Facilitating Effective Stakeholder Communication in Software Development Processes

Vladimir A. Shekhovtsov(✉), Heinrich C. Mayr, and Christian Kop

Institute for Applied Informatics, Alpen-Adria-Universität Klagenfurt, Klagenfurt, Austria
{Volodymyr.Shekhovtsov,Heinrich.Mayr,Christian.Kop}@aau.at

Abstract. Effective communication in software development is impaired when parties perceive communicated information differently. To address this problem, the project QuASE has been established. It aims at a solution that supports understandability and reusability of communicated information as well as the quality of decisions based on such information. In this paper, we focus on the architectural aspects of the QuASE system and on its knowledge base which consists of two ontologies: a site ontology defining the site-specific communication environment, and a "quality ontology" that incorporates all knowledge necessary for supporting communication. We describe the overall architecture of the system, introduce the ontologies as well as their interplay, and outline the approach for gathering knowledge necessary to form the QuASE site ontology.

Keywords: Stakeholders · View harmonization · Software development process · Software quality · Communicated information · Communication environment

1 Introduction

Software development processes require a continuous involvement of the affected business stakeholders in order to be successful (this requirement, in particular, is reflected by the ISO/IEC standard for software life cycle processes [9]). A prerequisite of such involvement is establishing an appropriate communication basis for the different parties. In particular, such a basis is needed for coming to terms and agreements on the quality of the software under development. Without this, quality defects are often detected only when the software is made available for acceptance testing.

Clearly, besides of enabling effective communication, the communicated quality-related information has to be managed properly and made available during the software development lifecycle; moreover, as past-experience may help to take the right decisions, such information should be provided in a way that allows for easy access and analysis.

The QuASE project[1] [22–29] aims at a comprehensive solution for these issues (Fig. 1). In particular, this solution will provide support for managing (1) the

[1] QuASE: Quality Aware Software Engineering is a project sponsored by the Austrian Research Promotion Agency (FFG) in the framework of the Bridge 1 program; Project ID: 3215531.

S. Nurcan and E. Pimenidis (Eds.): CAiSE Forum 2014, LNBIP 204, pp. 116–132, 2015.
DOI: 10.1007/978-3-319-19270-3_8

understandability of quality-related *communicated information*, (2) the **reusability** of that information, and (3) the **quality of decisions** based on that information.

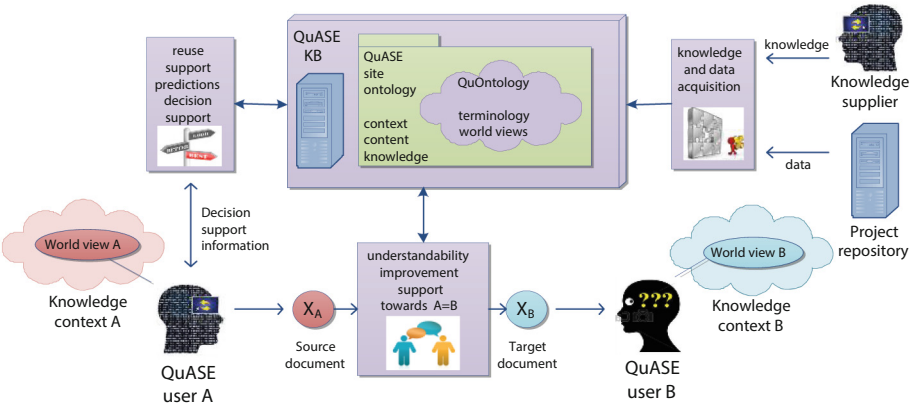

Fig. 1. General activities of the QuASE approach

Figure 1 presents different support scenarios.

1. User A and B use the *QuASE tool* to achieve a common understanding of some aspects of a software under development (SUD) that are described in a document X. In this scenario, the *QuASE knowledge base* (QuASE KB) provides the means of improving the understandability [25] of the document by translating or explaining its content where needed. Figure 1 shows the case where a source document X_A of user A is transformed into a version X_B that can be understood by B. Clearly, the translations/transformations can act in both directions.
2. User A uses the QuASE tool for decision support. As a means of such support, QuASE KB facilitates analysis of previous communications and the parties involved there-in; such information may be reused to improve the future communications, for predicting particular attributes, or for supporting decisions on the strategy of communication with the given party in the given context.

For supporting such scenarios, fundamental knowledge related to the communication domain has to be gathered and stored in the QuASE KB. There are two strategies: (1) knowledge supplier enters the data into the KB; (2) it is filled automatically from the project repositories e.g., the JIRA issue tracking database.

In the QuASE KB, the information about the given communication environment (who communicates, which documents store the communicated information etc.) has to be separated from the information about the knowledge being communicated; the latter can contain the set of core notions acting as a glue between the knowledge specific to different parties. This assures that the views of communicating parties can be harmonized [22].

The QuASE KB, therefore, consists of two parts:

1. *QuASE site ontology:* defines the site-specific communication environment;
2. *QuOntology:* contains knowledge about/around the communicated information (both common and site-specific).

The QuASE data acquisition process and the understandability management activities have been presented in previous papers [24, 25]. In this paper, therefore, we focus on the architectural aspects of the whole QuASE system, the QuASE KB and its knowledge structures.

The paper is structured as follows. Section 2 introduces the QuASE KB with its two parts and explains their features as well as their interplay. Section 3 continues with a description of the overall generic architecture of the QuASE system and the QuASE KB. It explains how other modules (e.g., QuASE site ontology generator, QuASE QuOntology generator, data acquisition engine etc.) work together. Furthermore Sect. 3 explains two alternative QuASE KB implementation strategies. Section 4 focuses on the support for gathering knowledge necessary to form the QuASE site ontology. After a short discussion of related work in Sect. 5, the paper concludes with a summary and an outlook on future research (Sect. 6).

