The COMETs Inspector: Towards Run Time Plasticity Control Based on a Semantic Network

Alexandre Demeure[1], Gaëlle Calvary[1], Joëlle Coutaz[1], and Jean Vanderdonckt[1,2]

[1] Université Joseph-Fourier
CLIPS-IMAG, BP 53
F-38041 Grenoble Cedex 9, France
{Alexandre.Demeure, Gaelle.Calvary, Joelle.Coutaz}@imag.fr
[2] Louvain School of Management,
Université catholique de Louvain
Place des Doyens, 1
B-1348 Louvain-la-Neuve, Belgium
jean.vanderdonckt@uclouvain.be

Abstract. In this paper, we describe the COMETs Inspector, a software tool providing user interface designers and developers with a semantic network in order to control the plasticity of their User Interfaces (UI) at run-time. Thanks to a set of predefined relationships, the semantic network links together various concepts ranging from the final UI (i.e., the UI described in terms of technological spaces) to the concrete and abstract UIs (i.e., the UI respectively described in terms of concrete interaction objects independently of any technological space, and abstract individual components and containers independently of any interaction modality) up to the tasks and concepts of the interactive system. In this way, plasticity can be addressed at four levels of abstraction (task and concepts, abstract, concrete, and final user interface) for forward, reverse, and lateral engineering. The end user exploits the semantic network at run time to adapt his/her UI to another context of use by identifying, selecting, and applying plasticity suitable operations.

Keywords: Abstract user interface, Active model, Ambient intelligence, COMET, Concrete user interface, Model-based approach, Plasticity, Semantic network, Task modeling, User interface eXtensible Markup Language.

1 Introduction

In an ever-changing world, end users of interactive systems are constantly demanding a higher level of adaptation of their User Interfaces (UI) to fit their purpose and better address their needs and wishes. The wide availability of different computing platforms makes this desire even stronger as the aspiration for executing the same interactive system on these different platforms is expressed, while minimizing the changes in the UI across these platforms. In these circumstances, the notion of plasticity plays a fundamental role as it denotes the "capacity of a UI to withstand

K. Coninx, K. Luyten, and K.A. Schneider (Eds.): TAMODIA 2006, LNCS 4385, pp. 324–338, 2007.

variations of context of use while preserving predefined usability properties" [2]. Supporting plasticity is more sophisticated than merely ensuring UI adaptation. Any kind of UI adaptation always induces some disruption from the end user's point of view as parts or whole of the UI may change during adaptation. Simple adaptation does not necessarily guarantee any level of quality. In contrast, plasticity aims at maintaining a certain level of usability by explicitly addressing the evolving context of use in which the user is carrying out his/her interactive task. By context of use [2], we hereby refer to the combination C of a user U working with a platform P in a given physical environment E: $C = <U,P,E>$. Although the adaptation in general and the plasticity in particular both consider the three aspects of this context definition, it is noteworthy to observe that the P aspect is the most frequently and extensively researched area (among them are [3], [4], [6], [8], [9], [11], [14-18], [20]): the platform is probably the facet which affects the UI the most immediately and concretely. This is challenging since a UI which was designed for a given platform in mind may no longer fit another one with extended or reduced interaction capabilities if they were not considered before.

The premises for supporting any form of plasticity are twofold: first, the availability of any valuable *information on the context of use* that may influence the UI adaptation, and secondly, the relationships between this contextual information and the reshuffled UI (remolded and/or redistributed) for that context. The Model-Based UI Development (MB-UIDE) community typically addresses the former aspect by context modeling [2], [4], [14], [17], [20] enriching task and system modeling [6], [15], whereas for the latter, the problem is often characterized as a *mapping problem* between the models [3], [10], [12], [19]. Thanks to the combination of context modeling and a technique for solving the mapping problem, it is possible to adapt the UI presentation, dialog and/or deployment after a context of use variation [13].

The literature identifies three significant instants when this combination occurs depending on the time when the models and their relationships are used: *at design time* to foresee future plastic UI, *at installation time* to take into account the current context of use (especially the platform that is foreseen at that time), and *at run time* to take into account contextual information which is known only at that time. Most recent works are devoted to design and installation time. The few works dedicated to run time are mostly addressing plasticity at the concrete UI level where only the UI look and feel is changed.

In this paper, we present a software tool which goes beyond this situation by supporting plasticity at run time at any level of abstraction (ranging from the final UI to the task and the domain) thanks to a semantic network that solves the mapping problem in a more elaborated way than existing techniques. To prove this, Section 2 summarizes the current trends in design- and installation-time plasticity, and identifies the most recent advances in run-time plasticity so as to locate this work as a next step in the progress. Section 3 provides a general definition of the semantic network that is used throughout this paper and illustrates it with an excerpt centered on the task type of choice. It exemplifies the case study along with a series of *plasticity questions* which can be addressed thanks to this network, and that cannot be addressed by existing systems. Section 4 presents the COMETs Inspector, a software that exploits this network at run time. Section 5 concludes the paper by highlighting the strengths

and the shortcomings of the current version of the system and introduces new families of UIs with even a higher level of plasticity to be researched in the future.

