Eliciting User Interface Requirements and Deriving Usability Problems from Scenario Textual Descriptions

Josefina Guerrero-García[*,a], Juan González-Calleros[a]

[a] Computer Science Faculty, Benemérita Universidad Autónoma de Puebla, Puebla, Mexico.

Abstract. *Scenario Textual Descriptions (STD) are general-purpose natural language descriptions of a narrative scenario of end users, real or potential, using an existing or a future interactive system. STDs may take many forms: use cases, structured scenarios, user stories, and natural language expressions of user actions. As such, these STDs contain useful information for initiating the development life cycle of a user interface of this interactive system. On the one hand, when the end user expresses some interaction through these STDs, user interface requirements can be elicited by deriving model fragments from them: user model, task model, domain model, process model, etc. On the other hand, when the end user refers to any previously used system to feed the requirements, usability problems can be derived from user interfaces critiques: usability problems by interaction object, by dialogue box or window, by entire application. Both approaches feed a bidirectional approach where requirements and usability problems co-exist in the same STD. This article presents how FlowiXML supports the entire approach based on a real-world case study for a distributed system for managing teaching students.*

Keywords. Automated User Interface Generation • Business Modelling • Requirements Elicitation

1 Introduction

Scenario-based design (Rosson and Carroll 2009) as well as Participatory Design and other forms of user-centred design initiate typically initiate the development life cycle of the User Interface (UI) of an interactive application using *Scenario Textual Descriptions* (STD), which are general-purpose natural language descriptions of a narrative scenario of end users, real or potential, using an existing or a future interactive system.Such STDs usually consist of informal but structured narrative descriptions of interaction sequences between the users and the interactive system, whether it is existing or envisioned. Scenarios have been proved (Rosson and Carroll 2009) to be a valuable mean to elicit, improve, and validate UI requirements. On the other hand, descriptions of the UI domain itself and the UI requirements are also expressed using conceptual models depicting either static (Tam et al. 1998) or dynamic (Fliedl et al. 2003) aspects of the interactive system. The models resulting from this process are supposed to raise the level of abstraction with respect to the implementation (Tam et al. 1998). The models are frequently expressed in a formal way so as to enable model reasoning. The process which ultimately leads to these descriptions, whether they are informal (such as scenarios) or semi-formal (such as models) is Requirement Engineering (RE) (Haumer et al. 1998). STDs have the advantage to describe UI requirements from captured or imagined user interactions through concrete examples (Garland et al. 2001) of the user carrying out her task. This

* Corresponding author.
E-mail. jguerrero@cs.buap.mx
Note: In a previous work (Lemaigre et al. 2008), we have introduced the process of how to elicit model requirements from textual scenarios. This article generalizes the whole process based on Scenario Textual Descriptions with two original aspects: automatic text interpretation to select elements with CRUDS-based UI to support the handling of selected objects and deriving usability problems from the same source.

form is much more representative and evocative for an end user to validate UI requirements than models that are mainly used by software engineers. Models, e. g., domain models, user models, are expressed in a way that maximizes desirable properties such as completeness, consistency, and correctness (Vanderdonckt 2005). But their expression is significantly less understandable for end users who are often in trouble of validating their UI requirements when they are confronted to models. Consequently, both types of descriptions, scenarios and models, are needed interchangeably in order to conduct a proper process that will effectively and efficiently feed the remainder of the UI development life cycle. STDs could be also effectively used in various configurations of the user interface development life cycle, like forward engineering (Simarro et al. 2005), reverse engineering (Bouillon et al. 2004), adaptation of user interfaces (López-Jaquero et al. 2007), and automated evaluation (Beirekdar et al. 2002).

We introduce model elicitation as the activity of transforming textual scenarios into models that are pertaining to the UI development. The remainder of this article is structured as follows: some related work is reported in Section 2. Three levels of model elicitation are defined in Section 3 and consistently described and discussed in the light of a model elicitation tool implementing these techniques. Section 4 gives another example. Section 5 will sum up the benefits and the shortcomings of the model elicitation techniques investigated so far and will present some future avenues for this work.