2 Knowledge Structures for Representing the Communication Environment and the Communicated Information

We distinguish the structures representing (1) the knowledge about the communication environment (such as the communicating parties, the documents containing communicated information etc.) and (2) the communicated knowledge itself (concepts and facts being communicated, related information). In establishing such structures, we aimed at the following goals: (a) flexibility: the permitted structures have to be easily adapted to the particular deployment site; (b) support for understandability management and analysis activities.

2.1 Project Repositories

Prior to defining the knowledge structures, it is necessary to enumerate the possible sources of data corresponding to these structures. Software development projects, as a rule, keep communicated information within *project repositories* such as (1) *project databases* controlled by issue management systems (IMS), e.g., JIRA [11], MantisBT [6] and others; such databases contain communicated information in form of issues (generalizing bug reports or feature requests) and related discussions; (2) *file-based repositories* containing meeting minutes, requirement and design specifications etc.; these files are usually kept in some kind of a directory tree; (3) *wiki-based systems*.

Consequently, QuASE considers these types of repositories as sources of information and therefore provides data acquisition interfaces to them.

2.2 QuASE Site Ontology

The data collected from project repositories are interpreted and mapped into the QuASE KB. Figure 2 depicts the high-level structure of the QuASE site ontology (in UML-like notation) which consists of the following key concepts:

1. *site:* owner of the given QuASE installation, e.g. a software provider.
2. *context:* units having particular views on communicated information, e.g. projects, organizations and their departments, involved people (stakeholders) etc. Context units are characterized by context attributes and can be connected to other units; a context configuration, for example, could include the representation of the whole organizational hierarchy or the whole portfolio of projects defined for a particular IT company. In addition, the set of context units can include the categories of such units (i.e. "IT company", "Business stakeholder") as it is possible to define the views on communicated information belonging to such categories (i.e. the view on quality belonging to IT companies).
3. *content:* units shaping communicated information originated from project repositories: they serve as containers for such information or organize such containers. Examples of content units are issues and their sets, issue attribute values, and requirement specifications. Content units can be related to context units.
4. *knowledge:* units encapsulating quality and domain knowledge that is subject of communication and harmonization. Every QuASE knowledge unit is a triple (o, v, r) composed of the following components:
   (a) *ontological foundation:* a reference to the conceptualization of the particular piece of quality or domain knowledge through ontological means: such conceptualizations form a modular ontology (*QuOntology*) providing a framework for translating between world views.
   (b) *representation:* the representation of the knowledge unit in a format that could be perceived by the communicating parties (e.g. plain text). Representation units are contained in content units and can be connected to each other.
   (c) *resolution means:* the means of resolving understandability conflicts related to the particular knowledge unit (such as explanations or external references).

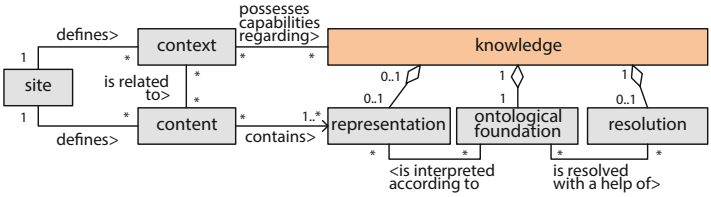

Fig. 2. Base structure of the QuASE site ontology

Context units possess *capabilities* to deal with knowledge units; in particular, these capabilities could refer to the ability of understanding a given knowledge unit at hand (i.e. its representation); or of explaining a knowledge unit using resolution means.

2.3 QuOntology

QuOntology covers all knowledge that has/can be communicated in the respective environment. Initial research on QuOntology has been published in [28], whereas the current version of the relevant conceptualizations is presented in [22]. Figure 3 shows the key components of that ontology as well as its relation to the site ontology.

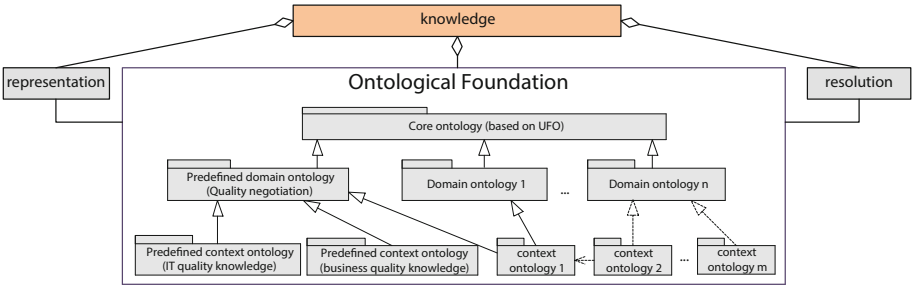

Fig. 3. Organization of QuOntology

QuOntology is organized in three layers [see also Fig. 3]:

1. *QuOntology core* represents a stable subset of the knowledge available from research and industrial practice; it does not depend on the particular problem domain. We use the Unified Foundational Ontology (UFO) [7, 8] as a foundation for QuOntology core; the motivation for this selection is provided in [22].
2. *Domain ontologies* [8] represent the specifics of the particular problem domain which is addressed by the given software under development (finance, oil and gas etc.); as a part of the project, we implement for this layer a Domain Ontology for software quality [22].
3. *Context ontologies* represent the knowledge related to particular components of the QuASE context: they contain organization-specific, project-specific etc. concepts. As a part of the project, we implement for this layer ontologies for business-specific and IT-specific views on quality.

As both QuASE site ontology and QuOntology are specialized ontologies supporting only the concepts depicted in Fig. 3, we propose to allow knowledge suppliers to use for their definition domain-specific visual modeling languages with limited set of constructs which are easier to understand and learn than ontology languages such as OWL. The models defined by means of these languages are then transformed into specific ontology instantiations. The process of specifying such models and transforming them into ontological format is described in the following section.