2 Related Work

FormsVBT [1] pioneered the field of *plasticity at design time* by providing the UI designer with three views: a view on TeX-based UI specifications, a view on the UI presentation and dialog, and a view on the final UI. These three views are coordinated: any change brought in one view is automatically reflected in the others, thus providing the end user with a mean to directly validate or invalidate a UI crafted for a specific platform. The Graceful Degradation plug-in [8] for GrafiXML editor (www.usixml.org) provides UI designers with a series of transformations to be manually applied on a UI tailored for an initial platform. The resulting UI should be adapted to a computing platform exhibiting reduced interaction capabilities, especially a smaller resolution or reduced widgets set. The Context Toolkit [4] embeds multiple widgets compositions in one single widget with plasticity capabilities. This system is still design time: although the appropriate UI composition is selected at run time, the available compositions are pre-computed at design time. The system only switches from one composition to another depending on the changes of the context of use. This observation is similar for the Ubiquitous Interactor [16], the vocabulary of Generic Widgets found in [18], and the ADUS system [14].

For *plasticity at installation time*, in AUI [20], the UI is also shipped with different compositions which are selected when the interactive application is installed on a particular platform. In the same vein, TERESA [17] automatically generates multiple UIs for multiple platforms, but one UI is used at a time for each considered platform. TERESA also supports some plasticity by achieving transmodality, i.e. a change of modality after a platform change.

For *plasticity at run time*, Keränen and Plomp [11] present an algorithm for repurposing a UI layout depending on its container dimensions. An interesting feature consists in its animation of the adaptation process. ARNAULD [9] is relying on games theory for eliciting the most preferred UI at run time. It is based on SUPPLE, a system which automatically generates a UI layout based on weights of its contents. ARNAULD shows very interesting plasticity questions such as widget substitution, layout reshuffling and re-portraiting. In this paper, we will show that the COMETs Inspector supports more sophisticated forms of what we will define as *plasticity questions*.

Puerta & Eisenstein [19] defined a computational framework for managing relationships within and across the various models (e.g., the task, the domain, the abstract UI, the concrete UI, the system, the context) to solve the mapping problem. Teallach [10] is probably the first implementation of this framework, although it is not targeted at plasticity, but merely UI development. Since then, several attempts have been made to expand this form of plasticity, as in [3] for ambient intelligence and in [12] for multi-platform UIs. The predefined usability involved in the plasticity in [3] is the consistency, while it is the UI guidance in [12].

All the aforementioned efforts to support plasticity involve some form of information on the context of use (usually in a context model) and some ways to infer a UI from this context (typically as a system of inference rules, as a knowledge base,

as a set of transformations). Next section introduces our semantic network, our new approach to condensate UI design knowledge captured at design-time, but to be exploited at run time.

3 A Semantic Network for Run Time Plasticity

This section provides a general definition of a semantic network (3.1). It is then applied to plasticity (3.4) based on concepts and relationships (3.3) defined in the CAMELEON reference framework (3.2). The section concludes with plasticity questions that are covered by the approach (3.5).

3.1 General Definition

Sowa [21] defines a semantic network as "a graphic notation for representing knowledge in patterns of interconnected nodes and arcs. Computer implementations of semantic networks were first developed for artificial intelligence and machine translation, but earlier versions have long been used in philosophy, psychology, and linguistics". Each semantic network may exhibit one or many of the following dimensions [21]:

- *Definitional networks* emphasize the subtype or "is-a" relation between a concept type and a newly defined subtype. The resulting network, also called a generalization or subsumption hierarchy, supports the rule of inheritance for copying properties defined for a supertype to all of its subtypes.
- *Assertional networks* are designed to assert propositions. Unlike definitional networks, the information in an assertional network is assumed to be contingently true, unless it is explicitly marked with a modal operator.
- *Implicational networks* use implication as the primary relation for connecting nodes.
- *Executable networks* include some mechanisms, such as marker passing or attached procedures, which can perform inferences, pass messages, or search for patterns.
- *Learning networks* build or extend their representations by acquiring knowledge from examples.

By defining the concepts and relationships appropriate for UI plasticity (3.3), we argue that our semantic network combines the five above dimensions. Concepts and relationships for plasticity are based on the CAMELEON reference framework.

3.2 CAMELEON Reference Framework

The CAMELEON Reference Framework (www.plasticity.org) structures the development life cycle of multi-target UIs according to four levels: (1) the *Final UI* (FUI) is the operational UI, i.e. any UI running on a particular platform either by interpretation (e.g. through a Web browser) or by execution (e.g., after the compilation of code in an interactive development environment); (2) the *Concrete UI* (CUI) expresses any FUI independently of any term related to a peculiar rendering engine, that is independently

of any markup or programming language; (3) the *Abstract UI* (AUI) expresses any CUI independently of any interaction modality (e.g., graphical, vocal, tactile) via the mechanisms of Abstract Interaction Objects (AIO) [22] as opposed to Concrete Interaction Objects (CIO) for the CUI; and (4) the *Task & Concept* level, which describes the various interactive tasks to be carried out by the end user and the domain objects that are manipulated by these tasks. We refer to [11] and to www.usixml.org for its translation into models uniformly expressed in the same User Interface Description Language (UIDL), selected to be UsiXML (which stands for User Interface eXtensible Markup Language). In Figure 1, two contexts of use are represented with the possibility of moving from one context to another one through three relationships: *abstraction*, *reification* and *translation* for respectively reverse, forward and lateral engineering.

