Implementing Tool Support for Analyzing Stakeholder Communications in Software Development

Vladimir A. Shekhovtsov, Heinrich C. Mayr, Matija Kucko
Application Engineering Group, Institute of Applied Informatics,
Alpen-Adria-Universität Klagenfurt,
Klagenfurt, Austria
{volodymyr.shekhovtsov,heinrich.mayr,mkucko}@aau.at

***Abstract*—The problem with communicated information originated from stakeholders participating in software development processes is related to the fact that such information is not always available for analysis and reuse. As a rule, it is collected in project repositories (such as issue or bug databases) and only consulted "as-is": such activities do not always satisfy the analytical needs of the software companies. The goal of a tool-supported QuASE approach presented in this paper is to allow for different kinds of analysis of the available communicated information, in particular, we support reuse of such information, prediction of the characteristics of the elements of the communication environment and the communicated information itself, issuing targeted recommendations and supporting selecting the preferred alternatives for decisions related to such information or to the elements of the communication environment. We describe the motivation for our approach, supported analytical scenarios, modeling and ontological support for analytical concepts, and the specifics of the implementation of the analysis component of the end-user software tool.**

Keywords—analysis; prediction; decision support; software development; project repository; ontology; knowledge base

I. INTRODUCTION AND MOTIVATION

This paper is devoted to the implementation aspects of the analytical components of the software tool developed as a result of the QuASE (Quality Aware Software Engineering) project established in a consortium with four Austrian software companies. The motivation of implementing these components lies in a fact that currently the information related to communication between stakeholders in software development processes (stored in such systems as Jira or Bugzilla) is often not accessible for analysis. As a rule, it is only consulted "as-is" in a form of natural language texts: such activities do not always satisfy the analytical needs of the software companies: the information is not reusable, it is not possible to predict future behavior of the involved parties, no recommendations based on past behavior could be provided, communication-related decisions cannot be supported based on the outcomes of similar decisions available from the past.

The tool-supported approach implemented as a result of the QuASE project is aimed at two main goals: (1) managing understandability of the communicated information and (2) analyzing stakeholder communication in software development processes. The motivational problem introduced above is related to the second goal: in this paper, we provide the description of the analytical components of the system, the understandability management components addressing the first goal were introduced in the previous publications [11, 12].

The QuASE approach is supported by the following software components [12, 13]:

1. An interactive modeling tool (based on ADOxx meta-modeling platform) which supports a domain-specific language allowing the modelers to describe a site-specific communication environment and the communicated knowledge, and to specify a mapping between this model and the project repositories deployed at the particular site,
2. A set of utilities for converting the specified model into ontological representation, and for acquiring communication-related data from the mapped project repositories into the knowledge base corresponding to the specified ontology, and
3. An end-user web-based QuASE tool supporting analytical scenarios on top of the established knowledge base accessible through SPARQL queries.

The QuASE tool supports: (1) reuse of communicated information; (2) issuing targeted communication-related recommendations; (3) forecasting the values of attributes for communication-related environment elements (such as the communicating parties or the communicated pieces of information); (4) providing suggestions with respect to communication-related decisions. For implementing this support, fundamental knowledge related to the communication domain has to be gathered and stored in the QuASE knowledge base: either automatically from the project repositories such as

The QuASE Project was sponsored by the Austrian Research Promotion Agency in the framework of the Bridge 1 program; Project ID: 3215531

2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)
13th User Symposium on Software Quality, Test and Innovation (ASQT 2015)
978-1-4799-1885-0/15/$31.00 ©2015 IEEE

JIRA issue tracking databases, or interactively from knowledge suppliers. The QuASE knowledge base is based on the QuASE site ontology defining the site-specific communication environment (the communicating parties, the documents which store the communicated information etc.)

The rest of the paper is structured as follows. Section 2 provides the background information related to the modeling and knowledge conversion components of the system, Section 3 introduces the basic concepts of the QuASE analysis support, it is followed by a description of the analysis support in the QuASE site model, ontology, and knowledge base in Section 4. Section 5 is devoted to the description of the analytical scenarios as implemented in QuASE tool, it also covers the specifics of implementing the analytical support in the tool by interacting with the Apache Mahout framework; it is followed by conclusions, and the description of the future research.

II. QUASE SITE MODEL, ONTOLOGY, AND KNOWLEDGE BASE: BACKGROUND INFORMATION

A. QuASE Core Knowledge Structures

Prior to describing the implementation of the QuASE analysis support in detail, it is necessary to introduce the set of QuASE core knowledge structures [13, 14]:

1. *QuASE site:* an owner of the given QuASE installation, e.g. a software provider.
2. *Context units:* units having particular views on communicated information, e.g. projects, organizations and their departments, involved people (stakeholders) etc. Context units are characterized by attributes and can be connected to other units; a context configuration e.g. could include the representation of the whole organizational hierarchy for an IT company. The set of context units can also include the categories of such units (e.g. "Business stakeholder" or "IT person") making able creating the views on communicated information belonging to such categories.
3. *Content units:* 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 comments, and issue attribute values. Content units can be related to context units.
4. *Knowledge units:* units encapsulating quality and domain knowledge that is subject of communication and harmonization; we described these concepts in detail in [13], they are not directly used for analysis support.

B. QuASE System Architecture

In an overview of the QuASE architecture we follow the architectural description in [13] taking into account the extensions introduced in the current implementation.

The site-specific configuration of the communication environment is interactively defined as a QuASE site model in the domain-specific visual modeling language QuASE site DSL supported by a *QuASE site editor* tool implemented by means of ADOxx framework. It provides the DSL support based on the QuASE site metamodel.

Obtained as a result of the above activities, the QuASE site model contains the description of the permitted structure of a site-specific set of concepts describing the communication environment (including context and content configuration) and their correspondence to the project repository structures (e.g. JIRA database tables). This model, by the means of *QuASE ontology builder* utility, is transformed into the QuASE site ontology and the QuASE repository mapping specification. The representation of the QuASE site ontology obtained as an output of the QuASE ontology builder utility describes the site-specific set of environment concepts by means of OWL 2.