2 Related Work

Model elicitation consists of transforming scenarios into models so that they are usable in the rest of the UI development life cycle (Haumer et al. 1998), for instance by conducting a model-driven engineering method (e. g., Clerckx et al. 2006; Vanderdonckt 2005). Model verbalization (Jarrar et al. 2014) is the inverse process: it consists of transforming model elements into textual scenarios while preserving some quality properties (e. g., concision, consistency). Any model may be considered for this purpose: models found in HCI (e. g., task, user) or in RE (e. g., domain, organization). In (Bono and Ficorilli 1992), the system restates queries expressed on a domain model (here, an entity-relationship attribute model) into natural language expression. As such, model elicitation is not new in software engineering (Fliedl et al. 2005, 2004), but at least four significant works have been conducted in Human-Computer Interaction (HCI):

1. U-TEL (Lu et al. 1999) is a user-task elicitation software that enables designers to allocate elements of a textual scenarios into elements of three models: actions names (relevant to the task model), user classes (relevant to a user model), and objects names (relevant to a domain model). This allocation can be conducted manually or automatically.
2. In (Caffiau and Portet 2017), a task model is expressed through a STD and a mapping (Simarro et al. 2005), manipulate, and correct better a STD representation of the task model than the task model itself, which may involve a special notation.
3. T2T (Paris et al. 2002) is a tool for automatic acquisition of task elements (names and relationships) from textual documents such as manuals. Another version exists for the same purpose from a domain model (here, an object-oriented diagram) (Lu et al. 1999) and fro multiple heterogeneous sources (Lu et al. 2002).
4. Garland et al. (Garland et al. 2001) present general software for gathering UI requirements from examples containing various elements that are relevant for different models, but models are not constructed per se.

From these works, we observed the following shortcomings: some, e. g., (Garland et al. 2001) do not produce a genuine model in the end, some other produce model elements that are relevant to HCI (e. g., Garland et al. 2001; Paris et al. 2002; Paternò and Mancini 1999), but the only some model elements are derived (e. g., task names)

or they mostly focus on task models whereas several models are typically found in HCI, not only the task model. When other models are considered, e. g., the user and the domain (Lu et al. 1999), only the names of the classes are captured. In this article, we would like to capture all elements (classes, attributes, and relationships) of several interrelated models to inform the UI development life cycle. It is however fundamental that the task model is considered to initiate a full model-driven engineering life cycle (Clerckx et al. 2006; Paternò and Mancini 1999). DYNAMO-AID (Clerckx et al. 2006) provides a distribution manager which distributes the sub-tasks of a task model to various computing platforms in the same physical environment, thus fostering a task-based approach for distributing UIs across locations of the physical environment. In the next section, an elicitation of UI model elements is provided according to three levels of sophistication.

3 User Interface Model Elements Elicitation

In order to effectively support UI model elicitation, the model elements that are typically involved in the UI development life cycle should be considered. Figure 1 reproduces a simplified version of the ontology of these model elements that will be used throughout this article: only classes and relationships are depicted here for concision, not their attributes and methods. The complete version of this ontology along with its definition and justification is detailed in (Guerrero Garcia et al. 2008). We choose this ontology because it characterizes the concepts used in the development life cycle of UIs for workflow systems, which are assumed to have the one of the largest coverage possible. Any other similar ontology could be used instead. In this ontology, tasks are organized into processes which are in turn ordered in a Workflow. A job consists of a logical grouping of tasks, as we know them (Paternò and Mancini 1999). Jobs are usually assigned to organizational units (e. g., a department, a service) independently of the workers who are responsible to conduct these jobs. These workers are characterized thanks to the notion of user stereotype. But a same task could require other types of resources such as material resources (e. g., hardware, network) or immaterial resources (e. g., electricity, power). A task may manipulate objects that can invoke methods in order to ensure their role. Figure 1 represents the conceptual coverage of model elements that will be subject to model elicitation techniques. This coverage is therefore larger than merely a task, an object, a user as observed today in the state of the art. In the next subsections, three progressively more sophisticated elicitation techniques based on this ontology will be described, motivated, and exemplified on a running textual scenario. This scenario explains the workflow for obtaining administrative documents in a town hall.