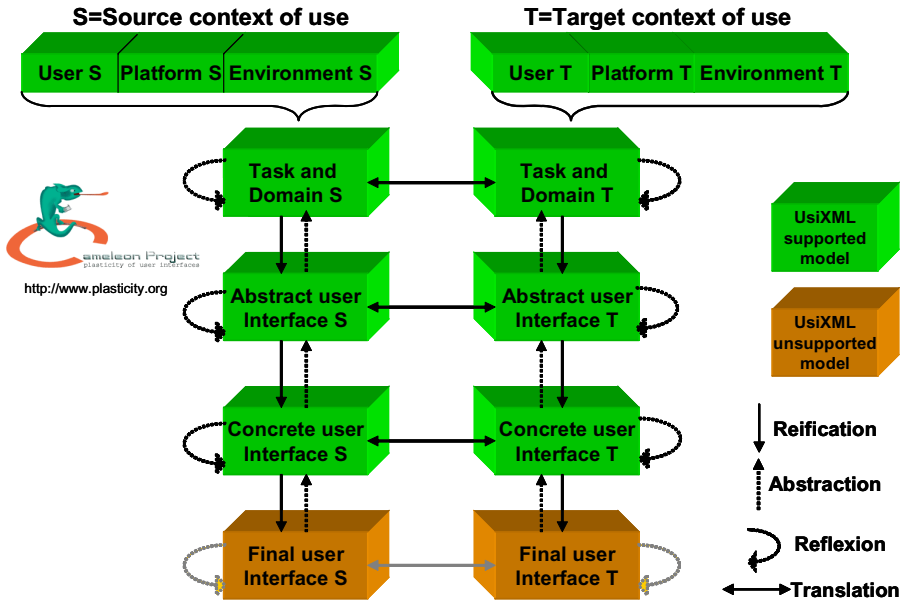

Fig. 1. The four levels of the CAMELEON framework

3.3 Concepts and Relationships for Plasticity

The concepts are those that are involved at each level of the CAMELEON reference framework (Fig. 1), which can be found in UsiXML (www.usixml.org): the "task & domain" level manipulates a task model (which consists of a recursive decomposition of a task into sub-tasks ordered with temporal relationships) and a domain model (which consists of a UML class diagram). In UsiXML, each task is associated with a task type: acquire, convey, select, navigate, compute, print, publish, etc. The task type is associated to an attribute, a group of attributes, or a class in the domain model. Therefore, the data type and the definition of the domain and co-domains are inferred from the domain model.

At the AUI level, any AUI consists of a decomposition of Abstract Containers into Abstract Individual Components (AIC). Each AIC exhibits one or many facets among input, output, control, etc. For instance, a task "select the value of an attribute" could be mapped onto an AIC "input an element from a collection".

At the CUI level, the AUI is reified into Concrete Containers and Concrete Interaction Objects satisfying the constraints imposed by the AUI. In our example ("input an element from a collection"), any CIO matching the AIC could work, such as a list box, a combo box, a radio box.

The concepts of the network are structured with multiple types of relationships such as inheritance, aggregation, composition, etc. The relationships themselves are arranged in an inheritance hierarchy, as presented in Fig. 2. Therefore, the semantic network is represented as a graph (i.e. a set of nodes and edges between the nodes), whose nodes represent fragments of models appearing at any level of abstraction and edges consist of transformation between nodes. The transformations represent a key aspect of exploiting UI design knowledge [19].

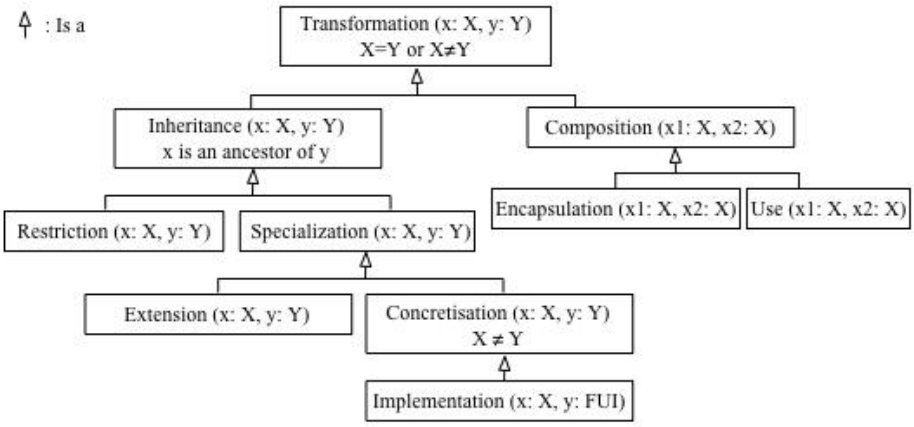

Fig. 2. Inheritance hierarchy between the relationships

The transformations are the following ones:

- *Inheritance. y* inherits from *x* if *y* refines *x*. The relation can be *total* versus *partial*, *exclusive* versus *non exclusive*. *Total* means that *x* cannot be instantiated as is: only *y* can exist. *Exclusive* means that there can be no *t* inheriting from both *y* and *z* if *y* and *z* refine *x* in an exclusive way.
- *Restriction.* Restriction refers to cuts that make of *y* a sub-case of *x*. As a result, *y* and *x* are no more substitutable. One example is the type restriction.
- *Specialization*. Specialization refers to inheritance that preserves properties. If *y* specializes *x* then *y* satisfies all the properties of *x*. As a result, *y* can be seen as an *x* making it substitutable to *x*.
- *Extension*. *y* extends *x* if *y* adds new descriptions to *x*, but *x* is still an *X*. This kind of inheritance is always partial.