QuASE repository mapping specification controls the execution of the *QuASE knowledge base builder* utility. It is responsible for acquiring the raw project data directly from the project repositories and obtaining non-repository data (e.g. the values of attributes not represented in project repositories) interactively from the users of these repositories through the external data acquisition component of the QuASE tool. QuASE site ontology forms the structural part (TBox in description logics) of the QuASE knowledge base (QuASE KB). The individuals comprising the ABox of this KB are provided by the QuASE knowledge base builder utility based on the raw project data.

QuASE tool includes components responsible for providing services to the end users (including analysis support). All interaction between this tool and the KB is performed through an API referring only to the information from the QuASE metamodel. By providing this API, it is guaranteed that the code of the tool is decoupled from the particular site definition.

Besides end-user services, the current architecture of the QuASE tool also provides external data acquisition functionality. The information is obtained as a result of interaction with the knowledge suppliers and then transferred back to the knowledge base builder utility to be transferred into OWL individuals combined with repository-based individuals to form the QuASE knowledge base. The architecture of the QuASE tool will be described in more detail in Section 5.

C. QuASE Site Model

QuASE site DSL was introduced in [13], here we outline its basic concepts which are important for implementing the QuASE analysis support.

On the foundational level we defined Modeling Element consisting of Attributes which specializes to Entity and Relation; Entities can again consist of entities. Relations consists of Perspectives connecting them to Entities; in turn, Entities are related through Relations.

On the QuASE site level (Fig.1), we instantiate the foundational concepts as metaclasses. Separate entity metaclasses are defined to specify modeling constructs corresponding to context units (Context Unit) and content units (Content Unit), the corresponding "relation to itself" metaclasses can be instantiated to connect modeling constructs belonging to the same metaclass (e.g. the instances of Content

Unit Relation metaclass can connect the instances of Content Unit), also relation metaclasses such as Context-Content Relation can be instantiated to connect instances of different entity metaclasses.

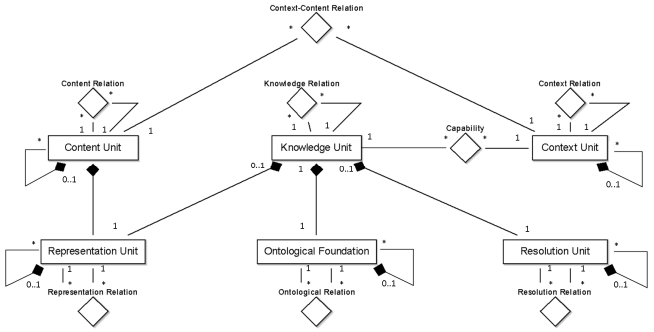

Fig. 1. Metamodel for the QuASE site DSL (from [11])

Fig.2 on the next page shows a snapshot of the user interface of the QuASE site DSL editor tool displaying an instance of the QuASE site model. We use colored boxes and colored diamonds to denote entity-based and relation-based constructs; relations have incoming and outgoing connectors. Categories of constructs are distinguished by their background color; for context-content relations, two colors are used separated by the vertical line. The names of categories are also shown in angle brackets. Attributes are specified using a notation similar to the one used in UML class diagrams.

D. QuASE Site Ontology

To incorporate the QuASE site model into the structure of the QuASE KB, it has to be transformed into computational ontology representation based on OWL 2 by means of QuASE ontology builder utility. It applies a set of rules to the XML-serialized QuASE site model to generate corresponding OWL 2 constructs. In particular,

1. The set of metamodel-based OWL classes is included in the ontology prior to processing the model elements: *ContextUnit* ⊑ *Entity* ⊑ *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: *JiraUser* ⊑ *ContextUnit* ⊑ *Entity* ⊑ *ModelingElement*,
3. A relation element is transformed into an OWL class for the element: *PersonCategory.Categorizes.JiraUser* ⊑ *Categorizes* ⊑ *ContextRelation* ⊑ *Relation* ⊑ *ModelingElement*, a set of object properties for perspectives and for the relation itself: *[source_for]PersonCategory.Categorizes.JiraUser* ⊑ *[source_for]ContextRelation* ⊑ *[source_for]Relation*, and a set of axioms connecting the resulting class and its target entity classes through perspective properties;
4. Attributes are transformed into data properties and the corresponding axioms connecting these properties to the possessing classes.

More detailed treatment of these rules is included in [13].

E. QuASE Knowledge Base

The information specified in the site model is used to map repository and external data into the QuASE knowledge base. This knowledge base contains individuals belonging to QuASE site ontology classes which are connected to other individuals with site ontology object properties and to literal values with site ontology data properties. To form the set of knowledge base individuals from the project repository data, QuASE knowledge base builder executes database queries specified in the site model for entity and relation model elements and forms the IRI for these individuals based on the returned information.

For an entity individual its IRI is formed using the value of the *id-designated column* in the dataset returned as a result of the execution of the query defined by the *entity query* attribute (the name of this column is also specified as the entity attribute) and the entity name: *Jira-Issue-4312 : Jira-Issue* ⊑ *ContentUnit* ⊑ *Entity*. Among data properties connecting such individuals to literals is (1) the property holding the name of the repository entity (e.g. the issue name): *Jira-Issue-4312.__name* ⊑ *Entity.__name*, <*Jira-Issue-4312*, "Installation problem"> : *Jira-Issue-4312.__name* and (2) the property for its unique identifier (based on *Entity.__id*).

Relation individuals are formed according to the mapping specification attributes defined for relation elements in the model which determine the character of the connection: the instances of *"connect existing elements"-type relation* are formed by finding matches between the value of the foreign key-designated column of the source entity query (the name of this column is defined as the relation attribute in the site model) and the value of the id-designated column of the target entity query, whereas the instances of *"use association query"-type relation* are formed based on the dataset returned as a result of the execution of the *association query* specified for a relation model element (the values of the columns in this dataset have to match the values of the id-designated columns at both sides of the relation).

Building the knowledge base is also possible in incremental mode (KB synchronization). To support it, an attribute can be specified for every entity element in the site model holding the name of the last update timestamp column in the entity query; such column is used to limit the data obtained from the project repository to the rows changed since the last synchronization.

