Given that the definition of requirement in the domain ontology has been kept generic enough to accommodate the semantics of that concept in all the local models, we define a specialization from the local class (renamed into `XX-Requirement`, being `XX` the name of the partner) to the domain ontology class. The declaration of this specialization implies removing from `XX-Requirement` all the attributes and associations that are inherited from the domain ontology, e.g., id, name and text. This way, schematic and semantic heterogeneities are automatically fixed. Fig. 6(a) shows the details for one of the OpenReq cases.

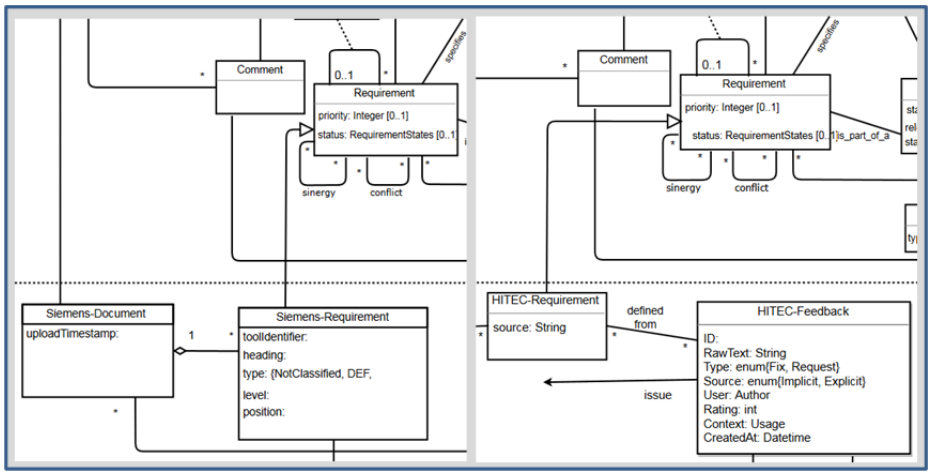


Fig. 6. Mappings with the OpenReq RE domain ontology: (a) mapping SIEMENS requirements; (b) Mapping HITEC user feedback.

Second, there are several concepts in the local data models that do not specialize any class in the domain ontology but are related. For instance, this is the case of HITEC's `Feedback` class. User feedback is not a type of requirement but one possible source of requirements; therefore, a subclass has been added to the HITEC ontology that specializes `Requirement` and a new association `definedFrom` so that a requirement may have its origin after an undetermined number of feedback instances (see Fig. 6(b)).

10 Step V: Definition of the Metamodel (M2)

Finally, we aim at consolidating the core concepts of the domain ontology into a metamodel. This allows a more compact view of the concepts at hand and support future evolution and extension of the domain ontology as new business cases and opportunities arise. Some points worth to remark are:

- The metamodel shall be such that most concepts of the domain ontology are instances of metaclasses. However, it may be the case that some concepts are not, if they are not considered in the backbone of the domain ontology. In addition, the classes introduced as abstract for convenience (in the case of OpenReq, the class `Element`, see Section 7) are not intended to be instances of any metaclass.
- The definition of the metamodel can require slight adjustments in the domain ontology. The mappings defined in the former step need to be adjusted accordingly.

The OpenReq case. We consider as starting point a metamodel for requirements proposed by UH based in the metamodel presented in [15] for the area of variability modelling. The upper part of Figure 7 shows the metamodel (M2), and the lower part shows an excerpt of some of its instantiations at the domain ontology (M1).

- Requirements as introduced in the domain ontology have been assumed to be free text in natural language. However, if we aim at having a comprehensive framework, it is necessary to allow other formats: diagrams, formulae, etc., or even natural language according to a template or user stories. Therefore, we define in M2 the `RequirementType` metaclass, which allows the definition of a requirements class in M1 that instantiates this metaclass for each format of requirements that is necessary. As attribute, apart from the `name`, the `contents` (i.e., the requirement in any type of notation) is declared as `Object`. In the current domain ontology at M1, just one class of requirement is needed, and we change the name of the `Requirement` class into `NL-Requirement` and adjust the mappings to this name change.
- In addition to the attributes included in the domain ontology, we consider that in other contexts there can be other attributes of interest. Therefore, we associate `RequirementType` to a new metaclass, `Attribute`. We relate this metaclass with a new one, `AttributeType` (note that for simplicity we do not include in Figure 7 the metaclasses corresponding to enumerates or sets). Names as identifiers are the only attributes in these two metaclasses.