- *Concretisation*. Concretisation refers to reification (Fig. 1). *y* concretises *x* if *y* adds concrete descriptions to *x* but *x* is not changed.
- *Implementation*. Whatever *x* is except an FUI, *y* is an FUI corresponding to *x*.
- *Composition*. *y* is part of *x* if *y* is included as is in *x*. *y* can be seen as a subsystem of *x*. Mappings between *x* and *y* are weaved.
- *Encapsulation*. Encapsulation means that *y* is embedded in *x*. *y* is consumed. It does no more exist as is.
- *Use*. Conversely to encapsulation, if *y* is used in *x*, then *y* still exists.
- *Abstraction* and *reification* are two other kinds of transformations. They are defined accordingly to Fig. 1.

Based on these concepts and relationships, next section presents a semantic network for plasticity.

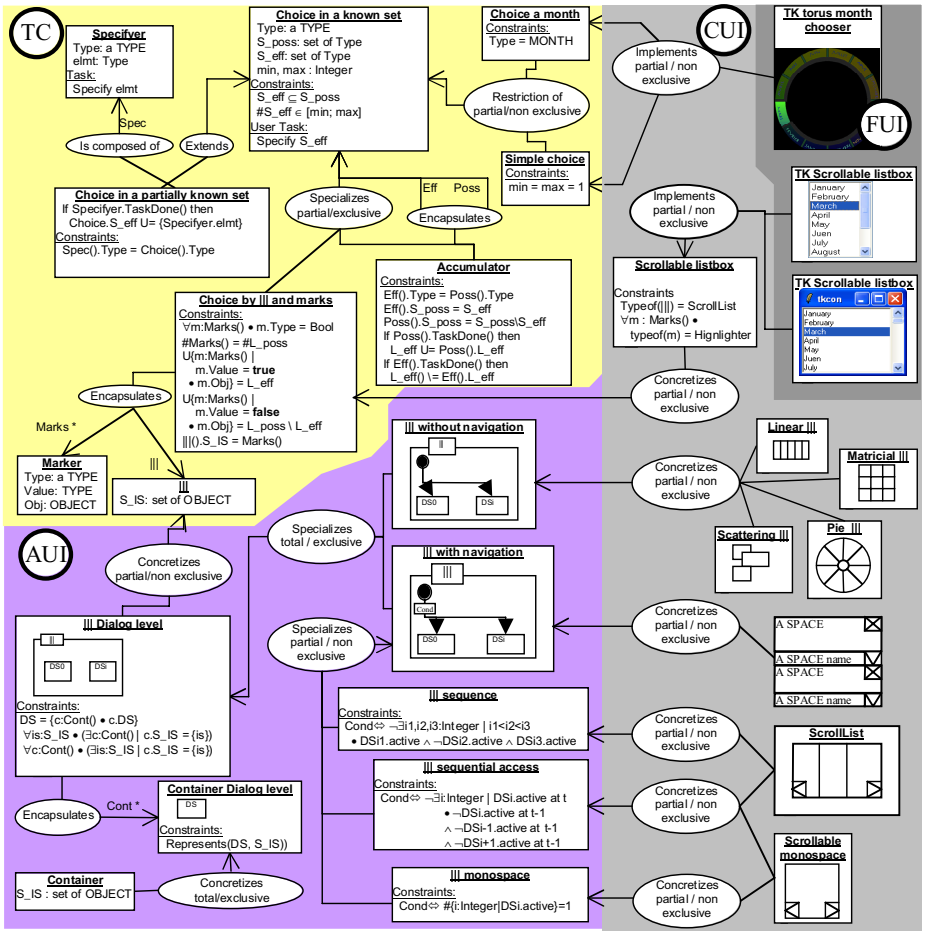

Fig. 3. Excerpt of the semantic network for the “Choice” case study

3.4 The Semantic Network for Plasticity

For legibility, this subsection focuses on an excerpt of the entire semantic network: the portion related to the "Choice" task type (Fig. 3). We have selected this portion because many interactive systems involve some form of choice among items, objects, menus, actions, etc. In addition, the available widget set for implementing a choice is wide: list box, drop-down list, combination box, drop-down combination box, radio button, check box, etc. In addition to these typical widgets, specialized widgets exist too: fast scrolling list box, accumulator, pie menu, season selector, calendar, etc. Usually, usability guidelines convey information to the designer on how to choose, format, and implement a choice widget in a UI. But this knowledge remains always subject to human interpretation and is never provided in an explicit, exploitable way. Our semantic network tackles this problem.

As illustrated in Fig. 3, the semantic network collects descriptions of a same entity (here the "Choice") in a same schema and makes explicit the relationships between them. The concepts and relationships are those that have been elicited in subsection 3.3. For legibility, the level of abstraction to which the descriptions belong is indicated by colors and labels: TC for Task & Concepts, AUI, CUI, FUI.

A description is provided for each node. For instance, at the TC level, the task "Choice in a known set" (of elements) makes explicit that:

- It manipulates elements of a given type TYPE.
- Elements can be chosen in a set of possible elements (S_poss).
- The selected elements are stored in a set of effective elements (S_eff).
- The number of selected elements can vary between a minimum (min) and a maximum (max).
- And of course (constraints part), S_eff is a subset of S_poss, and the number of effective elements is comprised between the min and max values.

The task "Choice a month" is a restriction of "Choice in a known set" as the type of the elements is constrained to be a month (see the constraint "Type=MONTH" in Fig. 3). A round FUI is provided as an example of implementation ("TK torus month chooser"). It is interesting to note that this FUI is an implementation of both "Choice a month" and "Simple choice" tasks. They are both restrictions of "Choice in a known set" (of elements). "Choice a month" is a restriction along the type of elements, whereas "Simple choice" restricts the number of selectable elements (see the constraint min=max=1).

