Task Modelling for Context-Sensitive User Interfaces

Costin Pribeanu[1,2], Quentin Limbourg[1], and Jean Vanderdonckt[1]

[1] Université catholique de Louvain, Institut d'Administration et de Gestion
Place des Doyens, 1 - B-1348 Louvain-la-Neuve, Belgium
limbourg, vanderdonckt@isys.ucl.ac.be
[2] National Institute for Research and Development in Informatics
Bd Averescu 8-10 - R-71316 Bucharest, Romania
pribeanu@acm.org

Abstract. With the explosion of devices, computing platforms, contextual conditions, user interfaces become more confronted to a need to be adapted to multiple configurations of the context of use. In the past, many techniques were developed to perform a task analysis for obtaining a single user interface that is adapted for a single context of use. As this user interface may become unusable for other contexts of use, there emerges a need for modelling tasks which can be supported in multiple contexts of use, considering multiple combinations of the contextual conditions. For this purpose, the concept of unit task is exploited to identify a point where traditional task models can break into two parts: a context-insensitive part and a context-sensitive part. A widespread task model notation is then used to examine, discuss, and criticise possible configurations for modelling a context-sensitive task as a whole. One particular form is selected that attempts to achieve a clear separation of concern between the context-insensitive part, the context-sensitive part, and a new decision tree which branches to context-sensitive tasks, depending on contextual conditions. The questions of factoring out possible elements that are common across multiple contexts of use and representation of the resulting task model are discussed.

1 Introduction

User interfaces (UIs) of today's applications seem no longer condemned to be executed in a single fixed context of use. Indeed, a multiplication of possible contexts of use is provoked by a new heterogeneity of computing platforms, each of them with its own device capabilities that are always posing new constraints. This heterogeneousness results in computing platforms exhibiting drastically different interaction capabilities. Coping with these differences implies reconfigurations of the UI that are beyond traditional UI change [6, 7], such as widget resizing, reduction of a full widget to its scrollable version, graceful degradation of a sophisticated widget into a moderate one, or redistribution of widgets across windows. Beyond this classic problem of having cross-computing platform UIs is appearing a new collection of constraints that are no longer imposed by the

computing platforms themselves, but rather by other factors, such as the type of user, the external resources manipulated by the UI (e.g., the network, the ambient environment).

For instance, we studied a registration procedure which should be performed in two completely different contexts of use. In the first context, a secretary registers a child for a leisure activity on her personal computer in a quiet environment, with all accessible facilities. In the second context, she is equipped with a PalmPilot and should register many children (at least, 50) arriving with their parents in a very short amount of time (typically 20 minutes). This atmosphere is very tense because she cannot stop the children for a long time nor she want to catch the parents to collect all required information. Moreover, children are running everywhere in the entrance, making the context of use a very stressed one. For these reasons, the interactive task is reduced to its minimum vital. UIs are consequently expect to gracefully evolve in a rather changing environment where several changes of the context of use may occur. And these changes may endanger the predicted usability of a predefined UI [1, 16, 18].

The *context of use* is hereby defined as the complete environment in which the user is carrying out an interactive task to fulfil a given role played in a specific organisation. Two types of characteristics simultaneously determine the context of use:

1. Characteristics *internal* to the application and its UI (e.g., the computing platform, the software/hardware parameters, the interaction devices, the network bandwidth, the latency, the screen resolution).
2. Characteristics *external* to this technical system (e.g., the type of user, her skills and knowledge, her preferences, the sound and light conditions, her geographic position in a building, the stress level, the organisation structure, the information channels).

Any change of at least one of these characteristics may generate one possible change of the context of use. Of course, a combination of changes on multiple characteristics also impose a further complicated change of the context of use. The challenge for the UI and for the designer is to match the UI configuration (e.g., look and feel) with the set of constraints imposed by the new characteristics of the resulting context of use. *Context-sensitivity* is hereby referred to as the ability of a UI to execute reconfiguration after a context of use variation so as to stay adapted to the new context. The importance of designing UIs for multiple contexts of use has already been stressed and studied in some work, such as, but not limited to, [2, 4, 1, 7]. In particular, it is underlined in [10] that a return to considering model-based approaches to address the issues of context-sensitivity might be expected, due to the wide variety of contexts of use. Indeed, it seems critical that models should allow an elegant way to segment different aspects of UI design that are relevant to different contexts of use and to isolate context-generic issues from context-specific ones.

In model-based approaches [3, 8, 11, 16], the task model is typically used for a user-task elicitation process that results into a task model and a user model,

possibly along with a domain model. At least, the task model is recognised as one fundamental point to initiate a UI design process which is user-centred.

In this paper, we would like to address the issues raised by task modelling for context-sensitive UIs. Section 2 reviews some efforts undertaken towards modelling context-sensitivity in task models and examines how these efforts can be situated on a four-steps method for designing context-sensitive UIs. To initiate this method, Section 3 analyses some possible approaches for modelling a task for context-sensitive UIs, based on a selected task model and associated notation. The resulting model is then exploited in Section 4 to highlight various model configurations for a varying amount of context-sensitive sub-tasks and for a single or multiple contexts of use. The problem of factoring out common elements in such models is then studied in Section 5. A case study is provided in Section 6. A conclusion explains the main benefits of this approach and how this model is currently being used for tool support and the rest of the method.