3 QuASE System Architecture

Figure 4 gives an overview of the QuASE system architecture. The components are described one by one in the following sections.

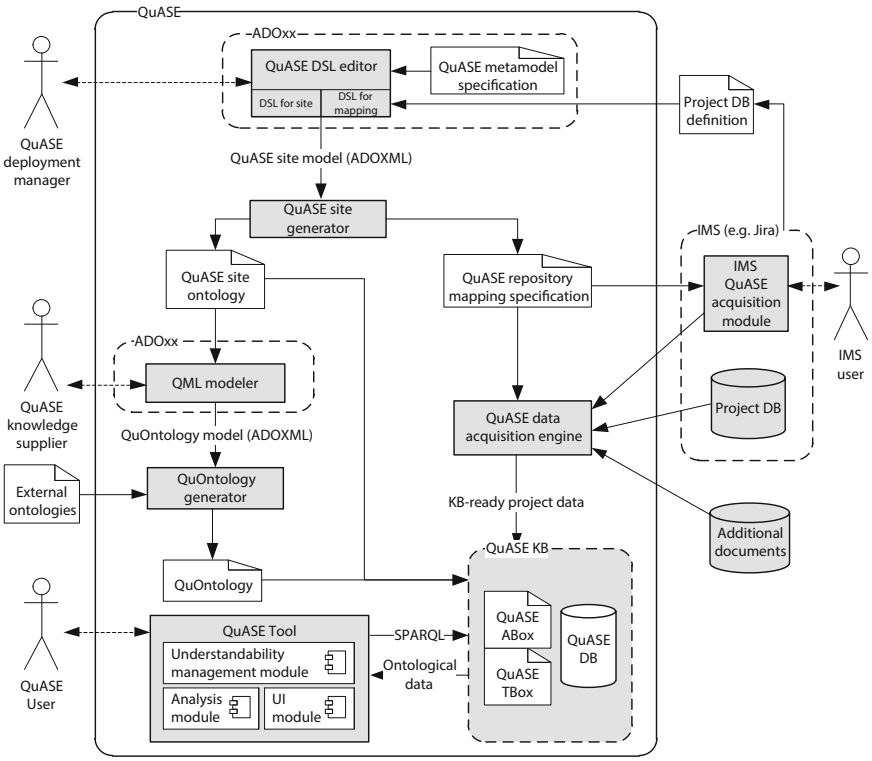

Fig. 4. Generic QuASE system architecture

3.1 Architectural Components and Their Interaction

The site-specific configuration of the communication environment is interactively defined as a *site model* in the domain-specific visual modeling language *site DSL* supported by a *site editor* tool. Such model can be also predefined by the supplier of the system based on the information collected on site. The site editor provides the DSL support based on the QuASE *site metamodel*; this support also uses the definition of the source project repository to enable defining the correspondence between QuASE concepts and this repository.

Obtained as a result of the above activities, the site model contains the description of the permitted structure of the site-specific set of concepts related to the communication environment (context and content configuration together with the knowledge elements) and their correspondence to the project repository structures (e.g. JIRA tables). This model, by the means of *QuASE site generator*, is transformed into the QuASE site ontology (introduced in Sect. 2) and the *QuASE repository mapping specification*. The site DSL and the corresponding editor are described in Sect. 4 together with a partial example of the site model.

The representation of the site ontology obtained as an output of the site generator describes the site-specific set of environment concepts by means of OWL 2. The transformation between the site model and the site ontology again is described in detail in Sect. 4.

QuASE repository mapping specification controls the execution of the *QuASE data acquisition engine*. The process of data acquisition is described in detail in [24]. In particular, this engine is responsible for the following functionality:

1. acquiring the raw project data directly from the project repositories (both IMS-controlled project databases and file-based repositories)
2. obtaining the non-repository data (e.g. the values of a content unit or context unit attributes not represented in project repositories) interactively from the users of these repositories through the extended IMS interface.

QuOntology construction is performed similarly to the construction of the QuASE site ontology. A knowledge supplier interacts with the *QML editor* tool supporting a visual modeling DSL (*Quality Modeling Language, QML*); QML is based on a metamodel that reflects knowledge unit configuration from the QuASE site model. By means of this tool, the supplier creates a *QML model* which is then transformed into an OWL 2 representation of QuOntology. QuOntology could also reuse the existing knowledge by importing external domain ontologies. The description of QML and the process of constructing QuOntology will be presented in a separate publication.

Both QuASE site ontology and QuOntology are combined to form the structural part (TBox in description logics) of the QuASE KB. The individuals comprising the ABox of this KB are provided by the QuASE data acquisition engine based on the raw project data. An approach to defining QuASE KB architecture is described in Sect. 3.2.

QuASE tool includes components responsible for providing understandability and decision support to the end users. All interaction between this tool and the KB is performed through an internal API referring only to the information from the QuASE metamodel; these API calls transparently form and execute SPARQL queries to KB and return the data based on their results. By providing this API, it is guaranteed that the code of the tool is decoupled from the particular site definition; this definition is used as the set of ontology individuals treated as the data structures by the tool code.

3.2 Defining QuASE KB Architecture

We used an architectural approach to define QuASE KB (Fig. 5) based on transparent ontology storage: the ontology ABox is explicitly formed by the QuASE data acquisition engine as a set of individuals corresponding to the classes defined in the QuASE TBox. Together TBox and ABox form an OWL 2 ontology which is then transparently stored into the triple store; we are using the capabilities of the Jena framework (Jena TDB triple store, http://jena.apache.org/documentation/tdb/) to implement this storage; all queries are handled by the SPARQL engine.

