Autom Softw Eng (2015) 22:241–288
DOI 10.1007/s10515-014-0144-4

A semantic web enabled approach to reuse functional requirements models in web engineering

Samad Paydar · Mohsen Kahani

Received: 17 April 2013 / Accepted: 9 February 2014 / Published online: 27 February 2014

Abstract Web engineering has emerged as a new software discipline to specifically address the challenges and complexities of developing high quality web applications. A main theme in different web engineering methodologies is to employ model driven development approaches. This increases the level of abstraction and formalism to the extent that machines can better involve in the development process and provide more automation, e.g. automatic code generation from the models. Despite their benefits, a main problem of these model-driven methodologies is that developing each new web application implies creating a probably large number of models from scratch. Hence, model reuse can be considered as the main solution to this problem. In this paper, a semantic web enabled approach is proposed for reusing models, specifically functional requirements models. It takes the brief description of the functional requirements of a new web application in terms of UML use case diagram, and semi-automatically generates the draft of the corresponding detailed description in terms of a set of UML activity diagrams. This is performed by utilizing a repository which contains semantic representation of the models of the previous web applications. The proposed approach is based on novel algorithms for annotating activity diagrams, measuring similarity of use cases, and adapting activity diagrams. The experimental evaluations demonstrate that the proposed approach is promising, and it has good precision and effectiveness.

Keywords Web engineering · Semantic web · Annotation · Adaptation · Use case similarity · Reuse

S. Paydar (✉) · M. Kahani
Web Technology Lab., Department of Computer Engineering,
Ferdowsi University of Mashhad, Mashhad, Iran
e-mail: samad.paydar@stu-mail.um.ac.ir

M. Kahani
e-mail: kahani@um.ac.ir

1 Introduction

Web, in addition to being an elegant infrastructure for publishing and accessing information, has become a major software platform which enables delivering the application logic across the internet. In spite of the considerable contributions of the web application development field to software engineering, it has also introduced new challenges and issues which, due to specific characteristics and inherent complexities of web applications, cannot be fully addressed by the traditional software engineering practices (Murugesan 2008).

Web engineering has emerged in response to this problem as a software engineering discipline which specifically deals with the systematic design and development of web based systems (Murugesan 1999).

A number of web engineering methodologies have been introduced which seek to provide an effective process for web application development, for instance *OOHDM* (Schwabe and Rossi 1994), *WSDM* (De Troyer and Leune 1998), *UWE* (Koch 2000; Koch et al. 2008), and *WebML* (Ceri et al. 2000). These methodologies mostly employ a model based development approach, since it helps to overcome the development complexities by increasing the level of abstraction, and providing more machine support or automation. The general proposal of these methodologies is to separately model different aspects, e.g. presentation or navigation, of a web application, and further, to use a mechanism for automatically or semi-automatically transforming these models to other required artifacts, e.g. code or other types of models.

Despite the benefits of applying model based approaches, they have the disadvantage that development of each new application requires creating new models from scratch, and for large applications it means that several models must be developed, all of which need to be carefully thought, designed and verified.

Great improvement will be achieved if model reuse is supported by a web engineering methodology. This makes it is possible to use models of the previous web applications for creating models of a new but similar web application. Considering the fact that there are many similarities and commonalities between the functionalities of different web applications, providing a model-reuse method for web engineering becomes more appealing.

In this paper, a semantic web enabled reuse approach is proposed which can be applied for web engineering. It is based on the idea that the requirements models are the first and the most important types of models, and many web applications have similar requirements, especially functional requirements. Therefore, it is possible to create the requirements models of a new web application by using or adapting the requirements models of the similar existing applications.

Among the different types of requirements models, the current work focuses on the functional requirements, and it is assumed that they are modeled as UML use case diagrams and UML activity diagrams, correspondingly for the brief and the detailed description. The proposed approach takes as input the partial specification of the requirements of a new system in terms of its use case diagram, and using a repository of the models of existing systems, it semi-automatically completes the requirements models by creating the corresponding activity diagrams.

Since an important component in any reuse approach is a search facility, and the semantic web technologies are known to be successful in improving search and information retrieval, the proposed approach employs the semantic web technologies to provide a flexible and effective search and analysis mechanism. Further, the semantic web data sources and ontologies are used for providing the required knowledge during the reuse process.

The contributions of this work can be briefly mentioned as proposing:

- A semi-automatic reuse process for reusing functional requirements model. This process employs:
  - a new method for creating a semantic layer over the UML models
  - a new metric for measuring the semantic similarity of UML use cases, in their diagrammatic format
  - a novel algorithm for UML activity diagram adaptation

There are few works which address reuse based on using UML use case diagrams as input. Since this diagram is considered as the first and the easiest type of models created for the requirement specification, the proposed approach inserts reuse at the very early stage of the development with low cost. Further, while most existing works only deal with retrieval of reusable assets, the proposed approach seeks to address adaptation of the retrieved assets, and therefore provides a complete reuse process.

The initial experiments demonstrate that the proposed approach is sound and promising, although more works are still needed to fully achieve the potentials.

The paper is organized as follows: a brief literature review is described in Sect. 2. In Sect. 3, the proposed approach is introduced and its underlying concepts and algorithms are described in details. The evaluation of the proposed approach is presented in Sect. 4, and finally, Sect. 5 concludes the paper and mentions some of the future works.

2 Related work

The proposed approach spans different fields, e.g. web engineering, requirements engineering, and software reuse. As a result, there are many works that are in some way related to it. This section provides a review of the related works under different categories.

2.1 Software reuse

Software reuse is an old subject in software engineering (it can be traced back to McIlroy 1968), which focuses on the use of existing software assets to create new software instances, with the goal of improving software quality and productivity (Frakes and Kang 2005; Mili et al. 1995; Mohagheghi and Conradi 2007).

It is acknowledged that implementing a systematic reuse approach in the software development lifecycle is a complex issue that not only has technical requisites, but also involves non-technical and human factors (Card and Comer 1994). Morisio et al. (2002) discuss the main factors that lead to success or failure of a company-wide reuse program. For instance, one of the root causes of reuse failure is the misconception that just using the object oriented technology, or just having a repository of software assets necessarily leads to successful reuse. Some of the barriers of successful software reuse are lack of tool support, lack of management commitment, and inability to precisely estimate the return on investment when building component for potential reuse in the future (Catal 2009; Sherif and Vinze 2003; Jalender et al. 2010). Regarding the last issue, there are some works that propose economical models for analyzing the cost and benefit of developing software for reuse and with reuse (Malan and Wentzel 1993; Lim 1994; Milli et al. 2000; Postmus and Meijler 2008).

In Mens et al. (1999), it is discussed that a requirement of disciplined reuse is the ability to preserve a high level of consistency between reusable components and the systems in which these components are reused, considering the fact that both parts are subject to changes during their evolution. The UML metamodel is extended to support the notion of reuse contracts (Lucas 1997; Steyaert et al. 1996), and to enable automatic detection of conflicts between the reuser system and the reused component, with regard to the reuse contract.

In Anguswamy (2012) the author discusses and analyzes the reuse design principles which are more frequently used to support *Design for Reuse*, i.e. designing components in such a way that they possess a good level of reusability. Further, different factors which affect *Design with Reuse* are experimentally studied. For instance, the relationship between reusability of a component and its size in terms of SLOC[1] is evaluated. In a similar work, (Anguswamy and Frakes 2012) study the relationship between complexity of a component, its reusability and the reuse design guidelines which are followed in building that component.

Reuse at source code level has long been explored by the researchers and used by the practitioners (Lemos et al. 2007). Different terms are used in the literature for referring to works that provide reuse at source code level, for instance *source code search engines* (Hamid 2013; Bajracharya et al. 2014; McMillan et al. 2012a) and *source code recommendation* (Holmes et al. 2006; McMillan et al. 2012b). However, software reuse is not limited to source code (Cheng and Budgen 2012). Jones (1993) identifies ten aspects of software projects which are potentially reusable, for instance, designs, requirements and test cases.

It is known in the literature that the earlier reuse is conducted in the development life cycle, the more beneficial it is (Sen 1997; Barros 2010; Smialek et al. 2010). For instance, reuse at design stage has more benefits compared to reuse at implementation phase (Karlsson 1995; Falessi et al. 2011). On the other hand, reuse at design level is also more challenging, due to complexities of this step, e.g. the variety of models to be designed and the variety of aspects to be modeled (Ali and Du 2004; Robles et al. 2012).

There are a number of works focusing on model reuse. *Moogle* (Lucredio et al. 2008) is a general purpose model search engine capable of indexing and searching UML models. It supports keyword-based as well as faceted search. Bozzon et al. (2010) propose a method for model search which uses information retrieval (IR) techniques. Another model search engine is also proposed by Nowick et al. (2005). Kling et al.

[1] Source line of code

(2011), propose *MoScript*, a *Domain Specific Language* (*DSL*) through which users can store, retrieve, and manipulate models within a model repository by a query based approach. The goal is to provide an efficient facility for reusing models or model fragments. In Bislimovska et al. (2011), the existing models are represented as labeled graphs and stored in model repositories. The input of the reuse approach is also a labeled graph model. Then a graph-based similarity measure is used to search over the model repositories to find the similar models. As another example of model reuse, an approach is presented in Zhuge (2002) for retrieval of workflow processes.

Due to the importance of UML as the de-facto standard modeling language in software domain, there are many works that address reuse of some type of UML models. For instance, Robles et al. (2012) propose an ontology-based approach for retrieving UML class diagrams. It uses the ontologies for measuring semantic similarity of the model elements, and also for query expansion. As another example, Gomes et al. (2007) also propose an approach for reusing class diagrams. In Salami and Ahmed (2013), existing works on reusing UML artifacts are surveyed and four categories are identified for techniques used to compare a query artifact with artifacts in the repository. These include techniques based on (1) graph matching algorithms (Robinson and Woo 2004; Park and Bae 2010; Salami and Ahmed 2012), (2) ontology-based similarity measurement (Robles et al. 2012; Bonilla-Morales et al. 2012), (3) case based reasoning (CBR) (Gomes et al. 2002, 2004), and (4) IR methods (e.g. classification, clustering, vector space similarity) (Bloc and Cybulski 1998; Alspaugh et al. 1999; Ali and Du 2004).

While about half of the UML diagram types are not yet considered for reuse, UML class diagrams, sequence diagrams and use case diagrams are the three kinds of diagrams that are considered the most in existing works (Salami and Ahmed 2013). Among the UML models, use cases play an important role, especially for software requirements specification (SRS) (Jacobson 1992). They are among the earliest artifacts generated in the development life cycle. Hence, it is very interesting to provide a use case based reuse process (Jacobson 2004). The existing works on use case based reuse can be classified to two groups: (1) reusing use case descriptions which include textual description of the basic flow of events and probably some other information, e.g. post-conditions (Saeki 1999), and (2) reusing use case diagrams in their diagrammatic format. While most of the works belong to the first group Salami and Ahmed (2013), but Srisura and Daengdej (2010) discuss that it is important to be able to reuse use cases in their diagrammatic form, i.e. not their textual description.

In the first group, Udomchaiporn et al. (2006) propose an approach for retrieval of use case descriptions based on their structure. A set of 12 features is considered for each use case descriptions, e.g. use case name, its objective, and the names of the use cases which it generalizes. For each existing use case description, the words appearing in these features are extracted and indexed using traditional IR techniques. In the retrieval process, the user builds his query by specifying the words that should appear in each of the 12 features of the desired use case description. Further, the user has to specify a weight value (from 1 to 5) for each feature. Finally, weighted textual similarity is used for comparing user query with repository models. No semantic similarity measurement is considered. As another example, Bloc and Cybulski (1998) introduce a method for retrieval of UML use case descriptions. It compares the uses cases by measuring similarity of their corresponding sequences of event flows. This is based on the similarity of the term vectors built from the names of the event flows. It also employs a clustering method to organize existing use cases in clusters, in order o reduce complexity of the comparison step. This clustering is performed manually by the experts, which provides very reliable and accurate clusters. However, as the authors state, it has *'considerable expense'*. Further, it is possible that the results are biased by the expert's points of view.

In the second group, Srisura and Daengdej (2010) employ CBR for retrieval of UML use case diagrams. The main component of their approach is a metric which measures similarity of two use case diagrams along two dimensions. In the first dimension, textual similarity of the use case diagrams is computed by exact matching of the names of the use cases, actors and the subject systems. The second dimension deals with different relationships in the use case diagrams by considering type, direction and multiplicity of the relationships. As another example, an approach is proposed in Bonilla-Morales et al. (2012) for reusing use case diagrams. Input use case diagrams are processed and information of the actors and use cases are extracted and transformed to ontological representation. This representation is stored in a repository, and finally an interface, actually a search form, is provided for retrieving use case diagrams through simple keyword-based search.

A complete reuse approach should provide support for four steps: representation, retrieval, adaptation, and incorporation (Prasad and Park 1994). Review of the existing works demonstrates that most works address only the first two steps, i.e. they only support retrieval of the artifacts which are promising for reuse, but no specific facility is provided for actually reusing the retrieved artifacts for the context of the new problem, e.g. (Robles et al. 2012). In other words, the adaptation phase within which the reuse is actually performed is left to the user.

Current paper proposes a new approach for model reuse. It is different from most existing works mainly in the sense that it provides a complete reuse process, which specifically addresses the adaptation step by a novel adaptation algorithm. The approach is semi-automatic and the level of required user's intervention is low. Further, the proposed approach employs ontological representation of the models, and also a novel annotation algorithm to augment existing models with helpful information. These are used to address the 7th and the 8th requirements discussed by Elias and Johannesson (2012) for a model repository.

The proposed approach focuses on reuse of UML use case diagrams in their diagrammatic format, not their textual description, and therefore it reduces the cost of initiating reuse. There are very few works in this regard. Compared to Srisura and Daengdej (2010) and Srisura and Daengdej (2010), our approach proposes a new metric for measuring similarity of two use cases, not two use case diagrams. Further, it covers both text and relationship dimensions, but it employs semantic similarity instead of simple textual exact matching.

In Frakes and Terry (1996), six categories of metrics and models for software reuse and reusability are reviewed and a classification of the types of software reuse is proposed. Based on this classification, the reuse approach of the current paper involves:

- Development scope: external (reusable models are from external projects)
- Modification: white box (the adaptation algorithm is intended to do internal modifications)
- Approach: leveraged
- Domain scope: horizontal (since it is possible that the reused models belong to different domains)
- Management: ad-hoc (the proposed reuse approach cannot not yet be considered systematic)

2.2 Software similarity metrics

The proposed approach of this paper includes a novel metric for measuring similarity of UML use cases. Therefore, some existing works on software similarity metrics are briefly reviewed in this section. Different metrics are proposed in the literature for measuring similarity of various types of software artifacts. They are proposed in different application areas, for instance:

- Software clone detection. Clones are known to reduce productivity and increase maintenance cost (Monden et al. 2002). In Deissenboeck et al. (2008) an approach is proposed which uses a graph-based similarity measure for automatic detection of clones in large models in the context of model driven development (MDD).
- Difference computation and version management. Due to the collaborative and incremental nature of many software development activities, some mechanism is required for computing differences of two possibly consecutive versions of an artifact. Different methods for computing differences of two UML models are proposed (Kelte et al. 2005; Treude et al. 2007). A review of these works is provided by Selonen (2007). As the author acknowledge, most existing works focus on UML class diagrams. For instance, in Girschick (2006) new algorithm is presented for calculating differences between UML class diagrams, based on different characteristics, e.g. name, stereotype, type and signature, of the class diagram elements.
- Software reuse. An essential component in any reuse method is the similarity metric used for comparing the query artifact with the repository artifacts during the retrieval step. Bildhauer et al. (2009), describe the method which is used for measuring similarity of requirements specifications in the context of ReDSeeDS project. Requirements are specified in terms of (1) requirement statements in natural language sentences, and (2) use cases scenarios in a constrained languages. Similarity of the requirements specifications of two software cases is then computed by a combination of word and sentence similarity algorithms, e.g. (Li et al. 2006), and structural similarity metric. For the latter case, use case scenarios are transformed to graph representation and graph-based similarity metrics are employed using SiDiff (Kelte et al. 2005).

Software similarity measurement is also studied at different levels of granularity and for different types of artifacts, e.g. similarity of complete software systems (Yamamoto et al. 2005, 2007), program similarity (Walenstein et al. 2007), and model similarity (Stephan and Cordy 2012).

There are some works which compute similarity of UML models by transforming them to graphs and then employing graph matching techniques, e.g. (Woo and Robinson 2002). However, as discussed by Salami and Ahmed (2013), it is recommended that the retrieval techniques for each type of UML models should consider semantics of that model type, instead of using generic metrics.

With regard to these related works, the current work proposes a new metric which is specifically designed for measuring similarity of two use cases in the context of their use case diagrams. This specificity makes it possible to utilize semantic of use cases. A similar work is described in Srisura and Daengdej (2010) however our work is different with it in a number of ways. First, the metric of Srisura and Daengdej (2010) uses exact string matching and does not employ semantic similarity of the concepts. Further, it measures similarity of two use case diagrams not two use cases. The latter is what the proposed reuse method of the current paper requires. Further, an important novelty of our metric, which is an example of considering use case semantics, is that it distinguishes between behavior and concepts of a use case, and provides simple algorithms to detect them.

It must be noted that the metrics introduced for computing differences of the models were not applicable for the current paper, since first, they produce as output a list of differences not a real value. This has the problem that if a new model M_1 is compared with two repository models M_2 and M_3, it is not clear how to compare the output of the comparisons, i.e. the differences of M_1 and M_2 with the differences of M_1 and M_3. This is required for identifying which of M_2 and M_3 is more similar to M_1. Further, none of the existing metrics specifically address two use cases with considering semantics of these models.

