Interface Model Elicitation from Textual Scenarios

Christophe Lemaigre, Josefina Guerrero García, and Jean Vanderdonckt

Belgian Laboratory of Computer-Human Interaction (BCHI)
Louvain School of Management (LSM), Université catholique de Louvain (UCL)
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
E-mail: {christophe.lemaigre@, josefina.guerrero@student,
jean.vanderdonckt@}uclouvain.be

Abstract: During the stage of system requirements gathering, model elicitation is aimed at identifying in textual scenarios model elements that are relevant for building a first version of models that will be further exploited in a model-driven engineering method. When multiple elements should be identified from multiple interrelated conceptual models, the complexity increases. Three method levels are successively examined to conduct model elicitation from textual scenarios for the purpose of conducting model-driven engineering of user interfaces: manual classification, dictionary-based classification, and nearly natural language understanding based on semantic tagging and chunk extraction. A model elicitation tool implementing these three levels is described and exemplified on a real-world case study for designing user interfaces to workflow information systems. The model elicitation process discussed in the case study involves several models: user, task, domain, organization, resources, and job.

Keywords: Model-driven engineering, requirements gathering, user interface development method, user task elicitation, workflow information systems.

1. Introduction

In recent years, there has been a lot of interest for scenario-based design (Rosson, 1997) and other forms of User-Centred Design (UCD) (Paterno, 1999) to initiate a development life cycle of User Interfaces (UI). Textual scenarios found in scenario-based design consist of informal but structured narrative descriptions of interaction sequences between the users and the interactive system, whether this system exists already or is simply envisioned. Scenarios have been proved (Rosson, 1997) to be a valuable tool to elicit, improve, and validate UI requirements.

Please use the following format when citing this chapter:

Lemaigre, C., García, J.G. and Vanderdonckt, J., 2008, in IFIP International Federation for Information Processing, Volume 272; *Human-Computer Interaction Symposium*; Peter Forbrig, Fabio Paternò, Annelise Mark Pejtersen; (Boston: Springer), pp. 53–66.

On the other hand, descriptions of the UI domain itself and the UI requirements are also expressed using conceptual models depicting either static (Tam, 1998) or dynamic (Fliedl, 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, 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, 1998).

Scenarios have the advantage to describe UI requirements from captured or imagined user interactions through concrete examples [8] of the user carrying out her task. This 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 RE process that will effectively and efficiently feed the rest of the UI development life cycle. We introduce *model elicitation* as the general activity of transforming textual scenarios into models that are pertaining to the UI development.

The remainder of this paper 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 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 development life cycle (Hemmecke, 2006), for instance by conducting a model-driven engineering method (Brasser, 2002; Vanderdonckt, 2005). *Model verbalization* (Jarrar, 2006) is the inverse process: it consists of transforming model elements into textual scenarios while preserving some quality properties (e.g., concision, consistency). Any model can be considered for this purpose: models found in HCI (e.g., task, user) or in RE (e.g., domain, organization). In (Bono, 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 (SE) (Fliedl, 2004, 2005b), but at least five significant works have been conducted in Human-Computer Interaction (HCI):

1. U-Tel (Tam, 1998) 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. ConcurTaskTrees Editor (Paterno, 1999) contains a module for task elicitation where designers copy task names found in a textual scenario and paste them in a graphical editor for representing a task model. Designers can then refine the task model, e.g., by specifying a task type, temporal relationships between tasks.
3. Similarly, T2T (Paris, 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,1998) and for multiple heterogeneous sources (Lu, 2002).
4. 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.
5. Brasser & vander Linden (Brasser, 2002) developed a task elicitation system for the Isolde task modeling environment: based on a 25-state Augmented Transition Network (ATN) derived from written narratives, this system extracts two kinds of information: domain information (i.e., actors and objects) and procedural information (e.g., "when the user saves a file,…")

From these works, we observed the following shortcomings: some do not produce a genuine model at the end, for instance (Garland, 2001), some other produce model elements that are relevant to HCI, for instance (Lu, 1998; Paris, 2002), but 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, 1998), only the names of the classes are captured. In this paper, we would like to capture all elements (i.e., classes, attributes, and relationships) of many interrelated models to inform the development. It is however fundamental that the task model is considered to initiate a full model-driven engineering life cycle (Clerckx, 2006; Paterno, 1999). Dynamo-AID (Clerckx, 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.