III. QUASE ANALYSIS SUPPORT: INTRODUCTION

A. QuASE Reuse and Analysis Support Scenarios

QuASE supports reuse and analysis scenarios where the latter are further subcategorized into prediction, recommendation, and decision support scenarios. In their description we follow [14], modifying the list provided in that paper to reflect currently implemented functionality.

The *reuse scenarios* are based on applying similarity search techniques, in particular, content-based similarity search, where the user selects the metrics of a given content unit (e.g. a specific issue). The values of these metrics then are used for the search of content units in the QuASE knowledge base, the similarity distance of which is below a predefined threshold. It

is also possible to perform context-based similarity search where the user looks for context units (e.g. stakeholders) close to the given context unit; the similarity distance is calculated based on a subset of attributes of the selected context unit.

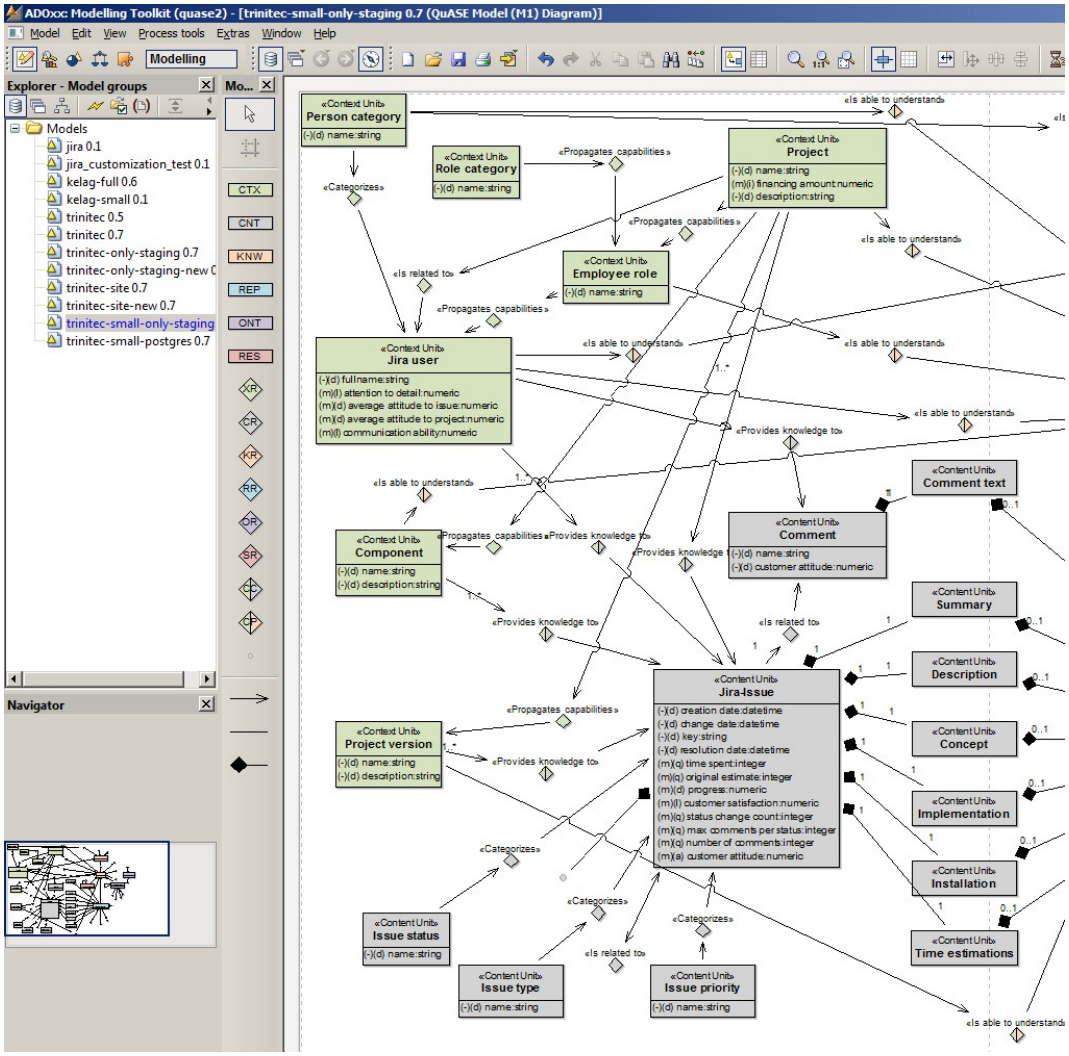

Fig. 2. Fragment of a particular QuASE site model in ADOxx-based DSL editor

The user goal in the *recommendation scenarios* is to obtain a set of recommendations based on the user reaction to the recommendations issued in the past. The recommendations could e.g. define the way of working with a given issue, or for communicating with a particular stakeholder. We distinguish behavior-related recommendations (describing the desired communication behavior) and capability-related recommendations (addressing user capabilities required to deal with e.g. an issue.) As an example, a user might select an issue and obtain in reply the recommendation that 'a high level of IT knowledge' is required for dealing with this issue.

In the *prediction scenarios* the predictions are based on historical data from past communications available in the QuASE knowledge base. They support the selection of communication strategies or issue handling schedules (based on estimated processing times). Computing predictions starts from the metric values of the given content or context unit. Current QuASE implementation supports value prediction scenarios where the user e.g. selects an attribute of the issue and obtains the predicted (computed) value of this attribute.

In the *decision support scenarios* the goal is to obtain, for a given decision case, recommendations based on prior decision data: either as assessments of the current decision alternatives or by ranking alternatives according to particular criteria. Example: the user selects an issue and, as decision type "selecting contact point for communication"; as a result, he obtains a ranked set of alternatives for this decision extracted from prior decisions made for similar issues.

B. Basic QuASE Analysis Concepts

The implementation of the analysis support in QuASE applies machine learning techniques of Apache Mahout [6] to the set of QuASE KB individuals: similarity search for implementing information reuse, regression analysis for predicting attribute values, and hybrid (partially supervised) learning for recommendation and decision support.

Establishing similarity search and prediction support relies only on accompanying the knowledge base individuals with normalized metric values which can be used for calculating similarity distances and building regression models used for predictions. In the following section, the support for these metric values will be described in detail.