2 Designing Context-Sensitive User Interfaces

Based on [4, 9, 16], four major steps for producing a context-sensitive UI may be identified (Fig. 1):

1. Production of a context-sensitive task model: this task model should foresee all sub-tasks that might be considered in a single context of use or in multiple contexts of use. This activity may involve adding sub-tasks that are especially supported in a particular context of use and withdrawing of sub-tasks that are not supported in a particular context of use. This task model should obey to the separation of concern principle : the context-insensitive part should be clearly separated from the context-sensitive part, while showing relationship between them.
2. Production of a generic UI model: this generic UI model is supposed to model a UI independently of any context of use, thus considering the context-insensitive part of the context-sensitive task model.
3. Production of a specific UI model: this specific UI model is supposed to model a UI depending on the constraints of intended contexts of use, thus considering the context-sensitive part of the context-sensitive task model. Due to variation of these constraints, multiple UI models may be produced. In [4], one model covers as many contexts of use as possible, provided that the usability properties are preserved and that a same UI can hold the expected properties simultaneously.
4. Production of a final running UI: the previous model can then be exploited at design time to automatically generate code (e.g., in HTML with CSS cascading style sheets for Web-based contexts, WML for cellular phone contexts, in XML with XSL eXtensible Style Sheets for XML-compliant browsers), for a particular con-text of use or be interpreted at run-time to produce the expected UI.

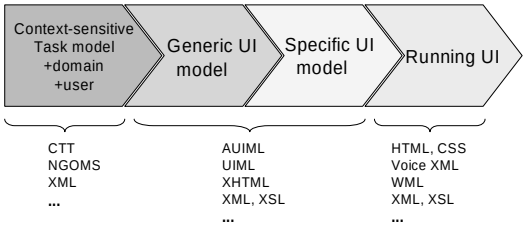

Fig. 1. Steps of the method for designing context-sensitive user interfaces

Fig. 1 highlights that some approaches or languages are more targeted at some steps of this method. For example, UIML [2] does not encompass any task model, but precisely supports a smooth transition from a platform-independent UI model to a platform-dependent UI model, which is in turn converted into code of a programming language of the target computing platform.

To locate points in the task model where a separation points between the context-insensitive part and the context-sensitive part of the task can be specified, we have to look back to the concept of basic task and unit task. The question is therefore: what is the most useful base unit for task decomposition between the two parts? We suggest to answer this question by relying upon the concept of *unit task*.

Although identification of the elementary task is an important concern for over a decade there is not a generally agreed definition yet. Card, Moran Newell [5] defined the unit task as the lowest level task a user really wants to perform. Later on, Tauber [17] defined the basic task as a task for which the system provides with one single command or unit of delegation. According to van Welie, van der Veer and Eliens [19], a unit task should only be executed by performing one or more basic tasks. The relationship between unit tasks and basic tasks is important because it shows problems a user may experience when she tries to accomplish her goals with a given UI. However, they give no indication how this task decomposition should be done.

In the ConcurTaskTree (CTT) notation elaborated by Paternó [12, 13, 11], basic tasks are defined as tasks that could not be further decomposed. Basic tasks are classified in three types: user tasks, application tasks and interaction tasks. This classification extends the definition of Tauber to other tasks than those related to a single function provided by the system. In order to design the user interface he starts from the specification of the so-called User's Virtual Machine (UVM) which is the conceptual model of a "competent user", i.e. a user knowing how to delegate a task to the computer. The specification of basic tasks is done in terms of conceptual events (commands), objects, states, attributes, values and places, each being an entry in UVM.

This definition of Tauber relates basic tasks to UI objects but it leaves outside the specification of user tasks that are not related to UI objects. Basic tasks as defined by Tauber roughly correspond to interaction tasks in CTT. Basic

task classification proposed by Paternó is more complete since it comprises also user tasks and application tasks that reflect cognitive activities of the user. We therefore consider that unit tasks are potential candidates to separate a task model between the context-insensitive part (which stops a unit tasks which are device- or context-independent) and the context-sensitive part (which ends up with basic tasks which are device- or context-dependent).

In [15], a method is developed to support task models that address multiple types of user, which is one form of multiple contexts of use. This method is based on polymorphic tasks. The main advantage of this method is its ability to incorporate style variations depending on user types into a single model, rather than having different models, possibly inconsistent.

In the next section, possible approaches for modelling a context-sensitive task that could initiate any instance of the above four-steps method are analysed.

3 Possible Approaches for Modelling Context-Sensitive Tasks

A *context-sensitive task* is hereby defined as any task that may require at least one context of use switching for its performance. After a user has performed a particular sub-task in a given context, some other sub-tasks, or possibly task units located deeper in the task hierarchy, may require that the user changes the current context of use in which the task is carried out to another one to complete the main task. Or the varying context of use itself imposes a change to the user. It is not necessary that all sub-tasks to be context-sensitive for a task to be context-sensitive itself. Rather, any task unit that require context of use switching implies that the encompassing task is context-sensitive. In addition, some task unit may be particular to only one context of use, some other, to multiple contexts of use, which are perhaps similar in nature. It could even be imagined that the context-sensitivity is intrinsically part of carrying out the global task.

To model a context-sensitive task, we can start from scratch and invent a totally new way of modelling tasks to take into account their context-sensitivity. We preferred to rely upon an existing task modelling technique that has received attention, theoretical soundness, and experimental studies enough to warrant its selection.

The ConcurTaskTree notation [12] has been selected for these reasons and because a ConcurTaskTree editor already exists [11]. Based on previous work [13] for expanding the original CTT notation to support co-operative tasks, there are at least three possible approaches to extend CTT to explicitly modelling context-sensitive tasks: a monolithic approach, a graph oriented approach, and an additional context-sensitive part.