The three levels of model elicitation presented in this paper, i.e., manual classification, dictionary-base classification, and nearly-natural language classification, are presented in this order only for structuring purposes. This does not mean that the elicitation process should be conducted in that order. Indeed, one may desire eliciting model elements in a mostly automated way, then refine the classification manually. Or one may prefer first designating the most important model elements if they do not fit well from the identified ontology and then apply more automated techniques in order to propagate these manual classifications.

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 paper: 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, 2008).

We choose this ontology because it characterises 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 (Paterno, 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.

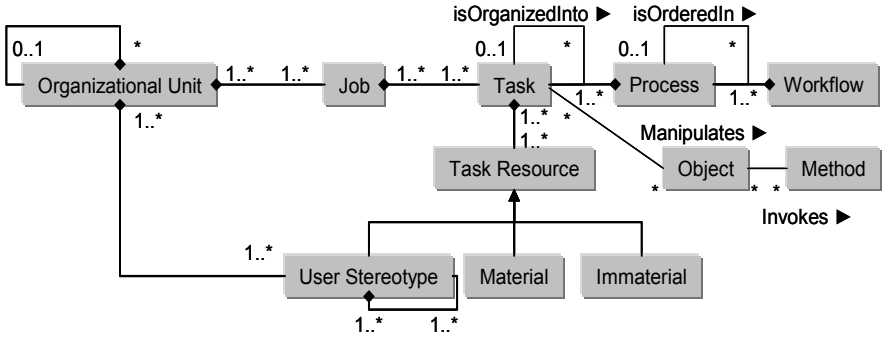

Figure 1. Simplified ontology of the model elements (Guerrero, 2008).

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. The ordering of the three classification levels in the text is just a way to structure the article. Not an order the program user would comply in order to get a result.

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 parameterized 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 in Figure 2 and is consequently assigned to the corresponding hierarchy in the related tab. Since a model element may appear in the scenario in multiple alternative forms (e.g., 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".

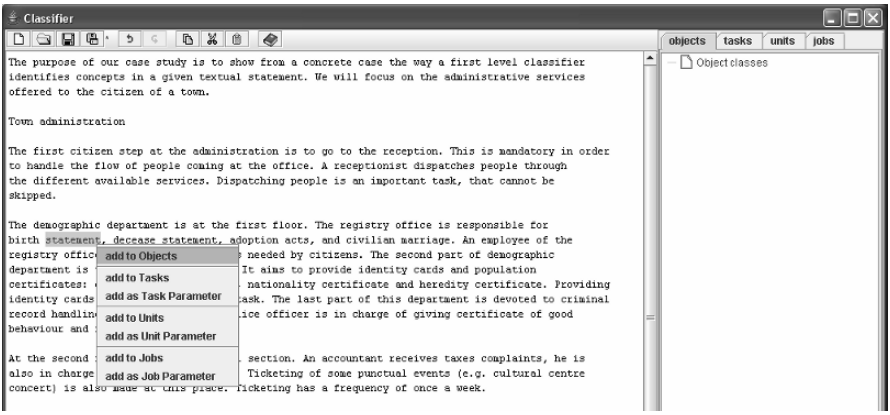

Figure 2**.** Elicitation of a class (here, an object) in manual classification.

In UCD, tasks, users, and objects are often considered as elements of primary interest. Therefore, it is likely that a designer 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 task type which belongs to one of the three following task types (Figure 4):

