UIPLML: Pattern-based Engineering of User Interfaces of Multi-Platform Systems

Nguyen Thanh-Diane, Jean Vanderdonckt
Louvain School of Management
Université catholique de Louvain
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
{thanh-diane.nguyen,jean.vanderdonckt}@uclouvain.be

Ahmed Seffah
Innovation and Software, School of Business and Management, Lappeenranta University of Technology
P.O.Box 20 – FI-53851 Lappeenranta (Finland)
ahmed.seffah@lut.fi

***Abstract*— Information systems become more accessible as a service offered to anybody, anywhere, at any time, via almost any device and computing platform. The continuous growth and the heterogeneity of these devices induce diverse user experiences depending on the device and challenge designers to creating methods and tools for engineering usable, yet accessible, information systems. Instead of repeating a similar development life cycle, design patterns concentrate design solutions with embedded usability and accessibility. Once a pattern is selected, the developer is responsible for adequately program the pattern code, which is a tedious and error-prone task. In order to address these challenges, this paper presents UIPLML (User Interface Pattern Language Markup Language), a XML-compliant markup language for defining user interface patterns for multiple contexts of use, e.g., for different users carrying out a task on different devices in different environments. A meta-model with new expressiveness enables multi-facet pattern matching. To validate it, four UIPLML pattern databases have been created: a base of 237 entries for multi-platform systems, a base of 42 entries for context-aware interfaces, a base of 10 entries for culturally-aware interfaces, and a base of 52 entries for accessibility. The master/detail pattern is in particular supported by a software for generative pattern-based approach where application parameters and contextual data govern automated user interface XML creation which, in turns, generates code for multi-platform information systems.**

Keywords*— *Design pattern; generative pattern; multi-context information system, multi-platform; pattern language; user interface description language, user interface pattern

I. INTRODUCTION

Nowadays, end users choose how, when, and where they would like to carry out their interactive task: they are no longer tied to a particular device or environment. They benefit from *Multi-context User Interfaces* (MUIs), which are User Interfaces (UIs) attached to an interactive information system or service for multiple contexts of use [1,2], which is hereby referred to as a user, a device, and an environment [3]. However, MUIs designers and developers face various challenges, such as a technical development and components to accommodate different types of users [2], devices/platforms [4,5], and environments [6] that result into context-aware systems.

UI patterns express a solution to a common UI design problem in a generic way that provides designers and developers with practical guidance on solution finding and application [2,7,8]. UI patterns set up the best design practices from distilled experience from real life. A pattern language [2] formalizes the definition and description of a pattern with structured semantic and syntax [9]. It helps the pattern users in understanding them as well as supporting the pattern-based design.

Several issues remain open for current UI patterns languages as a way to represent and use patterns: although significant efforts, such as PLML [10], have been devoted to uniformly document UI patterns [11], no standard form has been accepted so far and widely used [12], patterns are delivered mostly in narrative text (*descriptive patterns*), there is no true validation (patterns are provided "as is") and no software for supporting their right application (no *generative patterns*) [13], the guidance for applying them remains on low level of abstraction [14] with a limited usability approach [15] or is too specialized to one level of abstraction, their interrelationships are static and not context-oriented because they are expressed independently of any context of use (which is defined as user, platform, and environment [3,16]), thus making them totally inappropriate for MUIs. Some formal reasoning [17] and methodology should guide their creation and their pattern languages, which is not the case [2]. More challenges in defining, using UI patterns are discussed in [18].

In order to address these shortcomings, this paper provides a framework and a methodology for defining and using MUI patterns based on UIPLML (User Interface Pattern Language Markup Language), a XML-compliant markup language defining explicitly a reference to any applicable context of use, such as for which user, for which task, for which device or platform, and in which environment. For this purpose, a meta-model defines the semantics of this pattern language and a XML schema is derived from this meta-model to specify the syntax of UIPLML. In addition to providing an explicit link to applicable contexts of use, UIPLML enables creating a MUI by linking the pattern to a multi-level definition in UsiXML (User Interface eXtensible Markup Language), a User Interface Description Language (UIDL) that enables the semi-automated generation of MUI code [19], here considered in HTML5.

Since validation matters [15], an application of this framework is demonstrated through four databases of UIPLML patterns for different MUI dimensions. Instead of relying on a UI peculiar pattern language, UIPLML enables the creation of different pattern databases which could be related to each other and be complemented. UIPLML's separation of concerns facilitates the introduction of a new UI pattern in the collection.

This paper is structured as follows: Section 2 discusses the gap existing between the various MUI stakeholders in the literature: author (documentation), UI designer (quality), and software developer (implementation). Section 3 explains a meta-model for MUI design patterns into abstraction models and the language resulting of it. Section 4 shows the approach on 4 pattern databases and a software to support pattern description and its application and Section 5 presents the conclusion.

II. STATE OF THE ART

A. User Interface Patterns

Alexander introduced a pattern language to encapsulate in a common structure real world elements in order to describe good design practices to be reused in other similar cases, thus simplifying the problem solving part of development [20]. The Gang-of-Four later on introduced design patterns for object-oriented software development based on a format [21] covering behaviour, functional and structural aspects, such as in Coad-Nprth-Mayfield pattern collection [22]. Similarly, several UI pattern collections –representative examples being Tidwell's "patterns for effective interaction design" [23], van Welie's design patterns [24], pattern-based approaches of Borchers [26], Granlund [26], Pemberton [27], Perrins [28], Portland [29], Coram [30]– have been progressively introduced through simultaneous, yet uncoordinated, efforts that typically reflect different viewpoints: the UI pattern author/user (documentation), the UI designer (quality), and the software developer (implementation). These viewpoints define fundamental properties required by a UI pattern to be effectively used [2,8,23,31,32,33]:

- Provide a predefined structure for the implementation.
- Document encapsulation, capture detail, existing, well-proven design experience.
- Contain a local, self-standing process prescription (realization), generativity (capacity of automation) characteristic.
- Provide a homogeneous description, common semantic and syntaxes of UI pattern description.
- Identify reusable properties for software development.
- Separate functional from non-functional design aspects.
- Guide its implementation via a high level of abstraction.
- Deal with complexity of software in providing information implementation.
- Maintain some equilibrium between forces and constraints elements (minimization conflict between both elements).

