Improving Communication in Requirements Engineering Activities for Web Applications

Pedro Valderas and Vicente Pelechano

Department of Information System and Computation.
Technical University of Valencia, Spain
Cami de Vera s/n 46022
{pvalderas, pele}@dsic.upv.es

Abstract. We present a requirements engineering environment which provides techniques and tools to improve communication in Requirements Engineering activities. First, a technique based on requirements ontologies is proposed to allow customers to describe their needs. This technique is supported by a tool. This tool provides analysts with structured descriptions of the customers' needs that facilitate analysts to understand the problem to be solved. Next, both a model-to-text transformation and a model-to-model transformation are introduced to automatically obtain a textual requirements specification and a task-based requirements model respectively. The textual specification facilitates customers to validate requirements. The task-based requirements model facilitates programmers to interpret the requirements specification.

1 Introduction

We present a Requirements Engineering (RE) environment for Web applications that provides techniques and tools to improve the process of communication among customers, analysts and programmers. First, this environment introduces a tool which makes customers a set of questions throughout a guided process. This tool analyzes the information provided by customers in order to obtain a structured description of their needs. To do this, the tool uses a set of *requirements ontologies*. The obtained description of the customers' needs is clearly defined by means of concepts of the requirements ontologies which facilitates analysts to understand it. Furthermore, the proposed RE environment introduces two transformations: (1) A *model-to-text transformation* which transforms the ontology-based description of the customers' needs into a requirements specification defined in natural language. This aspect facilitates customers the validation of requirements. (2) A *model-to-model transformation* which transforms the ontology-based description of the customers' needs into a requirements model based on the concept of task. This aspect facilitates programmers to interpret the requirements specification.

The rest of the paper is organized as follows: Section 2 introduces both the concept of requirements ontology and the tool which uses them. Section 3 introduces the model-to-text transformation. Section 4 introduces the model-to-model transformation. Finally, conclusions are comment on in Section 5.

L. Baresi, P. Fraternali, and G.-J. Houben (Eds.): ICWE 2007, LNCS 4607, pp. 242 – 247, 2007.
© Springer-Verlag Berlin Heidelberg 2007

2 Facilitating Customers to Describe Their Needs

We introduce next a strategy which facilitates customers to describe their needs. It is based on two elements: (1) Requirements ontologies and (2) a tool which asks customers for their needs by using these ontologies**.** We present next both elements.

2.1 Requirements Ontologies

A *Requirements Ontology* specifies the concepts and the relationships between concepts that represent a Web application of a specific type (E-commerce applications, web portals, directories, etc). Figure 1 shows a partial view of the requirements ontology for E-commerce applications. This ontology defines concepts such as *On-Line Purchase*, *Shopping Cart,* or *Products* (concepts that characterize E-commerce applications).

To define ontologies of this kind, we use the approach presented in [1]. According to this approach, two kinds of concepts can be defined, namely lexical concepts (enclosed in dashed rectangles) and nonlexical concepts (enclosed in solid rectangles). A concept is *lexical* if its instances are indistinguishable from their representation. *Date* (see Figure 1) is an example of lexical concept because its instances (e.g. "21/05/2005" and "04/09/2004") represent themselves. A concept is nonlexical if its instances are object identifiers, which represent real-world objects. *User* (see Figure 1) is an example of nonlexical concept because its instances are identifiers such as "ID1", which represents a particular person in the real world who is a user. The main concept in a type ontology is marked with "->•". We designate the concept *On-line Purchase* in Figure 1 as the main concept because it represents the main purpose of an E-commerce application.

Figure 1 also shows a set of relationships among concepts, represented by connecting lines, such as *Product has Property.* The arrow connection represents a one-to-one relationship or a many-to-one relationship (the arrow indicates a cardinality of one), and the non-arrow connection represents a many-to-many relationship. For instance, *Auction offers Item* is a many-to-one relationship (i.e. in each auction only an item can be offered but an item can be offered in several auctions) and *Product has Property* is a many-to-many relationship (i.e. a product can have several properties, and a property can be defined for several products). A small circle near the source or the target of a connection represents an optional relationship. For instance, it is not obligatory for a category to belong to another category. A triangle in Figure 1 defines a generalization/specialization with a generalization connected to the apex of the triangle and a specialization connected to its base. For instance, *Direct Purchase* is a specialization of *On-Line Purchase*.

Finally, we have extended this notation by introducing *abstract concepts*. An abstract concept is a concept that depends on the domain of the Web application and need to be instantiated. These concepts are marked with a vertical line on the right side (see Figure 1, concepts *Product*, *Property* and *Category*). For instance, *Product* is an abstract concept because we know that every E-commerce application must allow users to purchase products; however, we do not know what kind of products they are (they can be CDs, Books, software, etc.). This information depends on the E-commerce application domain and must be instantiated by customers. To do this, a tool has been developed. It is introduced in the next section.