The advantage of this approach is the simplicity of the architecture: only the ontology has to be formed; the mapping between this ontology and the permanent storage is provided by TDB transparently. Another advantage is the maturity of the supporting

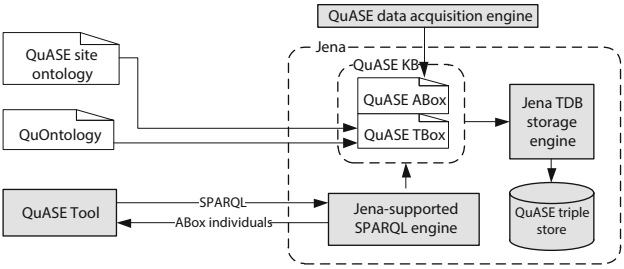

Fig. 5. Knowledge base architecture based on transparent ontology storage

technology (Apache Jena, and, in particular, TDB storage solution, are currently widely used in both academia and industry).

While considering this approach, the performance and scalability of such configuration were under question, as achieving the optimal performance of ontological reasoning over the database-backed ontology is still an open research challenge. But we ultimately decided in favor of it after achieving an acceptable performance of the QuASE site KB populated from the industrial-size project repository of one of the consortium partners (collected since 2008: over 8000 issues resulting in OWL 2 KB containing 1.8 million axioms).

The rejected alternative was based on using the OBDA (Ontology-Based Data Access) techniques [15] where the OBDA reasoner hides the independently configured relational database (QuASE DB) behind the ontological interface: this approach was rejected as (1) the OBDA solution turned out to be more difficult to configure and less capable (e.g. the available OBDA reasoners are restricted in OWL support with OWL2 QL which is not sufficient for expressing QuASE site ontology and KB), and (2) acceptable performance was achieved using the transparent storage approach.

3.3 QuASE System Usage Scenarios

In this section, we provide examples of using the proposed architecture to facilitate communication between stakeholders in a software project. We distinguish site definition, knowledge acquisition, and QuASE tool usage scenarios.

1. *Site and knowledge definition scenario:*
   (a) The knowledge supplier *K* (an employee of the company *E*) creates the *E*'s site model using the QuASE site modeler tool. Then he/she initiates the site ontology generation for *E* based on the specified model.
   (b) *K* extends the *E*'s site model adding the repository mapping specification referring to *E*'s JIRA installation.
   (c) *K* provides QuOntology concepts, representations, and explanations using QML editor and initiates the generation of QuOntology.
   (d) As a result, the *E*'s site ontology, QuOntology, and repository mapping specification are created; the system becomes ready to acquire knowledge.

2. *Knowledge acquisition scenario – incremental case:*
   (a) The Jira user U creates the new issue X_U and adds extra information (e.g. attributes and related decisions with preferred alternatives) to X_U based on E's site model and mapping information, the information is stored in E's local QuASE database (by QuASE Jira plug-in) and E's project repository (by Jira).
   (b) K initiates the process of populating the site KB based on the site ontology and QuOntology specifications, the repository mapping specification, X_U data from the project repository, and X_U extra data contained in the QuASE internal DB.
   (c) As a result, the QuASE site KB becomes synchronized with the current state of the project repository and the extra information specified by U.
3. *QuASE tool usage scenarios.* A complete set of usage scenarios for the QuASE tool is defined in [29], so here we limit ourselves only with a short outline of the instantiation of two basic scenarios presented in Sect. 1:
   (a) *Understandability support:* Users A and B use the QuASE tool to achieve common understanding of a new issue X_A. The issue is transferred into the QuASE tool, where the target context is set to B. After that, X_A's text is parsed by the tool and the problematic terms are looked up in the QuASE KB via SPARQL queries. B-specific representations and/or explanations for these terms are obtained as a result of these queries and shown to B forming a resolved issue X_B.
   (b) *Decision support:* User A uses the QuASE tool for decision support concerning his/her future communication with the stakeholder B. He/she selects the subset of B's metrics and asks QuASE tool to look for all similar stakeholders (similarity search), for recommendations related to B, and for predicted values of the selected metrics. The tool queries the QuASE KB using SPARQL and provides the requested information to A based on the existing knowledge.

4 Defining QuASE Site Model and QuASE Site Ontology

In this section, we define the QuASE site DSL through its metamodel, and show how QuASE site models are created by means of this DSL. We also describe an approach to transforming these models into the instances of QuASE site ontology.

4.1 Metamodel for QuASE Site DSL

We define the metamodel for the QuASE site DSL on two levels: foundational level shown in Fig. 6; and QuASE site level shown on Fig. 7.

On the foundational level (shown using an UML-like notation), we restricted ourselves with a very lean set of concepts: *Modeling Element* consisting of *Attributes* forms the top of the hierarchy; it specializes to *Structural Modeling Element* and *Descriptional Modeling Element*; *Entity* and *Relation* are specializations of Structural Modeling Element; Entities can again consist of entities. Relations consists of *Perspectives* connecting them to Entities; in turn, Entities are related through Relations. *Attribute* and *Perspective* are Descriptional Modeling Elements.