Establishing recommendation and decision support also relies on normalized metric values, but it also requires extending QuASE site model, ontology, and knowledge base with additional concepts: in particular an additional metamodel element (Decision Unit) has to be introduced together with corresponding relations, these elements will be described in the following section.

While defining the recommender system in QuASE we treat both recommendations and decisions as the items to be recommended (analogous to product titles for online store recommenders). For this problem statement, making decisions and issuing recommendations can be reduced to the same problem (with recommendations equivalent to decision alternatives). In addition, we treat decision targets (e.g. context or content units) as the objects to receive recommendations (analogous e.g. to the users of the online recommender system). We treat the process of connecting the particular decision target (e.g. issue) and the particular decision outcomes similarly to "liking" the particular alternative in the online store, i.e. we show different decision outcomes to the QuASE user and ask which one is true from that user's point of view.

QuASE uses content-based recommendation algorithm where, based on the available metric data, the system allows to obtain the preferred decision outcomes for the incoming decision target individuals (e.g. issues).

IV. IMPLEMENTING ANALYSIS SUPPORT IN QUASE SITE MODEL, ONTOLOGY, AND KNOWLEDGE BASE

The initial support for analysis is defined on the metamodel level by specifying the necessary elements of the metamodel for the site DSL, these elements are then converted into ontological representation, and then the corresponding information is acquired from the project repositories and . The following elements are specified on the metamodel level: QuASE metrics, QuASE decision units, and decision-specific QuASE relations.

A. QuASE Metrics

1) QuASE metrics in the site model

We define the QuASE metric as a special kind of attribute applicable to context and content units (owner units): the metrics accept numerical values and are ready to be used for analysis; by defining the particular attribute as a metric, the modeler informs the QuASE system that this attribute can be selected to perform analytical activities (e.g. only the metrics can be selected to perform similarity search, only the metric values can be predicted, the recommendation and decision support algorithms are also based on the values of the metrics).

The metric in the model is defined as a tuple <name, data type, mapping mode, mapping data> where *data type* defines the set of allowed values for the metric; QuASE supports integer and numeric metrics; *mapping mode* [10] defines the way of collecting the values of the metric while building the knowledge base; *mapping data* contains additional information necessary to perform value collecting, its semantics depends on the mapping mode.

QuASE site DSL allows the modelers to define direct, query-base, calculated, and interactive mapping modes.

The values for *direct metrics* are collected from the results of the SQL query provided as a part of the owner unit definition, the mapping data for such metric is the name of the source column which has to be present in the query;

The values for *query-based metrics* are collected from the results of the additional metric-defining SQL query provided as its mapping data value. The query has to return a set of tuples <*id*, *value*> where *id* corresponds to the value of the unique identifier of the metric owner (to establish a connection between the metric instance and the instance of its owner), *value* corresponds to the value of the metric.

The values for *calculated metrics* are obtained as a result of executing the special routines defined in the code of the knowledge base builder utility; currently the only way of locating such routines is by the name of their Java class (which has to match the name of the metric combined with the name of its owner). In future, we plan to utilize the scripting language (such as Groovy) to allow for scriptable calculated metrics.

The values for *interactive metrics* correspond to the information which is not available in the project repositories and have to be additionally provided by the knowledge suppliers by means of their interacting with the QuASE tool.

Among interactive metrics: (a) *level metrics* accept the values defining the level of some characteristic, to specify the value of such metric the user have to select from the list of labels connected to numeric constants: (from "very high" corresponding to 1, "high" – to 0.75, down to "very low" (0)); (b) *attitude metrics* are similar to the level metrics but reflect the user's attitude to some fact (the values range from "very positive" (1), "positive" (0.75) down to "very negative" (0)); (c) *input metrics* accept arbitrary numeric values specified by the user (the interaction is supposed to provide an input field instead of a select list).

2) QuASE metrics in the site ontology and knowledge base

In the site ontology, the support for metrics is based on the ontological concept of Attribute further specialized to Metric: *Jira-Issue.customer_attitude* ⊑ *IntegerMetric* ⊑ *Metric* ⊑ *Attribute* ⊑ *ModelingElement*. Metrics are connected to their owners with *hasAttribute* object property.

While building the knowledge base, to establish support for analytical activities, every metric individual has to be accompanied with properties connecting it to both absolute and normalized values, and with a unique identifier of the metric class: < *Jira-Issue-4513.customer_attitude*, 0.75 > : *Jira-Issue.customer_attitude.hasValue*.

The values for direct metrics are obtained from the mapped columns of the JDBC record set formed by the execution of the entity query, the data for query-based metrics is obtained from

the separate record sets formed by execution of the metric-defining queries, the data for calculated metrics is formed by the code of knowledge base builder utility, and the values of the interactive metrics, collected in the QuASE tool, are obtained from its state provider component.

B. QuASE Decision Units: Explicit Support

1) Explicit model support for decision units

The explicit support for the decision units in the site DSL allows the modeler to specify the corresponding model elements directly in the model by means of the site modeler tool. Four new modeling elements are introduced: Decision Unit (based on the Entity meta-class) (examples on Fig.3 below are labeled with (2), (4), (6), and (8); further in this section, the numbers in parentheses will refer to the labels on the picture), Decision Relation ((3), (5), (7), (12), and (13)), Context-Decision Relation (10), and Content-Decision Relation (9) (all based on the Relation meta-class);

Decision and recommendation support can be defined for either Context Units or Content Units; further we will call such units Decision or Recommendation Targets. For every decision or recommendation target (further referred to as <*target*>), the configuration shown on Fig.3 is supposed to be added to the model. The depicted <*target*> is Jira-Issue (1).