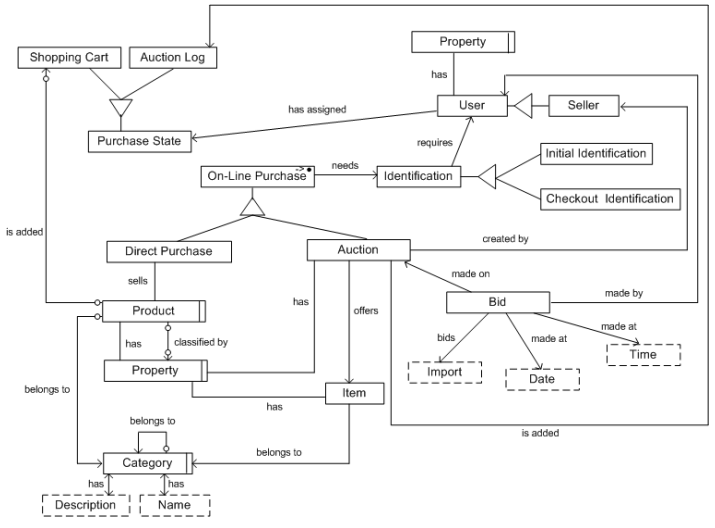

Fig. 1. Requirements Ontology for E-commerce applications

2.2 A Web Application Requirements Elicitation Tool

In this section, we introduce a requirements elicitation tool that supports customers in the description of their needs by means of requirements ontologies. To do this, the tool perform three main steps:

First, the tool allows customers to describe the Web application that they need by using natural language. This description is used by the tool to know the general requirements of the web application and then to select the proper requirements ontology. To do this, we use a technique based on data frames [2]. The data frame approach allows us to describe information about a concept by means of its contextual keywords or phrases, which may indicate the presence of an instance of the concept. We define data frame contextual information for each Web application type that is represented by a requirements ontology. The tool uses this contextual information to recognize the web application type and then select the proper ontology.

Next, the tool must obtain the information that cannot be systematically extracted from a requirements ontology in order to obtain the description of the customers' needs. This information is related to domain-dependent features such as for instance the kinds of products that must be on sale in an E-commerce application (e.g. CDs, DVDs, Books, etc.) (Abstract concepts, see Figure 1). This information must be introduced by customers. To do this, the tool provides them with an appropriate interface. For instance, Figure 2 shows the HTML interface that allows customers to determine which products must be on sale in the running example.

Finally, the information introduced by customers, together with the general features of the application domain (defined in the requirements ontology), allow the tool to obtain a description of the Web application that customers need. This description is defined as a view over the selected requirements ontology where abstract concepts

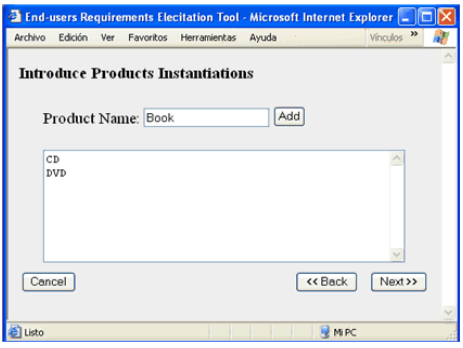

Fig. 2. HTML interface

(e.g. *Product*) are replaced by their instantiations (e.g. *CD*) and relationships among abstract concepts are replaced by relationships among instantiations (e.g. *Product has Property* has been replaced by *CD has Title*). These descriptions are stored in OWL.

3 Obtaining Textual Requirements Specifications

In this section, we introduce a model-to-text transformation that allows analysts to transform the Web application description based on ontology concepts into a textual requirements specification. Furthermore, each textually specified requirement is complemented with a list of real examples where customers can see an implementation of it. This list of real examples facilitates customers both to understand the requirements and to check that it is really what they need.

```
1 <xsl:template match="owl:Class" name="FindShoppingCart">

2 <xsl:variable name="concept" select="@rdf:ID" />
3 <xsl:if test="$concept='Shopping_Cart'">

4 <b>Requirement Number {$num_requirements}:</b>
5 <u>Name:</u> Shopping Cart.

6 <u>Description:</u> The E-Commerce Application must allow
  users to add products to a Shopping Cart. A shopping cart
  is a 'persistent' store for products that can be accessed
  from the whole Web application. The option of 'add to
  shopping cart' is attached to each product in order to al-
  low users to add it. Furthermore, other operations are as-
  sociated to manage the cart such as eliminating a product,
  changing quantities, making an order, etc.

7 <xsl:call-template name="ShoppingCartRealExamples" />

8 </xsl:if>

9 </xsl:template>
```

Fig. 3. Example of a XSL Transformation

To achieve this, we have implemented a set of XSL Transformations which take as source the ontology-based description of the customer needs and then create the corresponding textual requirements specification. The XSL Transformations are created in order to match with the concepts and relationships between concepts that appear in the description of the customer's needs. Figure 3 shows the XSL Transformation that generates a textual requirement specification from the concept "Shopping Cart". In order to better understand it we must know that concepts are represented in OWL by means of the label `owl:Class` and the name of each concept is defined by the attribute `rdf:ID`.

