The use of Ontologies as a backbone for use case management

Bart Wouters and Dirk Deridder and Ellen Van Paesschen

Abstract

Towards end-users, use cases provide an intuitive and easy to understand notation to capture the requirements of a system. This is a major benefit, and is also the main reason why software engineers favor them above other notations. However, as a result of this simplicity, they are less suitable to communicate the requirements in a precise, and unambiguous format towards developers. Also managing large sets of use cases becomes a troublesome and almost impossible task. In this paper we present ongoing research on a case based semi-formal approach for use cases, supported by a notion of ontologies, to resolve this drawback amongst others.

1 Introduction

In a UML based environment, use case diagrams are used to model the requirements of a software product. For a major part, they are made up out of textual descriptions of these requirements, in a free format. The main reason for this very informal representation of requirements, is the communicability towards end users. The main purpose of use cases towards them to validate the results of the requirements gathering phase. However they are also used to express the requirements towards software developers. These require an unambiguous representation (i.e. more formal) of the requirements so that they have a reference as a starting point for the development and to verify the different development results.

Although the textual description of use cases in an informal format offers quite a lot of benefits for communicating the requirements towards the end-user, problems arise regarding browseability, maintainability and scalability. Where the maintainability and scalability problem are mainly a consequence of the lack of browseability of textual descriptions.

When one wants to build a tool that tackles these three problem areas, there are three aspects that need to be considered. Firstly the unstructured format of use case descriptions. Secondly the ambiguity coming from the natural language in which use cases are expressed and thirdly how to provide a means to browse large sets of use cases in a structured way.

In this paper we present a first step towards a more powerfull environment as described above by combining a case based, semi-structured format (section

2.2) for use cases with a notion of ontologies (section 2.3).

2 Case-Based, Semi-Formal Structure for Use cases

2.1 Cases in Natural Languages

Cases in natural language are used to make the purpose of a word in a sentence explicit. Most of the time this is done by adding suffixes to the root of a word.

The most populare are the five latin cases, but also other languages like German and Finnish still use them. All sentence parts in these languages are covered by a specific case. Languages like Dutch and English do not use cases anymore, except for a small set of historical leftovers where one still could recognize an old form of a case.

The 5 traditional cases are: nominative (subject), accusative (direct object, direction), genitive (possession), dative (indirect object) and ablative (manner, instrument, goal).

It soon was quite obvious that the 5 cases available in Latin were not enough to cover all constructs encountered in a use case in enough detail. E.g. the part-whole relationship is a relation that is often encountered in use cases. Although this could be classified as a genitive, using the genitive-partitive principle in Latin cases, we prefer not to do so, because there would be too much overloading and ambiguity remaining. Therefore we took a look at the Finnish language, which uses no less than 15 different cases. The part-whole, e.g. is there covered by a specialised part-whole case i.e. the partitive. Apart from the 5 traditional cases mentioned above, a.o. the following Finnish cases are available (and desirable): ablative of manner, ablative instrumentalis, ablative of goal, direction, locative (position), comitative (cooperation), partitive (whole-part), etc.

Although the case-based approach has some problems (e.g. certain propositions need a specific case, independent of it's meaning), we believe that this approach offers more benefits – like the fact that certain prepositions and articles can be omitted – than disadvantages.

It is obvious that quite a lot of situations handled by natural language cannot be handled by this approach. They are only deducible from a quite large knowledge of the context. We believe that in a first phase our approach should not provide a solution to these problems, and we can allow ourselves to enforce some rules on the way use cases should be written (active/passive sentences, etc.).

The next section explains how we are going to use this idea of cases in order to construct a semi-formal structure for use cases.

2.2 Semi-formal structure for use cases

The use cases of Jacobson [Jac94] are used in the analysis part of the software development cycle; they describe the functionality of a certain part of a system

```
Use Case:
    Access to this functionality is only allowed to users
    that are identifiable as sysadmins.

 Semi-formal structure:
   ((verb is allowed)
    (nominatif ((nounname access)
                (possessif ((nounname functionality)
                            (adjectif this))))
    (ablatif_manner ((nounname only)))
    (datif ((nounname users)
            ((verb are identifiable)
             (ablatif_manner ((nounname sysadmins))))))))
```

Figure 1: Example of semi-formal structure of a use case

in natural language and should be understandable for both the developer and the client. Most of the time, they are stored in a free text format.

Although use cases are frequently used and take an important place in the development cycle, they can lower the quality of the final result. Consisting of natural language, use cases are difficult to reuse, are time-consuming for the developer to write down, and are often source of a lot of confusion between the client and the developer. Sometimes the developer interprets the use cases not "exactly" as they were intended by the client. Even if the final system satisfies all use cases, it may still differ from what the client expects. This can result in a system that is not accepted by the client and the developer needs to (partially) re-design and implement the application.

The idea of inserting a semi-formal description, based on cases as proposed in section 2.1 on top of the natural language offers solutions to reduce some of these disadvantages. This description should be unique: use cases with a different description in natural language, but with the same meaning should return the same formalized use case.

The semi-formal structure, contains 2 major parts. The first part is the traditional use case, written down in human readable format. The second part is the semi-formal description, in which all 'important' words are tagged by their case. Next to the case annotation, some more technical features are needed like e.g. a notion of sets, logical operators, kwantors etc. We do not discuss these features in detail as they are well known.