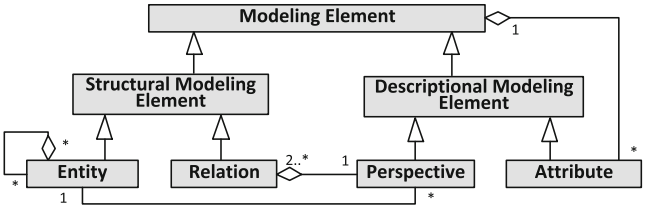

Fig. 6. Foundational concepts for the metamodel of the QuASE site DSL

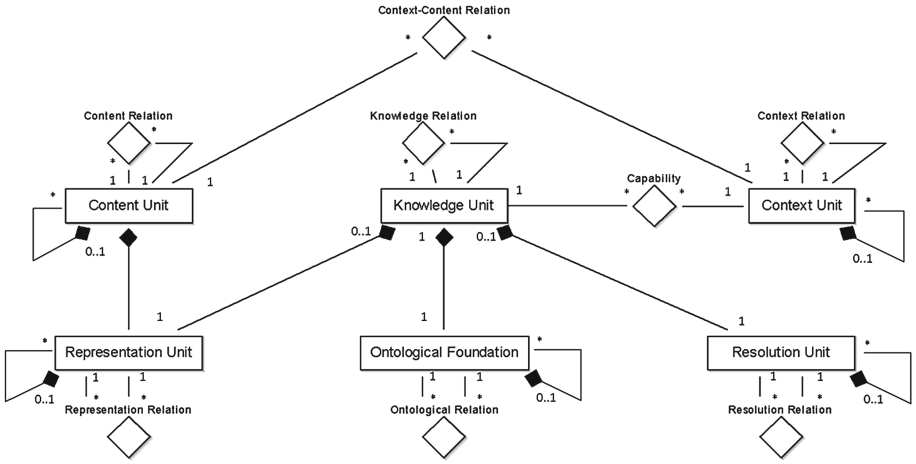

Fig. 7. Metamodel for the QuASE site DSL (the notation uses notions from ER)

On the QuASE site level, we instantiate the foundational concepts as metaclasses. In this level's metamodel:

1. Separate entity metaclasses are defined to specify modeling constructs corresponding to context units (*Context Unit*), content units (*Content Unit*), and knowledge units (*Knowledge Unit*). The Knowledge Unit is composed of a triple of *Representation Unit*, *Ontological Foundation*, and *Resolution Unit* metaclasses.
2. For every entity metaclass, the corresponding "relation to itself" metaclass is defined which can be instantiated to connect modeling constructs belonging to this metaclass (e.g. the instances of *Content Unit Relation* metaclass can connect model elements which are the instances of Content Unit; a model level example of such relation could define that Issue Versions are related to Issues).
3. Part-whole composition meta-relations are defined for entity metaclasses.
4. Two relation metaclasses are defined that can be instantiated to connect instances of different entity metaclasses: the instances of *Context-Content Relation* connect Context Units and Content Units (defining e.g. that the Persons are able to provide their knowledge while forming Issues), whereas the instances of *Capability* connect Context Units and Knowledge Units (defining e.g. that the Persons are able to understand Notions).

5. Context Units can contain Representation Units (e.g. the Description of the particular Issue can contain a set of Plain Text Fragments).
6. For every relation and entity metaclass, the *type* is defined as an additional meta-attribute (not shown on a diagram); in particular, for capabilities the following predefined relation types are available: “is able to understand”, “is able to understand with explanation”, “is able to explain”, and “is able to perform”. Such types are not defined as separate metaclasses to restrict the number of metamodel elements; it allows keeping it stable while allowing defining additional types for the particular instance of the QuASE site model.

4.2 Defining the QuASE Site Model by Means of the QuASE Site DSL

The QuASE site DSL editor tool is produced by implementing the QuASE site meta-model in the ADOxx meta-modeling framework (http://www.adoxx.org). Figure 8 shows a snapshot of its user interface displaying an instance of the QuASE site model.

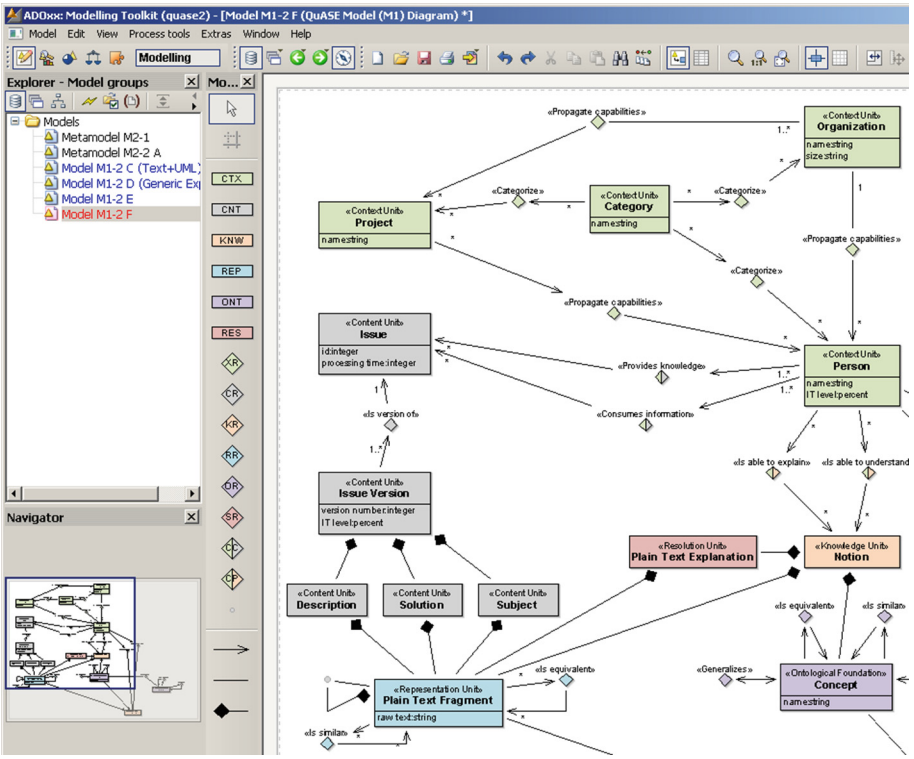

Fig. 8. Fragment of a particular QuASE site model in ADOxx-based DSL editor

On the current stage of the project, we restricted ourselves with a limited set of notational elements; the notation is going to evolve alongside the progress of the project.