"Choice in a known set" of elements can be specialized in many ways: for instance accumulators ("Accumulator"), and interleaving and markers ("Choice by ||| and marks"). For legibility, accumulators are not described in Fig. 3. They are typically concretized as two lists exchanging elements according to the user's selection. Fig. 3 elaborates further on the interleaving and markers specialization. A marker is a Boolean that indicates whether the corresponding element is selected (true) or not (false). Markers are managed by interleaving. Scrollable list boxes are typical concretizations (Fig. 3): the scrollbar corresponds to the interleaving, whereas the highlighting color corresponds to the marker (true). Two TK implementations are provided in Fig. 3. Check boxes are another option, whereas radio buttons would concretize both "Choice by ||| and marks" and "Simple choice".

At the AUI level, interleaving ("|||") is concretized as a dialog space ("||| dialog level") managing the elements that are interleaved. One dialog space is associated per element. They are nested in the interleaving dialog space. Two specializations are mentioned whether there is or not a navigation between the interleaved dialog spaces ("||| with navigation", "||| without navigation"). By navigation, we mean articulatory user's actions that do not directly contribute to the user's task but that are necessarily to access to the dialog spaces in which the user will perform his/her task. For instance, opening a menu is an articulatory task. One CUI with navigation is provided (Fig. 3): the user has to deploy the menu before achieving his/her task. This CUI contrasts with a linear, grid, scattering or pie interleaving that directly makes observable all the dialog spaces: no navigation is required (Fig. 3).

As pointed out in Fig. 3, interleaving with navigation ("||| with navigation") can be specialized in many ways. Three variants are mentioned:

- *Sequence* ("||| sequence"): the possible elements are browsed in a sequential way. The scroll list is a typical CUI example;
- *Sequential access* ("||| sequential access"): the possible elements are browsed in sequential way, parcel by parcel, whatever the size of the parcel is (i.e., the number of elements that are browsed step by step). Roughly speaking, it is not possible to switch from X to X+2 without first displaying X+1. The scroll list is another implementation;
- *Monospace* ("||| monospace"): only one dialog space is observable at a time. An example of FUI is provided in Fig. 3.

Besides this organized capitalization of knowledge, the semantic network promotes creation through composition. Composition is supported as a Cartesian product. It is for instance possible to combine any specialization of interleaving with any specialization of marker to create new interactors that had never been seen in the past. This is powerful for exploring new possibilities at design and/or run time: for instance, what about a monospace multiple choice with highlighters?

Now that the principles of the semantic network have been roughly introduced, let us examine how it can help in designing or plastifying UIs. Exploitation may be driven by strategies, such as:

- "Select the existing FUI that is the most compliant with the functional requirements". That means that producing FUIs manually or automatically is not an option. An existing FUI has to be selected. In that case, only three FUIs are available: the TK torus month chooser and the two TK scrollable list boxes. Again, for legibility, all the existing widgets supporting the "Choice" task have not been mentioned on Fig. 3.
- "Identify the element that map the best with all the functional and non functional requirements and if necessary generate an FUI from that point". Of course, the new FUI will be inserted in the network at the right place to enrich the knowledge for further designs and/or adaptations.
- "Prefer general purpose widgets" such as list box, combo box, pie menu that serve the simple choice with no restriction. As they are less exotic, they will probably be more familiar to the user.

Next section elaborates on the relevance of the semantic network for solving plasticity questions.

3.5 Covered Plasticity Questions

Since plasticity is a particular form of adaptation, it is equally submitted to the problems to be solved by adaptation. The main goal of performing some adaptation consists in defining an adaptation goal, identifying and executing adaptation rules in order to reach the adaptation goal. The literature abounds in providing adaptation rules, but seems more silent in defining properly adaptation goals by linking them to adaptation rules which could be executed for this purpose. Similarly, it is expected here to uncouple the adaptation goals from the adaptation rules. Therefore, we define a *plasticity question Q* as a couple $Q = (G, S)$ where G denotes a *plasticity goal* to reach when performing plasticity and S denotes a set of *plasticity solutions* which are potential actions to be executed to reach the plasticity goal. Let us assume that a plasticity goal G would be "migrate a graphical UI from a desktop to a PDA". The reduced screen real estate of the PDA stems for trying to reduce the surface of the UI widgets, a possible solution among others. For instance, "a list box could be turned into a drop-down list", "a radio box of radio items could be transformed into a drop-down list" are two possible plasticity solutions. The main shortcoming observed in the state of the art is that the set S is usually defined in extension by hard-coding opportunistic plasticity solutions in the adaptation engine, thus leaving little or no room for flexibility and modifiability. In this paper, the definition of S is given in comprehension so that the definition of plasticity questions remains unchanged: any extension of the semantic network will be automatically incorporated in the related plasticity questions.

A plasticity question is said to be *simple*, respectively *composite*, if and only if its goal G involves concepts and relationships of at most, respectively at least, one level of the CAMELEON reference framework (Fig. 1).