In Bin et al. (2009), a hybrid metric is introduced for measuring semantic similarity of two concepts. It is based on a combination of different graph-based metrics [e.g. length of the shortest path between the two concepts on a concept tree extracted from WordNet (Miller 1995], local density and connect power of the edge) and information content metrics.

In Trakarnviroj and Prompoon (2012) a reuse method for Software Product Line (SPL) is introduced. In this method the requirements models and analysis models are stored in a repository. These include use cases, feature models, static and dynamic models. For each model in the repository, all of its words are extracted and indexed based on the conventional TF/IDF measure. Further, a keyword-based search mechanism is provided which enables the user to retrieve models. This is performed by vector space cosine similarity computation between the query keywords and the index terms.

The proposed approach introduces a new metric for measuring similarity of UML use cases. It is different from other works like Trakarnviroj and Prompoon (2012) since it considers not only textual, but also the relational aspects of the use cases. Further it is not limited to simple text similarity, and it employs semantic similarity techniques.

2.3 Domain analysis

Domain analysis is the process of identifying the features, and abstracting the concepts and relationships which are common to a family of similar systems in a specific application domain (Neighbors 1981). Domain analysis has long been identified as a crucial activity in establishing software reuse program (CAMP 1987).

In Prieto-Diaz (1990), the basic concepts and ideas of domain analysis are described, and a domain analysis process model is proposed. A discussion on the relationship of domain analysis and software reusability is presented in Prieto-Diaz (1987), and domain analysis is considered as a key factor affecting success of software reusability.

In Frakes et al. (1998), DARE is introduced as a CASE tool for domain analysis with the goal of providing an infrastructure for systematic reuse (Frakes and Isoda 1994). DARE uses the notion of Domain Book for organizing large amount of domain information collected from different sources.

The proposed approach of this paper does not introduce new techniques for domain analysis. However, it seeks to provide insights on the idea that the semantic web can be used as a source of domain information, and the kind of information that result from domain analysis, can be retrieved from the semantic web sources. Considering the increasing growth in the semantic web data volume, e.g. the Linking Open Data (LOD)[2] cloud, and the fact that new sources are continuously joined the semantic web data space, this idea becomes more promising.

2.4 Traceability

Traceability is the ability to identify, establish and follow the traces between different artifacts during the software development and maintenance lifecycle (Ramesh and Jarke 2001). Traceability links are a more concrete form of these traces which determine the reason for existence of an artifact or its relationships to other artifacts, probably through a predecessor-successor or master-subordinate relationship (IEEE 1990).

Traceability links can be utilized for different purposes, e.g.:

- validating downstream artifacts against upstream requirements or specifications (Watkins and Neal 1994)
- understanding the system and its artifacts (Sabetzadeh and Easterbrook 2005; Paige et al. 2008)
- improving maintainability of the artifacts and enabling reengineering activities (Arkley and Riddle 2005; Ebner and Kaindl 2002)
- finding reusable elements and enabling creation of new variants of an existing system (Winkler and Pilgrim 2010)

In Winkler and Pilgrim (2010) the authors provide a detailed survey of the current approaches, applications and challenges of traceability in two domains of MDD and requirements engineering.

There are different categorizations of traceability and traceability links in the literature. For instance, Gotel and Finkelstein (1994) have identified two categories: pre-requirements specification (pre-RS) traceability and post-requirements specification (post-RS) traceability. Ramesh and Edwards (1993) distinguish between horizontal

[2] http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

and vertical traceability. The former is the ability of following the traceability links between artifacts of the same development phase or the same level of abstraction. The latter deals with traceability links between artifacts of different development phases or different abstraction levels. Spanoudakis and Zisman (2005) also classify the traceability links to eight categories.

The proposed approach of this paper includes a new algorithm for activity diagram annotation. The purpose of this algorithm is to create explicit links between the labels of the elements of an activity diagram and the names of the classes, attributes and actors. These links, i.e. annotations, are later used by the proposed adaptation algorithm to identify the points an existing activity diagram where labels should be modified to generate a new activity diagram. Consequently, these annotations can be considered as a kind of traceability links. The way these traceability links are created and used by the annotation and adaptation algorithm is an important novelty of the current paper.

Regarding the classification of Spanoudakis and Zisman (2005), the annotations of the proposed annotation algorithm are of type dependency links. The idea behind the proposed adaptation algorithm is actually how to utilize these dependencies to perform required changes.

Based on the classification of Gotel and Finkelstein (1994), the traceability links generated by the proposed annotation algorithm are post-RS traceability links, since they establish links between elements of the functional requirements specification (i.e. the activity diagrams) and the elements of the design models (e.g. classes in the class diagram). Further, these annotations are vertical traceability link, based on the classification of Ramesh and Edwards (1993), since they establish a relationship between artifacts of the requirements specification phase and the design stage.

2.5 Semantic web enabled software engineering

The potential applications of the semantic web technologies for software development, along with their advantages and challenges are described in different works like (Bauer and Roser 2006; Calero et al. 2006; Happel and Seedorf 2006).

The formalism, inference capability, and machine-understandability that are provided by ontologies and the semantic web data model are frequently utilized in software engineering domain for different goals. Two main approaches can be identified in the existing works: (1) using the semantic web technologies, and (2) using the sole semantic web.

In the first approach, the semantic web technologies like ontologies and OWL reasoners are used to provide a better representation format, so that analysis and processing of data is improved. This theme is followed for different purposes, like:

- Integrating different types of software artifacts (Iqbal et al. 2009)
- Improving search and retrieval of software components (Yao et al. 2008; Alnusair and Zhao 2012, 2010; Hartig et al. 2008; Durao et al. 2008; Keivanloo et al. 2010)
- Software analysis (Tappolet et al. 2010)
- Automating or facilitating software engineering tasks like design pattern detection (Paydar and Kahani 2012) or configuration management (Shahri et al. 2007)
- Providing a flexible and extensible data infrastructure (Nyulas et al. 2009)

In the second approach, the semantic web, as it is realized already, is used as a set of sources which provide machine understandable data. These sources include, for instance, the well-known and rich ontologies, linked data (Berners-Lee 2006) sources and semantic web search engines. This theme is followed in the literature for different purposes, like:

- Providing common-sense knowledge for machines to automate a processes (Castaneda et al. 2010; Korner and Gelhausen 2008)
- Improving *Natural Language Processing* (NLP) tasks by eliminating natural language ambiguities (Castaneda et al. 2012; Korner and Brumm 2009)

Reviewing works in the domain of semantic web enabled software engineering demonstrates that while the first approach is used in many works (Tappolet et al. 2010; Paydar and Kahani 2012), the second is considered in very few works, e.g. (Korner and Brumm 2009). In other words, the use of the sole semantic web is underestimated in software engineering tasks.

The proposed approach is related to the semantic web since it uses both approaches described above. It uses the *semantic web technologies* to provide an ontology-based data model. This enables a flexible and effective mechanism for accessing and analyzing information of the software assets, and also benefits from the reasoning capability of ontologies to tackle with some complexities. In addition, it uses the *semantic web sources* (e.g. linked data sources) as a large data space which provides background and domain knowledge in a machine-processable format. More specifically, the proposed adaptation algorithm uses the semantic web sources like LOD SPARQL Endpoint as an external repository and searches it for the kind of information which is found in UML class diagrams. This utilization of the semantic web sources for software reuse is less considered in existing works.

With regards to reusability, the use of the semantic web technologies as a representation format has the advantage that it increases *availability*, *searchability* and *understandability* of the reusable software assets, the three properties that improve reusability of the assets (Frakes and Kang 2005). Further, it provides reasoning, the facility which its necessity is acknowledged in the reuse research domain (Frakes and Kang 2005; Alnusair and Zhao 2010, 2012). It is also believed that a keyword-based retrieval mechanism is not sufficient for the effective reuse, and semantic aware retrieval methods provide better results (Robles et al. 2012).

2.6 Automatic model creation

Creating precise and valid models, especially for large systems, is a time-consuming task, which can be improved by increasing the machine involvement. As a result, there are many works on automatic model creation, whether by extracting the model from textual specifications of the requirements, or by transforming some other types of models, or by hybrid techniques.

For instance, an approach is proposed in Ilieva and Boley (2008) which takes the textual requirements specification as input and uses NLP techniques to translate it to a special kind of graph. This graph is then used to generate the corresponding UML diagrams (e.g. class or use case diagram).

In Korner and Gelhausen (2008), an approach is proposed for extracting domain models (in terms of UML class diagrams) from natural language textual specifications of the system. It is stated that some level of common-sense knowledge is required in order for the machines to be able to overcome ambiguities of natural language texts. An ontology, specifically the *RCyc* ontology[3], is used to provide such knowledge. In Samarasinghe and Some (2005) also an approach is proposed for producing domain models from use cases specified by a controlled natural language grammar.

Use case modeling is a traditional approach for the specification of the requirements (Jacobson 2004; Dobing and Parsons 2006; Klimek and Szwed 2010), in which UML use case diagrams are used to provide a high-level definition of the requirements while another technique, e.g. UML activity diagrams or textual templates, are used for more detailed definition (Bendraou et al. 2010; Lauesen and Kuhail 2012).

Due to the importance of use case modeling as an early step in the design stage, many researchers have worked on approaches for automatic extraction of other types of models from the use case specifications, for instance (Anda and Sjoberg 2005; Liang 2003; Yue et al. 2010a, b; Gronmo and Moller-Pedersen 2011). A review of some of these works is presented in Yue et al. (2011). A main difference of these works with the current work is that they take the detailed specification of the use case as input, e.g. (Some 2003), while the current work takes only the use case diagram as input, and therefore it has a lower cost to apply.

In Yue et al. (2013), the authors present a set of well-defined rules that restricts the way users specify the textual specification of use cases. Enforcement of these restrictions reduces the possible ambiguity of the textual specification, hence provides more potential for the derivation of analysis models through NLP or other techniques. The authors have evaluated their rules and concluded that while they are easy to understand and use, they also improve the quality of the derived models.

MDD Mellor et al. (2003), as a software development trend, focuses on the models as the main driver of the development process. Some kinds of models are developed manually, and then they are used by automatic or semi-automatic model transformation techniques, to create other kinds of models or even the final code (which is still a kind of model) (Selic 2003; Koch et al. 2012). Textual specifications are not much interesting from the point of view of *MDD*, due to their ambiguities and the lack of precision and formalism (Vilain et al. 2000). In Atkinson and Kuhne (2003) the authors discuss the importance of visual modeling in model-driven architecture.

In Kang et al. (2010) a transformation mechanism is proposed for synthesizing UML activity diagrams from scenario-based specifications which are modeled as UML sequence diagrams. The transformation is performed based on a set of rules that define the mapping for a number of predefined patterns in the sequence diagrams.

An essential component in all of these works is a set of rules for transforming models of the source type to models of the target type. Fig. 1 (top) shows the traditional approach followed in these works.

The proposed approach of the current paper seeks to get use case diagrams as input and generate activity diagrams as output. Since these two types of diagrams are at

[3] CyCorp. ResearchCyc. http://research.cyc.com.

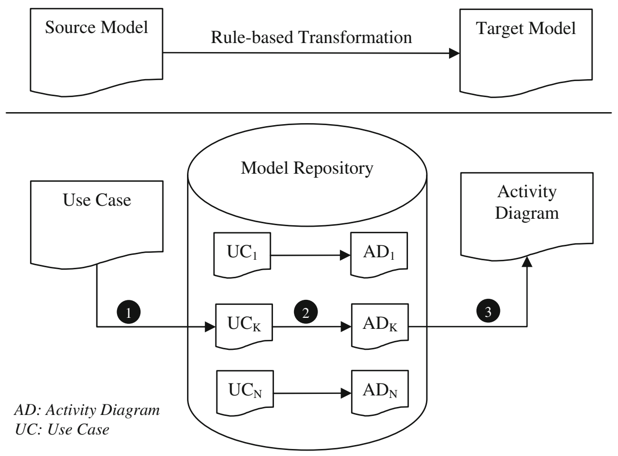


Fig. 1 Traditional approaches of automatic model creation (*top*), the proposed approach of this paper (*bottom*)

very different levels of abstraction, the traditional approach is not promising since providing a rule set for the required transformation is very challenging, if possible. To tackle with this problem, a model reuse approach is employed. As shown in Fig. 1 (bottom), this approach bridges the input and output using three links numbered in this figure. The first link is established by the proposed use case similarity metric which identifies, from the model repository, the most similar use case to the input use case. Since the associated activity diagram of this use case already exists in the model repository, the second link is already established in the repository. The third link is established by the proposed adaptation algorithm which adapts the repository activity diagram to create the new activity diagram. This approach has eliminated the above challenge and actually no transformation rules between two different model spaces are defined in the proposed approach.

2.7 Web engineering

Web engineering is a software engineering discipline which specifically deals with the systematic design and development of web based systems (Murugesan 1999). Different web engineering methodologies have been introduced during the last decade, which mostly follow the *MDD* approach (Aguilar et al. 2010). This has led to emergence of model-driven web engineering (MDWE) as a specialization of model driven development for web engineering.

In Valderas and Pelechano (2011) the authors have surveyed nine *MDWE* methods that explicitly consider requirements modeling in the development process. The goal is to identify what technique they provide for the specification of data, functional and navigational requirements, the extent of their capabilities for specifying these requirements, and how they use these requirements models in a *MDD* way? As this survey shows, UWE (Koch 2000) receives good score from both perspectives of the requirement specification expressiveness and *MDD* support. *NDT* (Escalona and Aragon 2008) is also an interesting method which has many commonalities with *UWE*, but the extensive use of various formatted templates, which are not easy to complete, increases the complexity of modeling large web applications. As this survey shows, use cases and activity diagrams are very common for specification of the functional requirements in web engineering methods.

While the proposed approach of this paper does not depend on a specific web engineering methodology, but it is more interesting if considered in conjunction with *UWE* method. It seems promising to combine advantages of UWE with the benefits of the proposed approach. First of all, *UWE* has the advantage that its modeling notation is completely based on UML. Further, its design process provides good support for *Model Driven Architecture* (*MDA*) (OMG 2005). *UWE* initially used use case diagrams and textual use case descriptions for requirement specification. However, in its recent version, a UML profile named *WebRE* (web requirements) is used for this purpose (Escalona and Koch 2007). This profile is based on the use of use case diagrams along with activity diagrams, correspondingly for brief and detailed specification of requirements. *UWE* requirements models play the role of *CIM*[4] in *MDA*, and they are used through a model to model transformation to create the initial version of the navigational and content models. These models, after necessary refinement, can be treated as *PIM*[5] and automatically transformed to code by tools like *UWE4JSF*[6]. Current paper proposes a reuse method which takes the brief specification of the requirements in terms of use case diagram, and semi-automatically generates the detailed description of the requirements in terms of UML activity diagrams. Since *UWE* supports automatic derivation of content and navigational models from the requirements models, i.e. use case diagrams and activity diagrams (Koch and Kozuruba 2012), therefore the proposed approach can be pipelined with *UWE* to provide a high level of automation in creating models.

3 The proposed approach

It is very common that two different web applications have similar functionalities, and therefore similar functional requirements (Murugesan 2008). Therefore it is possible to reuse requirements models of an existing software case for modeling a new one (Cheng and Atlee 2007). Based on this idea, the proposed approach seeks to insert the reuse process at an early time in the design stage, i.e. when functional requirements are being specified. These requirements are considered to be described using UML use case diagrams for brief specification, and UML activity diagrams for detailed specification. The goal is to take the brief specification of the functional requirements of the new software, and semi-automatically create the draft of the detailed specification by reusing models of the previous software cases.

[4] Computational independent model

[5] Platform independent model

[6] http://uwe.pst.ifi.lmu.de/toolUWE4JSF.html

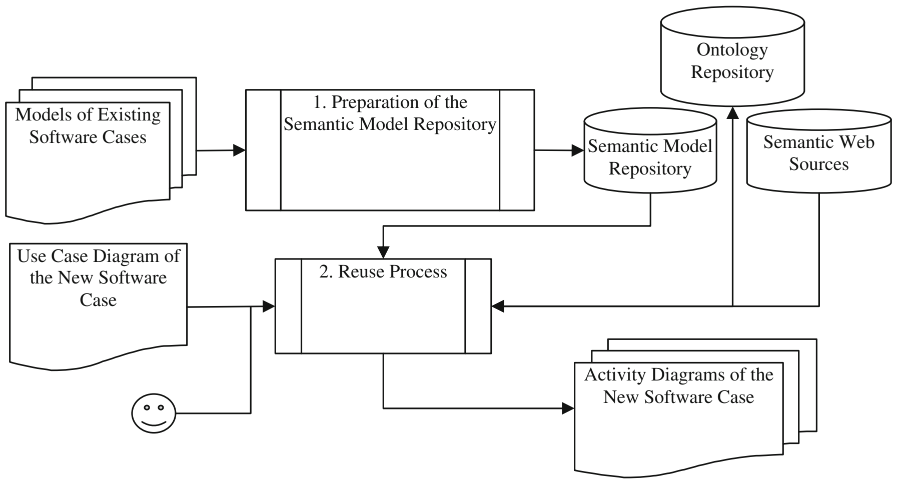

Ontology Repository
Models of Existing Software Cases
1. Preparation of the Semantic Model Repository
Semantic Model Repository
Semantic Web Sources
Use Case Diagram of the New Software Case
2. Reuse Process
Activity Diagrams of the New Software Case

Fig. 2 Overview of the proposed approach

The overview of the proposed approach is shown in Fig. 2. It consists of two main phases: (1) preparation of the semantic model repository, and (2) reuse process. In the first phase, a semantic repository is created which contains the UML models of previous software cases. For each software case, there is a use case diagram, a set of activity diagrams and a class diagram in the repository. It is assumed that each use case in the use case diagram is described in more details by an activity diagram. By semantic repository it is meant that the contents of the repository, i.e. UML models, are represented in the semantic web data model, which is an ontology-based data model.