3.1 Model Elicitation Level 1: Manual Classification

The UI designer is probably the most reliable person to identify in the textual scenario fragments that need to be elicited into model elements. Therefore, manual classification of model elements remains of high importance for flexibility, reliability, and speed. In a manual classification, any name that represents an instance of a model element belonging to the ontology can be manually selected, highlighted, and assigned to the corresponding concept, such as a task, a job, an organizational unit, etc. Consequently, all occurrences of this instance are automatically identified in the scenario and highlighted in the colour assigned to this concept. For instance, grey for an object, yellow for a user, red for an organizational unit, blue for a task. This colour coding scheme can be parametrized according to the designer's preferences.

Elicitation of a class. Any class belonging to the ontology can be manually classified according to the aforementioned technique. For example, "statement" is considered as an object instance in Figure 2 and is therefore assigned to the corresponding tab . Since a model element may appear in the scenario in multiple alternative forms (e. g.,

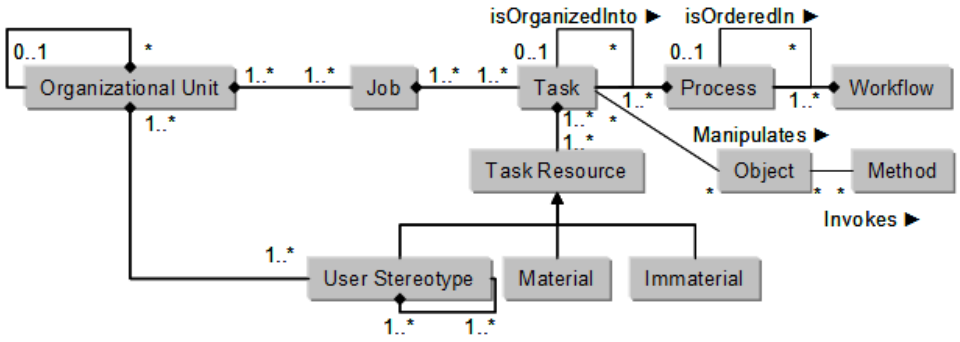

Figure 1: Simplified ontology of the model elements.

a plural form, a synonym), an alias mechanism enables designers to defines names that are considered equivalent to a previously defined one. For example, "statements" and "stated text" could be considered aliases of "statement".

In User-Centred Design (UCD), tasks, users, and objects are often considered as first-class citizens. Therefore, it is likely that the design will initiate the classification by identifying firstly tasks and related objects for instance. An object or a task could be of course elicited separately. In order to speed up this activity, the designer may directly associate a task to its related object when selected according to the same mechanism. All occurrences are highlighted similarly. Figure 3 illustrates this situation: a "birth statement" object is selected and a task "issuing" is attached to this object in order to create a complete task "issuing a birth statement".

A special support exists for tasks: at any time, the designer may specify for a task:

- A predefined task type: a taxonomy of task types (e. g., communicate, create, delete, duplicate) is made accessible for the designer to pick a name from, while a definition for each task type is displayed. This taxonomy consists of 15 basic task types that are decomposed into +/- 40 synonyms or sub-task types as used in the UsiXML User Interface Description Language [18]. Each predefined task type comes with a precise definition and scope of the task, some synonyms if any, and its decomposition into sub-tasks if any.
- A custom task name: any non-predefined task name can be entered, such as "issuing" in Figure 3.
- A pattern of tasks: any set of predefined task types and of custom task names. Such a set can be defined by the designer and reused at any time. For example, the pattern CRUD (acronym for Create, Read, Update, Delete) will automatically enter four predefined task types for a designated object.