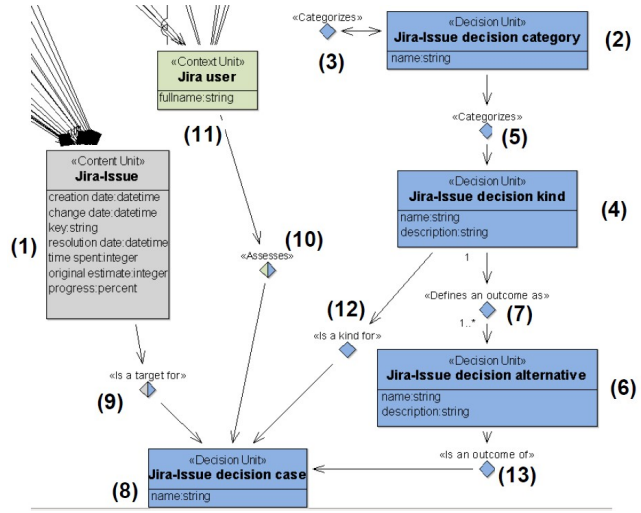

Fig. 3. Explicit decision support in the site model

The chain of Decision Units starts from "<*target*> decision category" (2), the "categorizes" relation (3) allows specifying a hierarchy of categories for decisions; example of a category can be "Communication-related issue decisions".

The decision kind (which is related to a set of alternatives) is defined as a "<*target*> decision kind" (4), examples of such kinds can be "Selection of the communication channel". Decision kinds are related to decision categories by another "categorizes" relation (5).

A particular decision kind defines as its outcomes a set of alternatives of a class "<*target*> decision alternative" (6). These alternatives are related to their decision kind via "defines an outcome as" relation (7). The unique identifier (id) of the alternative includes the id of its decision kind and its sequence number (the existence of the alternative depends on the existence of its kind). As an example, the "Select communication channel" decision kind (id: ID1) defines its outcomes as the following alternatives "Direct contact with stakeholder" (id: ID1-1), "Contact through key user" (id: ID1-2), "Contact through high-level manager" (id: ID1-3).

A fact of assessing the particular target by the particular person with selecting the particular decision outcome is represented by the instance of a class "*target* decision case" (8). Four incoming relations are defined for this class:

1. *"is a target for"* relation (9) from <*target*> (1) e.g. Jira issue;
2. *"assesses"* relation (10) from the *assessor* class (11); the assessor is the Context Unit representing the user performing the assessment, in the example case depicted on Fig.4 it is a Jira user;
3. *"is a kind for"* relation (12) from the "<*target*> decision kind" (4) representing the decision kind for this particular decision case;
4. *"is an outcome of"* relation (13) from the "<*target*> decision alternative" (6); representing the selected outcome of the decision of the particular kind.

Only one decision outcome for the particular assessor, the particular <*target*>, and the particular decision kind is allowed (users cannot store the outcomes of the same decisions more than once), as a result, the unique identifiers (ids) for decision cases are formed from the ids of decision assessors (11), decision targets (1), and decision kinds (4); they are used to compose the IRIs of decision case individuals. E.g. the decision case of selecting the outcome of the decision of the "Select communication target" (id: *ID1*) kind by the user *schwartz* for the issue with id of *29375* will have an id of *schwartz-29375-ID1*, it can be connected to the outcome with the id of e.g. *ID1-2* (which means that the second alternative has been selected).

2) Explicit ontology and knowledge base representation for decision units

To support communication-related decisions the following classes are added as subclasses of Decision Unit: *Decision Category*, *Decision Kind*, *Decision Alternative*, and *Decision Case*. The corresponding model-based decision classes are defined as their subclasses: *Jira-Issue decision category* $\sqsubseteq$ *Decision Category*, *Jira user decision category* $\sqsubseteq$ *Decision Category* etc. Similar classes have to be also added for recommendation support: *Recommendation category* etc.

The subclasses for the additional class *Decision Target* $\sqsubseteq$ *Entity* are entities which are the targets for decisions: *Jira-Issue* $\sqsubseteq$ *Decision Target*, *Jira user* $\sqsubseteq$ *Decision Target* etc. For recommendation targets, *Recommendation Target* $\sqsubseteq$ *Entity* class has to be introduced as well. The subclass of the additional class *Decision Assessor* $\sqsubseteq$ *Context Unit* is a Context Unit which corresponds to the persons able to assess the decision cases: *Jira user* $\sqsubseteq$ *Decision Assessor*.

The introduced relation classes are *Decision Relation* and *Decision-Target Relation*. Also, to simplify the connection to Mahout, the additional object property *is supplemented with*

connecting the instances of *Decision Case* and *Metric* is added into the ontology.

The knowledge base builder utility obtains the decision data from the QuASE tool (state provider component) and uses it to create necessary individuals and connect them with properties.

C. QuASE Decision Units: Implicit Support

The problem with adding the explicit analysis support into the site model is that it is necessary to have the repeating set of classes added to the model every time the modeler intends to add support for decisions or recommendations. These classes are parameterized by the target name: Jira-Issue decision category, Jira-Issue decision kind and corresponding relations for Jira-Issue, Jira user decision category for Jira user etc.

The solution for introducing implicit decision support into the model is to eliminate decision units and decision relation metaclasses from the metamodel used in the site modeler while adding options "Enable recommendations" and "Enable decision support" to Content and Context Units, so they can be enabled for the model elements, e.g. for Jira user (Fig.4).

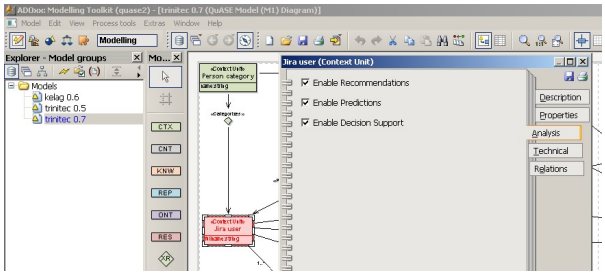

Fig. 4. Implicit model support for analysis

After that, during the generation of the ontology, the model is transparently extended with all the necessary classes and relations with the names auto-generated based on the name of the model element (<*target*>) with enabled support in the model, and, as a result, the generated ontology does not differ from the ontology generated from the model with the explicit support, but the model is kept less cluttered.

In addition, the options “Represents repository users” and “Represents QuASE users” are added to Context Units; they allow specifying in the model, which model-level Context Unit class can serve as an assessor in auto-generated model fragments (Fig.5).