UI pattern collections fall into three categories (Table I): *pattern catalogues*, which deliver lists of patterns in fixed collection without any evolution, typically in a book or a web site, *pattern managers*, offering a method-template approach to create, edit, delete patterns), and *pattern-based tools*, which provide some support in applying a selected pattern, typically through code generation. Table I compares UI pattern collection against two properties [13]: *descriptivity* refers to the UI pattern's ability to fully describe the conditions of application and the provided solution and is decomposed into *expressivity* (levels of details of the description) and *generality* (to which extent is this description generalizable); *generativity* refers to the UI pattern's ability to support pattern application, in particular through code generation and is decomposed into *coverage* (what is the scope covered by the support) and *genericity* (to which extent is this application generic enough to support MUIs). Table I compares UI patterns according to Harvey's Balls: '◔' when some limited elements, directives, examples exist, but no particular context is considered,'◐' when at least one context is specified, '◕' when more than one context is considered, '●' when patterns are described with two or more languages for many contexts of use, thus supporting MUIs. An empty field means either that no support exists or information is unavailable regarding this criteria in the literature.

Pattern collection		Descriptivity		Generativity	
		Expressivity	Generality	Coverage	Genericity
Pattern catalogues	Tidwell [23]	●	●	◔	◔
	van Welie [24]	●	●	◔	
	van Duyne [34]	●	◑	◑	
Pattern managers	Borchers [25]	●	●		
	Pemberton [27]	◑	◑	◑	
	Coram [30]	●	●	◑	
Pattern tools	Molina [35]	●	◑	●	●
	Henninger [36]	◑	◑	◑	

TABLE I. COMPARISON OF PATTERN COLLECTIONS.

Most UI patterns do contain information enough about the pattern itself (expressivity matters), but the level with which this description could be generalized to other cases is more or less supported (generality). Regarding generativity, only few patterns consider multiple contexts of use (coverage) and, when they do, their coverage is somewhat limited and their capability to actually generate a MUI is almost non-existent (genericity). One notable exception is JUSTUI [35], where a Presentation Model contains a set of conceptual patterns, which enable the design to model a MUI consisting of interaction units resulting from the application of patterns. The Presentation model is context-independent, but enables the developer to automatically generate UI code for three platforms: HTML, Java, and C++. A user model is currently being added in order to expand application to various users in their environments.

B. Comparison of UI patterns descriptions

The major UI pattern collections have been subject to a systematic analysis according to an analysis grid reported in Annexe1, allowing to identify the following shortcomings [11,31]:

S1. Lack of consistency. Collections deliver patterns according to different sets of attributes, links based on different taxonomies, and illustrations and application conditions with a varying level of details, thus making them inconsistent. Some attributes are revealed as being homonyms, homographs across collections. There has been some effort in proposing a consistent format, like PLML [10], but it is still inconsistently used, probably because of its incompleteness. van Welie [24] does not link any pattern, the "Gang of four" [37] and Portland [29] have only a related pattern.

S2. Lack of structure. Collections primarily present patterns as a flat list of attributes, with application conditions scattered across these attributes. When some structure emerges, it is not uniform. Tidwell presents a collection according to a question/answer allowing a quick understanding of the contents, but it is the only one. No conceptual model [23].

S3. Lack of implementation information. Conditions for applying a pattern are sometimes omitted, sometimes partially provided. Moreover, when such conditions are explained, the link with implementation is almost non-existent. Tidwell offers attributes "How" and "Figure" with regard to the information on the implementation; van Welie [24] has the attribute "How" to pilot the developer in implementing the pattern with example and explanation. The "Gang of four" [37] gives one concrete pattern implementation with a class diagrams while Portland [29] gives more freedom to users but defines a "story" in which they may find themselves.

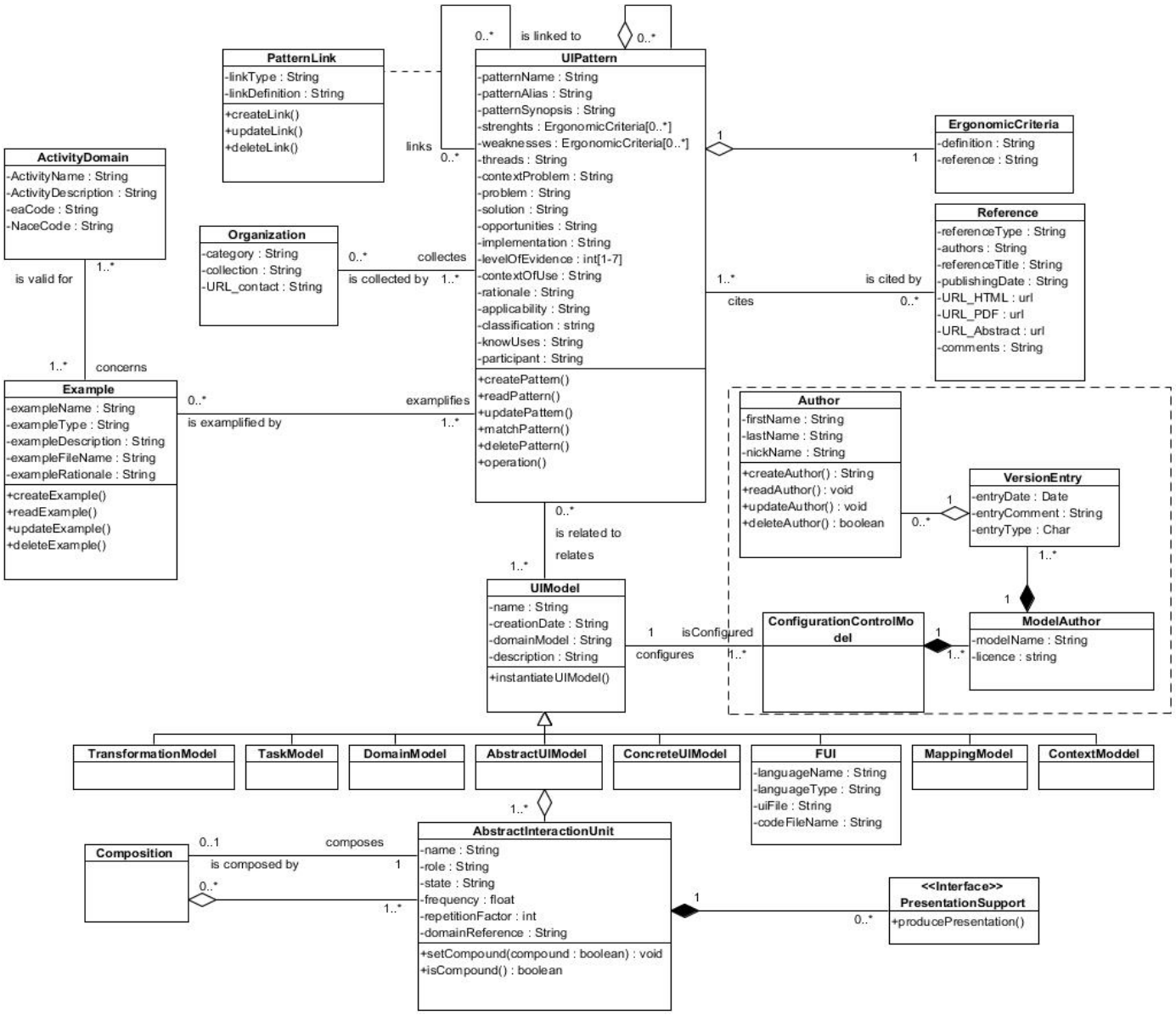

Fig. 1. The meta-model of UI patterns as described in UIPLML.