Entity-based and relation-based constructs are respectively denoted with colored boxes and colored diamonds; relations have incoming and outgoing connectors with associated cardinalities. Categories of constructs are distinguished by their background color; for context-content relations and capabilities, two colors are used separated by the vertical line. The names of categories are also shown in angle brackets similarly to the names of stereotypes in UML class diagrams. In addition, the UML-inspired composition connectors can be defined between elements if allowed by the metamodel.

Attributes are specified using a notation similar to the one used in UML class diagrams. The attributes for relations can also be specified (not shown on the diagram).

In Fig. 8, the QuASE site model includes the following context units as elements of the site-specific context configuration: Category, Organization, Project, and Person. Individual context units are connected by the relation of type "propagate capabilities" which denote that the capabilities belonging to the source unit are inherited by the target unit unless it overrides them, e.g. the capabilities of the particular project are inherited by the persons involved in the project.

The configuration of content units reflects the JIRA-like structure of the project database; it includes Issue and Issue Version as higher-level content units; in a project repository (e.g. a JIRA database), such units can correspond to interrelated tables. Description, Solution, and Subject are lower-level units (representation holders); their instances contain Plain Text Fragments (instances of representation units); in a project repository, these units can correspond to the attributes able to hold communicated information. The container for these units is Issue Version.

The only knowledge unit shown in Fig. 8 is Notion; it contains Plain Text Fragment as a representation unit, Concept as ontological foundation, and Plain Text Explanation as a resolution unit. I.e., the site model defines that the notions are represented and explained using plain text. Concepts can be connected to each other with "generalize", "is similar" and "is equivalent" relations; this fragment of the model is intended to define the means of specifying QuOntology.

The instances of Plain Text Fragments can be also connected to the other instances of this concept via "is equivalent" and "is similar" relations. This models the configuration where the means of representation can be compared and checked for similarity.

4.3 Generating QuASE Site Ontology

The QuASE site model defines a conceptual configuration of the communication environment as a set of interrelated entities. To incorporate this configuration into the structure of the QuASE KB, it has to be transformed into computational ontology representation; we propose to target OWL 2 representation in this transformation.

We have implemented Java-based a *QuASE site generator* tool aimed at this purpose. It obtains the serialized QuASE site model from the QuASE site editor and applies a set of rules to generate OWL 2 constructs corresponding to the QuASE site DSL constructs. These rules are based on a set of rules published in [3] to control the conversion of the generic conceptual modeling language OntoUML into OWL:

1. The set of *metamodel-based OWL classes* is included in the resulting ontology prior to processing the model elements; it is stable across conversions. Such classes represent metaclasses defined on both, foundational and QuASE site levels: *Context Unit* $\sqsubseteq$ *Entity* $\sqsubseteq$ *ModelingElement, Capability* $\sqsubseteq$ *Relation* $\sqsubseteq$ *ModelingElement*;
2. An entity model element is transformed into an OWL class as a subclass of the class which represents the entity's meta-class: e.g. the rule activated on arriving of a Context Unit model element named *Person* forms the following OWL class hierarchy: *Person* $\sqsubseteq$ *ContextUnit* $\sqsubseteq$ *Entity* $\sqsubseteq$ *ModelingElement*.
3. A relation model element is transformed into
   (a) an OWL class for the element (named based on the value of its *type* attribute and the names of connected entities: e.g. *Person_IsAbleToExplain_Notion*); this class becomes a subclass of a class that represents a particular value of its *type* attribute which in turn becomes a subclass of the relation's meta-class: *Person_IsAbleToExplain_Notion* $\sqsubseteq$ *IsAbleToExplain* $\sqsubseteq$ *Capability* $\sqsubseteq$ *Relation* $\sqsubseteq$ *ModelingElement*
   (b) a set of object properties for perspectives and for the relation itself named by prefixing the relation name with *[SOURCE_FOR]* or *[TARGET_FOR]* for perspective-based properties or with *[PROPERTY_FOR]* for the relation-based properties: *[SOURCE_FOR] Person_IsAbleToExplain_Notion* $\sqsubseteq$ *[SOURCE_FOR]Capability* $\sqsubseteq$ *[SOURCE_FOR] Relation* $\sqsubseteq$ topObjectProperty; inverse properties are additionally prefixed with *[INV]*.
   (c) a set of axioms connecting the resulting class and its target entity classes through perspective properties; following [3], we made these axioms depend on the cardinalities specified for the perspectives in the site model:
      (i) for the cardinality of 0..* (optional perspective) no axiom is added, only domain and range restrictions are specified for the particular property:

$$\top \sqsubseteq \forall\, [SOURCE\_FOR]\, Person\_IsAbleToUnderstand\_Notion.\ Person\_IsAbleToUnderstand\_Notion$$

$$\top \sqsubseteq \forall\, [SOURCE\_FOR]\, Person\_IsAbleToUnderstand\_Notion.Person$$

      (ii) for the cardinality of 1..* (mandatory perspective) existence axioms are added as equivalent classes to both the relation and the target classes:

$$Person\_IsAbleToExplain\_Notion \equiv [SOURCE\_FOR]\, Person\_IsAbleToExplain\_Notion.Person$$

$$Person \equiv [INV]\, [SOURCE\_FOR]\, Person\_IsAbleToExplain\_Notion.\ Person\_IsAbleToExplain\_Notion$$

      (iii) for quantity-based cardinalities (0..n, n, n..m, and n..*) cardinality axioms are added; e.g. if every issue must have at least 2 versions the axiom defining the relation class will be as follows: *IssueVersion_IsAVersionOf_Issue* $\equiv \geqslant 2$ *[SOURCE_FOR]IssueVersion_IsAVersionOf_Issue. IssueVersion*

4. Attributes are transformed into data properties and the corresponding axioms connecting these properties to the possessing classes.

On the technical level, as ADOxx-based modeling tools serialize models using XML-based ADOXML format, these transformation rules are applied to the units of data obtained from ADOXML stream. According to ADOXML specification, these units uniformly represent instances of the modeling elements defined on diagrams. The matching conditions are based on the values of attributes of these instance units.