The *monolithic approach* consists of drawing a global task model that directly encompass context insensitive parts and context-sensitive parts. In these parts, we can find every level of the task decomposition into sub-tasks and so forth. Indeed, for all classic task models represented as a tree decomposition, the

parent-child relationship is exploited. The question is therefore where to cut the tree to show a clear separation between context-insensitive and context-sensitive task units. As the context of use may occur wherever in the task accomplishment, this separation may be located everywhere in the task tree. A distinction can be drawn on the tree itself: dotted lines surround context-sensitive parts to make them distinguishable from the rest in a task tree (Fig. 2). For the simplicity, only the parent-child relationship is drawn in this task model: temporal relationships have been omitted temporarily.

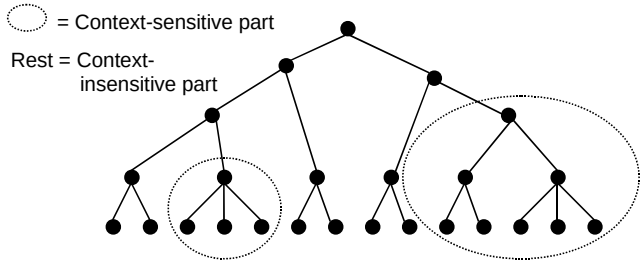

Fig. 2. The monolithic approach for modelling a context-sensitive task

The main advantage of this approach is the unicity of the task model as everything is modelled into a single model. However, within context-sensitive parts, it is hard to differentiate nodes branching to sub-trees to reflect possible changes of context from nodes indicating task units to carry out after the context of use changed. There are consequently no visual cue identifying changes of context. The *graph oriented approach* attempts to address this shortcoming by extracting sub-trees resulting from the context-sensitive part into separate trees that can be related where needed. It consists of establishing a relationship to all sub-trees describing task units resulting from changes of context (Fig. 3).

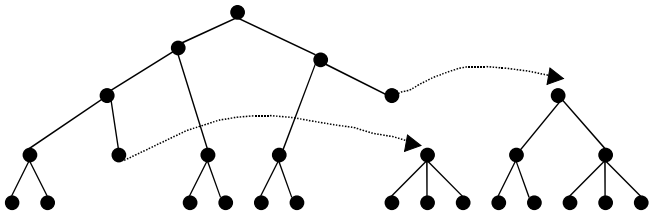

Fig. 3. The graph oriented approach for modelling a context-sensitive task

The biggest advantage of this approach relies in its distinction between non-context-sensitive and context-sensitive parts. However, the resulting model looses

its hierarchical structure to become a directed graph. This approach is introducing additional relationships that may increase model cluttering. According to [13], this kind of graph is also harder to interpret and to manipulate by a tool. Conversely, no representation of how and when the context of use may change in the task model is given. The *context-sensitive separation approach* attempts to solve this problem by recognising that the context-sensitive part actually holds two types of arcs and nodes:

1. Traditional arcs and nodes as one can find in a classic task tree: these nodes represent the task units at their various levels of decomposition and the arcs express their parent-child relationship.
2. Decision arcs and nodes so as to select the particular sub-tasks to carry out, depending on conditions expressed on properties of the context of use. These properties are called *contextual properties*, such as, for example, the type of computing platform, the screen resolution, and the network bandwidth. Conditions on these contextual properties are called in turn *contextual conditions*. The chaining of contextual conditions form a decision tree that properly branches to the appropriate task units, depending on the context of use selected.

The context-sensitive separation approach therefore consists in drawing (Fig. 4):

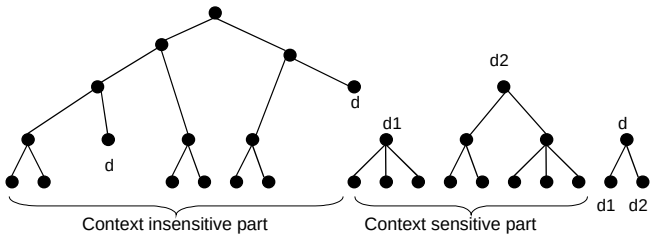

Fig. 4. The context-sensitive separation approach for modelling a context-sensitive task

- A context-insensitive tree containing the decomposition into tasks units that do not change if the context of use changes which ends up with decision points, where to branch to a decision tree for considering right context (*d* points in Fig. 4).
- A series of context-sensitive trees modelling task units for all supported contexts of use (middle in Fig. 4).
- A decision tree (right part of Fig. 4) summarising all contextual conditions. It is expected that each branch is labeled with one contextual condition at a time to avoid multiple logical formula and that all branches appearing at

a same level should form a partition of all possible values of a same contextual condition. For instance, a contextual condition could be "What is the currently used computing platform?", while the possible values could be "PC, Macintosh, WebTV". The decision tree could de detailed as needed with several levels of contextual conditions, and not only one, but one contextual condition is considered at a time. The leaves (d_1 and d_2 here) of this tree should branch to any context-sensitive tree (to the tree having d_1, respectively d_2, as root).

This approach makes more clear the distinction between the context-insensitive part, the context sensitive part, and the decision tree making the link between those parts. However, it is likely that different contexts of use may be considered similarly at different locations of the context-insensitive part. There is consequently no way to factor out the common parts of the context-sensitive part, depending on the selected context of use.

The *complete separation approach* is aimed at factoring out similar sub-trees which are possibly used at different locations in the global task tree, thus minimising duplication of these context-sensitive sub-trees. It consists of identifying (Fig. 5):

- A non-context sensitive part, which similarly produced by the context-sensitive separation approach.
- A series of separate decision trees representing branching to all the possible contexts of use with decision points as root (middle in Fig. 5).
- A series of sub-trees of factored task units to carry out when needed, one sub-tree for each considered value of the final contextual condition (dotted part right in Fig. 5). The difference with respect to the previous approach is that this context-sensitive part consists of a collection of sub-trees that may be called from different places in the decision tree. Therefore, each such sub-tree is unique.