Once the repository is prepared, the reuse process can be performed. It takes a use case diagram as input, and semi-automatically creates the draft of the activity diagram of each use case in the input.

Since the input of the reuse process is just a use case diagram, the cost of applying this process would be low, because creating use case diagrams is simple and easy (compared to creating other types of models). However, since use case diagrams are simple and concise, they do not convey much information. To address this lack of information, the proposed approach seeks to use the model repository, an ontology repository and the semantic web sources.

Next, detailed description of each phase of the proposed approach is presented.

3.1 Preparation of the semantic model repository

The goal of this step is to represent information of the existing models based on the semantic web data model. The reason is that it is experimentally acknowledged, for instance by Tappolet et al. (2010) and Paydar and Kahani (2012), that this transition to the semantic web data model provides benefits for accessing, retrieving and analyzing information of the software assets.

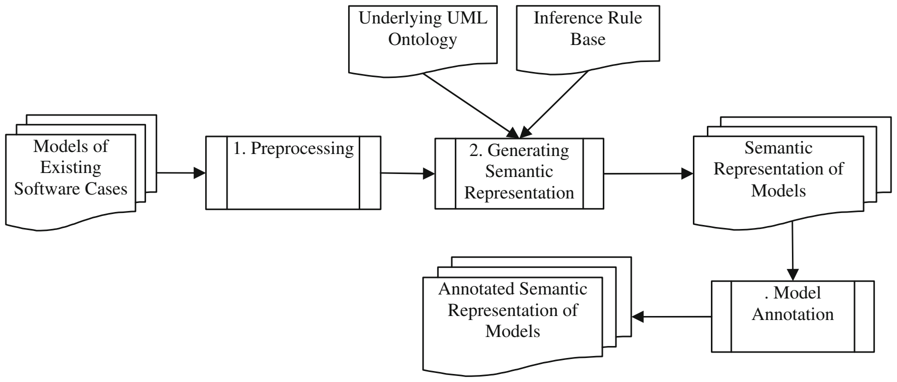

Underlying UML Ontology
Inference Rule Base
Models of Existing Software Cases
1. Preprocessing
2. Generating Semantic Representation
Semantic Representation of Models
Annotated Semantic Representation of Models
. Model Annotation

Fig. 3 Overview of preparation of the semantic model repository

In addition, an annotation algorithm is executed on the existing models to prepare the adaptation points, i.e. the points in the models where modifications can be performed to create new models.

The overview of this phase is shown in Fig. 3. It consists of three main steps: (1) preprocessing, (2) generating semantic representation, and (3) model annotation. Next, these steps are described in details.

3.1.1 Preprocessing

The models which are to be included in the model repository are intended to be reused in the future for the purpose of creating new models. Hence, it is reasonable to expect these models to have passed through some quality assurance mechanisms. This is the goal of the preprocessing phase. In this phase, the input models are checked and potential problems are reported to the repository manager, i.e. the person who is responsible for managing content of the model repository. After the manager has made his decision about the reported issue, the model is ready to pass to the next phase for being added to the model repository.

There are different kinds of problematic issues which can be checked in this step. Further, different approaches, e.g. automatic versus manual correction, are possible for handling the detected issues. Therefore, various mechanisms can be implemented in this step. Here, a simple mechanism which is used in the experiments of this paper is described. In this mechanism, for each label in the input model (e.g. name of the use case or label of an element in an activity diagram):

- The label is tokenized and then given to a spell checker. Any erroneous token, along with possible suggestions, is reported to the repository manager and he can decide whether to correct the reported issue or to ignore it. This is intended to eliminate the problems caused by misspellings, unintelligible abbreviations or dummy names in the models. These issues can reduce effectiveness of the proposed approach, since for example this approach includes an annotation phase in which n-gram string matching is employed.

- Each token is given to a stemmer. Any token which is not stemmed is reported to the repository manager to decide about. This stemming is included since use cases in the repository are later compared to new use cases, and this comparison involves stemming. It is better to correct stemming problems beforehand, if possible. It is important to note that some stemmer algorithms, like Porter stemmer (Porter 1980), have the property that for each input token, whether a correct token or an erroneous one, they produce some output. This is not appropriate for this step. The stemmer algorithm should have the ability of distinguishing correct tokens from incorrect tokens. We have used a stemmer algorithm in the experiments which has the nice property that for any input token, if it is stemmed, then the output is present in WordNet (Miller 1995) lexicon.

3.1.2 Generating semantic representation

In this step, models of existing software cases are processed to extract their information and represent it based on the semantic web data model. In this data model, an ontology is used to specify the schema of data. The ontology formally describes concepts of the domain of interest, along with their properties, relationships to other concepts, and also potential constraints on the properties or relationships (Antoniou and Harmelen 2009). Instance data of the domain, i.e. data of concrete instances of the concepts, are then represented based on the terminology provided by the corresponding ontology.

There are different languages for ontology definition, but *RDF*[7], *RDFS*[8] and *OWL*[9] are the main elements of the representation format for the semantic web data. Therefore the semantic representation of the current work is also based on these languages.

The proposed approach uses three components with regard to generating the semantic representation of the models: An ontology of UML model elements, A model parser, and a semantic model repository. These components are described in more details in the following.

UML ontology To provide the required constructs for generating the semantic representation of the UML models, an ontology is required which formally defines the main concepts of the UML models, along with their properties and relationships. However, UML is a full-featured modeling language and creating an ontology for the complete metamodel of UML is a complex and time-consuming task. Therefore, the ontology which is developed for the proposed approach does not cover all the concepts and features of UML. It only covers the elements that are used in the proposed approach, i.e. use case diagram, class diagram and activity diagram.

To create this ontology, first, the metamodel packages of the superstructure of *Object Management Group (OMG)* UML (2011) have been studied to determine which UML packages and elements are directly or indirectly used in the proposed approach. Then, an OWL ontology is developed for each of these packages to describe its required

[7] http://www.w3.org/RDF/

[8] http://www.w3.org/TR/rdf-schema/

[9] http://www.w3.org/TR/owl-features/

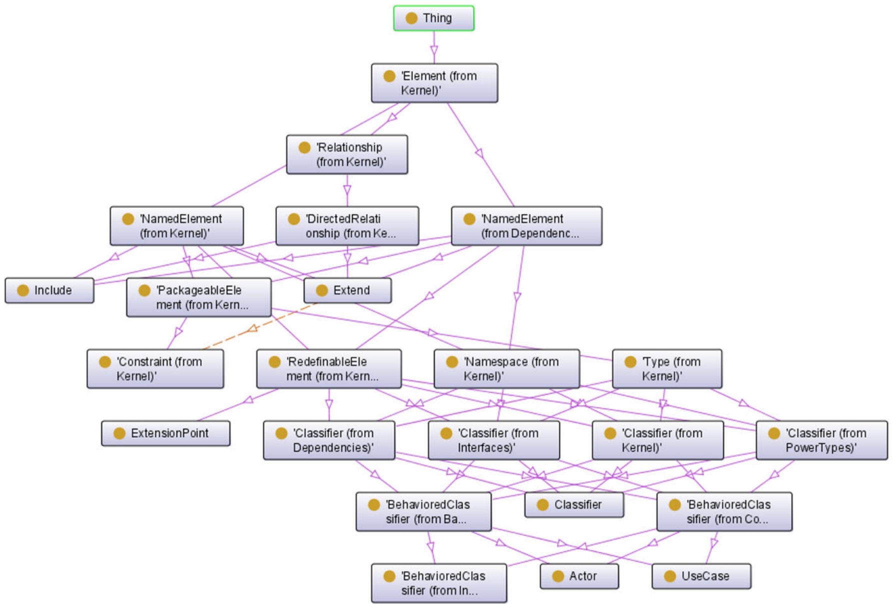

Thing
'Element (from Kernel)'
'Relationship (from Kernel)'
'NamedElement (from Kernel)'
'DirectedRelationship (from Ke...
'NamedElement (from Dependenc...
Include
'PackageableElement (from Kern...
Extend
'Constraint (from Kernel)'
'RedefinableElement (from Kern...
'Namespace (from Kernel)'
'Type (from Kernel)'
ExtensionPoint
'Classifier (from Dependencies)'
'Classifier (from Interfaces)'
'Classifier (from Kernel)'
'Classifier (from PowerTypes)'
'BehavioredClassifier (from Ba...
Classifier
'BehavioredClassifier (from Co...
'BehavioredClassifier (from In...
Actor
UseCase

Fig. 4 A part of the ontology of the package *'UseCases'*

elements. For instance, a part of the ontology of the package *UseCases* (from the metamodel level 1) is illustrated in Fig. 4. It defines classes like *Actor* and *UseCase*, and relations like *ownedUseCase* and *includingCase*. Further, it imports ontologies like those of the *Kernel* and the *Communications* packages.

Since the proposed approach includes an annotation algorithm, and new concepts like *annotation* or *annotationTarget* are introduced in this process, in order to semantically represent the generated annotations, an additional ontology is developed to describe these concepts.

These ontologies are developed in *Protégé*[10], which is a well-known ontology development environment. Since Protégé supports importing ontologies, all of them are integrated as the final version of the required ontology.

Model parser The model parser is responsible for parsing an input model file, and generating its semantic representation based on the proposed UML ontology. After parsing the input model file, the parser loads a set of predefined rules from a rule base and uses them to infer new RDF triples from the currently asserted triples.

Since the inference mechanism enriches the resulting semantic representation of the models, it can be stated that some functionalities of the parser are implemented through declaration of these inference rules. This capability of declarative implementation instead of hardcoding has the benefit that it reduces the cost of implementing the parser, and increases its flexibility, since after implementing the parser, it is possible to add some new functionality by adding new rules to the rule base.

[10] http://protege.stanford.edu/

An issue in the semantic representation is to assign *Uniform Resource Identifiers (URI)* to the entities which are being described. The parser uses a simple schema shown below for assigning URIs to entities.

$$\{baseURI\}\{scName\}\_\text{obj}\_\{value\} \tag{1}$$

where *baseURI* is a literal, *scName* is the name of the software case being processed (by default it is the name of the input model file), and *value* is the value of a counter. The counter is set to zero when processing a model file is started, and it is incremented by the parser whenever a new entity is encountered.

Semantic model repository The semantic model repository is a repository which stores the semantic representation of the models. The prime mechanism for accessing information of this repository is through a *SPARQL* endpoint provided by the repository. It is possible to send *SPARQL* queries to this endpoint and retrieve the results from the repository. SPARQL is the W3C standard query language for the semantic web and the RDF data model. It is syntactically very similar to SQL which is used for querying on the relational data. This similarity reduces cost and complexity of learning SPARQL by developers who are already familiar with SQL.

The more expressive is the underlying model ontology, and the more powerful is the parser, the richer is the information content of the repository, and there is more potential for retrieving model information through this query-based approach. It is shown in (Tappolet et al. 2010) that this query-based approach for analysis of software artifacts is very promising.

3.1.3 Model annotation

The goal of this step is to annotate labels of the repository activity diagrams with corresponding entities in other types of models. These annotations are the essential elements of the reuse process. Here, it is described how the annotations are generated, but the details of how they are used in the proposed approach will be described in Sect. 3.2.

In the model annotation step, for each existing activity diagram *AD* in the semantic model repository, the following steps are executed.

1. Three lists are created: $classes_{AD}$ which includes names of the classes in the class diagram of the software case associated with *AD*, $attributes_{AD}$ which includes names of the attributes of these classes, $actors_{AD}$ which includes the names of the actors that are associated with the use case corresponding to *AD*.
2. List of the labels of *AD*, i.e. $labels_{AD}$, is retrieved from the semantic model repository, which includes all the labels assigned to the elements (e.g. edges, decision points, actions) of *AD*.
3. For each label L in $labels_{AD}$, a set of possible annotations is generated by the following three steps:

3.1. The input label L is tokenized using simple heuristics. For instance, white spaces and case changes are treated as delimiters. That is, the labels *'request confirmation'* and *'ShowAddressBook'* are respectively split into the lists *{'request', 'confirmation'}* and *{'Show', 'Address', 'Book'}*.

3.2. Different *N-grams* of the elements of the tokens are searched in the lists $classes_{AD}$, $attributes_{AD}$ and $actors_{AD}$. When a match is found, an annotation is created which includes three parts:
- *Source*: the URI of the activity diagram element to which label L belongs
- *Target*: the URI of the entity (i.e. class, attribute or actor) that its name is matched with the N-gram
- *Text*: the matched N-gram

3.3. While it is probable that no annotation is generated for the input label L, it is also possible that more than one annotation is created for it. For instance, for the label *'ShowAddressBook'*, it is possible that the 2-gram *'AddressBook'* is matched with an element of $classes_{AD}$, e.g. the class *'AddressBook'*, and also the 1-gram *'Address'* is matched with an element of $attributes_{AD}$, e.g. attribute *'address'* from the class *'User'*. In such cases, a resolution process is performed to decide which annotation has more priority and which ones can be removed. In the resolution process, first, each annotation is assigned a priority level of *'high'*, *'medium'* or *'low'*. If the target of an annotation refers to attribute of a class, and name of that class is included in label L, then priority level of the annotation is *'high'*. If the target of the annotation is a class, one of its attributes, or an actor, then if name of that class or actor is included in the name of the corresponding use case (i.e. the use case associated with that activity diagram), then priority level is *'medium'*. If none of the two criteria mentioned above is met, then the priority level is *'low'*. After priority levels are assigned, each annotation is checked to see if it should be removed. An annotation $ANNOT_i$ is removed if and only if at least one of the following conditions is met:

- There is another annotation $ANNOT_j$ so that $Text_j = Text_i$ and PL_j is higher than PL_i, where PL_j and PL_i are correspondingly priority levels of $ANNOT_i$ and $ANNOT_j$.
- There is another annotation $ANNOT_j$ so that $Text_j$ includes $Text_i$.
- There is another annotation $ANNOT_j$ so that $Text_j = Text_i$, and $Target_j$ is a class while $Target_i$ is an actor.

4. Finally, the annotations are semantically represented using the proposed ontology, and the resulting RDF triples are added to the semantic model repository.

3.2 Reuse process

The overview of the reuse process is illustrated in Fig. 5. It takes as input the use case diagram of a new software case SC_{new}, and for each use case UC_{new} in this use case diagram, it creates the initial version of its activity diagram through three main steps: preprocessing, use case recommendation and activity diagram adaptation.

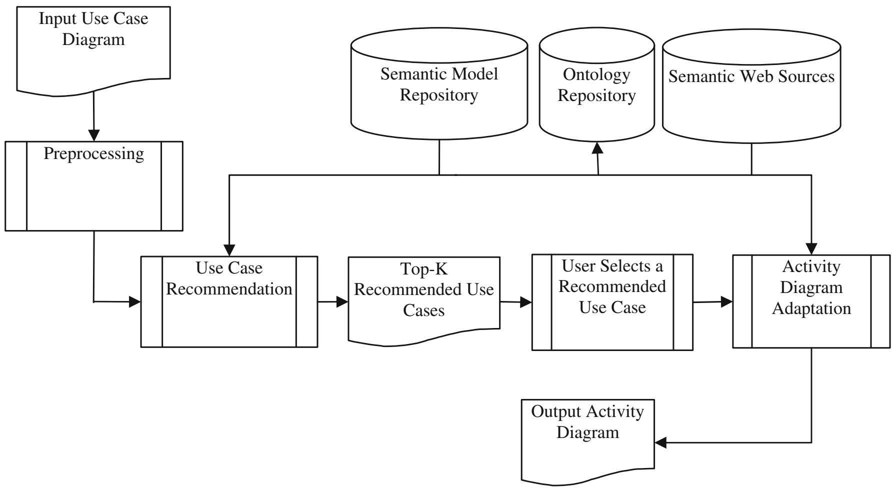


Fig. 5 Overview of the reuse process

3.2.1 Preprocessing

This step is intended to check some quality issues on the input model. The details are similar to what is described in Sect. 3.1.1 and therefore we ignore it here. The only difference is that if any problem is found in the input model, it is responsibility of the user to decide how to resolve it.

3.2.2 Use case recommendation

The use case recommendation algorithm inputs a use case and retrieves an ordered list of the top-K most similar use cases from the semantic model repository. To do so, a new metric is proposed for measuring the similarity of two use cases. In this section, first the basic definitions are described and then the proposed metric is introduced in details.

Basic definitions Each use case diagram includes a set of use cases, and each use case declares a proposed behavior of the subject system from a high level of abstraction. This behavior involves manipulating some concepts.

A very basic idea for comparing two use cases is to see each use case as a bag of words, including names of the use case, its associated actors and its related use cases. Then, it is possible to measure similarity of two use cases by employing textual similarity metrics. This approach is followed by some works like (Bonilla-Morales et al. 2012), however, it has some problems.

Generally, name of a use case is very short, and it is possible that no description is provided for the use cases. This leads to lack of enough textual data for textual similarity measurement. Further, problems of relying only on the textual similarity are very well highlighted in the literature, for instance it is possible that two semantically similar use cases might have been described by different words and therefore their textual similarity would be low, or vice versa.

To address these problems, a new metric is proposed which considers different aspects of a use case, in addition to its textual information.