- Similarly, we can think that eventually other type of relationships over requirements could be stated; therefore, we introduce a metaclass `RelationshipType` where the name of the relationship (synergy, conflict, …) is declared as attribute.

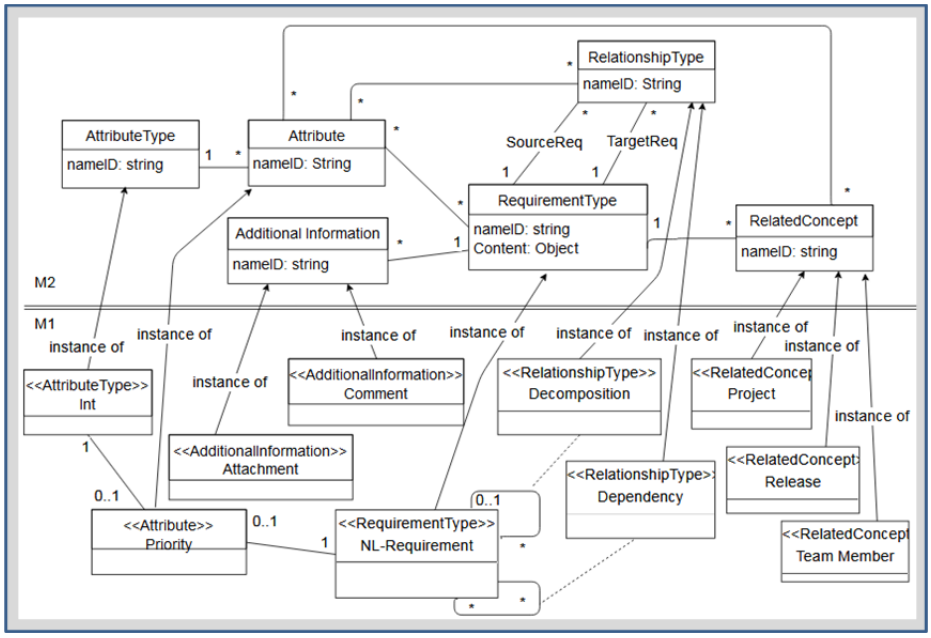


Fig. 7. The OpenReq metamodel for requirements.

On the other hand, we realize that there are two types of classes with respect to their relationship with Requirements:

- Classes extending requirements to provide richer information. These are `Comment` and `Attachment`. Note that they are optional (i.e., a requirement does not need a comment or attachment to exist). We define a metaclass `AdditionalInformation` to capture this concept.
- Classes defining elements that describe some context of the requirement. These are `Release`, `Project` and `TeamMember`. Note that they may be mandatory (i.e., a requirements needs to be defined in the context of a project) or optional (i.e., a requirement may be temporarily not assigned to any release). We define a metaclass `RelatedConcept` to capture these types of elements.

11 Discussion and Future Work

In this paper, we have presented a method to be used in the context of HIS construction in collaborative development projects. The method follows the principles of ontology-based integration balancing several somehow conflicting forces: semantic

rigour, practicality in terms of effort, fit for purpose and open adoption. The final product is a domain ontology with a corresponding metamodel, which also combines a general-purpose point of view (to serve an unforeseen portfolio of adopters) with a specific point of view (to satisfy the needs of the project partners).

The method has been used in the OpenReq project to develop a domain ontology for requirements engineering. The domain ontology and metamodel will be used to derive the schema for the implementation of OpenReq, i.e., platform and cloud services. The mappings among the local data models and the domain ontology will be used in the OpenReq interfaces to know how OpenReq concepts should be translated to local concepts. From a practical point of view, the ontology is being represented through JSON derived from the metamodel.

A lesson learned from this case study is the importance of understanding the local data models to be integrated. The role of glossaries (which are part of the ontologies) is key, and in fact they need to arrive to the level of defining the attributes. We found useful to use examples that complement definitions of terms. Even with these glossaries, some misunderstandings appeared and clarifications were needed. Thus, it was needed to ensure continuous and fluent communication among all ontology providers.