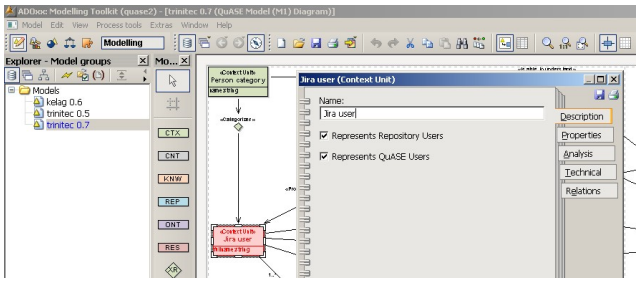

Fig. 5. Defining Assessor class in the model

The implementation of this approach uses a fragment of the serialized site model (in XML format) containing the necessary decision classes and relations as a template with placeholders corresponding to the information derived from the target class: this template is instantiated by means of Velocity template engine (http://velocity.apache.org) every time the decision-enabled class is encountered by the ontology builder and knowledge base builder, the fragment of the serialized model is generated out of it and integrated with the rest of the model.

V. QUASE TOOL: IMPLEMENTING ANALYSIS SUPPORT

A. QuASE Tool Architecture: Background Information

In the description of the QuASE tool architecture we follow [10], restricting ourselves with the components necessary for analysis support. The tool consists of the components belonging to three application tiers.

Data storage tier handles storing the data and knowledge into two categories of storage: (1) the storage for QuASE knowledge base; this category of storage is implemented by means of Jena TDB (http://jena.apache.org/documentation/tdb) triple store which provides a SPARQL query interface by means of Jena SPARQL endpoint; (2) QuASE internal data storage supported with relational database (exemplified by MySQL in the default configuration of the tool).

Application logic tier handles the operations defined on the available knowledge and the administrative logic. It consists of the following categories of components: (1) *data access layer* (encapsulates details of accessing data storage tier); (2) *functionality modules* (implementing main tool functionality such as analysis support); (3) *API module* (serves as an interface of the whole tier to be used by the web application tier; based on embedded Jetty web-server accessible through the REST interface); (4) *data model* (implementing storage-independent data model shared among other modules as a set of wrappers for individuals, data and object properties fetched from the knowledge base). In this paper further we will describe the functionality module responsible for analysis.

Web application tier consists of a rich web application based on AngularJS library integrated with plugin-based extension of the particular issue management system (e.g. Jira plugin [5]). It interacts with the embedded server through the REST API, provided by API module. The internal structure of the web application corresponds to the structure of functional modules in application logic tier. The layer implements primary security checks and input data validation.

B. QuASE Tool: Analytical Scenarios Support

In this subsection, we describe the current end-user support of analytical scenarios implemented in QuASE tool. We illustrate these scenarios with the screenshots of the current version of the tool.

Prior to performing analytical activities, it is necessary to select the target individual for such activities. QuASE tool interface for the target selection is depicted on Fig.6: it is possible to select the class for the selected item (the available classes depend on the selection of the corresponding “support” option in the site model), then the target individual has to be selected from the list (with keyboard-based name lookup).

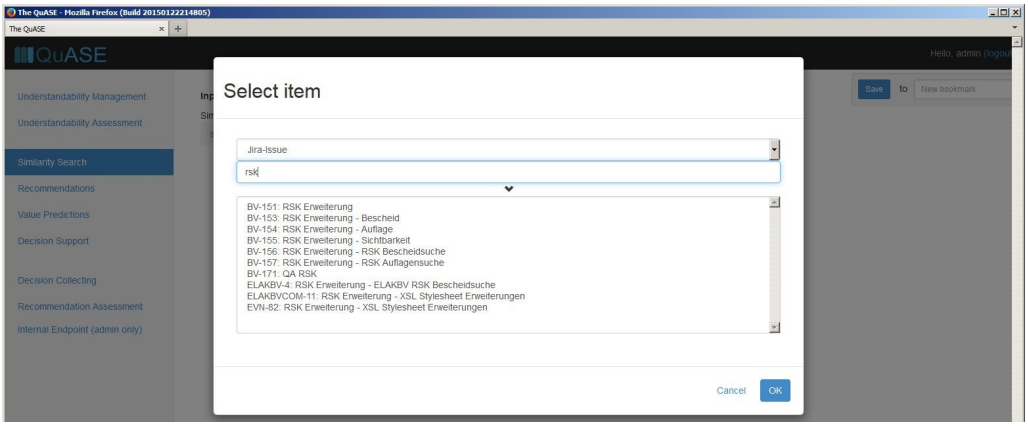

Fig. 6. Target selection interface of the QuASE tool

QuASE tool interface for similarity search is depicted on Fig.7: after selecting the target individual, the tool allows for selecting the subset of the relevant metrics, the list of results shows the values of these metrics for the target individual and for the list of the obtained similar individuals ordered by similarity distance.

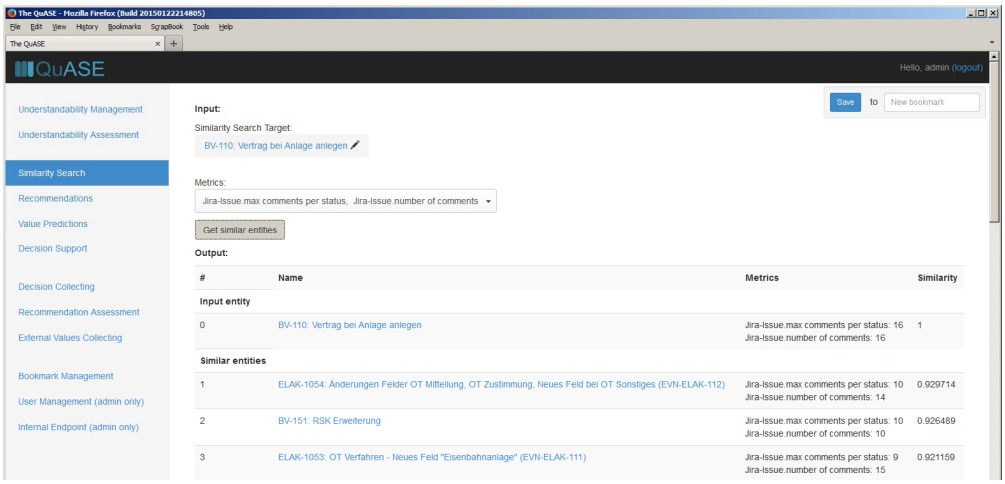

Fig. 7. Similarity search interface of the QuASE tool

QuASE tool interface for value predictions is depicted on Fig.8: after selecting the prediction target, the tool again allows for selecting the subset of the relevant metrics, the predicted values of these metrics will be available for the user after selecting the appropriate action.