In this paper, each use case UC_i is considered to be a tuple of the form:

$$UC_i = \{name_i, S_i, B_i, A_i, E_i, I_i, G_i, C_i\}$$

where

$name_i$ is the name of UC_i
S_i is the name of the subject system to which UC_i is associated
B_i is the set of declared behaviors of UC_i
$A_i = \{A|$ there is a direct/indirect *Association* relationship between actor A and $UC_i\}$
$E_i = \{UC_j|$ there is a direct/indirect *Extend* relationship from UC_i to use case $UC_j\}$
$I_i = \{UC_j|$ there is a direct/indirect *Include* relationship from UC_i to use case $UC_j\}$
$G_i = \{UC_j|$ there is a direct/indirect *Generalization* relationship from UC_i to use case $UC_j\}$
$C_i = \{C|$ behavior of UC_i involves manipulating concept $C\}$

Having a use case UC_i and the associated use case diagram UCD_i, it is easy to determine $name_i$, SC_i, A_i, E_i, I_i, and G_i, since they are explicitly specified in the use case diagram. However, identification of the B_i and C_i is not trivial. The idea of this paper is that if a use case has properly named, then its name must reveal its behavior and also associated concepts. Based on this idea, simple algorithms are proposed for identification of B_i and C_i from $name_i$. Next, these proposed algorithms are described and then, the proposed similarity metric is introduced.

Behavior detection The basic idea behind the behavior detection algorithm is to use a *Part Of Speech* (*POS*) tagger to tag the name of the use case, and then select the words that are tagged as verb.

However, there is an issue with this method. Names of the use cases are usually very short expressions and not complete sentences. As a result, the *POS* tagger may fail in determining the word which plays the verb role. For instance, for the string *'search contacts'*, the *Stanford POS Tagger*[11] assigns a name tag to both words *'search'* and *'contacts'*. To address this issue, the proposed algorithm uses the notion of *salt*. The *salt* is used to address the issue by making the expression more similar to a complete sentence. For instance, if name of the use case is *'search contacts'* and *salt='I want to '*, then the resulting expression is *'I want to search contacts'*. If the *POS* tagger is executed on this expression, then it assigns a verb tag to the word 'search' and a noun tag to the word 'contacts', which is a correct assignment.

The proposed algorithm for detecting behaviors of a use case UC_i from its name $name_i$ is described below.

11 http://nlp.stanford.edu/software/tagger.shtml

First, $name_i$ is converted to an expression *exp* by separating its words by a space. Then *exp* is changed to lower case, and a *salt* is added to its beginning. Finally, *POS* tagging is performed on *exp*, and the words after the *salt* which have a verb tag are selected as B_i.

In order to use the proposed similarity metric, it is required to once set an appropriate value for the *salt*. This must be performed experimentally by testing different intuitively potential values to identify behaviors of the use cases which are stored in the model repository, and selecting the salt which has the best results. Such an experiment is described in Sect. 4.2.1.

Concept detection It is important to make clear what is meant in this paper by concepts of a use case. The behavior declared by a use case involves manipulating some entities in the domain of the subject system. These entities belong to some specific types or classes. Further, these types or classes have a number of properties or attributes. As a result, concepts of the use case include names of these classes or attributes.

Our point of view is that if the subject system is properly modeled, then each concept of a use case must refer to one of these elements: (1) a class declared in the UML class diagram of the subject system, (2) an attribute of a class from the class diagram, or (3) an actor who uses the corresponding use case. Otherwise, i.e. if a use case concept refers to a target which does not exist among these elements, it is ambiguous and this ambiguity may lead to problems when the detailed description of the use case is being created.

The algorithm proposed for identifying concepts of a use case is very similar to the behavior detection algorithm. The only difference is that after *POS* tagging is performed, the words after the *salt* which have a noun tag are selected as C_i.

Use case similarity metric In order to compute similarity of two use cases, different information aspects of them must be considered. These aspects can be divided to two groups: *textual* aspects, and *relational* aspects. For a use case UC_i, textual aspects include $name_i$, S_i, B_i and C_i, and relational aspects include A_i, E_i, I_i and G_i. The proposed metric measures similarity of two use cases UC_i and UC_j as defined below:

$$sim\left(UC_i, UC_j\right) = W^*_{sem} semSim_{UC}\left(UC_i, UC_j\right) + W^*_{rel} relSim\left(UC_i, UC_j\right)$$

where $semSim_{UC}\left(UC_i, UC_j\right)$ is the semantic similarity of the textual aspects of the use cases UC_i and UC_j, and $relSim(UC_i, UC_j)$ is relational similarity of the two use cases, i.e. similarity of their relational aspects. Further, W_{sem} and W_{rel} are correspondingly the weights that determine the contribution of these two similarities to the overall similarity of UC_i and UC_j.

Next, it is described how the semantic similarity and relational similarity of two use cases are measured.

Semantic similarity Regarding the textual aspects, it is important to note that simple text comparison is not effective due to known problems of textual similarity. For instance, for the use cases UC_i and UC_j, if $name_i = \text{'}remove\,user\text{'}$ and $name_j = \text{'}delete\,user\text{'}$, then $B_i = \{\text{'}remove\text{'}\}$ and $B_j = \{\text{'}delete\text{'}\}$. Now if B_i and B_j are compared textually, they would have a very low similarity, although they are semantically very similar. As a result, for the textual aspects, the proposed metric uses the semantic similarity and not the pure textual similarity. This semantic similarity is based on *WordNet* (Miller 1995) which represents relationships of different words in a network of words. *WordNet* is used in many works for the task of word sense disambiguation, semantic annotation or semantic similarity computation (Bagheri et al. 2012; Gomes et al. 2002). Different algorithms are introduced for measuring semantic similarity of two words based on such a network, e.g. Lin (1998) or Wu and Palmer (1994).

In the proposed metric, the semantic similarity of two use cases UC_i and UC_j is computed as below:

$$semSim_{UC}\left(UC_i, UC_j\right) = W_S^* \, semSim_S\left(UC_i, UC_j\right) + W_B^* \, semSim_B\left(UC_i, UC_j\right) + W_C^* \, semSim_C\left(UC_i, UC_j\right)$$

where $semSim_S$, $semSim_B$ and $semSim_C$ are correspondingly the semantic similarity of the subjects, behaviors, and concepts of the two use cases computed as defined below, and W_S, W_B and W_C are the corresponding weights of these similarities.

$$\begin{aligned}
\mathrm{semSim_S}\left(\mathrm{UC_i}, \mathrm{UC_j}\right) &= \mathrm{semSim_T}\left(\{\mathrm{t_m}|\mathrm{t_m} \text{ is a token in } \mathrm{S_i}\}, \{\mathrm{t_n}|\mathrm{t_n} \text{ is a token in } \mathrm{S_j}\}\right)\\
\mathrm{semSim_B}\left(\mathrm{UC_i}, \mathrm{UC_j}\right) &= \mathrm{semSim_T}\left(\{\mathrm{t_m}|\mathrm{t_m} \in \mathrm{B_i}\}, \{\mathrm{t_n}|\mathrm{t_n} \in \mathrm{B_j}\}\right)\\
\mathrm{semSim_C}\left(\mathrm{UC_i}, \mathrm{UC_j}\right) &= \mathrm{semSim_T}\left(\{\mathrm{t_m}|\mathrm{t_m} \in \mathrm{C_i}\}, \{\mathrm{t_n}|\mathrm{t_n} \in \mathrm{C_j}\}\right)
\end{aligned}$$

where $semSim_T$ computes the semantic similarity of the two lists of terms using WordNet. Having two lists of terms $\mathrm{T} = \{\mathrm{t_1}, \mathrm{t_2}, \ldots, \mathrm{t_m}\}$ and $\mathrm{T}' = \{\mathrm{t}'_1, \mathrm{t}'_2, \ldots, \mathrm{t}'_n\}$ so that $|\mathrm{T}| \leq |\mathrm{T}'|$, $semSim_T$ is equal to the value of the best correspondence between T and T′. Each correspondence is a set of assignments of t'_j $(1 \leq \mathrm{j} \leq \mathrm{n})$ elements to $\mathrm{t_i}(1 \leq \mathrm{i} \leq \mathrm{m})$ elements where no $\mathrm{t_j}$ is assigned to more than one $\mathrm{t_i}$. The best correspondence is the one which has the largest value among all possible correspondences. Finally, value of a correspondence R is equal to:

$$2 \times \frac{\sum_{(\mathrm{t_i}, \mathrm{t}'_j) \in \mathrm{R}} \mathrm{WordNetSimilarity}(\mathrm{t_i}, \mathrm{t}'_j)}{|\mathrm{T}| + \left|\mathrm{T}'\right|}$$

where $\mathrm{WordNetSimilarity}(t_i, t'_j)$ is computed by first stemming t_i and t'_j, and then computing the semantic similarity of their stems by a WordNet-based algorithm.

Relational similarity Relational similarity measures the similarity of UC_i and UC_j by measuring similarity of the elements related to UC_i with the elements related to UC_j. The idea is that similarity of these related elements contributes to the similarity of the two use cases. For instance, if UC_m is a member of E_i and UC_n is a member of E_j, then the more similar are UC_m and UC_n, the greater is similarity of UC_i and UC_j.

In this paper, eight different types of directed relationships R_1 to R_8 are considered in which a use case may be involved. A relation of type $R_k(1 \leq k \leq 6)$ is considered to exist from UC_i to UC_j if and only if the condition $C_k(1 \leq k \leq 6)$ is met, where

C_1: UC_j is a member of EXT_i, i.e. UC_i extends UC_j
C_2: UC_i is a member of EXT_j, i.e. UC_i is extended by UC_j
C_3: UC_j is a member of INC_i, i.e. UC_j is included by UC_i
C_4: UC_i is a member of INC_j, i.e. UC_i includes UC_j
C_5: UC_j is a member of G_i, i.e. UC_j generalizes UC_i
C_6: UC_i is a member of G_j, i.e. UC_i generalizes UC_j

In addition, a relation of type $\underline{R_k}(7 \le k \le 8)$ is considered to exist from UC_i to an actor A if and only if the condition $C_k(7 \le k \le 8)$ is met, where

C_7: there is an association from UC_i to A
C_8: there is an association from A to UC_i

All of these relations are considered to be transitive. For instance, if there is an R_1 relation from UC_i to UC_j, and an R_1 relation from UC_j to UC_k, then there would be an (indirect) R_1 relation from UC_i to UC_k.

Further, we define

$$\begin{aligned} UC_{k,i} &= \{UC_x \mid \text{there is a } R_k \text{ relation from } UC_i \text{ to } UC_x\} \text{ for } 1 \le k \le 6 \\ A_{k,i} &= \{A_x \mid \text{there is a } R_k \text{ relation from } UC_i \text{ to actor } A_x\} \text{ for } 7 \le k \le 8 \end{aligned}$$

Based on the eight relation types mentioned above, the relational similarity of two use cases UC_i and UC_j is defined as:

$$\mathrm{relSim}(\mathrm{UC_i}, \mathrm{UC_j}) = \frac{1}{8}\sum_{k=1}^{8} \mathrm{F_k}(\mathrm{UC_i}, \mathrm{UC_j})$$

where $F_k(1 \le k \le 8)$ is a function that returns a value in the interval [0, 1] as the similarity of the use cases UC_i and UC_j, based on the corresponding relation R_k. These eight functions are defined as below.

$$\mathrm{F_k}(\mathrm{UC_i}, \mathrm{UC_j}) = \frac{\sum\limits_{\mathrm{UC_x} \in \mathrm{UC_{k,i}}} \sum\limits_{\mathrm{UC_y} \in \mathrm{UC_{k,j}}} (\mathrm{semSim_{UC}}(\mathrm{UC_x}, \mathrm{UC_y}))}{|\mathrm{UC_{k,i}}| \cdot |\mathrm{UC_{k,j}}|} \quad \text{for } 1 \le \mathrm{k} \le 6$$

$$\mathrm{F_k}(\mathrm{UC_i}, \mathrm{UC_j}) = \frac{\sum\limits_{\mathrm{A_x} \in \mathrm{A_{k,i}}} \sum\limits_{\mathrm{A_y} \in \mathrm{A_{k,j}}} (\mathrm{semSim_A}(\mathrm{A_x}, \mathrm{A_y}))}{|\mathrm{A_{k,i}}| \cdot |\mathrm{A_{k,j}}|} \quad \text{for } 7 \le \mathrm{k} \le 8$$

where $semSim_A(A_x, A_y)$ is the semantic similarity of two actors A_x and A_y, computed by the following formula:

$$\mathrm{semSim_A}(\mathrm{A_x},\ \mathrm{A_y}) = \mathrm{semSim_T}(\text{name of } \mathrm{A_x},\ \text{name of } \mathrm{A_y})$$

3.2.3 Activity diagram adaptation

After the use case recommendation algorithm has generated the ordered list of top-k most similar use cases to UC_{new}, the user selects one use case UC_{recom} from the recommendations. The adaptation algorithm then takes UC_{new} and UC_{recom} as input, and creates the draft of the activity diagram of UC_{new}, i.e. AD_{new}, by adapting the activity diagram of UC_{recom}, i.e. AD_{recom}.

The adaptation algorithm starts by retrieving AD_{recom} from the semantic model repository, and creating a copy of it as the initial version of AD_{new}. Then, for each label $label_{recom}$ of AD_{recom}, its annotations are processed by a label adaptation process. This process takes an annotation $annot_{recom}$ of the label $label_{recom}$ as input, and returns a label $label_{new}$ as output. Then $label_{new}$ replaces the corresponding label of $label_{recom}$ in AD_{new}.

The label adaptation process first initializes $label_{new}$ by a copy of $label_{recom}$. Then, the target of $annot_{recom}$ is retrieved which is an element E_{recom} (e.g. an actor) in SC_{recom}.

A search is performed to find an element E_{new} in the context of SC_{new} which is equivalent to E_{recom} in the context of SC_{recom}. If such an equivalent element is found, $label_{new}$ is updated by replacing the name of E_{recom} with the name of E_{new}.

The main issue is how to find the equivalent element of E_{recom}. Providing a complete matchmaking algorithm for addressing this issue is a challenge which requires more work, however, currently three different cases are considered based on the type of E_{recom}. Following rules are defined for handling these cases.

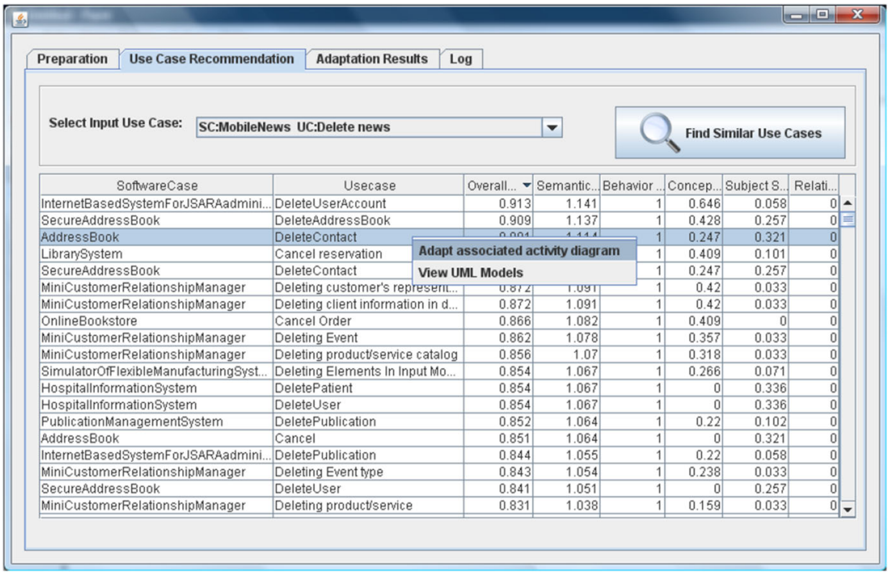
Adaptation Rule 1 If E_{recom} is a class in the class diagram of SC_{recom}, and if name of this class is included in $concepts_{recom}$, i.e. the set of concepts in the name of UC_{recom}, then E_{new} is any concept in $concepts_{new}$.

For instance, if the name of UC_{recom} is *'Comment Movie'*, and the name of UC_{new} is *'Comment Song'*, then $concepts_{recom} = \{`Movie'\}$ and $concepts_{new} = \{`Song'\}$. Now if the label *'selectedMovie'* in AD_{recom} has an annotation which its target, i.e. E_{recom}, is the class *'Movie'* in the class diagram of SC_{recom}, then the equivalent element of E_{recom} would be the concept *'Song'* and therefore the resulting label adapted from *'selectedMovie'* is *'selectedSong'*.

Adaptation Rule 2 If E_{recom} is an attribute of a class in the class diagram, and if name of the associated class exists in $concepts_{recom}$, then E_{new} is any appropriate attribute of every concept which belongs to $concepts_{new}$.

For instance, if the name of UC_{recom} is *'Buy Ticket'*, and the name of UC_{new} is *'Buy Book'*, then $concepts_{recom} = \{`Ticket'\}$ and $concepts_{new} = \{`Book'\}$. Now if the label *'show reservation number'* in AD_{recom} has an annotation which its target is the attribute *'reservationNumber'* of the class *'Ticket'*, then the equivalent element of E_{recom} would be every appropriate attribute of the concept *'Book'*, e.g. *'ISBN'* or *'publication year'*, and therefore the resulting label adapted from *'show reservation number'* is *'show ISBN'* or *'show publication year'*.

The main question is how to find the appropriate attribute, since yet there exist no class diagram for SC_{new}. The proposed adaptation algorithm first searches the model repository to find a class (from previous software cases) which its name is equal to the name of E_{new} (*'Book'* in the example above). If such a class is found, then its attributes are analyzed to find the more appropriate ones. Appropriateness of the attributes is decided based on their type (e.g. integer, float, string, …) and name similarity to the corresponding attribute (*'reservationNumber'* in the example above).