- A *predefined task type*: a taxonomy of task types (e.g., transmit, 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 (Vanderdonckt, 2005). This taxonomy has been established by relying on the Grounded Theory (Strauss, 1997), which means that it has been developed inductively from examining a corpus of data. In order to obtain this corpus, we have examined over time a series of interactive information systems and categorized the found task definitions into a corpus of task types that have been updated according to systematic deciphering scheme. Each predefined task type comes with a precise definition and scope, some synonyms, if any, and its decomposition into sub-tasks, if any. This taxonomy could be edited, e.g., by introducing some new task types, or by adding new synonyms to already existing task types.
- A *custom task name*: any non-predefined task name can be entered, such as "issuing" in Figure 3. In this way, any new task type that does not belong to the taxonomy may be introduced, such as task types for a particular domain of human activity. The custom task name is introduced mainly for specific case studies where one does not want necessarily to introduce a new task type in the taxonomy, for instance, in order to avoid deviations from these types.
- A *pattern of tasks*: a pattern of tasks is hereby referred to as any set of predefined task types and/or of custom task names. This should not be confused with a task pattern which is a pattern for task models. A pattern of tasks is aimed at gathering into one pattern a set of tasks that are typically, frequently carried out on an object. Instead of redefining every such task for an object, the pattern could be applied to the object, thus redefining the different tasks for this particular object. 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), one of the most frequently applied patterns in SE, will automatically enter four predefined task types for a designated object and specialize them for this objects in order to avoid ambiguity.

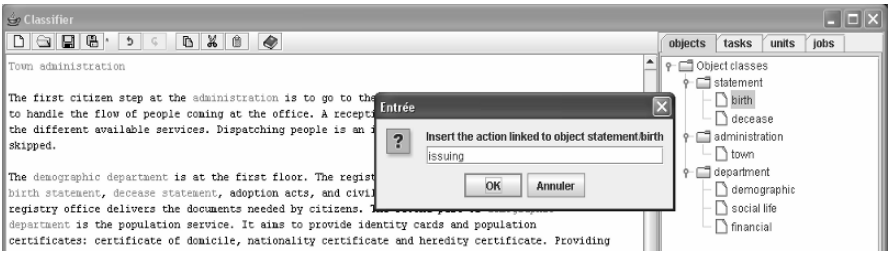

Figure 3. Assigning a task to a already defined object.

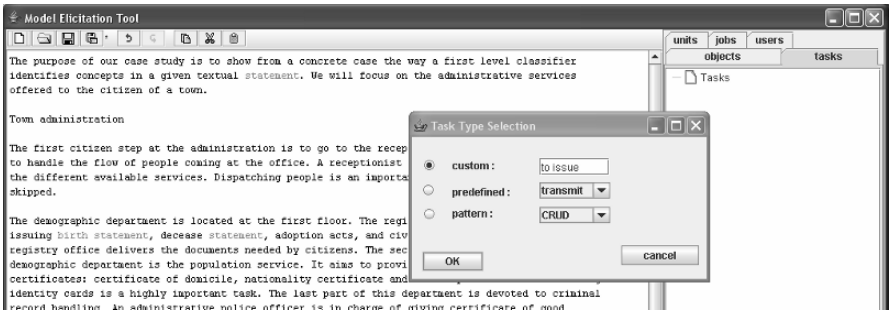

Figure 4**.** Introduction of various task types for a task model being elicited.

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 5, 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.

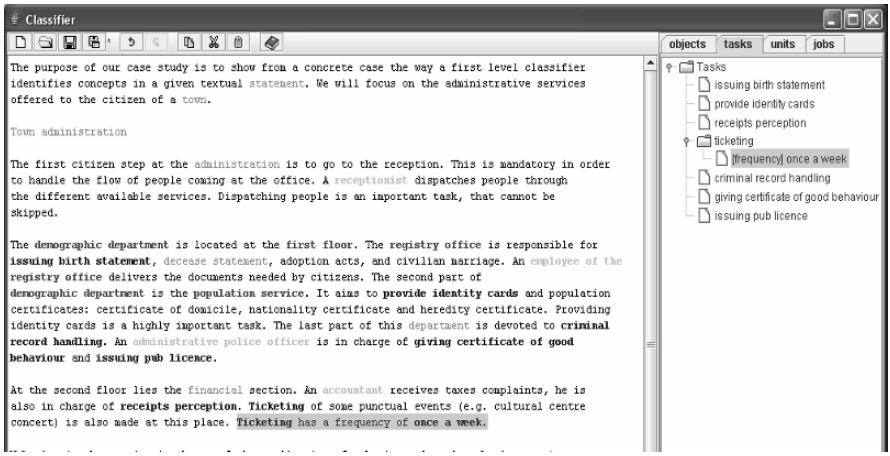

Figure 5**.** Elicitation of a predefined attribute for a task.