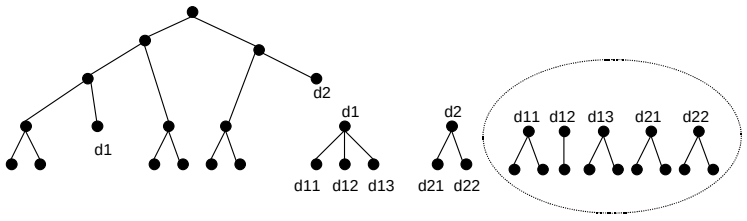

Fig. 5. The complete separation approach for modelling a context-sensitive task

The advantages of this complete separation approach are the following: everything is more obviously separated than in the context-sensitive separation approach (one context-insensitive part, one decision part, and one context-sensitive

part, all consisting of unique sub-trees); decision trees can be edited and maintained independently of each other, with their own logic (the context-sensitive and context-insensitive parts are task oriented while the other is context-detection oriented). The major inconvenience of this approach is that it dramatically suffers from a proliferation of trees, even if unique trees, with no direct linking between them. Moreover, maintaining such series of trees in a repository becomes quite complex, especially for future algorithms for UI derivation, and the modelling becomes very different from the initial task model as noted in CTT. For these reasons, we came to the conclusion that a better solution could be set up by combining the unique representation of the monolithic approach with decision tree and clear separation as discussed in the context-sensitive separation approach to produce a model that is closer to the initial CTT notation. The final solution consists of (Fig. 6):

1. A context-insensitive part as typically modelled in a CTT task model.
2. A decision tree represented by three CTT notation elements: the different contexts of use are linked sequentially as optional sub-tasks at the first level with the sequencing operator, then the different contextual conditions are expressed as sub-sub-tasks with choice operator (mutual exclusion), and finally the leaves of the decision trees are the roots for the context-sensitive part
3. A context-sensitive part as a series of sub-trees as typically modelled in CTTE.

This final approach is more close to the traditional CTT notation, while respecting the identification of the three areas, especially the decision tree. However, to preserve the hierarchical structure, there is a risk of sub-tree reproduction if one sub-tree of the context-sensitive part is used at different locations, which is not the case in the complete separation approach.

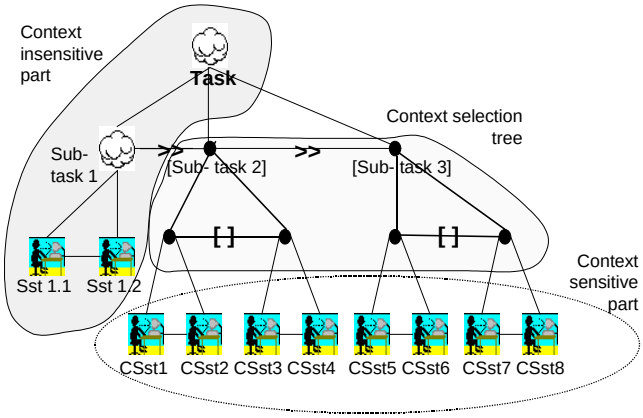

Fig. 6. The final approach for modelling a context-sensitive task

4 Configurations of Context-Sensitive Task Models

Adopting the formalism introduced in Fig. 6, we now may consider different configurations of this context-sensitive task model depending on the number of context-sensitive sub-tasks in the model and the number of possible contexts of use. Let us assume a single interactive task consisting of q sub-tasks at the first level of decomposition and at most m possible contexts of use. Depending on the values of q and m, different configurations can be examined as summarised in Table 1. To graphically depict and analyse possible configurations of context-sensitive task models, let us assume that a horizontal line represents any CTT operator and let us consider task tree with only one level of decomposition for all parts. A sub-task is said to be *context-sensitive* if at least one of its sub-sub-tasks or itself may change if the context of use changes.

Table 1. Possible configurations of context-sensitive task models

	Single context of use	Multiple contexts of use (m)
No context-sensitive sub-task	One context-insensitive tree (Fig. 7a)	One context-insensitive tree, one decision tree, no context-sensitive tree (Fig. 7b)
One context-sensitive sub-task	One context-insensitive tree, one decision tree with one branch, one context-sensitive tree (Fig. 8a)	One context-insensitive tree, one decision tree with m branches, m context-sensitive trees (Fig. 8b)
$p < q$ context-sensitive sub-tasks	One context-insensitive tree, p decision trees with one branch each, p context-sensitive trees (Fig. 9a)	One context-insensitive tree, p decision trees with m branches each, $p \times m$ context-sensitive trees (Fig. 9b)
q context-sensitive sub-tasks	One context-insensitive tree, q decision trees with one branch each, p context-sensitive trees (Fig. 10)	One context-insensitive tree, p decision trees with q branches each, $q \times m$ context-sensitive trees (Fig. 11b)

When there is no context-sensitive sub-task, the configuration is the same as traditionally found in a CTT-like model in a single context of use (Fig. 7a). The multiplicity of the contexts of use in this case does not affect anything since nothing is context-sensitive. Therefore, the decision tree gives rise to a void context-sensitive part (Fig. 7b) and is therefore unuseful.

When there is only one context-sensitive sub-task, the decision tree is reduced to considering only one branch (the one for the only context-sensitive sub-task) for a single context (Fig. 8a) or m branches for multiple contexts (Fig. 8b). In the first case, it means that the context-sensitive part is only conditionally performed if there is a change of context; in the second case, the occurrence of any new context of use may trigger a new decomposition of the context-sensitive part.