Select Input Use Case: SC:MobileNews UC:Delete news

SoftwareCase	Usecase	Overall...	Semantic...	Behavior ...	Concep...	Subject S...	Relati...
InternetBasedSystemForJSARAadmini...	DeleteUserAccount	0.913	1.141	1	0.646	0.058	0
SecureAddressBook	DeleteAddressBook	0.909	1.137	1	0.428	0.257	0
AddressBook	DeleteContact	[illegible]	[illegible]	1	0.247	0.321	0
LibrarySystem	Cancel reservation	[illegible]	[illegible]	1	0.409	0.101	0
SecureAddressBook	DeleteContact	[illegible]	[illegible]	1	0.247	0.257	0
MiniCustomerRelationshipManager	Deleting customer's represent...	0.872	1.091	1	0.42	0.033	0
MiniCustomerRelationshipManager	Deleting client information in d...	0.872	1.091	1	0.42	0.033	0
OnlineBookstore	Cancel Order	0.866	1.082	1	0.409	0	0
MiniCustomerRelationshipManager	Deleting Event	0.862	1.078	1	0.357	0.033	0
MiniCustomerRelationshipManager	Deleting product/service catalog	0.856	1.07	1	0.318	0.033	0
SimulatorOfFlexibleManufacturingSyst...	Deleting Elements In Input Mo...	0.854	1.067	1	0.266	0.071	0
HospitalInformationSystem	DeletePatient	0.854	1.067	1	0	0.336	0
HospitalInformationSystem	DeleteUser	0.854	1.067	1	0	0.336	0
PublicationManagementSystem	DeletePublication	0.852	1.064	1	0.22	0.102	0
AddressBook	Cancel	0.851	1.064	1	0	0.321	0
InternetBasedSystemForJSARAadmini...	DeletePublication	0.844	1.055	1	0.22	0.058	0
MiniCustomerRelationshipManager	Deleting Event type	0.843	1.054	1	0.238	0.033	0
SecureAddressBook	DeleteUser	0.841	1.051	1	0	0.257	0
MiniCustomerRelationshipManager	Deleting product/service	0.831	1.038	1	0.159	0.033	0

Fig. 6 Screenshot of the implemented prototype, use case recommendation tab

It is possible that there is no matching class in the model repository. Here, the ontology repository is searched to find ontologies that define a class with the required name. If such an ontology is found, properties of the associated class are evaluated to find the appropriate attributes. If no such ontology is found in the ontology repository, then a number of semantic web data sources are searched to find the required class. Results of this search, which are RDF triples, are stored in the ontology repository for future reuse.

There are different sources on the semantic web which can be used in the scenario described above. For instance, *SWOOGLE*[12] a semantic web search engine which enables searching for ontologies that contain a set of specific keywords. Other examples include the data sets on the LOD cloud like *DBpedia*[13]. The SPARQL endpoint of the LOD cloud is also interesting in this regard.

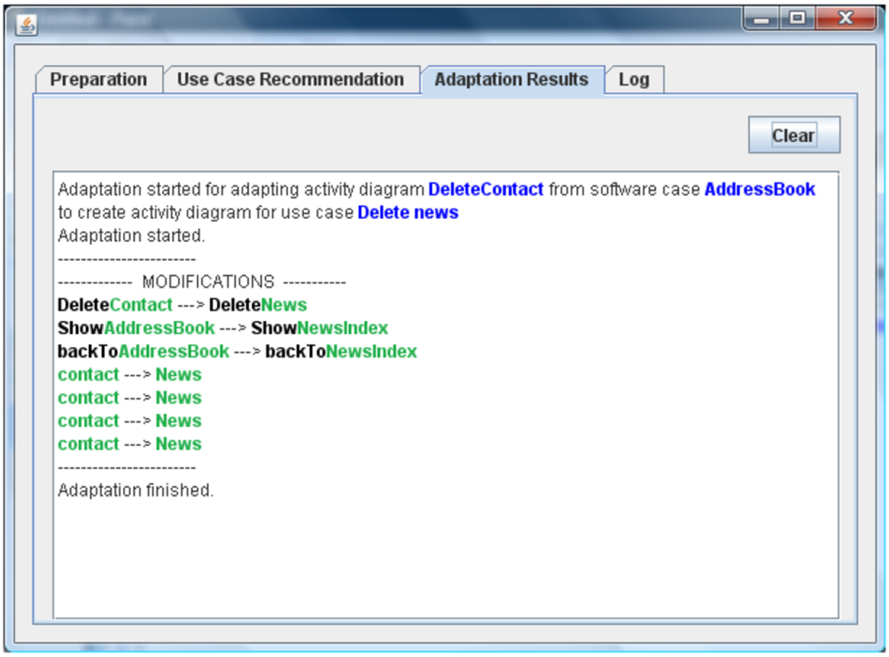
Adaptation Rule 3 If E_{recom} is an actor in the use case diagram of SC_{recom}, and if this actor is associated with UC_{recom}, then E_{new} is any actor associated with UC_{new}.

4 Experimental evaluation

For the purpose of evaluation, a prototype of the proposed approach is implemented in Java, and it is experimentally evaluated. In Figs. 6 and 7, two screenshots of the prototype is shown. The first figure shows the situation where the user has selected one of the use cases of the input use case diagram, i.e. use case *'Delete*

12 http://swoogle.umbc.edu/

13 http://dbpedia.org/



```
Adaptation started for adapting activity diagram DeleteContact from software case AddressBook
to create activity diagram for use case Delete news
Adaptation started.
------------------------
------------ MODIFICATIONS ----------
DeleteContact ---> DeleteNews
ShowAddressBook ---> ShowNewsIndex
backToAddressBook ---> backToNewsIndex
contact ---> News
contact ---> News
contact ---> News
contact ---> News
------------------------
Adaptation finished.
```

Fig. 7 Screenshot of the implemented prototype, the adaptation results tab

news', and the prototype has recommended a list of similar use cases from the model repository. The user has then selected one of the candidate use cases, i.e. use case '*DeleteContact*', and has initiated the adaptation of the associated activity diagram. In Fig. 7, the results of the activity diagram adaptation are presented. Currently, this is performed by printing as output the required modifications in the labels of the selected activity diagram in order to adapt it for the new use case. It is possible to improve the prototype to actually create the required XMI excerpt of the new activity diagram. However, this is not performed in the current state of the work.

The experiments are performed on a system with *Intel Core2 Duo* processor (2.26, 2.27 GHz), *4GB RAM*, and *64-bit Windows Vista* operating system. As mentioned in Sect. 3, the two main processes of the proposed approach are preparation of the semantic model repository, and reuse process. In this section, evaluation of these two processes is separately discussed.

4.1 Preparation of the semantic model repository

In order to prepare the semantic model repository, a set of sample software cases must be identified and their UML models must be transformed into the semantic representation. Further the model annotation process must be executed. In this section, first the collected dataset is described, and then the model annotation process is evaluated.

4.1.1 Dataset

Unfortunately no standard dataset of UML models of web applications is available. Alchimowicz et al. (2010) propose a use-case benchmark which specifies the typical attributes of use case based requirements specifications. Their work is based on precise analysis of a use case data base (UCDB) containing 16 industrial projects with a total of 524 use cases. While UCDB research team have not made these use cases publically available, but they have provided a set of 10 SRS in response to the request of the authors of the current paper. Therefore, for the purpose of the experiments, a dataset is created which includes:

- Models of 13 web applications downloaded from the website of *UWE* project[14].
- Models of the 10 SRSs from the UCDB team. These include systems like bibliometric information system, mini customer relationship manager, and project management system.
- Models of 10 projects which are developed by 4th year students of software engineering, during a course of Software Engineering II.
- Models of 27 sample software systems which their descriptions are presented in different UML-related books like (Conallen 2002; Hamilton and Miles 2006; Mellor and Balcer 2002).

For each of these 60 software cases, the corresponding models are prepared by *MagicDraw*[15] tool, and the associated *XMI* files, have been used as the input of the model parser.

4.1.2 Demographics

Due to the lack of a standard benchmark for evaluating the proposed approach, the experiments are based on the results obtained from a group of experts. This group is composed of 7 Ph.D. students of software engineering with more than 4 years of experiences in the software engineering domain. All the experts have passed three courses Software Engineering I, Software Engineering II, and Advanced Software Engineering during their bachelor or master degree. Further, three of the experts have also had the experience of teaching Software Engineering I and Software Engineering II in the recent 4 years. Finally, all the experts have experiences in developing web based systems and also object-oriented software.

For all the experiments described in the next sections, the experts were provided with both printed version of the test models and also with the associated files which can be opened in *MagicDraw* tool. Further, no specific output format is forced, and the experts are allowed to provide their results in any format which is more convenient for them, e.g. drawing and writing the results on the printed papers, or typing the results in a file.

[14] http://uwe.pst.ifi.lmu.de/

[15] http://www.nomagic.com/products/magicdraw.html

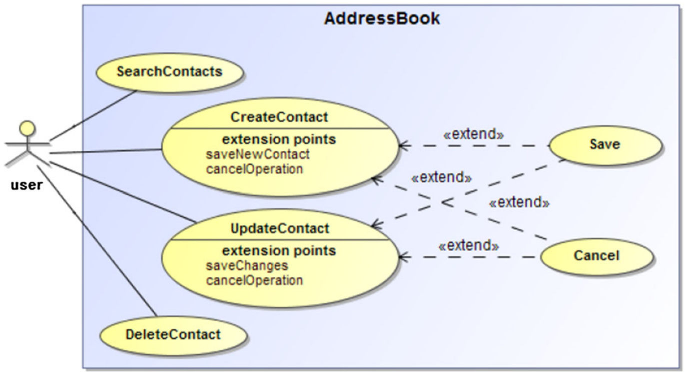


Fig. 8 Use case diagram of the software case ‘AddressBook’

4.1.3 Preprocessing

The models of the dataset are preprocessed based on the method described in Sect. 3.1.1. This identified a number of misspelling errors (e.g. ‘adress’), and non-stemmed words (e.g. ‘unregister’). All the misspelling errors are corrected, but other cases are not modified. The reason is that (1) unlike misspelling errors, abbreviations are considered to be used intentionally by the creator of the model, and therefore it is not appropriate to modify it, and (2) it was interesting to see how the proposed approach performs in the presence of such issues.

4.1.4 Model parser

The implemented model parser uses *SAX*[16] API for parsing the input model files, which are XML-based, and *Jena*[17] API for producing the semantic representation of the models. As an example, an excerpt of the generated semantic representation of the use case diagram illustrated in Fig. 8 is shown in Fig. 9.

In the current implementation, the rule base of the parser is a textual file in which 18 Jena rules are declared. This rule base reduced the cost of adding some functionality to the parser. It would have been more difficult and error-prone to add the functionality by modifying the source code of the parser. As an example, one of the inference rules used by the model parser is shown in Fig. 10. Briefly speaking, this rule states that if there is a generalization relationship between two classes, then the subclass inherits attributes of its superclasses.

The parser is executed over the sample dataset, and the resulting RDF triples are stored in a repository which is also implemented using *Jena* library. Some statistics about this process is presented in Table 1.

16 http://sax.sourceforge.net/

17 http://jena.apache.org

```
uml2rdf:AddressBook_obj_257      a     uml2rdf:UseCase ;
    uml2rdf:hasSoftwareCase      uml2rdf:AddressBook_obj_1 ;
    uml2rdf:name                 "CreateContact" .

uml2rdf:AddressBook_obj_252      a     uml2rdf:UseCase ;
    uml2rdf:hasSoftwareCase      uml2rdf:AddressBook_obj_1 ;
    uml2rdf:name                 "Save" .

uml2rdf:AddressBook_obj_250      a     uml2rdf:Extend ;
    uml2rdf:extendedCase         uml2rdf:AddressBook_obj_257 ;
    uml2rdf:extension            uml2rdf:AddressBook_obj_249 ;
```

Fig. 9 Parts of the semantic representation of the model illustrated in Fig. 8

```
[rule1:
(?superclass rdf:type uml2rdf:Class), (?subclass rdf:type uml2rdf:Class),
(?generalization rdf:type uml2rdf:Generalization), (?generalization uml2rdf:hasSpecific ?subclass),
(?generalization uml2rdf:hasGeneral ?superclass), (?superclass uml2rdf:containsAttribute ?attribute)
        -> (?subclass uml2rdf:containsAttribute ?attribute)]
```

Fig. 10 An example of the inference rules used by the parser

Table 1 Some statistics about the preparation of the semantic model repository

Number of input software cases	60
Number of resulting RDF triples	156364
execution time (ms)	85320

4.1.5 Model annotation

An experiment is conducted to evaluate how successful is the proposed annotation algorithm. In this experiment, a set of 20 activity diagrams are selected as the test cases from the model repository. These test cases are given to the experts in order to be manually annotated. The experts are asked to annotate labels of the elements of each activity diagram with appropriate entities from the associated models, i.e. models of the same software case. The allowed entities are the classes in the class diagrams, attributes of these classes, and actors in the corresponding use case diagram.

The implemented annotation algorithm is also executed on the test cases. For each test case, first, results of the algorithm are separately compared with results obtained from each expert, and then, the average is computed over the experts. Results are shown in Fig. 11. The minimum precision, recall and f-measure of the proposed annotation algorithm are respectively 58, 70 and 68 %. Further, the average precision, recall and f-measure are correspondingly 90, 88 and 89 %, with standard deviation of 13, 12 and 11 %. These results demonstrate that the proposed algorithm has good accuracy and effectiveness.

As an example, the activity diagram named *'BuyTicket'* from the software case *'IMDB'* is shown in Fig. 12. The annotations created by one of the experts are specified with a red line and are numbered in this figure. The annotation algorithm has been successful in finding 10 annotations out of 12, and it has missed annotations number 7 and 8.

While precision and recall of the proposed algorithm are good, but analysis of the results of this experiemtn have revealed two weaknesses of the proposed anno-

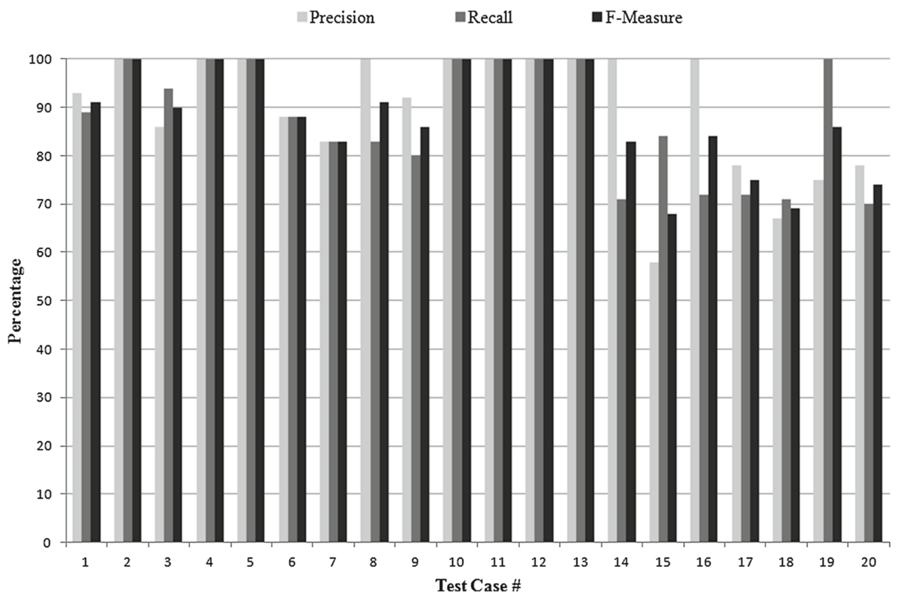


Fig. 11 Evaluation of model annotation algorithm

tation algorithm. First, since this algorithm uses n-gram matching, it is sensitive to small textual differences, and cannot handle semantically similar but textually different labels. For instance, in case of the activity diagram shown in Fig. 12, based on the expert's judgement, the word *'card'* in the labels *'card invalid'* and *'card valid'* must be annotated with the attribute *'creditCard'* of the class *'UserData'* (defined in the corresponding class diagram). However, the proposed algorithm is not able to identify that in this case, *'card'* and *'creditCard'* are the same, and therefore two annotations are missed. To reduce effect of this weakness, it is required that there is high level of consistency between the terms used in the labels of the activity diagram elements and the terms used in other models, e.g. class diagrams, of the same software case.

It must be noted that the requirement mentioned above is imposed on the models which are to be included in the model repository, and not on the models of the end user who is using the proposed approach, because the annotation algorithm is executed only on the models of the repository. The point of view is that models in the model repository are actually some kind of patterns which are instantiated by the adaptation algorithm. As a result, it is expected that these models are good-quality models and they meet some consistency requirements.