Since a FUI plasticity question only refers to elements of technological spaces, a restriction of the questions to be addressed is imposed. For instance, the plasticity goal "transcode a form from HTML to Java" is decomposed into similar sub-goals for all constituents of the form, such as "transcode a SELECT element from HTML into its counterpart in Java". If XUL is the target language, the goal becomes "transcode a SELECT element from HTML into its counterpart in XUL". To solve this question, the mappings between counterpart elements in various technological spaces are required. In terms of the semantic network, the plasticity solution consists of an abstraction of the SELECT element followed by a reification in the target platform, which is expressed as:

$$S = \{ rei_{c\text{-}f} (abs_{f\text{-}c} (\text{SELECT, HTML}), \text{Java}) \}$$

where $rei_{c\text{-}f}$ denotes the reification from CUI to FUI, $abs_{f\text{-}c}$ denotes the abstraction from FUI to CUI. If the previous plasticity goal is extended up to the CUI level, it would give "abstract a SELECT element from HTML into a CUI", a platform agnostic goal which is expressed as:

$$S = \{ abs_{f\text{-}c} (\text{SELECT, HTML}) \}$$

If the previous plasticity goal is extended up to the AUI level, it would give "abstract a SELECT element from HTML into a AUI", a modality agnostic goal which is expressed as:

$$S = \{ abs_{c\text{-}a} (abs_{f\text{-}c} (\text{SELECT, HTML})) \}$$

where $abs_{c\text{-}a}$ denotes the abstraction from CUI to AUI. If the previous plasticity goal is extended up to the TC level, it gives "abstract a SELECT element from HTML into a task and domain", a computing independent goal which is expressed as:

$$S = \{ abs_{a\text{-}tc} (abs_{c\text{-}a} (abs_{f\text{-}c} (\text{SELECT, HTML}))) \}$$

where $abs_{a\text{-}tc}$ denotes the abstraction from AUI to TC.

The original plasticity question in natural language could be generalized as "Give me all the widgets that are equivalent to this HTML widget" ($S = \{rei_{c\text{-}f} (abs_{f\text{-}c} (\text{SELECT, HTML}), X)\}$) where X denotes any technological space. If this widget is itself composed of other sub-widgets, the plasticity solution is recursively addressed. For instance, if a group box is composed of a group and a series of radio items, the plasticity solution is queried on the semantic network on the sub-nodes.

Other typical plasticity questions involve: "Give me all the possible reifications of this CIO for any technological space", or "for the X technological space", "Give me the abstraction of this CIO", "Give me the possible reifications of this AIO satisfying this property", "Give me the behaviorally-equivalent widgets in the same technological space corresponding to a given widget", "Give me a modality-equivalent CIO of this CIO", "Give me any equivalent CIO of this CIO independently of any modality", "Give me a browsable version of this observable interaction component", "Give me all the possibilities for implementing a simple choice".

Next section introduces a small case study that takes benefit from the semantic network at run time to solve few of these questions under the control of the end user.

4 A Case Study: The COMETs Inspector

The Home Heating Control System (HHCS) allows the user to manage the temperature at home depending on the month. In an interleaving way, the user selects the month and controls the temperature of the different rooms. They are here limited to the living room and the wine cellar (Fig. 4). HHCS has been implemented in COMETs (COntext Mouldable widgETs). COMETs are interactors specially fashioned for plasticity [2]. A COMET is "a self descriptive interactor that publishes the quality in use it guarantees for a set of contexts of use. It is able to either self-adapt to the current context of use, or be adapted by a tier-component. It can be dynamically discarded, respectively *recruited*, when it is unable, respectively *able*, to cover the current context of use" [2].

HHCS is made of four major COMETS:

- One for each user's task ("choose a month", "control living-room" and "control wine cellar"). Each COMET recursively embeds (encapsulates) other COMETs for both guiding the task (e.g., the label "Select a month") and sustaining interaction (e.g., the list boxes and sliders on Fig. 4a).

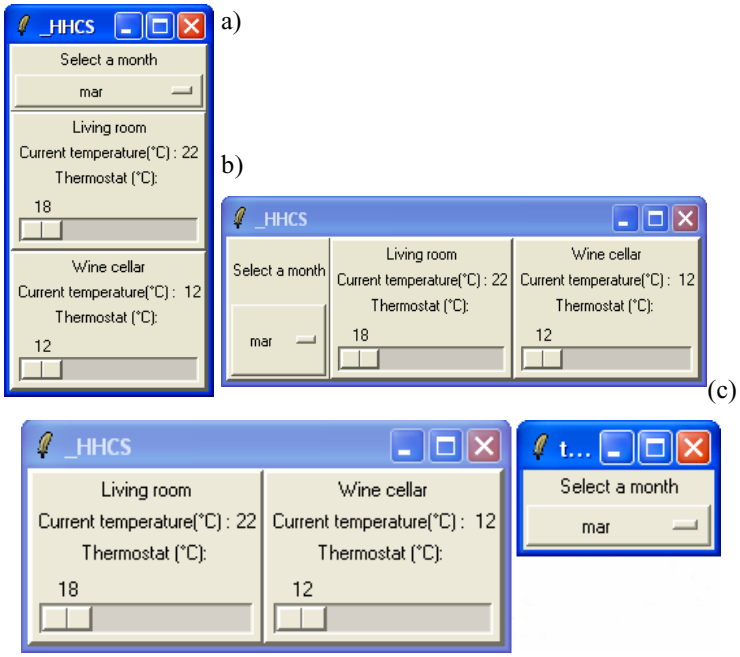

Fig. 4. A set of FUIs obtained by tuning the interleaving comet. Detachable windows are easily implemented thanks to COMETs.