Elicitation of an attribute. The same technique is used in order to elicit an attribute of a class: either this attribute is predefined in the ontology (e. g., "frequency" to denote the frequency of a task) or a custom name can be manually entered. For example, in Figure 4, the designer has identified in the scenario the expression denoting the frequency of task and therefore elicits this attribute for the corresponding task (here, "ticketing"). The attribute is then represented as a facer of the corresponding task. Figure 5 graphically depicts the three main steps for entering a custom name for an attribute, here an organizational unit. The location of an organization unit is an attribute that does not belong to the ontology. Therefore, once such a parameter has been selected (Figure 5a), it can be identified with a unique name (Figure 5b), and then included in the hierarchy (Figure 5c).

Elicitation of a relationship. By using drag and drop, the designer can arrange model elements

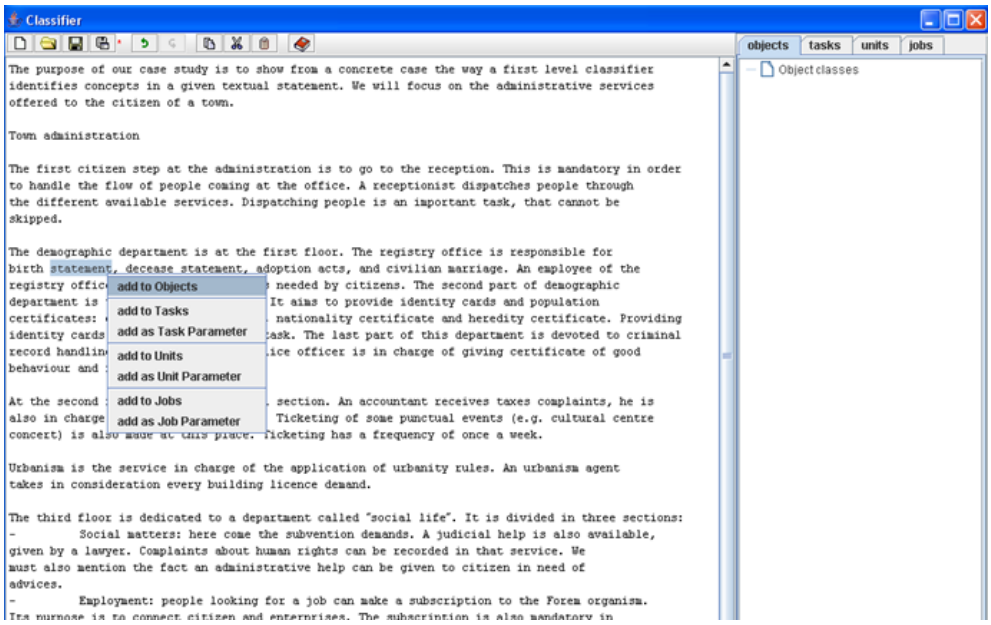

Figure 2: Elicitation of a class (here, an object) in manual classification.

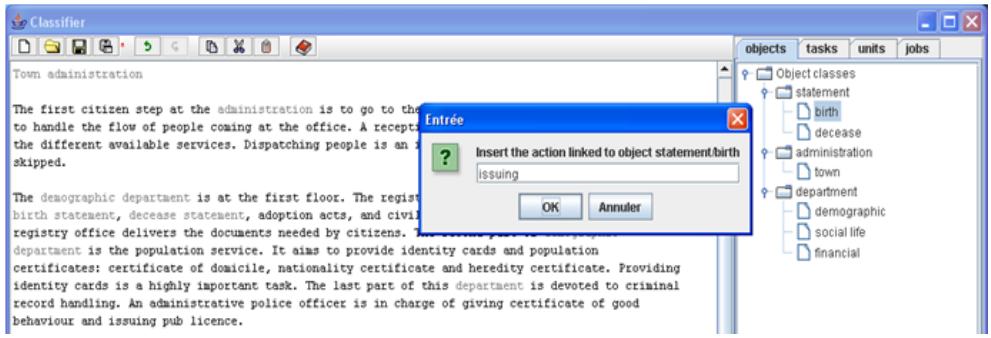

Figure 3: Assigning a task to a already defined object.

in their corresponding hierarchy in order to reflect the decomposition relationships of Figure 1. For example, a task is decomposed into sub-task, tasks are composed in a process, processes are composed into a workflow. Apart from these decomposition relationships, only the "manipulates" relationship between a task and an object can be encoded in this level because it can be recorded thanks to the special support for tasks described above. For example in the right pane of Figure 3, the object "statement" is further refined into the two sub-classes "birth statement" and "death statement", that automatically inherit from the attributes of the super-class.