The second weakness is that incomplete and imprecise conceptual modeling of the existing software cases reduces the effectiveness of the annotation algorithm, and consequently the reuse process. For instance, since there is no attribute or class named *'seat'* in the class diagram of the software case *'IMDB'*, there is no annotation for the term *'seat'* in the labels *'[not sufficient seats]'* and *'Book seats'* (see Fig. 12). This lack of annotation means that the term *'seat'* cannot be replaced by another term when the activity diagram of the use case *'BuyTicket'* is to be adapted for another use case,

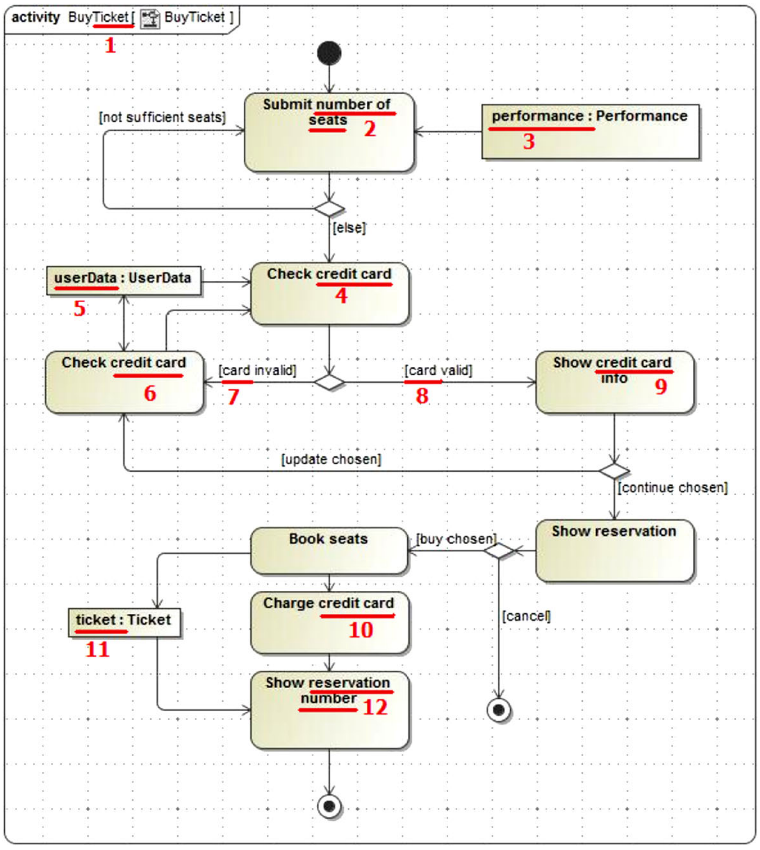


Fig. 12 Activity diagram *'BuyTicket'*, along with its annotations

e.g. *'Buy Book'*. The more precise and detailed are the class diagrams in the model repository, the more potential exists for activity diagram annotation and therefore for activity diagram adaptation.

4.2 Reuse process

The proposed reuse process includes two main steps: use case recommendation, and activity diagram adaptation. In this section, these two steps are evaluated separately.

4.2.1 Use case recommendation

The use case recommendation algorithm is based on the proposed similarity metric, and this metric involves detecting behaviors and concepts of the use cases. In the following, first the behavior and concept detection algorithms, and then the overall metric are evaluated.

Behavior/concept detection method In order to evaluate the proposed algorithms for behavior detection and concept detection, a set of 475 use cases are selected from the repository, and the experts are asked to identify the behaviors of each use case by identifying the words in the name of the use case that specify the use case behavior. Further, in case of concept detection, the experts are asked to manually determine in the use case name, the words that specify concepts of that use case. Based on how we defined the notion of use case concepts in Sect. 3.2.1, the experts are asked to consider this point: "a term C in the name of use case UC_i, is a concept of UC_i, if it is a potential candidate for naming a class in the associated class diagram, or an attribute of a class, or an actor of the corresponding use case".

The results created by the experts are regarded as the golden standard. It is worth noting that from the 475 use cases, 466 use cases were identified to have a behavior specified in their names. For the other 9 use cases, behavior is not explicitly specified in the name of the use case, for instance, the use case '*Home*' or '*general information*'.

The proposed behavior detection and concept detection algorithms are both executed on the test dataset in terms of nine different methods. The first method, i.e. M_1, uses an empty string as the salt. Actually, this method is considered to identify role of the salt. Methods M_2 to M_9 all implement the proposed algorithms but as shown in Table 2, use different values for the salt. All the methods use the *Stanford POS Tagger*. It is worth noting that it was experimentally identified that the selected *POS* tagger is case-sensitive and it generates better results for the lowercase expressions. As a result, before passing an expression to the *POS* tagger, it is changed to lowercase.

Results of the experiment are presented in Table 2 (the maximum value in each column is shown in bold). As it is shown in this table, in case of behavior detection, all of the methods have high precision, i.e. between 92 and 94 %. However, the recall values range from 55 to 93 %. Method M_1 has very low recall and this acknowledges the necessity of using an appropriate salt. From the eight different values tested, some have resulted in poor recall (e.g. the salt value of M_4 and M_9), however there are also three cases, i.e. M_2, M_5 and M_7, with high recall of 92 and 93 %.

Table 2 Evaluation of the behavior and concept detection algorithms

Method	Salt	Behavior detection algorithm			Concept detection algorithm		
		Precision (%)	Recall (%)	F-measure (%)	Precision (%)	Recall (%)	F-measure (%)
M_1		92	57	70	70	**94**	80
M_2	"I want to"	**94**	**93**	**93**	**87**	93	**90**
M_3	"I want to do"	**93**	65	77	74	**94**	83
M_4	"Please"	92	55	69	70	**94**	80
M_5	"I"	93	**93**	**93**	**87**	92	89
M_6	"I do"	93	73	82	77	93	84
M_7	"Do you"	**94**	92	**93**	86	93	89
M_8	"Are you"	93	80	86	80	92	86
M_9	"Let's"	92	55	69	69	**94**	80

In case of concept detection, all of the methods have high recall, i.e. between 92 and 94 %, but the precision values range from 69 to 87 %. Method M_1 has almost the lowest recall among the methods. This again highlights role of the salt in guiding the *POS* tagger. Further, the three methods M_2, M_5 and M_7 which had the best performance in behavior detection have also provided the best results in concept detection. This improves the reliability of the salt values associated with these three methods.

This experiment demonstrates that the proposed algorithms for behavior and concept detection have very good performance. Comparing results of the two algorithms, it is shown that the proposed behavior detection algorithm has better precision (considering similar salt values) than the proposed algorithm for concept detection. However, both algorithms have a similar recall value of about 93 %, when the best candidate salt is used.

In addition, it is shown that selecting an appropriate salt is feasible and more than one appropriate value exists. However, we do not provide a specific method for choosing the required salt. The point of view is that it is not costly, nor challenging, to find an appropriate salt by testing some suggestions. The values tested in this experiment are selected based on the intuition that usually name of a use case starts with a verb, and since this name is to be appended to the salt, the salt must be a string which can be meaningfully followed by a verb. Based on the results of this experiment the string *'I want to'* is selected as the salt value in the next experiments.

It is interesting to determine the reason why the concept detection algorithm has generated more false positives, and therefore its precision is reduced. By analyzing results of this algorithm it was identified that there are some general terms in the names of the use cases which are tagged as name by the POS tagger, but are not considered by the experts as the concepts of the use cases. For instance, for a use case named *'Save customer information'*, both words *'customer'* and *'information'* are tagged as name by the POS tagger, but none of the experts believed that the word *'information'* is qualified for being a use case concept. Actually, when asked about their decision, they had almost the same idea that the word *'information'* is too general and it is not a good choice for naming a class, an attribute or an actor.

From this analysis it was turn out that some kind of stop word removal might be helpful. However, our point of view is that such a stop words list needs to be created specifically for the context of the work, i.e. use cases. In other words, the traditional stop words lists, e.g. (Frakes and Baeza-Yates 1992), are not appropriate. For instance, the words *'order'* and *'number'* should not be considered as stop words in this context. To assess this idea, we modified the proposed concept detection algorithm to remove the stop words from the use case names. The list of stop words is taken from (Frakes and Baeza-Yates 1992). Then, the experiment was repeated. Results supported the idea that this general list is not helpful for the purpose of the proposed algorithm. Actually, precision and recall are reduced by about 1 % in all the nine methods. For instance, in case of method M_2, precision and recall are correspondingly reduced from 87 to 86 %, and from 93 to 92 %.

Use Case Similarity Metric The proposed similarity metric is evaluated by evaluating the results of the use case recommendation algorithm. In this experiment, ten use case diagrams from different software cases are selected from the model repository,

Table 3 Parameters of the proposed metric

Parameter	W_{sem}	W_{rel}	W_S	W_B	W_C
Value	0.8	0.2	0.2	1.0	0.2

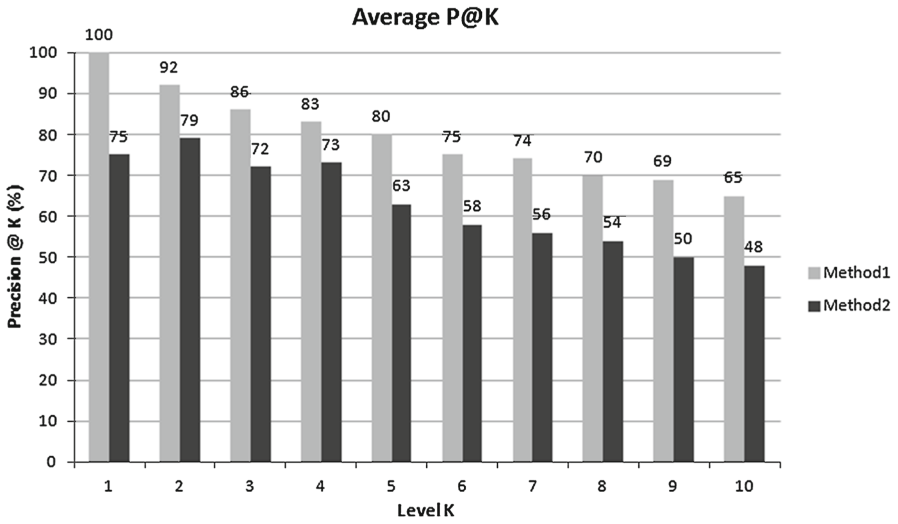


Fig. 13 P@K of the use case recommendation algorithm

and for each one, one of its use cases is selected as the test use cases. Further, the human experts are asked to manually assess the models inside the repository, and suggest for each test use case the top-10 most similar use cases. The same set of use case diagrams are given to the implemented prototype and the resulting recommendations for each of the ten use cases are obtained. Results associated with each use case are then compared with the suggestions of each expert, and then the average is calculated over the experts.

In the experiments discussed in this paper, *WordNetSimilarity* of two terms is computed based on*Lin* algorithm (Lin 1998). Actually, *JAWS API*[18] is used since it provides implementation of a number of *WordNet* based similarity algorithms. This API is also used for the purpose of stemming. Further, the values of the weight parameters defined in the proposed metric are experimentally set as shown in Table 3.

The recommendation algorithm is configured to return all the existing use cases, ranked by their similarity with the test use case. The evaluation involves assessment of the output ranked list, using some well-known metrics from the IR domain.

The first metric is *"Precision at k"*, or "$P@k$". It measures precision for the first k elements of the ranked list. For each test case, P@k is calculated for ten different values of $1 \leq k \leq 10$. In Fig. 13, the average P@k for all the ten test cases is shown for two methods. The first method, i.e. $Method_1$, uses the proposed metric with the weights shown in Table 3, and the second method, i.e. $Method_2$, uses this metric with all weights set to 1.0. The second method is used to determine the importance

[18] Java API for WordNet Searching: http://lyle.smu.edu/~tspell/jaws/index.html

Table 4 Sample results of the use case recommendation algorithm

Test use case		First recommended use case	
Software case name	Use case name	Software case name	Use case name
AddressBook	DeleteContact	SecureAddressBook	DeleteContact
SecureAddressBook	DeleteUser	HospInfo	DeleteUser
IMDB	View Movie	PVS	ViewPublicationText
Philoponella	Add Comment	VirtualOfficeAndMembers ManagementSystem	Add news
IMDB	Buy Ticket	SimpleMusicPortal	BuyAlbum
SimpleMusic Portal	ViewAlbumDetails	PVS	ViewPublication Details
Bibliometric InformationSystem	List publications	HospInfo	ListPatientNames
MiniCustomer RelationshipManager	Modifying customer's representative information (contacts)	InternetBasedSystemFor JSARAadministration	ModifyUserAccount
IMDB	Comment Movie	Philoponella	Add Image
MobileNews	Delete news	InternetBasedSystem ForJSARAadministration	DeleteUserAccount

of distinguishing between different features of the use cases. As shown in this figure, for $Method_1$, the average precision at $k = 5$ and $k = 10$ levels is respectively about 80 and 65 %. This means that on average, there are four correct results among the first five recommendations of the algorithm, and also six correct results among the first ten recommendations. This can be considered a good precision. As a result, it is reasonable to configure the algorithm to return only the first ten elements of the ranked list, in which case the user needs to evaluate a short list containing about seven good recommendations. For $Method_2$, the average precision at $k = 5$ and $k = 10$ levels is respectively about 63 and 48 %. These values are considerably smaller compared to the values associated with $Method_1$. This means that the idea of distinguishing between different features of the use cases is sound and promissing.

Mean reciprocal rank (*MRR*) is used as the second metric. It identifies the rank in which the first correct recommendation of the algorithm usually appears. In this experiment, for all the ten test case, the first correct result appears at rank 1, and therefore $MRR = 1$. The information of the ten test cases, and the associated top-most recommended use case is shown in Table 4.

As the third metric, *11-point Interpolated Average Precision (11pIAP)* is used to measure precision at eleven different recall levels 0.0, 0.1, …, 1.0. First, for each test case, *interpolated precision at level r*, i.e. $P_{interp}(r)$, is calculated which determines the highest precision achieved for any recall level $r' \geq r$. Then, *11-point Interpolated Average Precision* is created by computing average value of $P_{interp}(r)$ over the ten queris for the eleven different values of r. The result is shown in Fig. 14. As shown in this figure, it is possible to reach a precision of at most 71 % at the recall level of 0.6. In other word, at the level in which 60 % of the correct results are returned, at most 71 % of the returned results are correct. Further, the most achievable precision at the recall level 1 is 51 %, which means in order for the 100 % of the correct results to

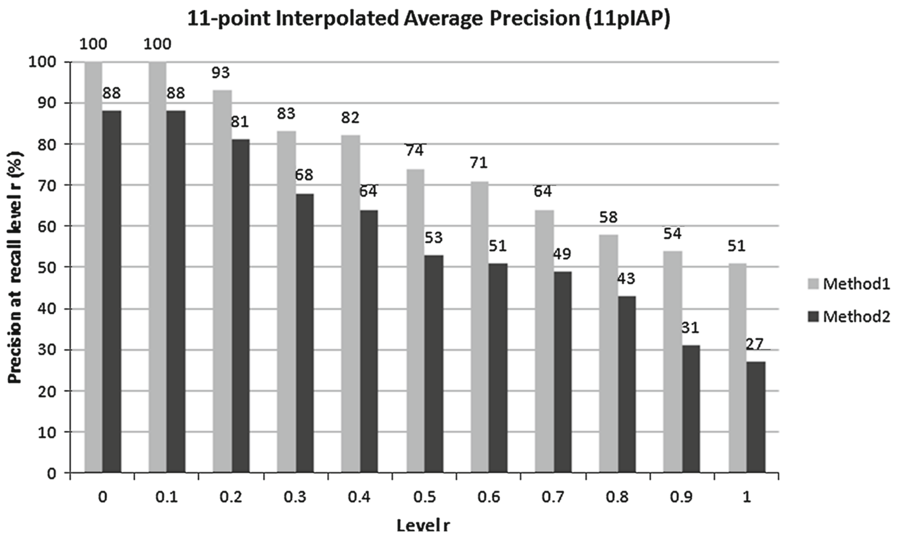


Fig. 14 11pIAP of the use case recommendation algorithm

be returned, it is necessary to accept a precision of at most 51 % (i.e. only half of the returned results are correct). Again, there is considerable difference between results of $Method_1$ and $Method_2$, which supports the idea of giving different weights to use case features.

Based on the three metrics mentioned above, it can be concluded that the proposed use case similarity metric and hence the recommendation algorithm is successful in effectively helping the user to find appropriate use cases for the purpose of reuse.

4.2.2 Activity diagram adaptation

In this section, an initial evaluation of the proposed adaptation algorithm is discussed. First, ten use cases are selected from the repository as the test cases. Each test case is given as input to the implemented prototype, and the recommended use cases are retrieved. Then each use case UC_{new}, along with its recommendation list is given to each of the experts to select the candidate use case UC_{recom} for adaptation. This resulted in a set of 43 different pairs of (UC_{new}, UC_{recom}), since different experts have selected different UC_{recom} for a specific UC_{new}. Finally, 10 pairs with different UC_{new} elements are randomly selected from this set so create a set of 10 test cases.

Each test case is then given as input to the adaptaion algorithm, and the activity diagram of UC_{new} is created by adapting the activity diagram of UC_{recom}. Each generated activity diagram, i.e. AD_{new}, is evaluated by each of the experts. More specifically, the experts are asked to correct AD_{new} by modifying its adaptations. The allowed modifications are changing the labels of AD_{new}. In other words, it is not allowed to modify the structure of AD_{new}, for instance by adding new activity diagram elements. After correcting AD_{new}, the adaptations that have not been modified by the experts are considered to be correct adaptations. This procedure is performed for the ten test use

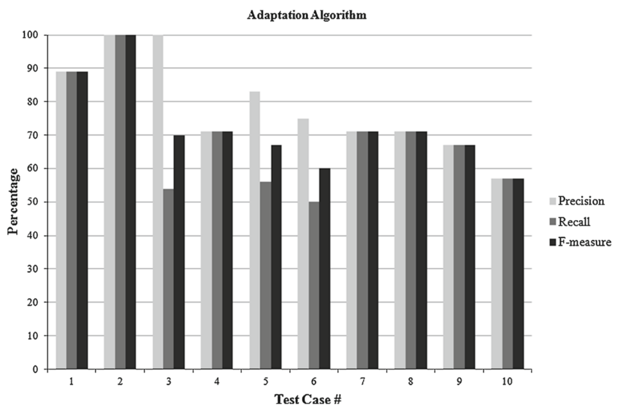


Fig. 15 Evaluation results of the adaptation algorithm

Table 5 Adaptation results for the first test case

#	Original label	Adapted label	Annotation type	Correctness
1	CreateContact	CreatePatient	1	✓
2	CreateContactButtonPressed	CreatePatientButtonPressed	1	✓
3	ContactDateInput	PatientDateInput	1	✓
4	name	name	2	✓
5	email	patient id	2	✓
6	postalAddrMain	age	2	✓
7	postalAddrAlternative	addr	2	✓
8	phoneMain	Patient's Birth Time	2	✓
9	phoneAlternative	sex	2	✓
10	newContact	newPatient	1	✓
11	reEditingContact	reEditingPatient	1	✓
12	saveNewContact	saveNewPatient	1	✓
13	newContact	newPatient	1	✓
14	showAddressBook	ShowReceptionists	3	×
15	newContact	newPatient	1	✓
16	saveContact	savePatient	1	✓
17	searchFor*Contact*InAddressBook	searchFor*Patient*InReceptionists	1	✓
18	searchForContactIn*AddressBook*	searchForPatientIn*Receptionists*	3	×

cases, and the average (over experts) results are shown in Fig. 15. The average precision, recall and F-measure are respectively 78, 69 and 72 % with standard deviation of 14, 16 and 13 %. The minimum precision and recall are correspondingly 57 and 50 %.