As in any other modelling endeavour, modelling was useful not only to produce a final result but also because the process of modelling uncovered some aspects in the original models that were subject of improvement, e.g. redundancies or non-optimal modelling solutions. We fixed these problems before defining the mapping.

The domain ontology obtained through our method should not be seen as a final product. For instance, we foresee that the OpenReq ontology will continue evolving as the OpenReq project does, including new elements that did not appear in the local data models. Candidate concepts at the moment are `Classifiers` (to allow grouping requirements by concepts, e.g. to organize a requirements document) and `External Elements` (such as code or tests cases) to be linked with requirements. Also new concepts may be necessary in local models of partners working on contributing to OpenReq new functionalities (see Section 3). In all the cases, the changes will be considered in order to evolve the domain ontology, and if it is necessary the metamodel.

Despite our effort to ensure validity, some threats could impact the results [16]. For instance, internal validity concerns are covered by the fact that we restrain ourselves to the retrieved evidence and expert knowledge. As for reducing reliability threats, we departed from a set of RE data models from different domains (usually with characteristics that are domain-dependent), some of them being already used by the industrial partners (i.e., SIEMENS and VOGELLA). Last, concerning external validity, we have produced a method with the aim to be general, but we need to be aware that it comes from a single experience in one particular domain, therefore further cases are needed.

Our future work spreads along several directions. First, as mentioned above, we envisage changes in the domain ontology due to evolution in the platform as the project progresses and also at the end of the project, as new local ontologies implementing additional functionalities will join the OpenReq platform.

Acknowledgments

This work is a result of the OpenReq project, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732463.

References

1. Gruber, T.R.: A Translation Approach to Portable Ontologies. *Knowledge Acquisition* 5(2), 1993.
2. Guizzardi, G.: On ontology, ontologies, conceptualizations, modeling languages, and (meta) models. *Frontiers in Artificial Intelligence and Applications* 155(18), 2007.
3. Wache, H., Voegele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., & Hübner, S.: Ontology-based Integration of Information-A survey of Existing Approaches. *IJCAI-01 workshop: ontologies and information sharing*. 2001.
4. Sowa, J.F.: Building, Sharing, and Merging Ontologies. Available at http://users.bestweb.net/~sowa/ontology/ontoshar.htm, 2001.
5. Calvanese, D., de Giacomo, G., Lenzerini, M.: Ontology of Integration and Integration of Ontologies. DL Workshop 2001 – CEUR 49.
6. De Giacomo, Giuseppe, et al.: Using ontologies for semantic data integration. In *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*. Springer, Cham, 2018. 187-202.
7. Visser, P.R.S., Jones, D.M., Bench-Capon, T.J.M., Shave, M.J.R.: An Analysis of Ontology Mismatches; Heterogeneity versus Interoperability. *AAAI Spring Symposium on Ontological Engineering*, Stanford University, California, USA, 1997.
8. Cranefield, S., Purvis, M.: UML as an Ontology Modelling Language. IJCAI 1999.
9. Franch, X., Palomares, C., Quer, C., Renault, S., de Lazzer, F.: A Metamodel for Software Requirement Patterns. REFSQ 2010.
10. Gómez, M., Adams, B., Maalej, W., Monperrus, M., & Rouvoy, R.: App Store 2.0: From Crowdsourced Information to Actionable Feedback in Mobile Ecosystems. *IEEE Software*, *34*(2), 2017.
11. Kurtanović, Z., Maalej, M.: Mining User Rationale from Software Reviews. RE 2017.
12. Goh, C.H.: *Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Sources*. PhD Thesis, MIT, 1996.
13. IEEE Std 610.12-1990 - *IEEE Standard Glossary of Software Engineering Terminology*, 1990.
14. Schenner, G., Bischof, S., Polleres, A., Steyskal, S.: Integrating Distributed Configurations with RDFS and SPARQL. confWS 2014.
15. Asikainen, T., Männistö, T., Soininen. T.: Kumbang: A Domain Ontology for Modelling Variability in Software Product Families. *Advanced Engineering Informatics* 21(1), 2007.
16. Wohlin C, Runeson P, Host M, Ohlsson MC, Regnell B, Wesslen A: *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers Norwell, MA, USA, 2012.