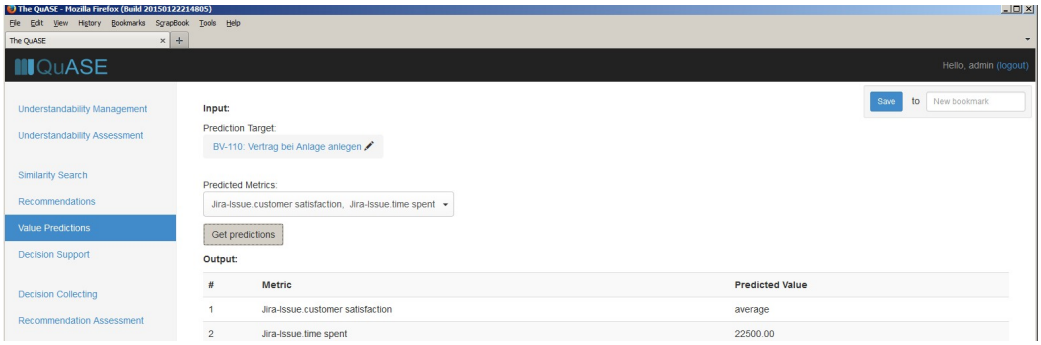

Fig. 8. Value prediction interface of the QuASE tool

QuASE tool interface for recommendation support is depicted on Fig.9. After selecting the recommendation target individual, the tool allows for selecting the recommendation kind out of the list of the kinds relevant for this target and then obtaining the list of preferred recommendations of this kind ordered by certainty rank.

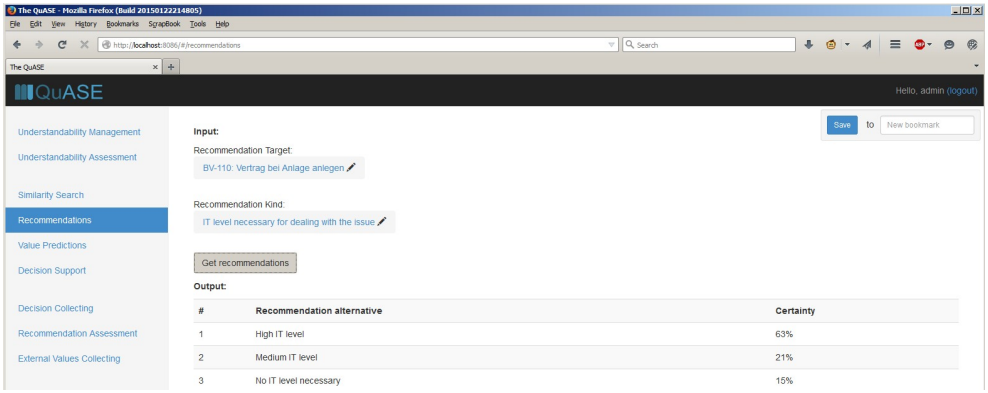

Fig. 9. Recommendation support interface of the QuASE tool

The interface for decision support is similar to the interface for recommendation support, the obtained results show ranked list of preferred decision alternatives.

Data collection subsystem allows the knowledge supplier to specify the data for decisions related to the selected target (Fig.10): after selecting both decision target and assessor, the list of appropriate decision kinds is shown to the user together with the information about the history of the selected alternatives. The tool then allows to select the decision alternative for every available decision kind and save the choice. Recommendation assessments are collected similarly to the decision data.

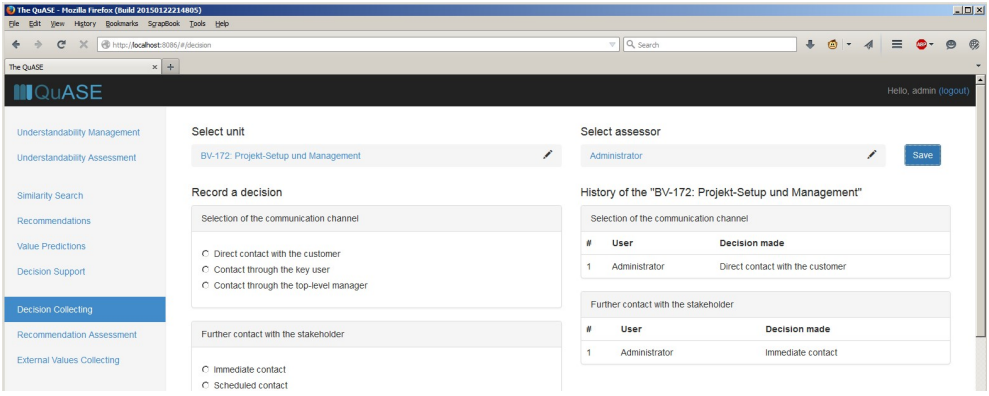

Fig. 10. Collecting decision data in the QuASE tool

To collect the values for interactive attributes, QuASE tool includes the functionality depicted on Fig.11. After selecting the individual, the list of available interactive attributes becomes available, the controls used in this list depend on the interaction type defined for the attribute (on Fig.11, customer attitude attribute is of type “attitude”, “customer satisfaction” attribute – of type “level”.)

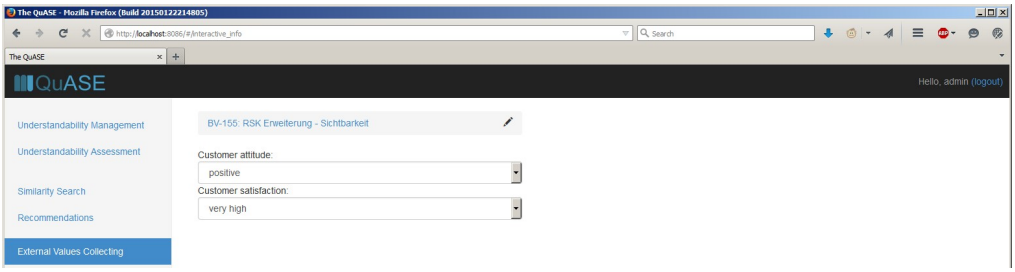

Fig. 11. Collecting values for interactive attributes in the QuASE tool

C. *QuASE Tool: Implementing Similarity Search*

In this section, we describe the implementation of similarity search in QuASE tool as a result of interaction with the Mahout framework. The input for the similarity search procedure is a data structure defining entities and their corresponding metrics.