Building a browser that supports the creation of use cases on the one hand, and the semi-formal description on the other hand would already provide some major help for a designer. As soon as the use cases have been given a semi-formal description (manual process), one can start browsing and searching for particular use cases in a structured way. (We already have a small prototype tool)

Until now, the only results a search could provide was the line number where a certain searchword was found. By introducing this semi-formal description, it now becomes possible to look for use cases where a certain word was used in a certain context. This offers us a the possibility to search more focused and more precise (e.g. find me all use cases where a human actor interferes with the userinterfaces).

2.3 The Role of the Ontology

Although the semi-formal description of use cases (section 2.2) addition would already mean quite a big help for designers having to write down and manage use cases, another vast problem still has not been handled, i.e. the ambiguity of natural language. What philologist define as the 'richness' of a language, is a nightmare for the developer. Reason for this is the existence of e.g. synonymy and hyper-/hyponymy.

The term 'ontology' originated as a science within philosophy, but evolved over time. With the introduction of ontologies into various domains of computer science, they evolved towards the name for a complex, generic repository in which it is possible to store any kind of information that can be found in the real world.

The existence of synonyms makes it quite difficult to find the use case we are looking for, as we have to include in our query all possible synonyms for our searchterm (cfr. queries on the internet). In order to solve such problems, it is necessary to integrate an ontology in our browser, and make search and verification algorithms use this ontology.

Our ontology is built based on three major categories of information: labels (the 'names' we tend to give to things), concepts (the things themselves) and relations (these are actually also a kind of concepts, they are used to combine items contained in our ontology). Two major classes of relations exist: relations combining labels and concepts (e.g. synonymy is a relation where multiple labels are combined with one concept), and relations combining concepts (e.g. the part-whole relation). Also tags and the use cases themselves are described in it.

Upon this ontology, a logic inference engine exists, allowing us to enforce rules on the one hand, and to query the data on the other hand.

One has to be aware of the fact, though, that in many cases, ontologies will be much too complex for the goal(s) we are aiming at. In many cases, it could already be sufficient to integrate a simple thesaurus or lexicon into tools. The decision on what to use and on how complex and extended the repositories have to be, should be made at design-time, based on the requirements.

3 Combining a Case-Based, Semi-Formal Structure for Use cases with a Notion of Ontologies

By combining the techniques mentioned in the previous sections, a whole new range of possibilities becomes available in order to make use cases more manageable.

Browsers will beyond any doubt profit from our approach. The benefits for browsing are situated on 2 levels: on the one hand, the writing down of use cases can benefit from this approach as one can start writing algorithms in order to verify whether a use case has not already been written down before [DH], or in a more general way. This is something that was not possible before.

On the other hand, consulting of use cases can be made more efficient and elegant than before. On an abstract level, one can state that consulting boils down to queries. Using the integrated techniques one can search for a (set of) use case(s) where a certain term plays certain role or is related to another term by a special relation. Something that could not be done before.

As browsers get extended by the features mentioned above, one can start thinking about more complex tasks and goals like e.g. reuse. Up until now, the biggest problem in reusing use cases was finding similar or related use cases to reuse. By using the extended browsing possibilities, this becomes possible. Even more, by writing appropriate queries, one can make virtual classifications[Hon98].

Suppose we want to make a classification containing all use cases concerning the interaction between a human user and the user interface. This is possible now because our system is aware of the fact that John, the secretary and Nancy, the system administrator are human users. The system is now also aware of the fact that the user interface is made up of buttons, windows, input fields, forms etc. So all use cases describing any human actor who interacts with any possible part of the userinterface are the one we are looking for. Our virtual classification would be defined as the following kind of query:

Human-GUI interaction Use Case Classification ::=
{ $\forall$ x $\in$ isA_hierarchy('human actor') AND $\forall$ y $\in$ partOf_hierarchy('GUI') AND
$\exists$ R $\in$ isA_hierarchy ('interaction') AND xRy }

Figure 2: Example of classification query

This kind of query can easily be expressed in the logical engine from our ontology.

Furthermore, thanks to the possibility of expressing such classifications, we can solve a lot of scalability problems with use cases. As our classifications are dynamic and offer the possibility of giving different views on the system in manageable chunks. One can make classifications as big or as small as he/she wants, by using more or less specific terminology in the queries which define the classifications.

We believe that our approach could ease the work of development teams

using use cases significantly in all phases of the software development life cycle: from requirement specification over implementation up to checking whether the system conforms to all requirements.

4 Conclusion

In this paper we proposed some ongoing research on techniques to make use cases easier to reuse, more maintainable, and less problematic when dealing with large and complext projects by combining a semi-formal structure and the notion of ontologies.

Although a lot of work remains to be done, we are convinced the ideas presented in this paper are only the beginning. In the near future we will introduce virtual links into the ontology in such a way that use cases can be virutally linked with the software artefacts they are defining. This way we would have an intelligent and extensible browsable indexing system for software artefacts.

References

[DH] Hercules Dalianis and Eduard Hovy. Integrating step schemata using automatic methods.

[Hon98] Koen De Hondt. *A Novel Approach to Architectural Recovery in Evolving Object-Oriented Systems.* PhD thesis, Vrije Universiteit Brussel, December 1998.

[Jac94] Ivar Jacobson. *Object-Oriented Software Engineering, A use case driven approach.* Addison-Wesley Object Technology Series. Addison-Wesley, March 1994.