- One for the interleaving. This comet is in charge of managing the three previous ones (they are nested in this COMET). Depending on the layout (Fig. 4 a and b) and whether the embedded containers are displayed as frames (Fig. 4 a and b) or windows (Fig. 4c), the rendering is updated, possibly implementing detachable/(re-)attachable windows (Fig. 4c).

In our approach, adaptation is placed under the control of the end user (yet the designer only, because of a too poor quality of the tool's UI). A COMETS inspector [5] supports the inspection of the UI and its modification thanks to the support of the semantic network. The TK torus month chooser has been selected in Fig. 5.

Only basic operations (i.e., Add, Remove and Substitute) are supported yet, for instance enabling the end-user to substitute one FUI with another one. Fig. 6 shows the inspector (the left window). It displays the hierarchy of comets (left part). A zoom in the selected one is provided (central part). The performable operations are listed in the right part according to the freedoms leveraged by the semantic network. On Fig. 6, the user is being to switch from a window-based to a frame-based presentation for the "control living room" COMET. This will have the effect of re-attaching the living-room window to the main HHCS window. Actually, the semantic network is outside the COMETs. We envision embedding local semantic networks in the COMETs to support a mix of open and close adaptations.

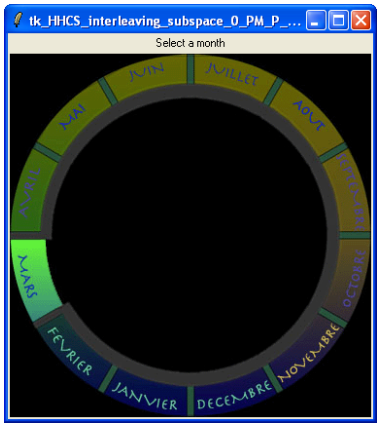

Fig. 5. The torus presentation for selecting a month

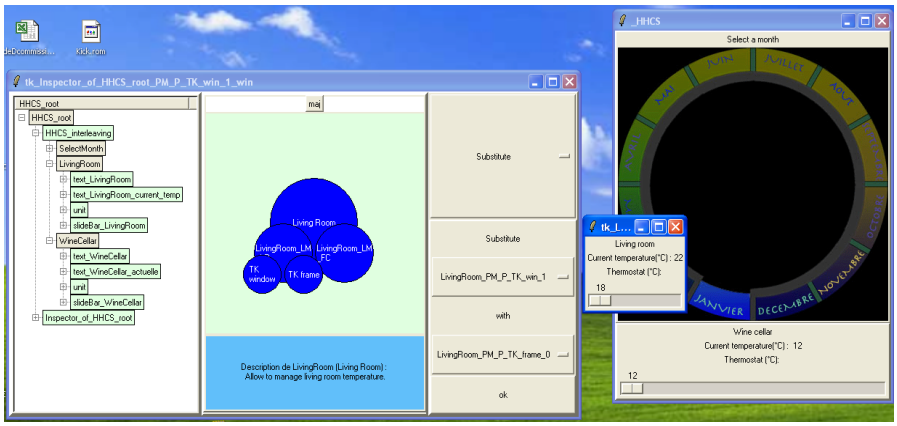

Fig. 6. Based on the semantic network, the comets inspector (left window) provides the user (yet, the designer; in the future, the end-user) with a set of operations (right part of the left window) that can be applied to the interactive system (the two right windows) for its design and/or adaptation. Here, the substitute operation will replace the living room window (the small middle window) with a frame that will be attached to the main window (right window).

5 Conclusion

First of all, it is important to emphasize that the semantic network defined in this paper is independent from its exploitation through the COMETs Inspector: whether you are using a COMET-compliant system [2] or not, it does not matter and it does not change the structure of concepts. The network structures the concepts throughout the four levels of the CAMELEON Reference Framework, thus enabling us to address plasticity questions at run time with an unprecedented level of flexibility and exploitation. Plasticity can now be based on the task and the concepts models. Since the network is exploited at run time to address the plasticity questions requested by

the end user, genuine run time plasticity could be achieved. The COMETS Inspector is just one implementation of a software which accesses this network and performs the desired operations. In the provided example, the task type was predefined (here, a choice). We could even imagine that this task type is provided at run time by the end user by asking "what task do you want to carry out on this object?". The user could then be presented by a series of options like "Insert an object, delete an object, list existing objects, select an object among several (our example)". This is compliant with the CRUD pattern (Create-Read-Update-Delete) design pattern usually found in the UML method and notation. Therefore, the design knowledge that is contained in the semantic network remains stable over time since the plasticity questions do not change. If, for instance, another widget should be added, it could be added only where it is required and the rest is re-composed straightforwardly. Changing the network is a matter of adapting the internal representation (a graph) of the network and exploiting it therefore becomes a problem of graph exploration according to predefined semantic relationships. Of course, the quality of the results heavily depends on the network quality.

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). Jean Vanderdonckt would like to thank Université Joseph Fourier for supporting his position as invited professor for two months since May 2006.