S4. Lack of pattern evaluation. Most pattern languages and collections do not provide any evidence regarding if the solutions are efficient in solving the identified problems. van Welie [24] enables its users to post comments on UI pattern usage, but this is far away from an impact factor. For a better assessment, some pattern languages miss code examples-implementation like UsiPXML [39].

S5. Limitation of contextual coverage. Most collections restrict the context of use (which covers user, platform, and environment) to solely the device or platform. In doing so, they usually focus on one device at a time [21], therefore providing little or no support to MUIs. The description of these contexts is mostly narrative and unstructured.

S6. Limited software support. Little or no software support exist for most collections [25,27,30], neither in searching for a pattern nor for matching one and applying one to a particular context of use. A notable exception is [35], which enables the designer to create a UI model based on patterns that lead to a MUI for three contexts of use.

S7. Specific pattern examples. While UI patterns are assumed to deliver a solution that is independent of any context of use, including any interaction modality, nearly all collections provide examples expressed as a Graphical User Interface (GUI) for a specific operating system or rendered in a specific environment that is not straightforward to transfer to another one. No other expression of a pattern exist, for instance at a higher level of abstraction than GUI [30,40].

S8. Limited usability consideration. Applying a pattern to a particular case study poses the question of deciding between various design options, thus leading to a usable solution with varying degree of quality. Most collections provide little or no information about the potential usability or user experience resulting from applying a pattern [14]. For instance, the usability approach of PaMGIS Patten Description Language (PPSL) (including PLML v1.1 and based on [3]) is limited by the use of comments like feedback.

In order to address these shortcomings, the following requirements have been elicited, also based on [15,39]:

R1. Structured and consistent conceptual model for MUIs.
R2. Explicit consideration of context of use for context-awareness and usability in pattern application.
R3. Integration of patterns in the whole development process.
R4. Consolidation of methods and techniques via software.
R5. Validation of patterns with an up-to-date usability guide.

III. User Interface Pattern Markup Language

A. UIPLML Description

In order to address requirements R1-R3 primarily and somewhat R5, Fig. 1 depicts the meta-model of User Interface Pattern Language Markup Language (UIPLML) represented as a UML 2.0 class diagram. This meta-model has been obtained by compiling all attributes from existing collections (Appendic I), but also significantly expanded several branches required for MUI development. The main aspects of this meta-model are hereby introduced, defined, and discussed.

The UIPattern class is firstly enriched by a complete SWOT analysis in which Strengths and Weaknesses are internal attributes referring to the usability quality properties that are ensured, respectively endangered by the pattern (here, ergonomic criteria are used [12]), whereas Opportunities and Threats are external attributes addressing the risks of pattern application. For example, the screen size may become a constraint in applying the pattern, thus representing a threat. The contextProblem attribute provides a reference to a context of use in which the pattern could be applied, where the contexts consists of a reference to a user model, a device model, and an environment model, while the resultingContext attribute describes the final context in which the MUI will be obtained. The levelOfEvidence attribute characterizes a scale of empirical evidence ranging from 1 (no evidence is provided in applying this pattern) to 7 (several different sources concur to conclude that this pattern has some impact on quality of MUIs). The rationale attribute justifies the pattern by providing a link to references where the pattern comes from; the rationale can range from '*weak*' when a pattern is suggested by a person to '*strong*' when a pattern is officially pushed. The implementation attribute authorizes three different approaches for implementing a pattern: *(i)* provide a code fragment, which is only possible for one context at a time and not for all MUIs, *(ii)* describe the pattern implementation using a high level notation, and *(iii)* provide a reference to one or many UIModels that describe the pattern at 4 levels of abstraction [3]:

1. A *Task Model* is a hierarchical decomposition of a global task, with constraints expressed on and between the subtasks. A notation exists to express a task model. The Domain Model completes the task model with a presentation of important entities of the particular application domain together with their attributes, methods and relationships. A domain model is typically expressed as a UML 2.0 class diagram.
2. The *Abstract User Interface* (AUI) model specifies a user interface independently of any interaction modality (we do not know yet whether it will be graphical, tactile, gestural, vocal, or multimodal in the future) and any technological space.
3. The *Concrete User Interface* (CUI) model specifies a user interface independently of any technological space, but for a given interaction modality. That means to define widgets layout and interface navigation independent of any computing device in case of a graphical user interface (GUI) or to define vocal elements in case of a vocal interface.
4. The *Final User Interface* (FUI) consists of source code, or a code skeleton in any programming (e.g., C++) or markup language (e.g. HTML5), which can be interpreted/compiled.

The AUI model represents a unique opportunity to describe a pattern independently of any context of use by definition, which is compliant with the W3C recommendation (www.w3.org/TR/abstract-ui/). Each UIPattern could be further decomposed into sub-patterns or linked to other patterns according to the following taxonomy of links (which is unique to UIPLML): isRelatedTo, ContradictsWith, isMoreImportantThan, isLessImportantThan, ShouldBeConsideredWith, MustBeConsideredWith, CouldBeConsideredWith ("MOSCOW" requirement). When appropriate, a UIPattern could be documented with an explicit link indicating which organisation, if any, is recommending, for instance a software vendor (e.g., Microsoft style guide, Google Material), a standard firm (e.g., ISO9241), a governmental agency (e.g., ANSI HFES) or any other corporate environment (e.g., a corporate style guide).