Figure 6 graphically depicts the three main steps for entering a custom name for an attribute, here an organizational unit. The procedure is similar for any other type of model attribute. 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 6a), it is identified with a unique name (Figure 6b), and then included in the hierarchy (Figure 6c). Its value is then entered in the model as well. There is no underlying definition of data types supporting this action since it is considered rather informal at this stage of the development life cycle.

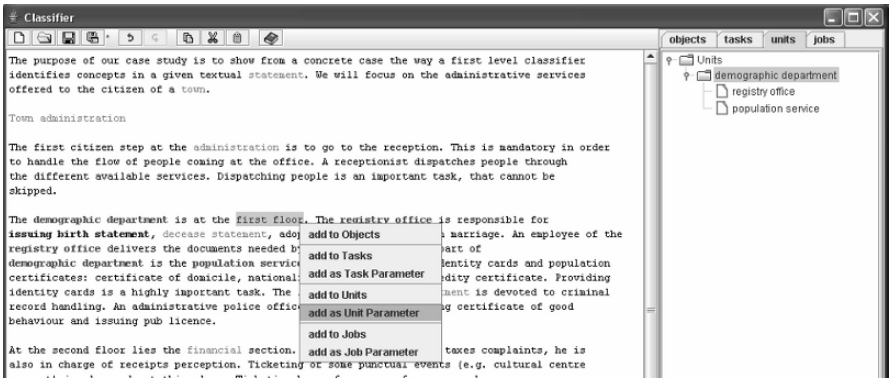

Figure 6a**.** Elicitation of a custom attribute for an organizational unit: selection.

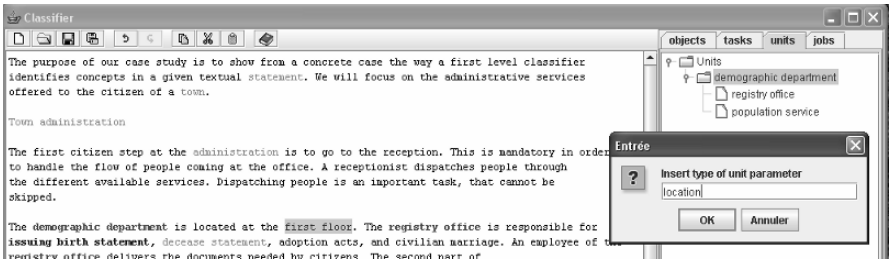

Figure 6b**.** Elicitation of a custom attribute for an organizational unit: identification.

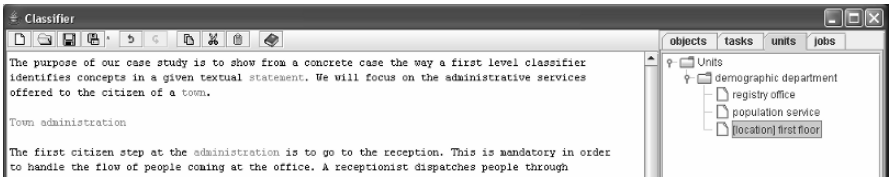

Figure 6c**.** Elicitation of a custom attribute for an organizational unit: inclusion.

Elicitation of a relationship. By using drag and drop, the designer can arrange model elements in their corresponding hierarchy in order to reflect the decomposition relationships of Figure 1. For example, a task is decomposed into sub-tasks, tasks are composed in a process, and processes are composed into a workflow. Or in the other way around, a workflow is decomposed into processes (e.g., business processes), which are in turn decomposed into tasks, to end up with sub-tasks. 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 subclasses "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 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, 1998):

1. *Positive filters* force some model terms to be considered anyway, whatever the domain or the contexts 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"), etc.

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 third level has been defined, which is the object of the next sub-section.