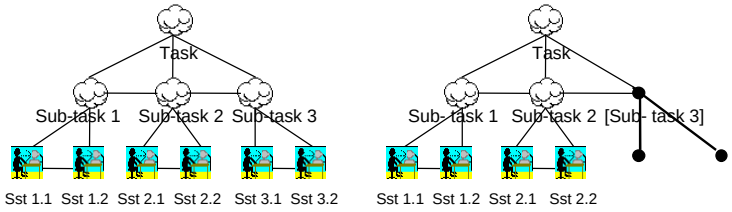

Fig. 7. Configurations with no context-sensitive sub-task: **(a)** single vs. **(b)**multiple contexts.

As discussed above, there is no guarantee that each of these context-sensitive sub-tasks remains unique.

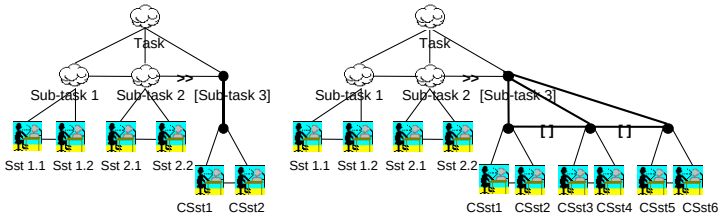

Fig. 8. Configurations with one context-sensitive sub-task: **(a)** single vs. **(b)** multiple contexts.

When there are numerous context-sensitive sub-tasks, let us say $p < q$, but not all of them, there are p possible decision trees in the decision tree part, one for each such sub-task. Each branch leads to one context-sensitive task, respectively p context-sensitive tasks, for a single context of use (Fig. 9a), respectively for multiple contexts of use (Fig. 9b). In the last case, we obtain a maximum of $p \times m$ context-sensitive trees. Indeed, it is likely that not all context-sensitive tasks should respond to all possible contexts of use. It is more likely that only a subset will appear.

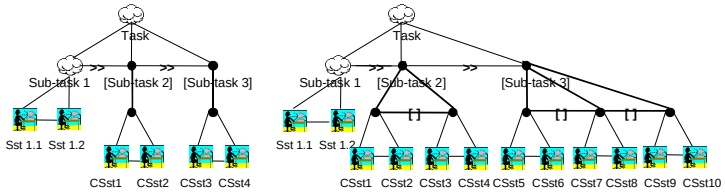

Fig. 9. Configurations with $p \times q$ context-sensitive sub-task: **(a)** single vs. **(b)** multiple contexts.

When all q sub-tasks are context-sensitive, of course there is no longer a need for keeping a context-insensitive part (thus, empty). In a single context of use, the task model is reduced to a decision tree with only one branch to test if we are in the right context of use and q context-sensitive trees related to the context-sensitive sub-tasks (Fig. 10).

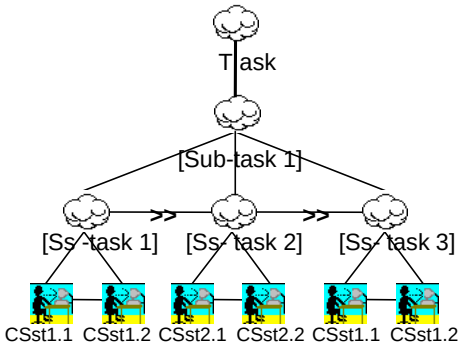

Fig. 10. Configurations with all context-sensitive sub-tasks in a single context.

In the case of m multiple contexts of use, the context-insensitive part is still empty, but leaves space for a potentially huge decision tree with m branches at the first level and at most $q \times m$ context-sensitive tasks at the subsequent level (Fig. 11). In these two last cases, the task model begins directly with the decision tree to branch to the context-sensitive sub-tasks which are appropriate to consider depending on the identified context of use resulting from parsing the decision tree.

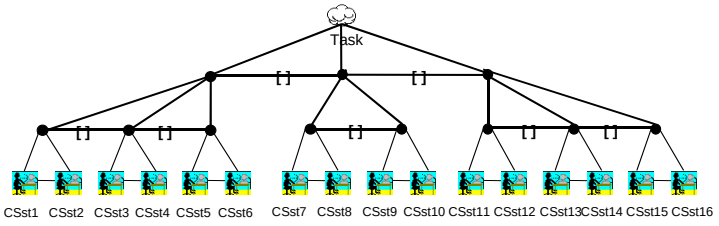

Fig. 11. Configurations with all context-sensitive sub-tasks in multiple contexts.

5 Factoring Out Common Parts for Multiple Contexts of Use

As criticised above, the possible configurations of task models which have been examined in the previous section do not address the problem of factoring out identical elements in the tree. The worst consequence of this is that some similar elements in the task model (e.g., some task units) may be reproduced several times, actually as many times as the same part is used. This raises the question of whether to concentrate all contextual conditions for branching to the different context-sensitive tasks into one decision tree or to distribute them among the global task tree to maximise factoring out of common parts and to avoid duplication. To discuss this issue, let use assume a task model for inputting personal information that needs to be carried out on three computing platforms (i.e. Personal Computer, cellular phone, and Web-based appliance with low or high resolution). This gives rise to four possible contexts of use. Let us also assume in this scenario that the task contains a sub-task displaying general information (i.e. company logo, date and time) whatever the context of use is. Let us also consider that any Web-based appliance should additionally display help links to provide guidance to visitors. A context-sensitive task model without any factoring out is represented in Fig. 12a. The model begins with a decision tree with two contextual conditions: one for identifying the computing platform, another for detecting the screen resolution. All sub-tasks are put as sub-trees of the leaves of the decision tree. In Fig. 12b, a hybrid factoring out is adopted. Observing that the sub-task "Display information" is carried out in all possible contexts of use, this sub-task is extracted from the rest to become the non-context-sensitive part. As all sub-tasks in the mobile phone case may be performed in a sequential and individual manner, this model prefers to leave this arrangement of sub-tasks as it is. A contextual condition states that if a Web appliance is detected, help links should be displayed. Since the arrangement of sub-tasks is unique for the Web case with low resolution, a context-sensitive decom-position is produced and maintained.