The similarity search is implemented on top of three Mahout interfaces: (1) the *DataModel* implementation

queries the KB to retrieve entities and metrics belonging to these entities, these queries utilize Jena SPARQL endpoint to get the requested individuals; (2) the *UserSimilarity* implementations provide similarity measures used for calculating the distance between the queried entity and other entities based on the metric data from the provided *DataModel*; (3) the *UserNeighborhood* implementation uses the *UserSimilarity* to find entities similar to the queried entity: in QuASE, the *NearestNUserNeighborhood* implementation is used which finds *n* closest entities.

The output of the similarity search routine is the data structure containing entities sorted by their similarity to the queried entity, this structure is transferred to the web application layer via QuASE API, that layer then renders it by means of AngularJS UI rendering functionality.

D. QuASE Tool: Implementing Decision Support

The input for recommendation procedure is the data structure containing decision targets (content units or context units) and their metrics.

The decision support is implemented on top of the implementations of *AbstractVectorClassifier* component of the Mahout framework that allows training the classification model and classifying the production data with the trained model. For each decision kind a separate classification model is trained. The training of the model is performed by providing the following training data: (1) *the predictor variables* containing metrics of all decision targets connected to the respective decision kind; (2) *the target variables* containing the outcomes (decision alternatives) of the decision cases connected to the respective decision kind. Classifying production data using the trained classification model is performed by querying the classification model using the predictor variables of the production data (queried decision target and metrics) in order to retrieve the *probable target variables* (suggested decision alternatives and the classification probabilities for each alternative).

The output of the procedure is the set of decision alternatives sorted by their classification probability.

VI. RELATED WORK

The QuASE approach belongs to the category of solutions that facilitate reusing and analyzing the development knowledge [8, 9] based on applying knowledge management techniques to software engineering [1, 2]. Among these, from the point of view of addressed software process qualities, we can distinguish solutions separately addressing decision support quality [7]. Specific approaches implementing techniques for analysis of the communicated information in software development are provided in [3, 4].

The advantage of the QuASE approach is that it aims at an integrated solution that (1) targets reusability of communicated information, prediction of the future communication behavior, as well as decision quality, (2) handles the analysis based on the attributes of the fragments of communicated information contained in project repositories, as well as provided by the knowledge suppliers.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we described the implementation of the analysis support in the QuASE software solution. The proposed approach facilitates effective communication in the software process by making possible analyzing the properties of communicated information, and learning from past communication experience. Experiences in rolling out the current version of the system to the partner companies showed that our approach scales well and is flexible enough to be adapted to different communication environments.

Future direction for extending the analysis support in QuASE are related to defining and implementing the support for value-based and decision-based what-if scenarios [14] and for better integration with Jira and other development support systems available in industry.

REFERENCES

[1] M. A. Babar, T. Dingsøyr, P. Lago, and H. van Vliet, Eds., *Software Architecture Knowledge Management: Theory and Practice*. Springer, 2009.

[2] F. O. Bjørnson and T. Dingsøyr, "Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used," *Information and Software Technology,* vol. 50, pp. 1055-1068, 2008.

[3] P. Heck and A. Zaidman, "An analysis of requirements evolution in open source projects: Recommendations for issue trackers," in *Proceedings of the 2013 International Workshop on Principles of Software Evolution*, 2013, pp. 43-52.

[4] B. Kenmei, G. Antoniol, and M. Di Penta, "Trend analysis and issue prediction in large-scale open source systems," in *12th European Conf. on Software Maintenance and Reengineering*, 2008, pp. 73-82.

[5] J. Kuruvilla, *JIRA 5.x Development Cookbook*: Packt Publish., 2013.

[6] S. Owen, R. Anil, T. Dunning, and E. Friedman, *Mahout in Action*: Manning, 2011.

[7] G. Ruhe, "Software engineering decision support–a new paradigm for learning software organizations," in *Advances in Learning Software Organizations*: Springer, 2003, pp. 104-113.

[8] K. Schneider, *Experience and Knowledge Management in Software Engineering*. Berlin-Heidelberg: Springer, 2009.

[9] N. Sharma, K. Singh, and D. Goyal, "Can Managing Knowledge and Experience Improve Software Process?-Insights from the literature," *Research Cell: An International Journal of Engineering Sciences,* vol. 4, pp. 324-333, 2011.

[10] V. A. Shekhovtsov and H. C. Mayr, "Managing Quality Related Information in Software Development Processes," in *CAiSE-Forum-DC 2014*, CEUR Workshop Proceedings, vol. 1164, 2014, pp. 73-80.

[11] V. A. Shekhovtsov and H. C. Mayr, "Towards Managing Understandability of Quality-Related Information in Software Development Processes," in *ICCSA 2014, Part V.*, LNCS, v.8583, B. Murgante, S. Misra et al., Eds.: Springer, 2014, pp. 572-585.

[12] V. A. Shekhovtsov, H. C. Mayr, S. Ianushkevych, M. Kucko, V. Lubenskyi, and S. Strell, "Implementing tool support for effective stakeholder communication in software development – a project report," in *Anwenderkonferenz für Softwarequalität Test und Innovation - ASQT 2014* Wien: ÖCG, 2015, in print.

[13] V. A. Shekhovtsov, H. C. Mayr, and C. Kop, "Facilitating effective stakeholder communication in software development processes," in *CAISE'2014 Forum Post-proceedings*, S. Nurcan and E. Pimenidis, Eds.: Springer, 2015, in print.

[14] V. A. Shekhovtsov, H. C. Mayr, and V. Lubenskyi, "QuASE: A Tool Supported Approach to Facilitating Quality-Related Communication in Software Development," in *QUATIC 2014*, A. R. da Silva, A. R. Silva et al., Eds.: IEEE, 2014, pp. 162-165.