3.2 Model Elicitation Level 2: Dictionary-based Classification

The model elicitation technique described in the previous sub-section, although flexible, reliable, and fast, represents a tedious task for the designer since it is likely to be repeated. Therefore, instead of manually designating in the textual scenario the fragments that are subject to model elicitation, these fragments could be automatically classified according to a dictionary of model terms. We distinguish two categories of dictionary:

1. Generic dictionaries contain fragments representing model elements that are supposed to

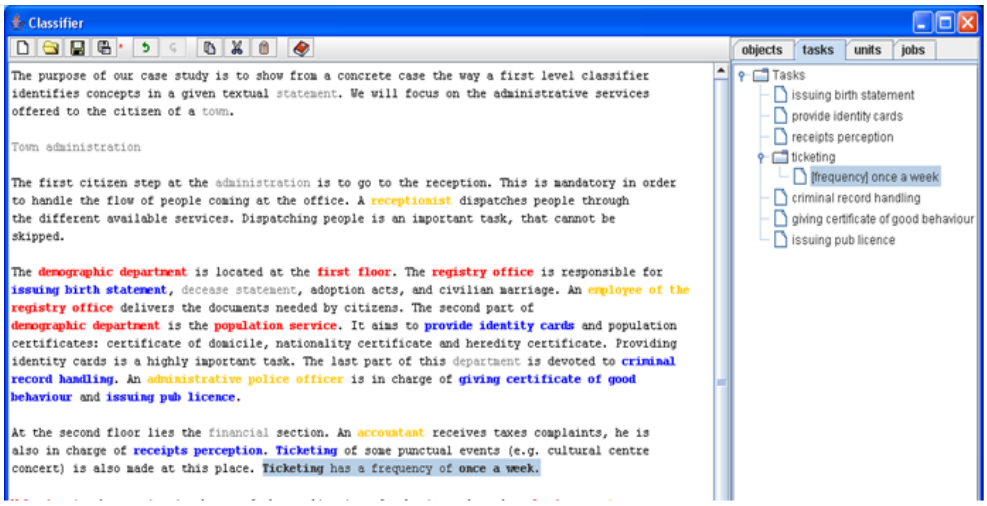

Figure 4: Elicitation of a predefined attribute for a task.

be domain-independent (e. g., "a worker", "a manager", "a clerk" for a user model; "create", "read", "update", "delete" for a task model, etc.)

2. Specific dictionaries that contain fragments representing model elements that are domain-dependent (e. g., a "physician", "a pharmacist" in medicine for a user model; "administrate" for a task model, "physiology " for a domain model).

Each dictionary may contain predefined terms (like the task types) and aliases (e. g. plural, synonyms) in order to maximize the coverage of the automatic classification. In order to tailor this classification, two types of filters could be applied Tam et al. 1998:

1. Positive filters force some model terms to be considered anyway, whatever the domain or the context of use are.
2. Negative filters prevent the automatic classification from classifying irrelevant terms, such as articles (e. g., "the", "a"), conjunctions (e. g., "with", "along").

The terms collected in such filters can be edited manually within any ASCII-compliant text editor. The advantage of this dictionary-based classification over the manual one is certainly its speed: in a very short amount of time, most terms belonging to the dictionaries, modulo their inclusion or rejection through the usage of filters, are classified. The most important drawback of this technique is that the identified terms are not necessarily located in the right place in their corresponding hierarchies. For example, a task hierarchy resulting from this process may consist of a one-level hierarchy of dozens of sub-tasks located in the same level without any relationships between them. In order to overcome this serious drawback, a semantic-based technique is required that is addressed in the next subsection.