References

1. Avrahami, G., Brooks, K.P., Brown, M.H. A Two-view Approach to Constructing User Interfaces. In *Proc. of SIGGRAPH'89* (Boston, July 31-August 4, 1989), Computer Graphics, *23, 3* (July 1989), 137-146
2. Calvary, G., Coutaz, J., Dâassi, O., Balme, L., Demeure, A. Towards a new Generation of Widgets for Supporting Software Plasticity: the "Comet". In *Proc. of 9th IFIP Working Conf. on Engineering for Human-Computer Interaction EHCI-DSVIS'2004* (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425. Springer-Verlag, Berlin, (2005), 306-324
3. Clerckx, T., Luyten, K., Coninx, K. The Mapping Problem Back and Forth: Customizing Dynamic Models while Preserving Consistency. In *Proc. of the 3rd Int. Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2004* (Prague, November 15-16, 2004). ACM Press, New York, (2004), 33-42
4. Crease, M., Gray, P.D., Brewster, S.A. A Toolkit of Mechanism and Context Independent Widgets. In *Proc. Of Int. Workshop on Design, Specification, and Verification of Interactive Systems DSVIS'2000* (Limerick, June 5-6, 2000). Lecture Notes in Computer Science, Vol. 1946. Springer-Verlag, Berlin, (2000), 121-133
5. Demeure, A., Calvary, G., Coutaz, J., Vanderdonckt, J. The Comets Inspector, Manipulating Multiple Interface Representations Simultaneously, In *Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'06* (Bucarest, June 3-5, 2006). Springer-Verlag, Berlin, (2006), 167-174
6. Dittmar, A., Forbrig, P. Methodological and Tool Support for a Task-Oriented Development of Interactive Systems. In *Proc. of 3rd Int. Conf. on Computer-Aided Design of User Interfaces CADUI'99* (Louvain-la-Neuve, Oct. 21-23, 1999). Kluwer Academics Pub., Dordrecht, (1999), 271-274

7. Fensel, D., Benjamins, V., Motta, E., Wielinga, B. UPML: A Framework for Knowledge System Reuse. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence IJCAI'99* (Stockholm, July 31-August 6, 1999). Morgan Kaufmann, San Francisco, (1999), 16-23
8. Florins, M., Montero, F., Vanderdonckt, J., Michotte, B. User Interface Graceful Degradation for Small Platforms. In *Proc. of 8th Int. Working Conference on Advanced Visual Interfaces AVI'2006* (Venezia, May 23-26, 2006). ACM Press, New York, (2006)
9. Gajos, K., Weld, D.S. Preference Elicitation for Interface Optimization. In *Proc. of the 18th Annual ACM Symp. on User Interface Software and Technology UIST'2005* (Seattle, Oct. 23-26, 2005). ACM Press, New York, (2005), 173-182
10. Griffiths, T., Barclay, P.J., Paton, N.W., McKirdy, J., Kennedy, J.B., Gray, P.D., Cooper, R., Goble, C.A., da Silva, P. Teallach: a Model-Based User Interface Development Environment for Object Databases. *Interacting with Computers 14, 1* (2001), 31-68
11. Keränen, H., Plomp, J. Adaptive Runtime Layout of Hierarchical UI Components. In *Proc. of the 2nd Nordic Conf. on Human-Computer Interaction NordiCHI'02* (Aarhus, October 19-23, 2002). ACM Press New York, (2002), 251-254
12. Limbourg, Q., Vanderdonckt, J. Addressing the Mapping Problem in User Interface Design with UsiXML. In *Proc. of the 3rd Int. Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2004* (Prague, November 15-16, 2004). ACM Press, New York, (2004), 155-163
13. McKinley, P.K., Sadjadi, S.M., Kasten, K.P., Cheng, B.H.C. Composing Adaptive Software. *IEEE Computer 37*, 7 (July 2004), 56-64
14. Mitrovi , N., Royo, J.A., Mena, E. ADUS: Indirect Generation of User Interfaces on Wireless Devices. In *Proc. of 7th Int. Workshop Mobility in Databases and Distributed Systems MDDS'2004* (Zaragoza, August 30-September 3, 2004). IEEE Computer Society, Los Alamitos, (2004), 662-666
15. Navarre, D., Palanque, P., Paternò, F., Santoro, C., Bastide, R. A Tool Suite for Integrating Task and System Models through Scenarios. In *Proc. of 8th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2001* (Glasgow, June 13-15, 2001). Lecture Notes in Comp. Science, Vol 2220. Springer-Verlag, Berlin, 88-113
16. Nylander, S., Bylund, M., Waern, A. The Ubiquitous Interactor – Device Independent Access to Mobile Services. In *Proc. of 5th Int. Conf. of Computer-Aided Design of User Interfaces CADUI'2004* (Funchal, January 13-16, 2004). Kluwer Academics, Dordrecht, (2005), 271-282
17. Paternò, F., Santoro, C. One Model, Many Interfaces. In *Proc. of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002* (Valenciennes, May 15-17, 2002). Kluwer Academics Pub., Dordrecht, (2002), 143-154
18. Plomp, C.J., Mayora-Ibarra, O. A Generic Widget Vocabulary for the Generation of Graphical and Speech-Driven UIs. *Int. J. of Speech Technology 5* (2002), 39-47
19. Puerta, A.R., Eisenstein, J. Towards a General Computational Framework for Model-based Interface Development Systems. *Knowledge-Based Systems 12,* 8 (1999), 433-442
20. Schneider, K.A., Cordy, J.R. Abstract User Interfaces: A Model and Notation to Support Plasticity in Interactive Systems. In *Proc. of 8th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2001* (Glasgow, June 13-15, 2001). Lecture Notes in Comp. Science, Vol. 2220. Springer-Verlag, Berlin, (2001), 28-48
21. Sowa, J.F. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks/Cole Publishing Co., Pacific Grove, (2000)
22. Vanderdonckt, J., Bodart, F. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In *Proc. of ACM Conf. on Human Aspects in Computing Systems INTERCHI'93* (Amsterdam, April 24-29, 1993). ACM Press, New York, (1993), 424-429