A UIPattern may be illustrated by copious examples which are themselves structured and attached to any possible domain of human activity. An activity domain is a set of tasks that are relevant to a specific context and is characterized by a list of EU NACE codes (Statistical Classification of Economic Activities in the European Community), e.g., Economics, Manufacturing. The three values for the implementation are thus covered: (i) by code: the link to a FUI gives access to a code fragment in particular language that may serve as a template or skeleton –note that this remains a manual approach, (ii) by model searching: the links to the various models enables the designer to look for patterns satisfying some conditions imposed on these models, and (iii) by multi-model: several models could be combined in the pattern. The MappingModel establishes mappings between models involved in a pattern, which may be mappings from T&C level downwards to FUI or transformations, such as Model-2-Model transformation from T&C to AUI, AUI to CUI, and Model-2-Code from CUI to FUI. Any UIModel may be subject to a configuration model enabling the pattern writer to maintain multiple versions of a pattern evolving over time, with different versions introduced by different authors.

B. UIPLML Syntax

To date, UIPLML is the only UI pattern language expressing MUIs in a way that is **compliant** with W3C recommendations on CRF, abstract UI, and configuration models. The UML 2.0 class diagram of Fig. 1 defines the semantics of UIPLML. It has been transformed into both a XML Schema for XML description and a W3C OWL 2.0 ontology. In this way, any software tool that is compliant with this schema or ontology may exchange patterns according to a standard format, thus addressing requirements R4-R5.

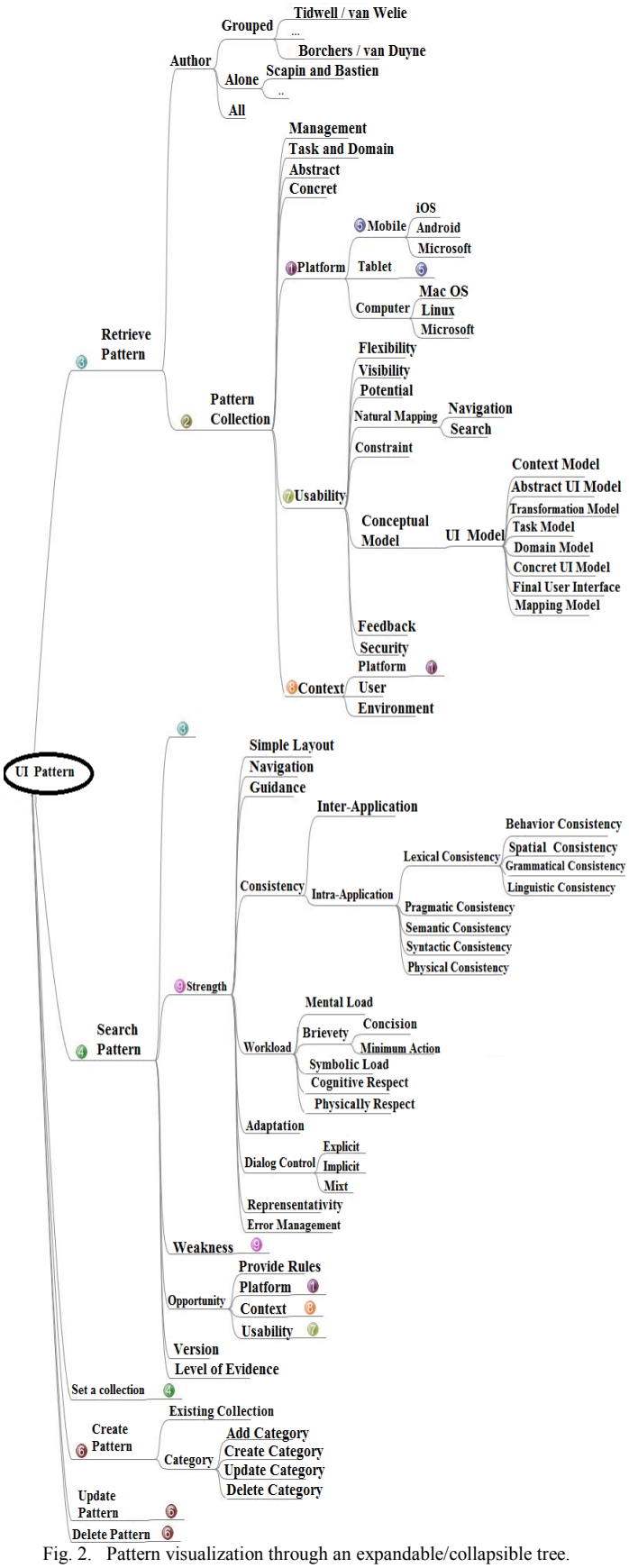

Fig. 2. Pattern visualization through an expandable/collapsible tree.

C. *UIPLML visualization*

Visualizing a pattern at once is impossible both from a machine screen viewpoint and from a human cognitive load viewpoint. Based on the UIPLM syntax, the categories, subcategories, attributes and their assigned values could be browsed according to an expandable/collapsible tree (Fig. 2). This tree is automatically produced from the XML-compliant description of a UIPLML pattern. Note in Fig. 2 that methods are also attached to a pattern depending on its status: for instance, a pattern promoted by an official standard cannot be modified nor deleted, but an instance could be created that becomes another pattern with a dependency link.

IV. APPLICATION OF UIPLML

A. *UIPLML Patterns databases*

In order to address requirements R4-R5 primarily with some impact on other requirements R1-R3, four UIPLML Patterns databases have been created as reported in Table II. Each pattern database is characterized by a name, an identifier, a main author (although other authors may edit patterns depending on their rights), and a description. The UIPLML patterns database system consist of two different pieces of software:

1. A *back-office*, representing the editor viewpoint (Fig. 3), is aimed at patterns authors and writers for them to create, retrieve, update, and delete patterns and patterns database. This back-office is implemented as a Java Web Start application (with .jnlp suffix) that can be downloaded from the web site and installed on the pattern writer/author workstation. After authentication, the writer is enable to edit any pattern database depending on the rights assigned. Each pattern database is managed by Atoms®, a multi-platform database management system. A pattern could be created first and then linked to other resources, such as users, platforms and devices, tasks, environments, activity domains, examples, links with other patterns (Fig. 3) or the other way around. The editor could first create individual resources such as users, platforms, and environments, and then link each of them to a particularly applicable pattern. At any time each pattern could be assigned to a status: inactive incomplete, inactive complete, and active. By drag and drop, the write may change the status of any pattern so as to make it visible for the second part. All patterns are stored in Atoms as XML documents which are compliant with the UIPLML syntax (Sub-section III.B).
2. A *front-office*, representing the end user viewpoint (Fig. 4), is aimed at patterns users for them to search for patterns, to apply pattern matching and use patterns databases. This front-office is implemented as a responsive web site in PHP and JavaScript that accesses the Atoms®-managed pattern databases. While the pattern writer may move a pattern from one database to another, the pattern user can access only one pattern database at a time.