3.3 Model Elicitation Level 3: Towards Semantic Understanding

Different techniques exist that elicit model elements from textual scenarios, but so far they have never been applied in HCI to our knowledge: syntactic tagging (Fliedl et al. 2003), semantic tagging (Fliedl et al. 2004), chunk parsing (Fliedl et al. 2004). Genuine semantic understanding requires natural language understanding and processing, which is far beyond the scope of this work. What can be done however is to substitute a semantic understanding by a combination of syntactic and semantic tagging (Fliedl et al. 2005, 2004) in order to recognize possible terms that express, depict, reveal model elements. For instance, a

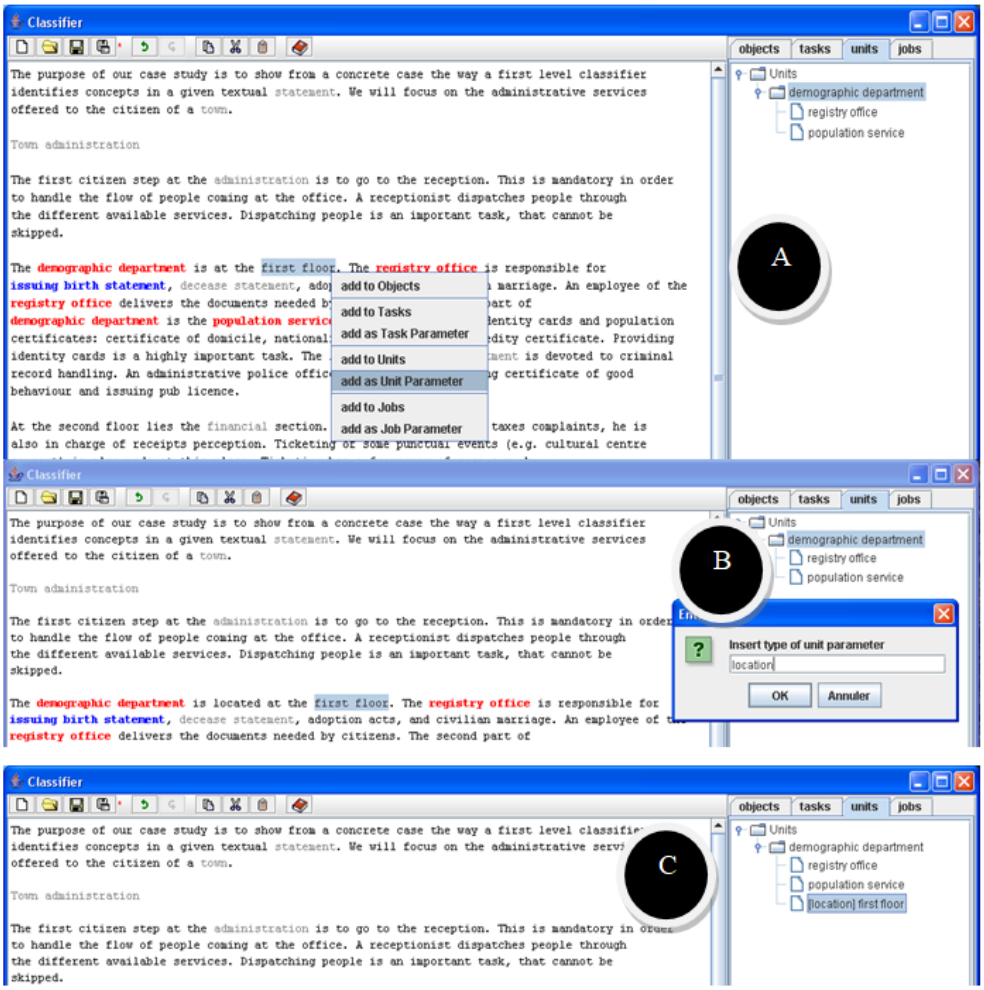

Figure 5: Section a. Elicitation of a custom attribute for an organizational unit: selection. Selection b Elicitation of a custom attribute for an organizational unit: identification. Section c. Elicitation of a custom attribute for an organizational unit: inclusion.