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, 2003), semantic tagging and chunk parsing (Fliedl, 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, 2004, 2005b) in order to recognize possible terms that express, depict, reveal

model elements. For instance, a 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. In order 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. The parsing decides when to break any textual fragment (e.g., a sentence, a series of propositions that form a sentence) into separate model elements using both textual (e.g., periods, commas, colons, semi-colons) and lexical (e.g., "and", "or", "by", "to") cues.

Table 1. Possible terms incorporated in taggable expressions for a task model.

Concept	**Possible terms**
Frequency	Every day, daily, day by day, day after day, every Monday, every week, weekly, monthly, each month, each year, yearly, two (#) times a day, each hour, two (#) times per month (day, year, hour), two (#) days (weeks, months) a week (month, year), occasionally, from time to time, every other day, on alternate days, each two (#) days (weeks, months, years, hours)
Importance	Very important, low, high, regular
Structuration level	Low, high, regular
Complexity level	Low, high, regular, trivial, very complex, simple
Criticity	Low, regular, high, very critic
Centrality	Low, high, regular, very central, minor, peripheral
Termination value	End, finish, last, final, finally, lastly, endings
Task type	Communicate, convey, transmit, call, acknowledge, respond/answer, suggest, direct, instruct, request, create, input/encode/enter associate, name, group, introduce, insert, (new), assemble, aggregate, overlay (cover), add, delete, eliminate, remove/cut, ungroup, disassociate, duplicate, clone, twin, reproduce, copy, filter, segregate, set aside, mediate, analyze, synthesize, compare, evaluate, decide, modify, change alter, transform, turning, rename, segregate, resize, move, relocate, navigation, Go/To, perceive, acquire/detect/search for/scan/extract, identify / discriminate / recognize, locate, examine, monitor, scan, detect, Reinitialize, wipe out, clear, erase, select/choose, pick, start, initiate/trigger, play, search, active, execute, function, record, purchase, stop, end, finish, exit, suspend, complete, terminate, cancel, toggle, activate, deactivate, switch
Task item	Collection, container, element, operation

Table 2. Possible terms incorporated in taggable expressions for a task relationship.

Relationship	Possible terms
Sequence	Then…after; following; first, second…
Parallel	And; at same time; in any order; in parallel; jointly; concurrently
Conditional	If… then … else; either… or; in case of … otherwise
Iterative	Each time
Suspend / resume	Stop; suspend; discontinue; cease

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; these values are the result of the exploration of existing literature. This pattern matching can be executed automatically or under the control of the designer 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. This XSD file can be downloaded from http://www.usixml.org/ index.php?mod=pages&id=5.

3.4 After Model Elicitation

The main goal of model elicitation is then to handle the textual statement from the beginning to the end and to ensure that all textual fragments that should be transformed into model elements are indeed elicited. In particular, the graphical highlighting in colours allows designers to quickly identify to which model type the element is relevant and to check in the end that the complete scenario has been exhausted, that no term remains unconsidered. In this way, they can check whether main model properties are addressed in an informal way, such as, but not limited to those model properties that are summarized in Table 3.

It seems of course impossible to automatically check these model properties at this level since only textual fragments are considered, even if they are linked with the ontology. However, this table may serve as a check list to ensure that the resulting models are the least incomplete, inconsistent, incorrect, etc. as possible.

After performing the elicitation of model elements according to any of the three aforementioned techniques, the model elicitation tool can export at any time the results in UsiXML files for the whole set of models or for any particular combination (e.g., only the tasks with the users or only the tasks with their related objects).

This file can then be imported in any other UsiXML-compliant editor in order to proceed with the remainder of the development life cycle. Several tools are candidates for this purpose (Vanderdonckt, 2005):

- IdealXML enables designers to graphically edit respectively the task and the domain models, in particular to automatically generate an Abstract UI.
- FlowiXML enables designers to edit the task, job, and organizational unit models in order to proceed with user interfaces for workflow information systems. Per se, it does not edit the domain model however. It is mainly targeted towards editing models that are involved in workflow information systems.
- Any general-purpose tool for applying model-to-model or model-to-code transformations, in particular any software that supports solving the mapping problem [20] between various models.

Table 3**.** Desirable quality properties of a model.