However, the arrangement of sub-tasks is identical in both the PC and Web with high resolution cases, thus leading to a last contextual condition gathering these two contexts of use. This factoring out is hybrid since, on the one hand, the "Display information" is correctly factored out, but on the other hand, the leaves of the tree are not unique (e.g., "Input first name" is duplicated two times). More regrettable is that the contextual conditions are no longer equivalent: they are no longer simple conditions with simple values as output and they are not necessarily mutually exclusive. To avoid complex contextual conditions and to avoid possible overlapping of their values, we suggest modelling a context-sensitive task with progressive factoring out. In Fig. 12c, the "Display information" sub-task is common to all possible contexts of use, thus it is kept in the context-insensitive part. The decision tree then begins with only one simple contextual condition: what is the currently used computing platform? The nodes associated with the values of this condition give rise to other individual sub-trees, one for each computing platform. Observing that the sub-task "Dis-

Fig. 12. Task model without factoring out (**a**), with hybrid factoring out (**b**) with progressive factoring out (**c**).

play links" is common to all sub-cases of the Web appliance case, it is added as a context-sensitive sub-task that is local to the sub-tree. Next, the second contextual condition is asked: what is the resolution of the Web appliance's screen? The two possible values give rise to two individual sub-trees detailing what are the specific context-sensitive sub-tasks to perform in that case. Since the "Input person" sub-task is common to both cases with same decomposition beneath, it could also make sense to isolate it by analogy with the "Display links" sub-tasks. This progressive way of factoring out identical elements of the context-sensitive part ensure that only simple contextual conditions are preserved. But the price to pay to this is that a mixing of decision tree and context-sensitive part arcs and nodes is introduced, thus increasing the importance of the notation to see the difference. Otherwise, it may also be desirable not to factor out common parts.

6 A Case Study

To illustrate how this can be applied, let us introduce a case study described by a user scenario of the task "child registration for a leisure activity". A child must be registered in a leisure club for children where various activities are organised. Each time a child wants to participate into any activity, he or she must register not only for organisation issues, but also for producing an invoice to any person who is responsible for the child. This interactive task should be carried out in two contexts of use:

1. A non-stressing context where a written registration form is sent to the club secretariat. In this context, the task may be decomposed into three sub-tasks:
   - Identifying a child for an activity: this subtasks covers all signaletic information such as first name, last name, birth date, sex, computed age, phone number of the responsible person. This information may also include public and private information (e.g., a free text in case of need). A picture can be added. To identify a child, the secretary searches a data base by first name and/or last name. This action may be processed iteratively until a child can be selected in the current list.
   - Identifying a responsible person for the child: for each possible person, a personal identification number is provided, along with first name, last name, title, address, city, zip code, country, up to four phone numbers.
   - Registering the child: depending on the computed age and space availability for activities, a list of possible activities is displayed. Each activity is char-acterised by a name, a monitor, a place, a date, the hours, an age interval, the current attendance, the maximum attendance. Depending on this information, the cost of the registration is computed and added in an invoice. If the child is not a club member, an automated registration is added with cost.
2. A stressing context where a vocal registration is communicated from the respon-sible person to a secretary in a very noisy and confusing environment.

The morning of the activity date, the secretary downloads on a PDA a list of current club members and the list of possible activities of the day. When a child suddenly arrives at the club entrance, often with a responsible person, the secretary asks her a first name and a last name and attempts to quickly locate her in the list. If the child is located, the secretary assumes that the rest can be found after without too much trouble. If not, in order not to stop the responsible person too long, the secretary input name and address. The remainder will be filled up afterwards. The secretary also takes a picture of the child with an ID. An activity is then selected, provided that constraints are met. The secretary works with pressure: many people arrive at the same time in a short amount of time and the running children prevent her to quickly grasp all information accurately. After this stressing period, the secretary comes back to her PC, transfers information, and completes the task by retrieving information from data bases as described above. Fig. 13 represents a possible task model for this task which is restructured depending on the context, but which holds the same sub-tasks.

In the task modelling process, the child registration task was simplified as follows :

- In the stressing context, the operator is searching the name of the child in the list; if found, then desired courses are chosen, else the name of the child, the name of the correspondent and his address are recorded and then the then desired courses are chosen. Hence, the responsible person registration is optional, only if the child was not previously recorded.
- In the normal context, the full registration of the child and his correspondent (for a new child) are performed. The case when the correspondent has more then one child was not considered.

This task modelling is progressively illustrated in Fig. 14, 15, and 16.

As it could be seen, there is another context of use, depending on the data source: written document or verbal. The second could be seen as co-operative (although the correspondent role is not so relevant). However it illustrates a second context of use. The first context of use was modelled as follows. For child registration, if not found, either input child name (stressing context) either full registration are performed ([ ] operator). For correspondent registration, in the stressing context only name and address are recorded. In the normal context, details are also recorded (optional task). The second context of use is modelled using [ ] operator.