scenario sentence like "An accountant receives taxes complaints, but she is also in charge of receipts perception" should generate: a task "Receive taxes complaint", a task "charge of receipts perception", both being assigned to the user stereotype "Accountant", and a concurrency temporal operator between those two tasks because no specific term is included to designate how these tasks are actually carried out by an accountant. We may then assume the most general temporal operator, like a concurrency temporal operator. To reach this goal, this level attempts to identify possible terms in a syntactical structure (e. g., a set, a list, a sequence) that depicts a pattern for inferring for instance a task, another task with a temporal constraint, etc. For each model element, a table of possible terms involved in this pattern structure is maintained in accordance with the semantics defined in Figure 1. On the one hand, this pattern matching scheme is syntactical because it is only based on detecting a particular combination of terms. On the other hand, those terms are assumed to reflect the precise semantics defined in the ontology. But we cannot say that this is a true semantic understanding anyway. Table 1 shows some excerpts of possible terms related to the concept

of task, along with its attributes, while Table 2 shows some possible terms for detecting possible temporal relationships between tasks. This pattern matching can be executed automatically or under the designer's control who validates each matching one by one. The reserved names for model elements (e. g., task, the task attributes, and the temporal operators between the tasks) are read from the XML schema definition of the underlying User Interface Description Language (UIDL), which is UsiXML (Vanderdonckt 2005) in this case. After performing the elicitation of model elements according to any of the three aforementioned technique, the model elicitation tool can export the results in UsiXML files for the whole set of models or for any particular combination (e. g., only the tasks with the users) at any time. Afterwards, this file can be imported in any other UsiXML-compliant editor, such as IDEALXML for graphical editing.

4 Conclusion

In this article, we have investigated three different techniques for eliciting model elements from fragments found in a textual scenario. These three techniques are progressively more advanced in terms of consideration of the possible terms found in the scenario: from purely manual syntactical classification until ontology-based pseudo-semantic understanding. Beyond the facilities for automated classification of terms into the respective models, that are compatible with the initial ontology, the model elicitation tool allows editing facilities within a same model and across models. Its main drawback today is the lack of graphical visualization of inter-model relationships or intra-model relationships, others than merely decomposition relationships. For the moment, these relationships are only collected in a table that can be further edited. In the near future, we would like to refine the level 3-technique in terms of possible combinations of terms in an expression to be subject for the pattern matching.

References

Beirekdar A., Vanderdonckt J., Noirhomme-Fraiture M. (2002) A Framework and a Language for Usability Automatic Evaluation of Web Sites by Static Analysis of HTML Source Code. In: Kolski C., Vanderdonckt J. (eds.) Computer-Aided Design of User Interfaces III, Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, May, 15-17, 2002, Valenciennes, France. Kluwer, pp. 337–348

Bono G., Ficorilli P. (1992) Natural language restatement of queries expressed in a graphical language. In: Entity-Relationship Approach—ER'92, pp. 357–374

Bouillon L., Vanderdonckt J., Chow K. C. (2004) Flexible re-engineering of web sites. In: Vanderdonckt J., Nunes N. J., Rich C. (eds.) Proceedings of the 9th International Conference on Intelligent User Interfaces, IUI 2004, Funchal, Madeira, Portugal, January 13-16, 2004. ACM, pp. 132–139 http://doi.acm.org/10.1145/964442.964468

Caffiau S., Portet F. (2017) La génération automatique de textes comme support de la compréhension de modèle de tâches en conception: une étude préliminaire. In: Proceedings of IHM2017, pp. 125–135 https://hal.archives-ouvertes.fr/hal-01578499/file/1030.pdf

Clerckx T., Vandervelpen C., Luyten K., Coninx K. (2006) A task-driven user interface architecture for ambient intelligent environments. In: Proceedings of the 11th international conference on Intelligent user interfaces. ACM, pp. 309–311