The tool can be extended with plugins encapsulating new type-level rules (additional types defined as new values for the type meta-attribute).

5 Related Work

In this section, we discuss several categories of the related work: (1) addressing the complete set of goals for QuASE, (2) addressing particular goals of QuASE, (3) addressing collecting knowledge about communication environment and communicated information and obtaining the data from project repositories for analytical purposes.

The approaches addressing the complete set of QuASE goals belong to the category of solutions that facilitate reusing, adapting, and analyzing the development knowledge. In particular they apply the existing body of research on knowledge management to the field of software engineering [2, 4]. A more specific category of solutions is related to managing past software engineering experience; they are known as experience management solutions [21]. With respect to our aims, such solutions bear the following shortcomings: (1) they do not specifically address quality-related issues, it is true especially for those issues that are available from existing repositories; (2) they collect the experience only as viewed from the developer side; the business stakeholder's view is mostly ignored, understandability management is not supported.

Current research *addressing understandability of the information in the software process* mainly deals with this quality characteristic defined for the following software process artifacts (we group these artifacts by the stage of the development process): (1) *requirement engineering-related artifacts*: in particular, understandability of the requirement specifications is addressed in [14], whereas [1] deals with understandability of the use case models; (2) *design-related artifacts*: in particular, the set of metrics for measuring understandability of the conceptual models is defined in [18], understandability of entity-relationships diagrams is introduced in [6], and the set of factors influencing understandability of the business process models is outlined in [20]; (3) *implementation-related artifacts*: in particular, the set of metrics for source code understandability is defined in [10, 16];

The differences between our approach and the above techniques are as follows: (1) most state-of-the-art techniques address understandability of the particular categories of development-related artifacts (such as requirements, source code, or conceptual models); our research, to the contrary, addresses understandability of the generic fragments of communicated information which could be contained in documents belonging to different categories; in this paper, these documents are exemplified by issues; (2) these techniques, as a rule, do not specifically address understandability of quality-related

information; (3) they do not employ ontology-based approach for establishing common understanding between parties in the software process.

The approaches to obtaining information from project repositories for analytical purposes typically belong to the research area of mining software repositories [12]. Particular examples include automatic categorization of defects [30], building software fault prediction models based on repository data [31], and using repositories to reveal traceability links [13]. Other approaches use repository information to analyze the applicability of specific development practices [5].

Repository mining solutions use software repositories as sources of quantitative code- and coding process-related information (such as the frequency of bugs, the time spent on various tasks, information about commits into repositories etc.). In contrast to that, QuASE uses repositories as sources for communicated information by looking into issue descriptions, negotiation opinions, wikis, and requirements documents. In addition to the difference in the general goals, our approach differs from these solutions as it is based on an established set of conceptual structures that represent communication environment and communicated knowledge.

6 Conclusions and Future Work

In this paper, we outlined a solution that is intended to support understandability and reusability of quality-related information, and thus may help to improve the quality of decisions in the software development process. The QuASE provides a knowledge-oriented interface to information that is communicated and collected in the course of software development projects. For this purpose, we introduced a set of knowledge structures representing both communication environment and the communicated information and described how they are incorporated into QuASE site ontology and QuOntology to form QuASE KB. We also defined the architecture of the QuASE system and introduced the techniques for creating QuASE site ontology by the knowledge suppliers.

Ongoing research within the framework of the QuASE project in the short term aims at implementing understandability and analysis scenarios in a QuASE tool based on the defined conceptual structures. It has to be achieved through implementing and testing the QuASE solution on top of establishing site models for all partner companies, generating QuASE site ontology instances based on these models, and collecting project data corresponding to the established site models.

We also aim at the following long-term research goals:

1. Establishing *QuASE process support* by elaborating the means of integration of the QuASE-specific activities into different software development process models (by implementing the scenarios implementing “QuASE for agile” etc.)
2. Generalizing QuASE as a means of implementing generic understandability management and issue-based analysis support:
   (a) through handling different representation formats (not only plain text): in particular to manage understandability of conceptual schemas or other artifacts;
   (b) by establishing generic process support to be specialized for different processes not limited to software development; the case study for this generalization can

be integrating QuASE capabilities into the HBMS framework [17, 19] for managing understandability in the Ambient Assisted Living domain.