Property	Definition
Completeness	Ability of the model to abstract *all* real world aspects of interest via appropriate concepts and relations
Graphical completeness	Ability of the model to represent *all* real world aspects of interest via appropriate graphical representation of the concepts and relations
Consistency	Ability of the model to produce an abstraction in a way that reproduces the behaviour of the real world aspect of interest in the same way throughout the model and that preserves this behaviour throughout any manipulation of the model.
Correction	Ability of the model to produce an abstraction in a way that correctly reproduces the behaviour of the real world aspect of interest
Expressiveness	Ability of the model to express via an abstraction any real world aspect of interest
Concision	Ability of the model to produce concise, compact abstractions to abstract real world aspects of interest
Separability	Ability of models to univocally classify any abstraction of a real world aspect of interest into one single model (based on the principle of *Separation of Concerns*)
Correlability	Ability of models to univocally and unambiguously establish relations between models to represent a real world aspect of interest
Integrability	Ability of models to concentrate and integrate abstractions of real world aspects of interest into a single model or a small list of them.

4. Conclusion

In this paper, we have investigated three different techniques for eliciting model elements from fragments found in a textual scenario in order to support activities of scenario-based design. 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. These three levels can be used in combination. Beyond

the automated classification of terms into the respective models that are compatible with the ontology, the model elicitation tool provides editing facilities within a same model and across models of this ontology. In order to support other models or other variations of the same model (e.g., a different task model or more attributes for the same task model), one may need to incorporate these definitions in the ontology. As empirical validation is an important component in understanding the capacity and limitations of the model elicitation tool, a series of case studies has been developed.

Its main advantage relies in its capability of supporting designers in identifying text fragments that should be considered for model elicitation and in helping them to informally check some desirable model properties.

Its main drawback today is the lack of graphical visualisation of inter-model relationships or intra-model relationships others than merely decomposition relationships (represented implicitly in the respective hierarchies). Advanced visualisation techniques, such as carrousel visualisation, may be considered. For the moment, these relationships are only collected as an entry 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 to semantic pattern matching.

Acknowledgments. We gratefully acknowledge the support of the SIMILAR network of excellence (http://www.similar.cc), the European research task force creating human-machine interfaces similar to human-human communication of the European Sixth Framework Programme (FP6-2002-IST1-507609) and the CONACYT program (www.conacyt.mx) supported by the Mexican government. We also thank the anonymous reviewers for their constructive comments.

References

Bono, G., and Ficorelli, P.: Natural Language Restatement of Queries Expressed in a Graphical Language. In: Proc. of the 11th Int. Conf. on the Entity-Relationship Approach ERA'92 (Karlsruhe, October 7-9, 1992), Lecture Notes in Computer Science, Vol. 645, pp. 357-373, Springer, Heidelberg (1992).

Brasser, M., and Vander Linden, K.: Automatically Elicitating Task Models from Written Task Narratives. In: Proc. of the 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002 (Valenciennes, 15-17 May 2002), pp. 83-90, Kluwer Academic Publishers, Dordrecht (2002).

Clerckx, T., Vandervelpen, Ch., Luyten, K., and Coninx, K.: A Task Driven User Interface Architecture for Ambient Intelligent Environments. In: Proc. of 10th ACM Int. Conf. on Intelligent User Interfaces IUI'2006 (Sydney, January 29-February 1, 2006), pp. 309-311, ACM Press, New York (2006).

Fliedl, G., Kop, Ch., Mayr, H., Hölbling, M., Horn, Th., Weber, G., and Winkler, Ch.: Extended Tagging and Interpretation Tools for Mapping Requirements Texts to Conceptual (Predesign) Models. In: Proc. of 10th Int. Conf. on Applications of Natural Language to Information Systems NLDB'2005 (Alicante, June 15-17, 2005), Lecture Notes in Computer Science, Vol. 3513, pp. 173-180, Springer, Heidelberg (2005).

Fliedl, G., Kop, Ch., and Mayr, H.: From textual scenarios to a conceptual schema. Data Knowledge Engineering, 55(1), 20-37 (2005).