As an example, result of the first test case is shown in Table 5 and Fig. 16. As it is shown in Table 5, all the experts have considered adaptations number 14 and 18 as being incorrect. It is interesting to note that for adaptations number 4–9, a search is

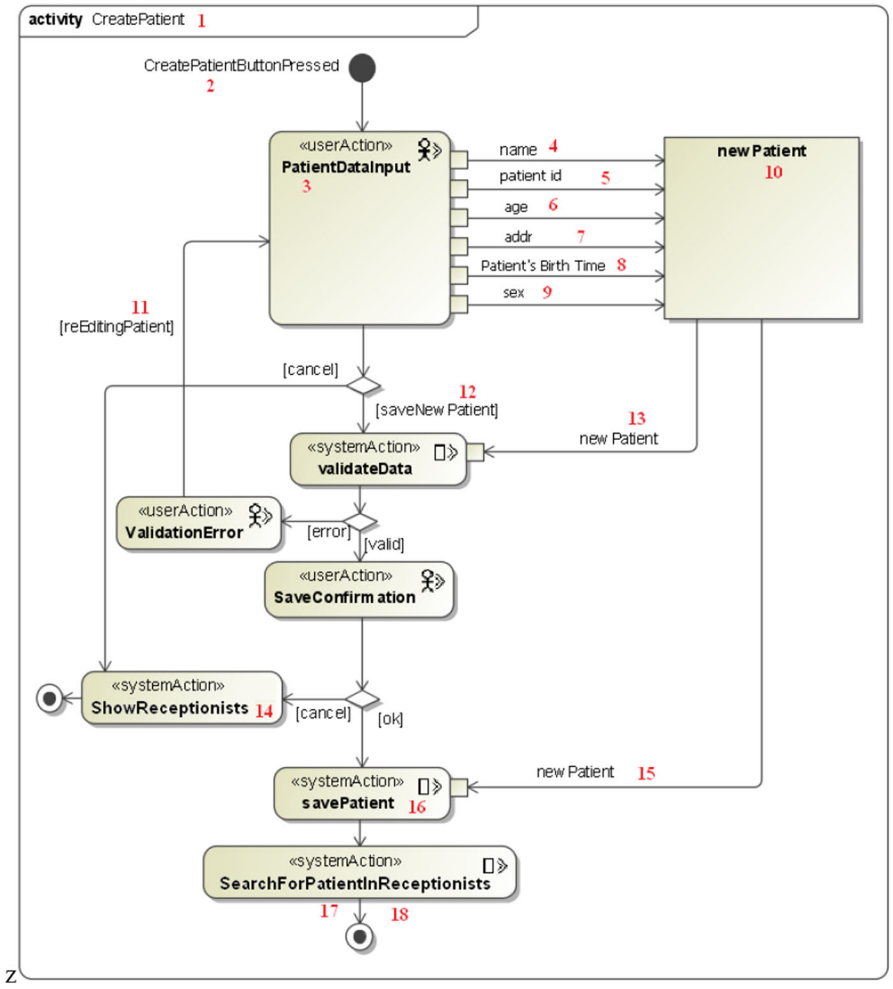

activity CreatePatient 1
CreatePatientButtonPressed 2
«userAction» PatientDataInput 3
name 4
patient id 5
age 6
addr 7
Patient's Birth Time 8
sex 9
newPatient 10
11 [reEditingPatient]
[cancel]
12 [saveNewPatient]
13 new Patient
«systemAction» validateData
«userAction» ValidationError
[error]
[valid]
«userAction» SaveConfirmation
«systemAction» ShowReceptionists 14
[cancel]
[ok]
new Patient 15
«systemAction» savePatient 16
«systemAction» SearchForPatientInReceptionists
17
18

Fig. 16 Activity diagram *'createPatient'* adapted from activity diagram *'createContact'*

performed on the semantic web to find attributes of a concept named *'Patient'*. This search is based on: (1) searching *SWOOGLE* ontology search engine for ontologies which declare a class named *'Patient'*, and also (2) searching *LOD* cloud by sending *SPARQL* queries to its endpoint. A simple version of the employed *SPARQL* query is shown in Fig. 17. Some of the results of this search are presented in

Table 6, which shows URIs of the entities from the semantic web that represent a class named *'Patient'*, along with some of their attributes.

This experiment demonstrates that the proposed adaptation algorithm has acceptable accuracy and effectiveness. However, more work is required on the details of the adaptation algorithm to improve its performance. Through analysis of the experimental results it was turn out that the proposed adaptation rules do not cover all possible cases. In other words, the proposed annotation algorithm might produce annotations

```
SELECT ?CL ?attribute WHERE {
          ?CL       rdf:type           rdfs:Class .
          ?CL       rdfs:label         "Patient"
          OPTIONAL { ?attribute rdfs:domain ?CL . }
}
```

Fig. 17 SPARQL query for finding classes named *'Student'* along with their attributes

Table 6 Some results of searching the semantic web for attributes of the concept *'Patient'*

Class URI	Some attributes
http://sw.opencyc.org/2008/06/10/concept/en/MedicalPatient	age, Primary Cause Of Death For Individual
http://www.loria.fr/~coulet/ontology/sopharm/version2.0/#SOPHARM_51000	patient id, sex, presents clinical item
http://purl.org/healthcarevocab/v1#IE.Patient	Patient Comments, Patient's Birth Time
http://veggente.berlios.de/ns/RIMOntology#Patient	veryImportantPerson Code, name, id, addr,

which are not handled by the proposed adaptation rules. Actually, among all the annotations created for the models of the model repository, the three adaptation rules apply correspondingly to 34, 45 and 2 % of the annotations. This means that about 19 % of the annotations are not handled by these rules. Consequently, a main direction in which the proposed approach can be extended is to add new adaptation rules.

It is also worth noting that as mentioned above, on average, 34 % of the annotations are handled by the first adaptation rule which is straightforward and deterministic in the sense that it does not require searching the semantic web. Regarding the second adaptation rule, this experiment uses *SWOOGLE* search engine and *LOD SPARQL* endpoint for collecting required information. However, it is interesting to investigate usefulness of other sources like *Sindice* search engine[19], or other *Linked Data* sources, and also identify which semantic web sources are more promising for the purpose of the proposed adaptation algorithm.

5 Conclusions and Future Work

An important issue in web engineering methodologies is specification of the functional requirements. Traditionally, UML use cases are used for the brief specification of these requirements, while another technique, e.g. UML activity diagrams, is used for the detailed specification.

In this paper, an approach is proposed which takes the use case diagram of a new software case as input, and semi-automatically generates corresponding UML activity diagrams by reusing models of existing software cases. The main benefit of the pro-

[19] http://sindice.com/

posed approach is that it provides reuse at the very early stages of the development, and at a very low cost. The approach is related to the semantic web in two ways. First, it uses the semantic web technologies to provide an effective representation layer, and further, it uses the semantic web sources for automatically obtaining its required information. The proposed approach is based on three main components: a novel metric for measuring similarity of use cases, and two algorithms, one for activity diagram annotation, and the other for activity diagram adaptation.

The initial experiments demonstrate that the proposed approach is promising. However it can be improved in different directions which are scheduled as our future works. For instance,

- the annotation algorithm currently uses n-gram matching, but it can be improved by employing more sophisticated techniques, e.g. semantic matching, which are more tolerant against textual changes.
- the proposed adaptation algorithm can be improved by extending its adaptation rule set. Currently, it has potential of handling about 81 % of the annotations.
- the adaptation algorithm performs only textual adaptations, and it does not address structural modifications. It is interesting to identify major types of structural adaptations which are required, and then providing adaptation rules for them.
- The way the semantic web sources are used by the proposed approach can be improved by identifying more promising sources and also providing more effective mechanisms for retrieving required information from these sources.

References

Aguilar, J.A., Garrigos, I., Mazon, J.N., Trujillo, J.: An MDA approach for goal-oriented requirement analysis in web engineering. Univers. Comput. Sci. **16**(17), 2475–2494 (2010)

Alchimowicz, B., Jurkiewicz, J., Ochodek, M., Nawrocki, J.: Building benchmarks for use cases. Comput. Inform. **29**(1), 27–44 (2010)

Ali, F.M., Du, W.: Toward reuse of object-oriented software design models. Inf. Softw. Technol. **46**(15), 499–517 (2004)

Alnusair, A., Zhao, T.: Retrieving reusable software components using enhanced representation of domain knowledge. Recent Trends in Information Reuse and Integration, Lecture Notes in Computer Science (LNCS), pp. 363–379 (2012)

Alnusair, A., Zhao, T.: Component search and reuse: an ontology-based approach. In: IEEE International Conference on Information Reuse and Integration (IRI), pp. 258–261 (2010)

Alspaugh, T.A., Ant, A.I., Barnes, T., Mott, B.W.: An integrated scenario management strategy. In: Proceedings of the 4th IEEE International Symposium on Requirements Engineering, pp. 142–149 (1999)

Anda, B., Sjoberg, D.I.K.: Investigating the role of use cases in the construction of class diagrams. Empir. Softw. Eng. **10**, 285–309 (2005)

Anguswamy, R.: A study of factors affecting the design and use of reusable components. In: International Doctoral Symposium on Empirical Software Engineering (IDoESE'12), Lund, Sweden, 21 Sep 2012

Anguswamy, R., Frakes, W.B.: A study of reusability, complexity, and reuse design principles. In: The 6th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM'12, Lund, Sweden (2012)

Antoniou, G., Harmelen, F.V.: Web Ontology Language: OWL. In: Staab, S., Studer, R. (eds.) Handbook of Ontologies, pp. 91–110. Springer, Berlin (2009)

Arkley, P., Riddle, S.: Overcoming the traceability benefit problem. In: Proceedings of the 13th International Conference on Requirements Engineering (RE'05), pp. 385–389 (2005)

Atkinson, C., Kuhne, T.: Model-driven development: a metamodeling foundation. IEEE Softw. **20**(5), 36–41 (2003)

Bagheri, E., Ensan, F., Gasevic, D.: Decision support for the software product line domain engineering lifecycle. Autom. Softw. Eng. **19**(3), 335–377 (2012)

Bajracharya, S., Ossher, J., Lopes, C.: Sourcerer: an infrastructure for large-scale collection and analysis of open-source code. J. Sci. Comput. Program. **79**(1), 241–259 (2014)

Barros, F.: Increasing software quality through design reuse. In: Proceedings of the 7th International Conference on the Quality of Information and Communications Technology, pp. 236–241 (2010)

Bauer, B., Roser, S.: Semantic-enabled software engineering and development. In: Jahrestagung, G.I. (ed.) Lecture Notes in Informatics (LNI) (2), pp. 293–296 (2006)

Bendraou, R., Jezequel, J.M., Gervais, M.P., Blanc, X.: A comparison of six UML-based languages for software process modeling. IEEE Trans. Softw. Eng. **36**(5), 662–675 (2010)

Berners-Lee, T.: Linked Data. Design Issues for the World Wide Web. http://www.w3.org/DesignIssues/LinkedData.html (2006). Accessed 19 Feb 2014

Bildhauer, D., Horn, T., Ebert, J.: Similarity-driven software reuse. In: ICSE'09 Workshop (2009)

Bin, S., Liying, F., Jianzhuo, Y., Pu, W., Zhongcheng, Z.: Ontology-based measure of semantic similarity between concepts. In: World Congress on Software Engineering (WCSE), pp. 109–112 (2009)

Bislimovska, B., Bozzon, A., Brambilla, M., Fraternali, P.: Graph-based search over web application model repositories. In: The 11th International Conference on Web Engineering (ICWE), Paphos, Cyprus, (2011)

Bloc, M.C., Cybulski, J.L.: Reusing UML specifications in a constrained application domain. In: Proceedings of the 5th Asia Pacific Software Engineering Conference (APSEC) (1998)

Bonilla-Morales, B., Crespo, S., Clunie, C.: Reuse of use cases diagrams: an approach based on ontologies and semantic web technologies. Int. J. Comput. Sci. **9**(1), no. 2, 24–29 (2012)

Bozzon, A., Brambilla, M., Fraternali, P.: Searching repositories of web application models. In: Lecture Notes in Computer Science, vol. 6189, pp. 1–15 (2010)

Calero, C., Ruiz, F., Piattini, M. (eds.): Ontologies for Software Engineering and Software Technology. Springer, Berlin (2006)

CAMP, Common Ada Missile Packages, Final Technical Report, vols. 1, 2, and 3, AD-B-102 654,655, 656, Air Force Armament Laboratory, AFATL/FXG, Elgin AFB, FL (1987)

Card, D., Comer, E.: Why do so many reuse programs fail? IEEE Softw. **11**(5), 114–115 (1994)

Castaneda, V., Ballejos, L., Caliusco, M.L.: Improving the quality of software requirements specifications with semantic web technologies. In: Workshop em Engenharia de Requisitos, Buenos Aires (2012)

Castaneda, V., Ballejos, L., Caliusco, M.L., Galli, M.R.: The use of ontologies in requirements engineering. Glob. J. Res. Eng. **10**(6), 2–8 (2010)

Catal, C.: Barriers to the adoption of software product line engineering. ACM SIGSOFT Softw. Eng. Notes **34**(6), 1–4 (2009)

Ceri, S., Fraternali, P., Bongio, A.: Web modeling language (WebML): a modeling language for designing web sites. Comput. Netw. **33**, 137–157 (2000)

Cheng, B.H.C., Atlee, J. M.: Research directions in requirements engineering. In: Future of Software Engineering (FOSE '07), pp. 285–303 (2007)

Cheng, Z., Budgen, D.: What do we know about the effectiveness of software design patterns? IEEE Trans. Softw. Eng. **38**(5), 1213–1231 (2012)

Conallen, J.: Building Web Applications with UML. Addison-Wesley, Boston (2002)

De Troyer, O., Leune, C.: WSDM: a user-centered design method for web sites. In: The 7th International World Wide Web Conference, pp. 85–94. Elsevier, Amsterdam (1998)

Deissenboeck, F., Hummel, B., Juergens, E., Schatz, B., Wagner, S., Girard, J.F., Teuchert, S.: Clone detection in automotive model-based, development. In: ICSE'08, pp. 603–612 (2008)

Dobing, B., Parsons, J.: How UML is used. Commun. ACM **49**, 109–113 (2006)

Durao, F.A., Vanderlei, T.A., Almeida, E.S., Meira, S.R.L.: Applying a semantic layer in a source code search tool. In: The 2008 ACM Symposium on Applied Computing (SAC '08), New York, pp. 1151–1157 (2008)

Ebner, G., Kaindl, H.: Tracing all around in reengineering. IEEE Softw. **19**(3), 70–77 (2002)

Elias, M., Johannesson, P.: A survey of process model reuse repositories. In: Proceedings of ICISTM 2012, pp. 64–76 (2012)

Escalona, M.J., Aragon, G.: NDT: a model driven approach for web requirements. IEEE Trans. Softw. Eng. **34**(3), 377–390 (2008)

Escalona, M.J., Koch, N.: Metamodelling the requirements of web systems. In: Lecture Notes in Business Information Process, vol. 1, pp. 267–288. Springer, New York (2007)

Falessi, D., Cantone, G., Kazman, R., Kruchten, P.: Decision-making techniques for software architecture design: a comparative survey. ACM Comput. Surv. **43**(4), 1–28 (2011)

Frakes, W., Isoda, S.: Success factors of systematic reuse. IEEE Softw. **11**(5), 14–19 (1994)

Frakes, W.B., Kang, K.: Software reuse research: status and future. IEEE Trans. Softw. Eng. **31**(7), 529–536 (2005)

Frakes, W., Prieto-Diaz, R., Fox, C.: DARE: domain analysis and reuse environment. Ann. Softw. Eng. **5**, 125–141 (1998)

Frakes, W., Terry, C.: Software reuse: metrics and models. ACM Comput. Surv. **28**(2), 415–435 (1996)

Frakes, W.B., Baeza-Yates, R.: Information Retrieval: Data Structures and Algorithms. Prentice-Hall, Upper Saddle River (1992)

Girschick, M.: Difference detection and visualization in UML class diagrams, Technical Report TUD-CS-2006-5 (2006)

Gomes, P., Gandola, P., Cordeiro, J.: Helping software engineers reusing UML class diagrams. In: Proceedings of the 7th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development, pp. 449–462 (2007)

Gomes, P., Pereira, F.C., Paiva, P., Seco, N., Carreiro, P., Ferreira, J.L., Bento, C.: Case retrieval of software designs using WordNet. In: European Conference on Artificial Intelligence (ECAI 02), pp. 245–249 (2002)

Gomes, P., Pereira, F.C., Paiva, P., Seco, N., Carreiro, P., Ferreira, J.L., Bento, C.: Using Wordnet for case-based retrieval of UML models. AI Commun. **17**(1), 13–23 (2004)

Gotel, O.C.Z., Finkelstein, A.C.W.: An analysis of the requirements traceability problem. In: Proceedings of the 1st International Requirements, Engineering Conference (RE'94), pp. 94–101 (1994)