References

1. Anda, B., Sjøberg, D., Jørgensen, M.: Quality and understandability of use case models. In: Lindskov Knudsen, J. (ed.) ECOOP 2001. LNCS, vol. 2072, pp. 402–428. Springer, Heidelberg (2001)
2. Aurum, A., Jeffery, R., Wohlin, C., Handzic, M. (eds.): Managing Software Engineering Knowledge. Springer, Heidelberg (2003)
3. Barcelos, P.P.F., dos Santos, V.A., Silva, F.B., Monteiro, M.E., Garcia, A.S.: An automated transformation from OntoUML to OWL and SWRL. In: ONTOBRAS 2013. CEUR Workshop Proceedings, vol. 1041, pp. 130–141. CEUR-WS.org (2013)
4. Bjørnson, F.O., Dingsøyr, T.: Knowledge management in software engineering: a systematic review of studied concepts, findings and research methods used. Inf. Softw. Technol. **50**, 1055–1068 (2008)
5. Ernst, N.A., Murphy, G.C.: Case studies in just-in-time requirements analysis. In: 2012 IEEE Second International Workshop on Empirical Requirements Engineering (EmpiRE), pp. 25–32. IEEE (2012)
6. Genero, M., Poels, G., Piattini, M.: Defining and validating metrics for assessing the understandability of entity–relationship diagrams. Data Knowl. Eng. **64**, 534–557 (2008)
7. Guizzardi, G.: Ontological foundations for structural conceptual models. Twente (2005)
8. Guizzardi, G., Falbo, R., Guizzardi, R.S.: Grounding software domain ontologies in the unified foundational ontology (UFO): the case of the ODE software process ontology. In: RESE 2008, pp. 244–251 (2008)
9. ISO: ISO/IEC 12207:2008, Information technology – software life cycle processes. International Organization for Standardization, Geneva (2008)
10. Jin-Cherng, L., Kuo-Chiang, W.: A model for measuring software understandability. In: Proceedings of CIT 2006, pp. 192–192 (2006)
11. JIRA Issue Tracking System. http://www.atlassian.com/software/jira. Accessed 8 May 2014
12. Kagdi, H., Collard, M.L., Maletic, J.I.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution. J. Softw. Maint. Evol. Res. Pract. **19**, 77–131 (2007)
13. Kagdi, H., Maletic, J.I., Sharif, B.: Mining software repositories for traceability links. In: ICPC 2007. pp. 145–154. IEEE (2007)
14. Kamsties, E., von Knethen, A., Reussner, R.: A controlled experiment to evaluate how styles affect the understandability of requirements specifications. Inf. Softw. Technol. **45**, 955–965 (2003)
15. Kontchakov, R., Rodríguez-Muro, M., Zakharyaschev, M.: Ontology-based data access with databases: a short course. In: Rudolph, S., Gottlob, G., Horrocks, I., van Harmelen, F. (eds.) Reasoning Weg 2013. LNCS, vol. 8067, pp. 194–229. Springer, Heidelberg (2013)
16. Lin, J.-C., Wu, K.-C.: Evaluation of software understandability based on fuzzy matrix. In: Fuzzy Systems, (IEEE World Congress on Computational Intelligence), IEEE International Conference on FUZZ-IEEE 2008, pp. 887–892. IEEE (2008)
17. Machot, F.A., Mayr, H.C., Michael, J.: Behavior modeling and reasoning for ambient support: HCM-L modeler. In: Ali, M., Pan, J.-S., Chen, S.-M., Horng, M.-F. (eds.) IEA/AIE 2014, Part II. LNCS, vol. 8482, pp. 388–397. Springer, Heidelberg (2014)

18. Mehmood, K., Cherfi, S.S.: Data quality through model quality: a quality model for measuring and improving the understandability of conceptual models. In: MDSEDQS 2009, pp. 29–32. ACM (2009)
19. Michael, J., Mayr, H.C.: Conceptual modeling for ambient assistance. In: Ng, W., Storey, V.C., Trujillo, J.C. (eds.) ER 2013. LNCS, vol. 8217, pp. 403–413. Springer, Heidelberg (2013)
20. Reijers, H.A., Mendling, J.: A study into the factors that influence the understandability of business process models. IEEE Trans. Syst. Man Cybern. A Syst. Hum. **41**, 449–462 (2011)
21. Schneider, K.: Experience and Knowledge Management in Software Engineering. Springer, Heidelberg (2009)
22. Shekhovtsov, V., Mayr, H.C., Kop, C.: Harmonizing the quality view of stakeholders. In: Mistrik, I., Bahsoon, R., Eeles, R., Roshandel, R., Stal, M. (eds.) Relating System Quality and Software Architecture, pp. 41–73. Morgan-Kaufmann, Waltham (2014)
23. Shekhovtsov, V.A., Mayr, H.C.: Let stakeholders define quality: a model-based approach. In: Linssen, O., Kuhrmann, M. (eds.) Qualitätsmanagement und Vorgehensmodelle - 19. Workshop der GI-Fachgruppe Vorgehensmodelle, pp. 101–110. Shaker Verlag GmbH (2012)
24. Shekhovtsov, V.A., Mayr, H.C.: Managing quality related information in software development processes. In: CAiSE-Forum-DC 2014. CEUR Workshop Proceedings, vol. 1164, pp. 73–80. CEUR-WS.org (2014)
25. Shekhovtsov, V.A., Mayr, H.C.: Towards managing understandability of quality-related information in software development processes. In: Murgante, B., Misra, S., Rocha, A.M.A., Torre, C., Rocha, J.G., Falcão, M.I., Taniar, D., Apduhan, B.O., Gervasi, O. (eds.) ICCSA 2014, Part V. LNCS, vol. 8583, pp. 572–585. Springer, Heidelberg (2014)
26. Shekhovtsov, V.A., Mayr, H.C., Kop, C.: Acquiring empirical knowledge to support intelligent analysis of quality-related issues in software development. In: Faria, J.P., Silva, A., Machado, R.J. (eds.) QUATIC 2012, pp. 153–156. IEEE Press (2012)
27. Shekhovtsov, V.A., Mayr, H.C., Kop, C.: Stakeholder involvement into quality definition and evaluation for service-oriented systems. In: USER 2012 Workshop at ICSE 2012, pp. 49–52. IEEE (2012)
28. Shekhovtsov, V.A., Mayr, H.C., Kop, C.: Towards conceptualizing quality-related stakeholder interactions in software development. In: Kop, C. (ed.) UNISON 2012. LNBIP, vol. 137, pp. 73–86. Springer, Heidelberg (2013)
29. Shekhovtsov, V.A., Mayr, H.C., Lubenskyi, V.: QuASE: A tool supported approach to facilitating quality-related communication in software development. In: da Silva, A.R., Silva, A.R., Brito, M.A., Machado, R.J. (eds.) QUATIC 2014, pp. 162–165. IEEE (2014)
30. Thung, F., Lo, D., Jiang, L.: Automatic defect categorization. In: WCRE 2012, pp. 205–214. IEEE (2012)
31. Vandecruys, O., Martens, D., Baesens, B., Mues, C., De Backer, M., Haesen, R.: Mining software repositories for comprehensible software fault prediction models. J. Syst. Softw. **81**, 823–839 (2008)