7 Conclusion

This paper presented a method for modelling context-sensitive tasks with the aid of the ConcurTaskTree notation [12]. Several approaches were considered and analysed to come up with a final approach combining clear separation of context-sensitive part, context-insensitive part, and context decision tree, while

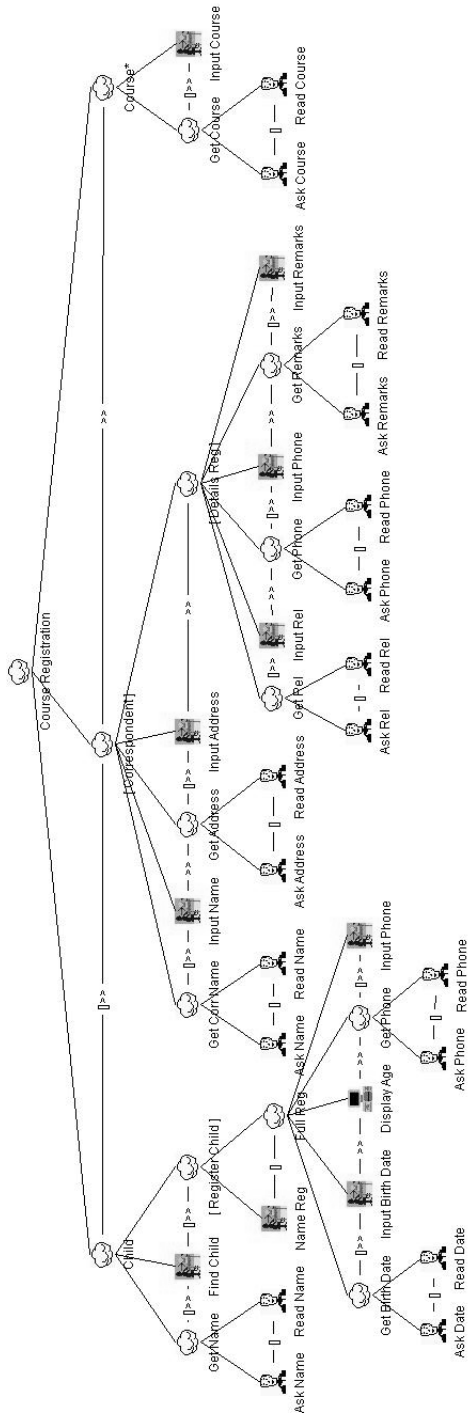

Fig. 13. A possible task model for the "Child registration for a leisure activity".

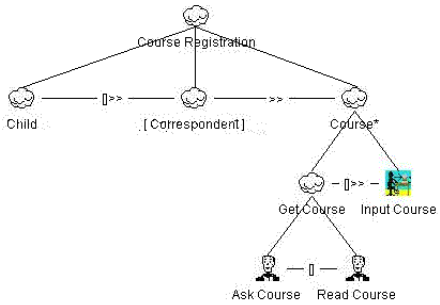

Fig. 14. Main task "Child registration for a leisure activity" partial modeling.

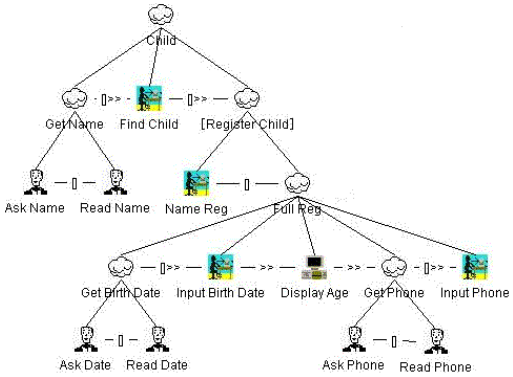

Fig. 15. Sub-Task "Identifying a child".

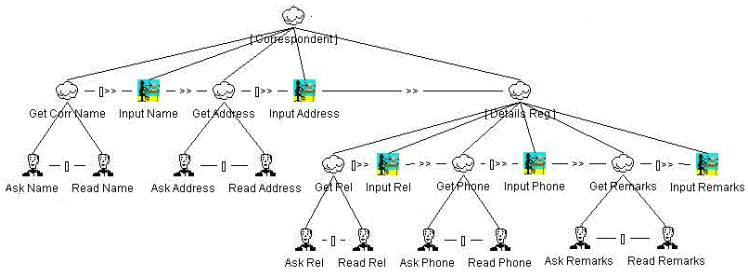

Fig. 16. Sub-Task "Identifying a responsible person".

preserving the hierarchical structure of the model, as expressed in CTT. In order to make a distinction between the normal choice operator and the ones used for the decision tree, we decided to introduce a marked structured annotation in the tasks representing the nodes of the decision tree. Each such task is then augmented with the current value (e.g., low/moderate/high) of the contextual condition considered at that time (e.g., what is the available bandwidth). We then choose to export the resulting CTT model into an XML-compliant file thanks to the facility provided by CTTE. This XML file can then transformed into an XIML-compliant file [14] using XSLT transformations. This XIML file can then be exploited by UI derivation algorithms to progressively derive a generic UI, then a specific UI, and a code generation/interpretation as described in Fig. 1. The study of this derivation process is another importance piece of research, where the context-sensitive task model, along with its domain model are separately or simultaneously considered to derive presentation and dialog models, that are in turn refined into a specific one after considering the user model. Today, some case studies of context-sensitive tasks have been performed with the above method, including the Map Annotation Assistant as described in [11] and the multi-context registration procedure (both in a stressed highly-constrained and in a quiet moderately-constrained environments). A limitation of this current model is relying in its inability to highlight change of context during task accomplishment. The extended model is able to show different sub-task trees depending on different contexts of use considered at a decision point. But it is unable to show how one of these sub-task trees can branch to another sub-task tree to denote a dynamic change of context, where the context is changing while the user is carrying out the task.