Gronmo, R., Moller-Pedersen, B.: From UML2 sequence diagrams to state machines by graph transformation. J, Object Technol. **10**, 1–22 (2011)

Hamid, A.: A source code search engine for keyword based structural relationship search. Thesis, the University of Texas at Arlinkgton (2013)

Hamilton, K., Miles, R.: Learning UML 2.0. O'Reilly (2006)

Happel, H.J., Seedorf, S.: Applications of ontologies in software engineering. International Workshop on Semantic Web Enabled Software Engineering, pp. 1–14 (2006)

Hartig, O., Kost, M., Freytag, J.C.: Automatic component selection with semantic technologies. In: The 4th International Workshop on Semantic Web Enabled Software Engineering (SWESE), Karlsruhe (2008)

Holmes, R., Walker, R.J., Murphy, G.C.: Approximate structural context matching: an approach to recommend relevant examples. IEEE Trans. Softw. Eng. **32**(12), 952–970 (2006)

IEEE: IEEE Standard Glossary of Software Engineering Terminology, IEEE Press, Piscataway (1990)

Ilieva, M.G., Boley, H.: Representing textual requirements as graphical natural language for UML diagram generation. In: Software Engineering and KnowledgeEngineering (SEKE), pp. 478–483 (2008)

Iqbal, A., Ureche, O., Hausenblas, M., Tummarello, G.: LD2SD: linked data driven software development. In: The 21th International Conference on Software Engineering and Knowledge Engineering (SEKE), Boston (2009)

Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, Reading (1992)

Jacobson, I.: Use cases—yesterday, today, and tomorrow. Softw. Syst. Model. **3**, 210–220 (2004)

Jalender, B., Gowtham, N., Kumar, K.P., Murahari, K., Sampath, K.: Technical impediments to software reuse. Int. J. Eng. Sci. Technol. **2**(11), 6136–6139 (2010)

Jones, C.: Software return on investment preliminary analysis. Software Productivity Research, Inc. (1993)

Kang, S., Kim, H., Baik, J., Choi, H., Keum, C.: Transformation rules for synthesis of UML activity diagram from scenario-based specification. In: 34th Annual IEEE Computer Software and Applications Conference, pp. 431–436 (2010)

Karlsson, E.A.: Software Reuse- A Holistic Approach. Wiley, New York (1995)

Keivanloo, I., Roostapour, L., Schugerl, P., Rilling, J.: Semantic web-based source code search. In: The 6th International Workshop on Semantic Web Enabled Software Engineering (SWESE 2010), San Francisco (2010)

Kelte, U., Wehren, J., Niere, J.: A generic sifference algorithm for UML models. In: Proceedings of the Software Engineering Conference 2005, Essen, Germany, pp. 105–116 (2005)

Klimek, R., Szwed, P.: Formal analysis of use case diagrams. Comput. Sci. **11**, 115–131 (2010)

Kling, W., Jouault, F., Wagelaar, D., Brambilla, M., Cabot, J.: MoScript: A DSL for querying and manipulating model repositories. In: Lecture Notes in Computer Science, pp. 180–200 (2011)

Koch, N.: Software engineering for adaptive hypermedia applications. Ph.D. Dissertation, Ludwig-Maximilians-University Munich, Munich, Germany (2000)

Koch, N., Knapp, A., Zhang, G., Baumeister, H.: UML-based web engineering: an approach based on standards. In: Olsina, L., Pastor, O., Rossi, G., Schwabe, D. (eds.) Web Engineering: Modelling and Implementing Web Applications, pp. 157–191. Springer, Berlin (2008)

Koch, N., Knapp, A., Kozuruba, S.: Assessment of effort reduction due to model-to-model transformations in the web domain. Web Eng. **7387**, 215–222 (2012)

Koch, N., Kozuruba, S.: Requirements models as first class entities in model-driven web engineering. In: 3rd Workhop on the Web and Requirements Engineering at ICWE 2012 (2012)

Korner, S.J., Gelhausen, T.: Improving automatic model creation using ontologies. In: Software Engineering and Knowledge Engineering (SEKE) (2008)

Korner, S.J., Brumm, T.: Improving natural language specifications with ontologies. In: Software Engineering and Knowledge Engineering (SEKE) (2009)

Lauesen, S., Kuhail, M.A.: Task descriptions versus use cases. Requir. Eng. **17**(1), 3–18 (2012)

Lemos, O.A.L., Bajracharya, S.K., Ossher, J., Morla, R.S., Masiero, P.C., Baldi, P., Lopes, C.V.: CodeGenie: using test-cases to search and reuse source code. In: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, pp. 525–526 (2007)

Li, Y., McLean, D., Bandar, Z.A., O'Shea, J.D., Crockett, K.: Sentence similarity based on semantic nets and corpus statistics. IEEE Trans. Knowl. Data Eng. **18**(8), 1138–1150 (2006)

Liang, Y.: From use cases to classes: a way of building object model with UML. Inf. Softw. Technol. **45**, 83–93 (2003)

Lim, W.C.: Effects of reuse on quality, productivity, and economics. IEEE Softw. **11**(5), 23–30 (1994)

Lin, D.: An informatino-theoretic definition of similarity. In: Proceedings of the 15th International Conference on Machine Learning, vol. 1, pp. 296–304 (1998)

Lucas, C.: Documenting reuse and evolution with reuse contracts. Ph.D. Dissertation, Vrije Universiteit Brussel (1997)

Lucredio, D., Fortes, R. P.M., Whittle, J.: Moogle: a model search engine. In: The 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS '08), pp. 296–310. Springer, Berlin (2008)

Malan, R., Wentzel, K.: Economics of software reuse revisited. In: Proceedings of the 3rd Irvine Software Symposium, University of California, Irvine, pp. 109–121 (1993)

McIlroy, M.D.: Mass produced software components. In: NATO Software Engineering Conference, Garmisch, Germany (1968)

McMillan, C., Hariri, N., Poshyvanyk, D., Cleland-Huang, J.: Recommending source code for use in rapid software prototypes. In: Proceedings of the International Conference of Software Engineering (ICSE), pp. 848–858 (2012)

McMillan, C., Grechanik, M., Poshyvanyk, D., Fu, C., Xie, Q.: Exemplar: A source code search engine for finding highly relevant applications. IEEE Trans. Softw. Eng. **38**(5), 1069–1087 (2012)

Mellor, S.J., Balcer, M.J.: Executable UML: A Foundation for Model-Driven Architecture. Addison-Wesley, Boston (2002)

Mellor, S.J., Clark, A.N., Futagami, T.: Model-driven development. IEEE Softw. **20**(5), 14–18 (2003)

Mens, T., Lucas, C., Steyaert, P.: Supporting disciplined reuse and evolution of UML models. In: Lecture Notes in Computer Science, vol. 1618, pp. 378–392 (1999)

Mili, H., Mili, F., Mili, A.: Reusing software: issues and research directions. IEEE Trans. Softw. Eng. **22**(6), 528–562 (1995)

Milli, A., Fowler, S.C., Gottumkkala, R., Zhang, L.: An integrated cost model for software reuse. In: Proceedings of the 22nd International Conference on Software Engineering, pp. 157–166 (2000)

Miller, G.: Wordnet: a lexical database for English. Commun. ACM **38**, 39–41 (1995)

Mohagheghi, P., Conradi, R.: Quality, productivity, and economics benefits of software reuse: a review of industrial studies. Empir. Softw. Eng. **12**, 471–516 (2007)

Monden, A., Nakae, D., Kamiya, T., Sato, S., Matsumoto, K.: Software quality analysis by code clones in industrial legacy software. In: Proceedings of the 8th International Symposium on Software Metrics (2002)

Morisio, M., Ezran, M., Tully, C.: Success and failure factors in software reuse. IEEE Trans. Softw. Eng. **2**(4), 340–357 (2002)

Murugesan, S.: Web application development: challenges and the role of web engineering. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modelling and Implementing Web Applications. Springer, London (2008)

Murugesan, S.: Web engineering: a new discipline for development of web-based systems. In: First ICSE Workshop on Web Engineering, Los Angeles, pp. 1–9 (1999)

Neighbors, J.: Software construction using components. Ph.D. Thesis, Department of Information and Computer Science, University of California, Irvine (1981)

Nowick, E., Eskridge, K.M., Travnicek, D.A., Chen, X., Li, J.: A model search engine based on cluster analysis of search terms. Libr. Philos. Pract. **7**(2), 1–6 (2005)

Nyulas, C., Noy, N.F., Dorf, M., Griffith, N., Musen, M.A.: Ontology-driven software: what we learned from using ontologies as infrastructure for software or how does is taste to eat our own dogfood. In: Proceedings of the 5th International Workshop on Semantic Web Enabled Software Engineering (SWESE 2009), pp. 58–72 (2009)

OMG: "Model Driven Architecture". Object Management Group. http://www.omg.org/mda/ (2005)

"OMG Unified Modeling Language (OMG UML) Superstructure, version 2.4. (2011)

Paige, R.F., Olsen, G.K., Kolovos, D.S., Zschaler, S., Power, C.: Building model-driven engineering traceability classifications. In: Proceedings of ECMDA Traceability Workshop (ECMDA-TW), pp. 49–58 (2008)

Park, W.J., Bae, D.H.: A two-stage framework for UML specification matching. J. Inf. Softw. Technol. **53**, 230–244 (2010)

Paydar, S., Kahani, M.: A semantic web based approach for design pattern detection from source code. In: The International Conference on Computer and Knowledge Engineering (ICCKE 2012), Mashhad, Iran (2012)

Porter, M.F.: An algorithm for suffix stripping. Program **14**(2), 130–137 (1980)

Postmus, D., Meijler, T.D.: Aligning the economic modeling of software reuse with reuse practices. J. Inf. Softw. Technol. **50**, 753–762 (2008)

Prasad, A., Park, E.K.: Reuse system: an artificial intelligence-based approach. J. Syst. Softw. **27**, 207–221 (1994)

Prieto-Diaz, R.: Domain analysis for reusability. In: Proceedings of COMPSAC'87, Tokyo, Japan, pp. 23–29 (1987)

Prieto-Diaz, R.: Domain analysis: and introduction. ACM SIGSOFT Softw. Eng. Notes **15**(2), 47–54 (1990)

Ramesh, B., Edwards, M.: Issues in the development of a requirements traceability model. In: Proceedings of the IEEE International Symposium on Requirements Engineering, pp. 256–259 (1993)

Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Trans. Softw. Eng. **27**(1), 58–93 (2001)

Robinson, W.N., Woo, H.G.: Finding reusable UML sequence diagrams automatically. IEEE Softw. **21**, 60–67 (2004)

Robles, K., Fraga, A., Morato, J., Llorens, J.: Towards an ontology-based retrieval of UML class diagrams. Inf. Softw. Technol. **54**(1), 72–86 (2012)

Sabetzadeh, M., Easterbrook, S.: Traceability in viewpoint merging: a model management perspective. In: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE '05), pp. 44–49 (2005)

Saeki, M.: Reusing use case descriptions for requirements specification: towards use case patterns. In: Proceedings of the 6th Asia Pacific Software Engineering Conference (APSEC), pp. 309–316 (1999)

Salami, H.O., Ahmed, M.A.: A framework for class diagram retrieval using genetic algorithm. In: Proceedings of the 24th International Conference on Software Engineering and Knowledge Engineering (SEKE 2012), pp. 737–740 (2012)

Salami, H.O., Ahmed, M.A.: UML artifacts reuse: state of the art. Int. J. Soft Computi. Softw. Eng. (JSCSE) **3**(3), 115–122 (2013)

Samarasinghe, N., Some, S.S.: Generating a domain model from a use case model. In: Proceedings of the ISCA 14th International Conference on Intelligent and Adaptive Systems and Software Engineering, July 20–22, Natural Sciences and Engineering Research Council of Canada, Toronto, Canada, pp. 23–29 (2005)

Schwabe, D., Rossi, G.: From domain models to hypermedia applications: an object-oriented approach. In: The International Workshop on Methodologies for Designing and Developing Hypermedia Applications (1994)

Selic, B.: The pragmatics of model-driven development. IEEE Softw. **20**(5), 19–25 (2003)

Selonen, P.: A review of UML model comparison approaches. In: Proceedings of Nordic Workshop on Model Driven Engineering, Ronneby, Sweden, 27–29 August 2007

Sen, A.: The role of opportunism in the software design reuse process. IEEE Trans. Softw. Eng. **23**(7), 418–436 (1997)

Shahri, H.H., Hendler, J.A., Porter, A.A.: Software configuration management using ontologies. In: The 3rd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2007), Innsubruk, Austria (2007)

Sherif, K., Vinze, A.: Barriers to adoption of software reuse, a qualitative study. J. Inf. Manag. **41**(2), 159–175 (2003)

Smialek, M., Kalnins, A., Kalnina, E., Ambroziewicz, A.: Comprehensive system for systematic case-driven software reuse. In: Proceedings of SOFSEM 2010: Theory and Practice of Computer Science, pp. 697–708 (2010)

Some, S.S.: An approach for the synthesize of state transition graphs from use cases. In: International Conference on Software Engineering Research and Practice (SERP'03), pp. 456–462 (2003)

Spanoudakis, G., Zisman, A.: Software traceability: a roadmap. In: Change, S.K. (ed.) Handbook of Software Engineering and Knowledge Engineering, vol. 3, pp. 395–428. World Scientific Publishing Co., Singapore (2005)

Srisura, B., Daengdej, J.: Retrieving use case diagram with case-based reasoning approach. J. Theor. Appl. Inf. Technol. **19**(2), 68–78 (2010)

Stephan, M., Cordy, J.R.: A survey of methods and applications of model comparison, technical report 2011–582 Rev. 3, School of Computing, Queen's University, ON, Canada (2012)

Steyaert, P., Lucas, C., Mens, K., D'Hondt, T.: Reuse contracts—managing the evolution of reusable assets. In: Proceedings of OOPSLA '96. SIGPLAN Notices, vol. 31, no. 10, pp. 268–286 (1996)

Tappolet, J., Kiefer, C., Bernstein, A.: Semantic web enabled software analysis. J. Web Semant. **8**, 225–240 (2010)

The RDF Advantages. http://www.w3.org/RDF/advantages.html

Trakarnviroj, A., Prompoon, N.: A storage and retrieval of requirement model and analysis model for software product line. In: Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS 2012), Hong Kong, 14–16 March (2012)

Treude, C., Berlik, S., Wenzel, S., Kelter, U.: Difference computation of large models. In: Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, pp. 295–304 (2007)

Udomchaiporn, A., Prompoon, N., Kanongchaiyos, P.: Software requirements retrieval using use case terms and structure similarity computation. In: Proceedings of the 13th Asia Pacific Software Engineering Conference, APSEC, pp. 113–120 (2006)

Valderas, P., Pelechano, V.: A survey of requirements specification in model-driven development of web applications. ACM Trans. Web (ACM) **5**(2), 1–51 (2011)

Vilain, P., Schwabe, D., Sieckenius, C.: Use cases and scenarios in the conceptual design of web application. Technical report MCC 12/00, Departamento de Informatica, PUC-Rio, Rio de Janeiro, Brasil, (2000)

Walenstein, A., El-Ramly, M., Cordy, J.R., Evans, W.S., Mahdavi, K., Pizka, M., Rama-lingam, G., von Gudenberg, J.W., Similarity in programs. In: Koschke, R., Merlo, E., Walenstein, A. (eds.) Duplication, Redundancy, and Similarity in Software, number 06301 in Dagstuhl Seminar Proceedings. IBFI, Dagstuhl (2007)

Watkins, r, Neal, M.: Why and how of requirements tracing. IEEE Softw. **11**(4), 104–106 (1994)

Winkler, S., Pilgrim, J.V.: A survey of traceability in requirements engineering and model-driven development. J. Softw. Syst. Models **9**, 529–565 (2010)

Woo, H.G., Robinson, W.N.: A light-weight approach to the reuse of use-cases specifications. In: Southern Association for Information Systems 2002 Conference, Savannah, GA (2002)

Wu, Z., Palmer, M.: Verb semantics and lexical selection. In: Proceedings of the 32nd Annual Meeting of the Associations for, Computational Linguistics, pp. 133–138 (1994)

Yamamoto, T., Matsushita, M., Kamiya, T., Inoue, K.: Measuring similarity of large software systems based on source code correspondence. In: Bomarius, F., Komi-Sirvio, S. (eds.) PROFES 2005, LNCS 3547, pp. 530–544 (2005)

Yamamoto, T., Matsushita, M., Kamiya, T., Inoue, K.: Similarity of software system and its measurement took SMMT. Syst. Comput. Jpn. **38**(6), 91–99 (2007)

Yao, H., Etzkorn, L.H., Virani, S.: Automated classification and retrieval of reusable software component. J. Am. Soc. Inf. Sci. Technol. **59**(4), 613–627 (2008)

Yue, T., Briand, L.C., Labiche, Y.: A systematic review of transformation approaches between user requirements and analysis models. Requir. Eng. **16**(2), 75–99 (2011)

Yue, T., Briand, L.C., Labiche, Y.: Automatically deriving a UML analysis model from a use case model. Technical Report 2010–2015, Simula Research Laboratory (2010)

Yue, T., Briand, L.C., Labiche, Y.: Facilitating the transition from use case models to analysis models: approach and experiments. Trans. Softw. Eng. Methodol. (TOSEM) **22**(1), 1–38 (2013)

Yue, T., Briand, L.C., Labiche, Y.: An automated approach to transform use cases into activity diagrams. In: Lecture Notes in Computer Science, pp. 337–353 (2010)

Zhuge, H.: A process matching approach for flexible workflow process reuse. Inf. Softw. Technol. **44**(8), 445–450 (2002)