Database	Entries	Purpose
MUI patterns	237	UI patterns for multi-platform (smartphone, tablet, notebook, PC, PocketPC, PDA) structured into 8 categories (windowing, home, menu, contents, actions, forms, help, specific)
Context-aware patterns	42	UI patterns for context-aware user interfaces
Culture-aware patterns	10	UI patterns for user interfaces to be localized/globalize for different languages, culture
Accessibility patterns	51	UI patterns addressing accessibility of graphical user interfaces depending on user

TABLE II. UIPLML PATTERNS DATABASES.

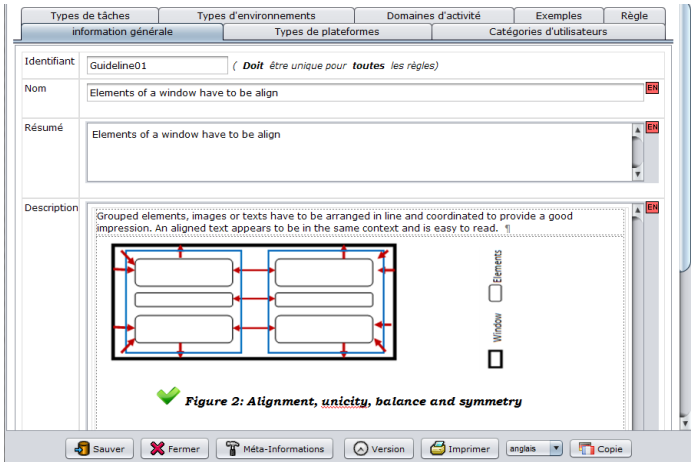

Fig. 3. A UI pattern in editing mode.

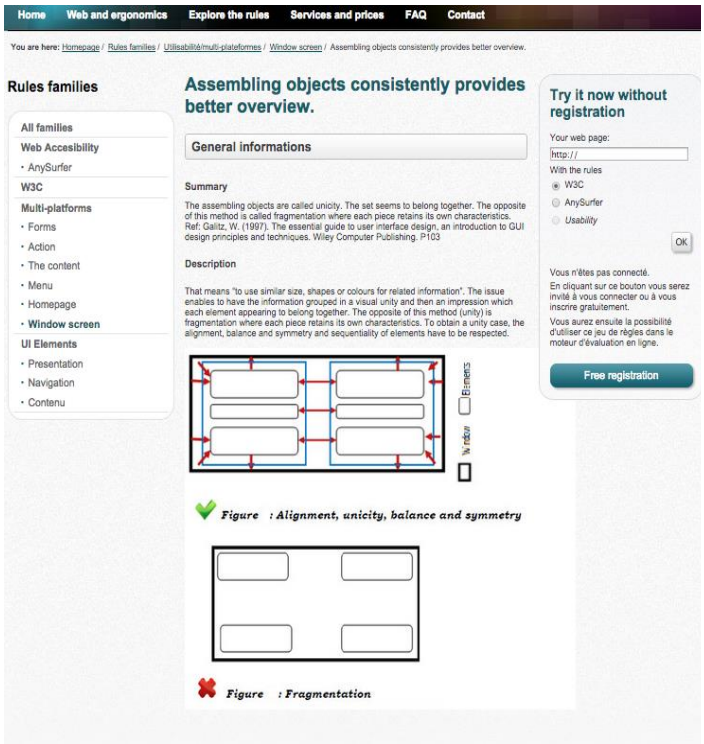

Fig. 4. A UI pattern in viewing/browsing mode.

B. UIPLML Pattern matching

Based on the meta-model defined in Fig. 1, pattern matching could be ensured by querying a UIPLML pattern database according to the following schemes:

- *Single-criteria single-class searching*: a query could be issued on any attribute of the UIPattern class on one database at a time. For instance, search for all patterns having a particular keyword, a particular string in the name (e.g., all patterns for lists), having a high level of evidence.
- *Single criteria multi-class searching*: a query could be issued by satisfying a constraint on a relationship between UIPattern and a related class. For instance, search for all patterns having "compatibility" as an ergonomic criteria for opportunities, search for all patterns documented in a particular bibliographical reference (useful for standard compliance), search for all patterns supported by examples in a particular domain of activity (e.g., all examples of forms for business administration), search for all patterns submitted by a particular author, documented in a given organisation.
- *Multi-criteria multi-class searching*: a query could be issued in order to satisfy a combination of the two aforementioned searching. For instance, search for all patterns that are empirically validated (having a high level of evidence) for online marketing domain (with a NACE code for marketing).
- *Full-text searching*: a query could be issued to perform a full-text searching based on a string on the complete UIPattern description. In this case, all attributes are considered for the search.
- *Single-model searching*: a query could be issued based on a particular model linked to patterns. For instance, give me all patterns for task model manipulating CRUDS methods (i.e., create, read, update, delete, search), give me all available patterns for a particular platform, say a smartphone. For the moment, only attributes of the models could be searched. A natural extension of this search would be the capability to search for models based on their structure and/or relationships, but this goes beyond the scope of this paper.
- *Multi-model searching*: a query could be issued based on properties of two or more models at once, possibly bound to each other by a mapping. Again, only attributes of the models and the mapping are supported. For instance, give me all patterns for a particular user carrying out a task on a specific device or independently of any platform/device.

C. UIPLML Master/Details Pattern

In order to explicitly address requirement R4 on software support, this section will focus on one particular design pattern that is particularly suitable for software support and for code generation. We selected the **Master/Details** (M/D) pattern because it is probably one of the most frequently used pattern, which exhibits many variations for MUIs, depending on constraints imposed by the context of use. A M/D pattern is typically selected to support a focus+context interactive approach in which a list of items, called *master*, is firstly displayed to provide an initial context, from which a particular item is selected that is subject for a detailed display, called *details*. The details need to have a usable navigation and to follow some us-

ability guidelines in order to respect users' requirements and to have an appropriate user experience. Here is an excerpt of the M/D pattern according to UIPLML:

<PatternName>: Master/Details
<PatternAlias>: Master/Slave, Director/Details
<Problem>: The user has to search in a list and select an item to have more details. A set of information units linked or not by a relationship has to be presented to the user. In this last scenario the master interaction unit determines information that details interaction unit will show.
<PatternSynopsis>:displays a master list and the details for the currently selected item.
<Classification/Template>: Structural/object centric
<Solution>: Perform a composed presentation in which master and detail data are shown in a synchronized way. In the master unit, its object guide and trigger the update in the details unit. Detail unit presentation is provided while Master unit presentation is changing.
<Restriction>: The constraint is to have synchronized information between the master information units and detail information units.
<Forces>: This pattern is used in numerous situations and contexts. The scenario of this pattern allows simplifying the user task. Indeed, navigation is decreased to get specific information. Moreover, information is maintained synchronized between the master and details units.
<Weaknesses>: The size of screen can discourage the presentation of this pattern. Less information can be shown at the same time on a screen.
<Applicability>: The M/D pattern is used when we need to interact with several objects aggregated.
<Structure>: In the case of an aggregated relationship, the master unit is the head element of details unit.
<Collaborations>: Objects can operate though their aggregated relationship or attributes.
<Participants>: One or two instances with an aggregated relationship. It is the presentation and instance interaction unit linked in the navigation with the M/D pattern.
<Known Uses>: Commercial system can use this pattern to show the case Invoice/Line.
<Consequences>: Need to adapt this pattern on different platforms. Knowledge about these devices is required.
<PatternLink>: Object presentation, Population unit, Instance Interaction Unit, Display Form, Table List Pattern
<Rationale>: Provide a presentation to reduce several navigation steps and to simplify the user task. Users need to interact with several objects aggregated or not. The scenario offered by M/D pattern allows getting details information aligned with its master component. Moreover, the purpose of this pattern is to make explicit information related to an instance.
<Context of use>: All type of users can use this pattern. All environments such as business environment can get this pattern and adapt it. For instance, we can use this pattern to show the cases Project/Employees or Invoice/Lines.
<Implementation>: The issue about using a unity class or aggregated classes is necessary before implementing this pattern.
<Threats>: Many systems offer the possibility to create this pattern into final user interfaces. The risk is to lack of graphical customization and interaction.
<Opportunities>: M/D pattern is frequently used in the real life. Different tools offer the possibility to no-expert developer (with few implementation software knowledge) to use template to perform it on specific operating system.
<levelOfEvidence>:7 (expert+validation description)

D. Task model for M/D pattern

A task model is a description of the tasks that a user will be able to accomplish while interacting with the system [14]. This description consists of a hierarchical decomposition of a global task recursively into sub-tasks, with constraints expressed on and between the sub-tasks. Two task models have been specified and encoded in UsiXML for the M/D pattern:

1. A task model (Fig. 5) in which a collection of objects (e.g., a list of cars) belonging to the same class (e.g., a CAR class) is subject to browsing first and each time one or many objects are selected, editing of respective attributes (e.g., changing the availability of a set of cars) and execution of respective methods could be ensured (e.g., a set of cars is booked at the same time for a company). A choice between sub-tasks is graphically denoted by a '[ ]' temporal operator, while a '|||" represents parallelism.
2. A task model (Fig. 6) in which the master is first manipulated as a collection of objects (e.g., a list of cars), and each detail (e.g., each car) is then subject to browsing, editing of attributes and/or executing related methods. A sequence with information passing is represented by "[ ]>>".

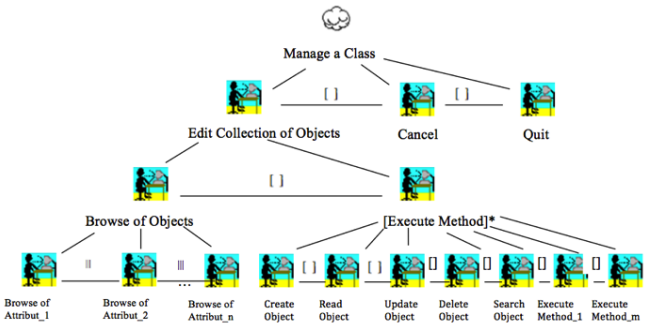

Fig. 5. Master/Details pattern defined by a unity class in the Task Model.

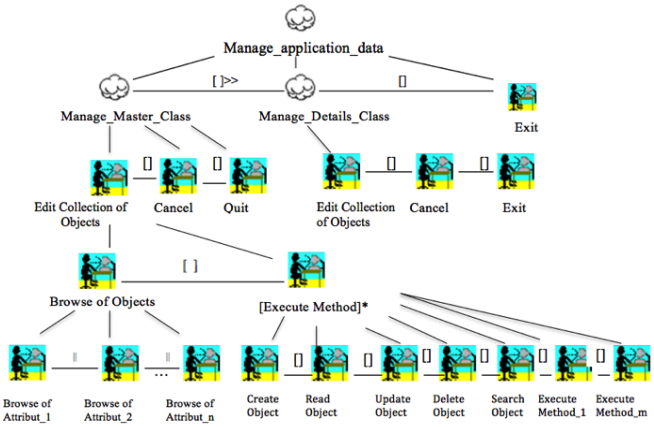

Fig. 6. Master/Details pattern defined by an aggregation relationship.

The second task model of the M/D pattern is related with an aggregation relationship between two different instances in a conceptual model [21]: "this pattern can be easily mapped to a many-to-one relation schema used within a database design [29]." For instance, an Employee class contains several attributes, some simple such as Name, LastName and some compound ones like Address, which could be further subject to another application of a M/D pattern since this repetitive attribute is decomposed into elements such as street, number, PO Box, zip code, city, and country. This could be mapped in an aggregation relationship between two distinct entities if the cardinality of the relation from master class to details class is 0..*n*.

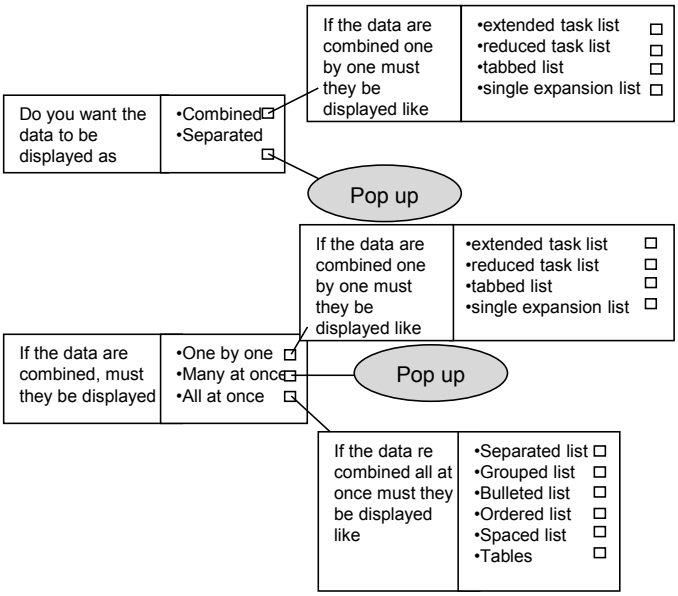

Fig. 7. Two nodes of UsiMAD decision tree: (a) a question with one answer and a sub-tree with further questions; (b) a question with one answer and two sub-trees with further questions.