References

1. G.G. Abowd and A.D. Key. Towards a better understanding of context and context-awareness. Technical Report Research report 1999-22, Georgia University of Technology, 1999. Accessible at ftp://ftp.cc.gatech.edu/pub/gvu/tr/1999/99-22.pdf.
2. M. Abrams, C. Phanouriou, A.L. Batongbacal, S. Williams, and J. Shuster. UIML: An appliance-independent XML user interface language. In A. Mendelzon, editor, *Proceedings of 8th International World-Wide Web Conference WWW'8 (Toronto, May 11-14, 1999)*, Amsterdam, 1999. Elsevier Science Publishers. Accessible at http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html.
3. B. Bomsdorf and G. Swillius. From task to dialogue: Task based user interface design. *SIGCHI Bulletin*, 30(4):40–42, 1998.
4. G. Calvary, J. Coutaz, and D. Thevenin. A unifying reference framework for the development of plastic user interfaces. In *Proceedings of IFIP WG 2.7 Conference on Engineering the User Interface EHCI'2001 (Toronto, May 11-13, 2001).*, London, 2001. Chapman Hall.
5. S.K. Card, T.P. Moran, and A. Newell. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, New York, 1983.
6. J. Eisenstein, J. Vanderdonckt, and A. Puerta. Adapting to mobile contexts with user-interface modeling. In *Proceedings of IEEE Workshop on Mobile Computing*

Systems and Applications WCSMA'2000 (Monterey, December 7-8, 2000)., pages 83–92, Los Alamitos, 2000. IEEE Press.

7. J. Eisenstein, J. Vanderdonckt, and A. Puerta. Applying model-based techniques to the development of user interfaces for mobile computers. In *Proceedings of ACM Conference on Intelligent User Interfaces IUI'2001 (Albuquerque, January 11-13, 2001)*, pages 69–76, New York, 2001. ACM Press.
8. P. Johnson. *Human-Computer Interaction: Psychology, Task Analysis and Software Engineering*. McGraw-Hill, Maidenhead, 1992.
9. V. Kaptelinin and B. Nardi. Activity theory: Basic concepts and applications. ACM Press (New York), 2000. CHI'2000 Tutorial Notes vol. 5.
10. B. Myers, S. Hudson, and R. Pausch. Past, present, future of user interface tools. *ACM Transactions on Computer-Human Interaction*, 7:3–28, 1. Accessible at www.acm.org/pubs/articles/journals/tochi/2000-7-1/p3-myers/p3-myers.pdf.
11. F. Patern. *Model-Based-Design and Evaluation of Interactive Applications*. Springer-Verlag, Berlin, 2000.
12. F. Patern, C. Mancini, and S. Meniconi. ConcurTaskTree: A diagrammatic notation for specifying task models. In S. Howard, J. Hammond, and G. Lindgaard, editors, *Proceedings of IFIP TC 13 International Conference on Human-Computer Interaction Interact'97 (Sydney, July 14-18, 1997)*, pages 362–369, Boston, 1997. Kluwer Academic Publishers.
13. F. Paternó, C. Santoro, and S. Tahmassebi. Formal model for cooperative tasks: Concepts and an application for en-route air traffic control. In P. Markopoulos and P. Johnson, editors, *Proc. Of 5th Int. Workshop on Design, Specification, and Verification of Intractive Systems DSV-IS '98 (Abingdon, June 3-5 1998)*, pages 71–86, Vienna, 1998. Springer-Verlag.
14. A. Puerta and J. Eisenstein. A representational basis for user interface transformations. In Ch. Wiecha and P. Szekely, editors, *Proceedings of CHI'2001 Workshop "Transforming the UI for Anyone, Anywhere - Enabling an Increased Variety of Users, Devices, and Tasks Through Interface Transformations" (Seattle, April 1-2, 2001)*, New York, 2001. ACM Press.
15. A. Savidis, D. Akoumianakis, and C. Stephanidis. *The Unified User Interface Design Method*, chapter 21, pages 417–440. Lawrence Erlbaum Associates, Mahwah, 2001.
16. P. Szekely, P. Sukaviriya, J. Castells, Muthukumarasamy, and E. Salcher. Declarative interface models for user interface construction tools : The MASTERMIND approach. In *Engineering for Human-Computer Interaction*, pages 120–150. Chapman Hall, London, 1996.
17. M. J. Tauber. ETAG: Extended task action grammar. a language for the description of the user's task language. In D. Diaper, D. Gilmore, G. Cockton, and B. Shackel, editors, *Proc. of the 3rd IFIP TC 13 Conf. On Human Computer Interaction Interact '90 (Cambridge, 27-31 August 1990)*, pages 163–168, Amsterdam, 1990. Elsevier.
18. D. Thevenin and J. Coutaz. Plasticity of user interfaces: Framework and research agenda. In A. Sasse and Ch. Johnson, editors, *Proceedings of 7th IFIP TC 13 International Confercence on Human-Computer Interaction Interact'99 (Edinburgh, August 30-September 3, 1999)*, pages 110–117, London, 1999. IOS Press.
19. M. van Welie, C.G. van der Veer, and A. Eliens. An ontology for task world models. In *Proc.of the 5th Int. Workshop on Design, Specification and Verification of Interactive Systems DSV-IS'98 (Abingdon, 3-5 June 1998)*, pages 57–70, Vienna, 1998. Springer Verlag.