The textual requirements specification that is obtained by means of the transformation in Figure 3 can be consider to be a very simple specification that is little useful throughout the rest of development process. This is true. However, this simplicity has been explicitly chosen in order to facilitate customers to understand them. More formal requirements specifications which can be taken as reference point throughout the development process are obtained in the next section.

4 Obtaining Task-Based Requirements Models

In this section, we introduce a model-to-model transformation that allows analysts to transform the Web application description based on concepts of a requirements ontology into a task-based requirements model. This model is presented in [3].

The transformation has been defined by using a graph transformation technique [4]. Graph transformations are graph rewriting rules made of basically a Left Hand Side (LHS) and a Right Hand Side (RHS). They are applied in the following way: when the LHS matches into a host graph G (which in this case represents the source model) then the LHS is replaced by the RHS.

Figure 4 shows two representative examples of transformation rules. Rule 1 transforms the main concept of a type ontology (which indicates the main purpose of a Web application type, see Section 2.1) into the root of a hierarchical task description. Rule 2 match with the concept "Checkout Identification" which indicate the type of identification that the E-commerce application must support (see Figure 1). According to this concept users must identify themselves when checkout. Then, this concept is derived into a task-based representation which indicates that users must first login and then handle payment in order to checkout.

We have chosen a graph transformation technique because several widely validated tools can be found. In particular, we have chosen the AGG (Attributed Graph Grammar System) tool [5]. The AGG tool can be considered to be a genuine programming environment based on graph transformations. It provides 1) a programming language enabling the specification of graph grammars and 2) a customizable interpreter enabling graph transformations. AGG was chosen because it allows the graphical expression of directed, typed and attributed graphs (for expressing specifications and rules). It has a powerful library containing notably algorithms for graph transformation, critical pair analysis, consistency checking and application of positive and negative conditions.

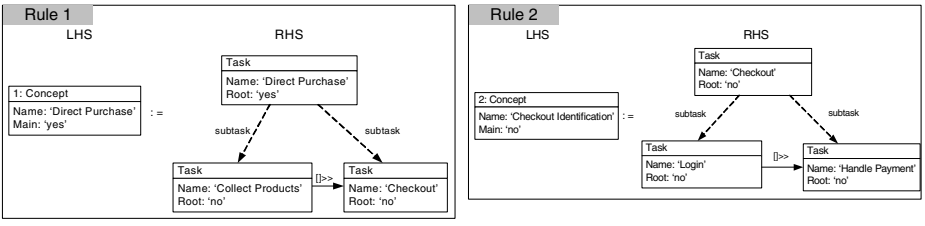

Fig. 4. Examples of transformation rules

5 Conclusions

In this paper we have presented a RE environment in order to improve the communication activity during the RE process.

We have introduced a technique based on Requirements Ontologies in order to facilitate customer to describe their needs. This technique is supported by a requirements elicitation tool which provides customers with an intuitive interface which allows them to generate a structured description of their needs. This helps analysts to understand which Web application customers need. Furthermore, two transformations have been presented: (1) A model-to-text transformation which transforms ontology-based descriptions of the customers' needs into textual requirements specifications. This facilitates customers to validate the requirements specification. (2) A model-to-model transformation which transforms ontology-based descriptions of the customers' needs into task-based requirements models. This provides precise requirement specifications to facilitate programmers interpret them.

References

1. AL-Muhammed, M., Embley, D.W., Liddle, S.: Conceptual Model Based Semantic Web Services. In: Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, Springer, Heidelberg (2005)
2. Embley, D.W.: Programming with Data Frames for every Items. In: Proceedings of AFIPS Conference, Anheim, California, pp. 301–305 (1980)
3. Valderas, P., Fons, J., Pelechano, V.: Developing E-Commerce Application From Task-Based Descriptions. In: Bauknecht, K., Pröll, B., Werthner, H. (eds.) EC-Web 2005. LNCS, vol. 3590, pp. 65–75. Springer, Heidelberg (2005)
4. Rozenberg, G. (ed.): Handbook of Graph Grammars and Computing by Graph Transformation. World Scientific, Singapore (1997)
5. The Attributed Graph Grammar System (AGG) v1.5: http://tfs.cs.tu-berlin.de/agg/