E. UsiMAD for supporting M/D pattern

UsiMAD (http://usimad.alwaysdata.net/tree) is aimed at supporting the designer in exploring alternative MUI designs for the same M/D pattern by assigning different values to various design options and by immediately previewing (with Balsamiq mockups [5]) how the future MUI corresponding to these assigned options will look like. UsiMAD then enables the developer to save the resulting MUI pattern application into a XML file that is then imported in the UsiXML software suite for HTML5 code generation, which is available mainly for three computing platforms or devices: smartphone, tablet, and desktop or all at once through responsive design HTML.

Since UsiMAD is equipped with a decision tree where each *node* represents a *question* (e.g., how would like to present the details of a M/D?) and the *branches* represent the possible *answers* to this question (Fig. 7a,b). When an answer is considered final, the decision tree has reached a leaf node (one in both Fig. 7a and 7b); when an answer is not considered final, the decision tree then branches to the sub-tree corresponding to the selected answer (one sub-tree in Fig. 7a and two sub-trees in Fig.7b) and so forth until a final conclusion is reached.

Fig. 8 graphically depicts a portion of UsiMAD's decision tree showing options for the M/D pattern. Note that questions asked in this decision tree are located at AUI level since there no reference to any interaction modality nor any technological space like a particular platform. Returning to our scenario on applying a M/D pattern to a range of cars, a designer may first decide the amount of items to be displayed simultaneously: one item of the collection at a time (i.e., cars one by one), many items in a group (i.e., a partial list of cars), or all items resulting from the search at once (i.e., a complete list of cars).

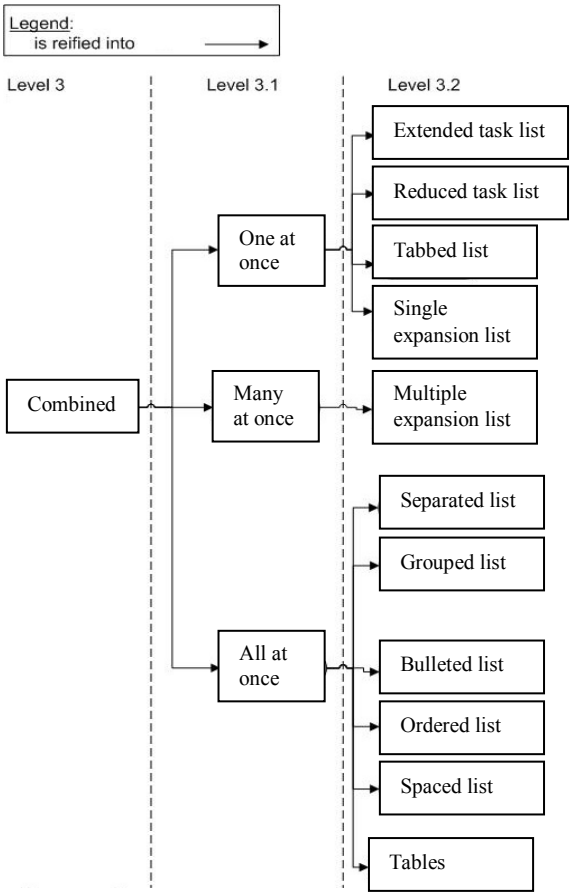

Fig. 8. A portion of UsiMAD decision tree for M/D pattern.

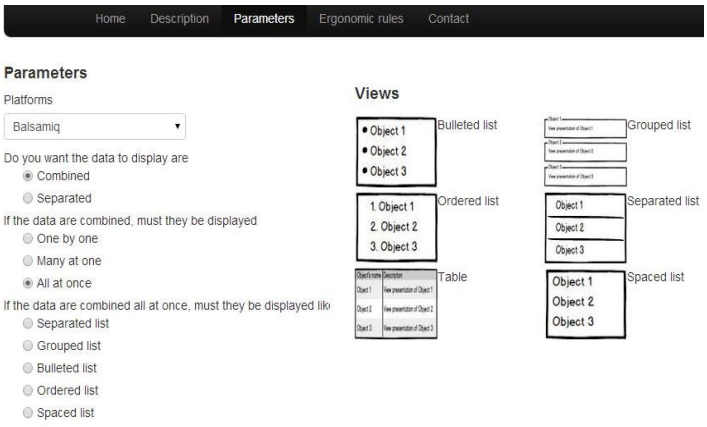

Fig. 9. One M/D pattern application in UsiMAD.

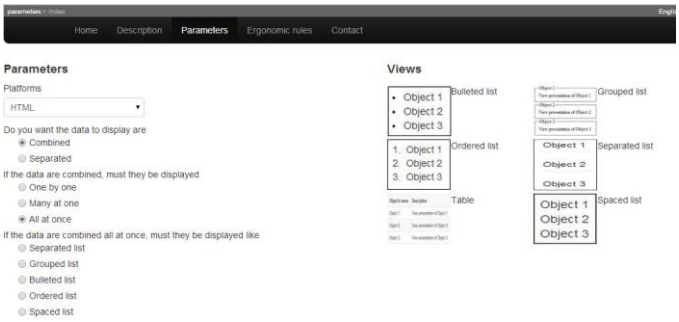

Fig. 10. Another M/D pattern application in UsiMAD.

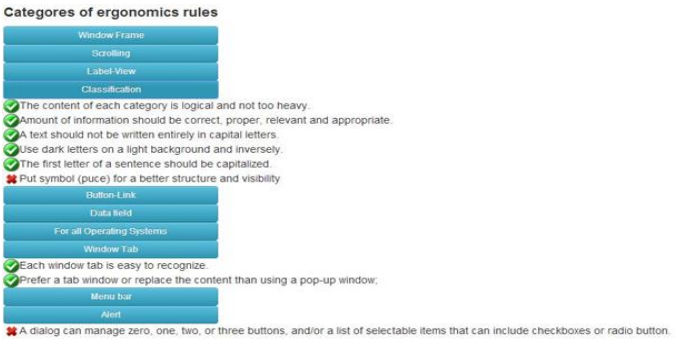

Fig. 11a UsiMAD Usability knowledge as rationale behind the M/D pattern.