Fliedl, G., Kop, Ch., Mayr, H., Winkler, Ch., Weber, G., and Salbrechter, A.: Semantic Tagging and Chunk-Parsing in Dynamic Modeling. In: Proc. of 9th Int. Conf. on Applications of Natural Languages to Information Systems NLDB'2004 (Salford, June 23-25, 2004), Lecture Notes in Computer Science, Vol. 3136, pp. 421-426, Springer, Heidelberg (2004).

Fliedl, G., Kop, C., and Mayr, H.: From Scenarios to KCPM Dynamic Schemas: Aspects of Automatic Mapping. In: Proc. of 8th Int. Conf. on Applications of Natural Language to Information Systems NLDB'2003 (Burg, June 2003), Lecture Notes in Informatics, Vol. 29, pp. 91-105, Gesellschaft für Informatik, Bonn (2003).

Garland, A., Ryall, K., and Rich, Ch.: Learning hierarchical task models by defining and refining examples. In: Proc. of the 1st Int. Conf. on Knowledge Capture K-CAP'2001 (Victoria, October 21-23, 2001), pp. 44-51, ACM Press, New York (2001).

Guerrero, J., and Vanderdonckt, J.: FlowiXML: a Step towards Designing Workflow Management Systems, Journal of Web Engineering, 4(2), 163-182 (2008).

Haumer, P., Pohl, K., and Weidenhaupt, K.: Requirements Elicitation and Validation with Real World Scenes, IEEE Transactions on Software Engineering, 24(12), 1036-1054 (1998).

Hemmecke, J., and Stary, Ch.: The Tacit Dimension of User Tasks: Elicitation and Contextual Representation. In: Proc. of 5th Int. Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2006 (Hasselt, October 23-24, 2006), Lecture Notes in Comp. Science, Vol. 4385, pp. 308-323, Springer, Heidelberg (2006).

Jarrar, M., Keet, M., and Dongilli, P.: Multilingual verbalization of ORM conceptual models and axiomatized ontologies. Technical report. STARLab. (Available via Vrije Universiteit, 2006). http://www.starlab.vub.ac.be/staff/mustafa/publications/[JKD06a].pdf. Accessed 14 April 2008.

Lu, S., Paris, C., and Vander Linden, K.: Computer Aided Task Model Acquisition From Heterogeneous Sources. In: D. Guozhong (Ed.), Proc. of 5th Asia Pacific Conference on Computer Human Interaction APCHI'2002 (Beijing, November 1-4, 2002), pp. 878-886, Science Press, Beijing (2002).

Lu, S., Paris, C., and Vander Linden, K.: Toward the Automatic Construction of Task Models from Object-Oriented Diagrams. In: Proc. of the IFIP TC2/TC13 WG2.7/ WG13.4 7th Working Conf. on Engineering for Human-Computer Interaction EHCI'98 (Heraklion, September 14-18, 1998), pp. 169-189, IFIP Conference Proceedings, Kluwer (1999).

Paris, C., and Vander Linden, K., Lu, S.: Automated knowledge acquisition for instructional text generation. In: Proc. of the 20th Annual Int. Conf. on Computer documentation SIGDOC'2002 (Toronto, October 20-23, 2002), pp. 142-151, ACM Press, New York (2002).

Paterno, F., and Mancini, C.: Developing task models from informal scenarios. In: Proc. of ACM Conf. on Human Aspects in Computing Systems CHI'99 (Pittsburgh, May 15-20, 1999), ACM Press, New York (1999).

Rosson, M.B., Carroll, J. M.: Scenario-based Design. In: A. Sears, J.A. Jacko (Eds.), The human-computer interaction handbook: fundamentals, evolving technologies, and emerging applications, CRC Press (2007).

Strauss A.L., Corbin, J.: Grounded Theory in Practice, Sage, London (1997).

Tam, R., Maulsby, D., Puerta, A.: U-TEL: A Tool for Eliciting User Task Models from Domain Experts. In: Proc. of ACM Int. Conf. on Intelligent User Interfaces IUI'1998 (San Francisco, January 6-9, 1998), pp. 77-80, ACM Press, New York (1998).

Vanderdonckt, J.: A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In: Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05 (Porto, June 13-17, 2005), Lecture Notes in Computer Science, Vol. 3520, pp. 16-31, Springer, Heidelberg (2005).