Fliedl G., Kop C., Mayr H. C. (2003) From scenarios to KCPM dynamic schemas: aspects of automatic mapping. In: NLDB, pp. 91–105

Fliedl G., Kop C., Mayr H. C. (2005) From textual scenarios to a conceptual schema. In: Data & Knowledge Engineering 55(1), pp. 20–37

Fliedl G., Kop C., Mayr H., Winkler C., Weber G., Salbrechter A. (2004) Semantic tagging and chunk-parsing in dynamic modeling. In: Natural Language Processing and Information Systems, pp. 440–446

Garland A., Ryall K., Rich C. (2001) Learning hierarchical task models by defining and refining examples. In: Proceedings of the 1st international conference on Knowledge capture. ACM, pp. 44–51

Guerrero Garcia J., Vanderdonckt J., Gonzalez Calleros J. M. (2008) FlowiXML: a step towards designing workflow management systems. In: International Journal of Web Engineering and Technology 4(2), pp. 163–182

Haumer P., Pohl K., Weidenhaupt K. (1998) Requirements elicitation and validation with real world scenes. In: IEEE Transactions on Software Engineering 24(12), pp. 1036–1054

Jarrar M., Keet C. M., Dongilli P. (2014) Multilingual verbalization of ORM conceptual models and axiomatized ontologies. In:

Lemaigre C., Guerrero J., Vanderdonckt J. (2008) Interface model elicitation from textual scenarios. In: Human-Computer Interaction Symposium. Springer, pp. 53–66

López-Jaquero V., Vanderdonckt J., Simarro F. M., González P. (2007) Towards an Extended Model of User Interface Adaptation: The Isatine Framework. In: Gulliksen J., Harning M. B., Palanque P. A., van der Veer G. C., Wesson J. (eds.) Engineering Interactive Systems - EIS 2007 Joint Working Conferences, EHCI 2007, DSV-IS 2007, HCSE 2007, Salamanca, Spain, March 22-24, 2007. Selected Papers. Lecture Notes in Computer Science Vol. 4940. Springer, pp. 374–392 https://doi.org/10.1007/978-3-540-92698-6_23

Lu S., Paris C., Vander Linden K. (1999) Toward the automatic construction of task models from object-oriented diagrams. In: Engineering for human-computer interaction. Springer, pp. 169–189

Lu S., Paris C., Vander Linden K. (2002) Computer Aided Task Model Acquisition From Heterogeneous Sources. In: Proc. of 5th Asia Pacific Conference on Computer Human Interaction APCHI, pp. 878–886

Paris C., Linden K. V., Lu S. (2002) Automated knowledge acquisition for instructional text generation. In: Proceedings of the 20th annual international conference on Computer documentation. ACM, pp. 142–151

Paternò F., Mancini C. (1999) Developing task models from informal scenarios. In: CHI'99 Extended Abstracts on Human Factors in Computing Systems. ACM, pp. 228–229

Rosson M. B., Carroll J. M. (2009) Scenario based design. In: Human-computer interaction. Boca Raton, FL, pp. 145–162

Simarro F. M., López-Jaquero V., Vanderdonckt J., González P., Lozano M. D., Limbourg Q. (2005) Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML. In: Gilroy S. W., Harrison M. D. (eds.) Interactive Systems, Design, Specification, and Verification, 12th International Workshop, DSVIS 2005, Newcastle upon Tyne, UK, July 13-15, 2005, Revised Papers. Lecture Notes in Computer Science Vol. 3941. Springer, pp. 161–172 https://doi.org/10.1007/11752707_14

Tam R., Maulsby D., Puerta A. R. (1998) U-TEL: A tool for eliciting user task models from domain experts. In: Proceedings of the 3rd international conference on Intelligent user interfaces. ACM, pp. 77–80

Vanderdonckt J. (2005) A MDA-compliant environment for developing user interfaces of information systems. In: International Conference on Advanced Information Systems Engineering. Springer, pp. 16–31