Fig. 11.b UsiMAD Usability knowledge as rationale behind the M/D pattern in validation view

Depending on this choice, UsiMAD's decision tree directs the designer to a new question: how attributes and methods of the item should be conveyed, here in a single expansion list, and so forth. Depending on the specifications of the user and the device/platform, UsiMAD may suggest an appropriate value based on usability knowledge available until a final node of the decision tree is reached. Fig. 9, respectively Fig. 10, reproduces two final situations, one for a smartphone and one for a tablet.

Fig. 11.a and b. reproduce a situation where the designer is requesting explanation of the recommendation suggested by UsiMAD. For this purpose, a section of various categories of usability guidelines (here called *ergonomic rules* [42]) is opened that is further expanded to discover underlying guidelines that are either respected (a green check box is displayed) or not (a red cross-as a result of the parameters selection). If the designer becomes aware of the usability respected or not by this parameter selection, another selection may be operated as depicted in the end of Fig. 11 for another selection. Once design options have been decided (note that default values are automatically assigned by UsiMAD), a transition from AUI to CUI, and from CUI level to FUI could be achieved. The cross-device pattern is available for three targets: in HTML5 for both desktop and mobile, in a responsive version, in Android for smartphones and tablets, and in Objective-C for iOS devices.

A *Model Voyageur* (see http://sites.uclouvain.be/mbui/ for the full case study) enables the designer and the developer to navigate through the 4 CRF levels of abstraction [3] starting from a same task model (based on Fig. 5 or 6) and for a predefined domain model. In this on-line version, MUIs have been produced for different targets, depending on various users and devices. Any project for a particular case could be added, edited, or removed from this library. Another library could be created for another problem, with different task and domain models. Each time a branch is expanded, possible values for decided design options are visualized and the branch is further expanded to the subsequent levels, thus facilitating how decided design options may affect the final user interface.

V. CONCLUSION

This paper presented the following contributions:

- A *conceptual contribution*: a meta-model for expressing design patterns for multi-context user interfaces (Fig. 1), which goes beyond existing state-of-the-art pattern language for mono- or multi-context user interfaces, based on a systematic literature review comparing the expressiveness of existing pattern collections (Appendix I). For this comparison, a series of 8 shortcomings has been introduced and discussed in order to elicit 5 main requirements driving the usable a MUI pattern language. From this meta-model, UIPLML has been defined as a XML-compliant markup language for defining multi-context UI patterns, e.g., for different users carrying out a task on different devices in different physical environments.
- A *practical contribution*: UIPLML is available as a XML schema and as a W3C OWL ontology for importing and exporting MUI patterns across various collections of MUI patterns, provided that they adhere to the same format.
- A *methodological contribution*: how to express a MUI pattern according to the provided meta-model with explicit reference to various dimensions such as: user, platform or device, environment, and UIModel with different levels of abstraction which are the one recommended by W3C, thus making it compliant with the W3C MBUID [16].
- A *support of the method and language*: four MUI pattern databases have been created (respectively, with 237 entries for multi-platform systems, 42 entries for context-aware interfaces, 10 entries for culturally-aware interfaces, and 52 entries for accessibility) and incorporated into a UIPLML software consisting of a front-end and a back-office for MUI pattern management and usage, in particular providing 6 major mechanisms for MUI pattern matching. One particular pattern, i.e. the master/detail, has been subject to the implementation of UsiMAD, a software for supporting the designer and the developer in choosing appropriate values for design options collected for this pattern.

Future avenue of this work will focus on revisiting other collections of software design pattern, especially Coad, North, and Mayfield [22], in order to come up with a decision tree that is similar to the M/D pattern, also with MUI generation preview.

REFERENCES

[1] D. Malandrino, F. Mazzoni, D. Riboni, C. Bettini, M. Colajanni, and V. Scarano, "MIMOSA: context-aware adaptation for ubiquitous web access," Personal Ubiquitous Comp., vol. 4, no. 4, pp. 301-320, May 2010.

[2] A. Seffah, "Patterns of HCI design and HCI design of patterns - Bridging HCI design and model-driven software engineering," Human Computer Interaction Series, Berlin: Springer, 2015.

[3] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, "A Unifying Reference Framework for Multi-Target User Interfaces," Int. with computers, vol. 15, no. 3, pp. 289-308, 2003.

[4] E.G. Nilsson, "Design patterns for user interface for mobile applications," Adv. in Engineering Software, vol. 40, no. 12, pp. 1318-1328, Dec. 2009.

[5] S. Geiger-Prat, B. Marín, S. España, G. Giachetti, "A GUI Modeling Language for Mobile Applications," Proc. of 9th IEEE Int. Conf. on Research Challenges in Information Science RCIS'2015 (Athens, May 13-15, 2015), Piscataway: IEEE Press, pp. 76-87, 2015.

[6] K. Breiner, M. Seissler, G. Meixner, P. Forbrig, A. Seffah, and K. Klöckner, "PEICS: towards HCI patterns into engineering of interactive systems," Proc. of the 1st Int. Workshop on Pattern-Driven Engineering of Interactive Computing Systems PEICS'2010 (Berlin, June 20, 2010), New York: ACM Press, pp. 1-3, 2010.

[7] Ch. Märtin, Ch. Herdin, and J. Engel, "Patterns and Models for Automated User Interface Construction - In Search of the Missing Links," in Proc. of 15th Int. Conf. on Human-Computer Interaction: Human-Centred Design Approaches, Methods, Tools, and Environments HCI International'2013 (Las Vegas), M. Kurosu, Ed., Lecture Notes in Computer Science, vol. 8004, Berlin: Springer, pp. 401-410, 2013.

[8] E. Folmer, M. van Welie, Jand J. Bosch, "Bridging patterns: An approach to bridge gaps between SE and HCI," Information & Software Technology, vol. 48, no. 2, pp.69-89, 2006.

[9] S. Montero, P. Díaz, and I. Aedo, "A Semantic Representation for Domain-Specific Patterns," in Proc. of Int. Symposium on Meta-Informatics MIS'2004 (Salzburg), Springer, pp. 129-140, 2004.

[10] S. Fincher, PLML: Pattern Language Markup Language, University of Kent, February 2006, accessible at https://www.cs.kent.ac.uk/people/staff/saf/patterns/plml.html

[11] N. Li, Q. Hua, S. Wang, K. Yu, and L. Wang, "Research on a pattern-based user interface development method," in Proc. of 7th Int. Conf. on Intelligent Human-Machine Systems and Cybernetics IHMSC'2015 (Hangzhou, August 26-27, 2015), IEEE Press, pp. 443-447, 2015.

[12] J. Deng, E. Kemp, and E-G.Todd, "Focusing on a standard pattern form: the development and evaluation of MUIP," in Proc. of the 7th ACM Int. Conf. on Computer-Human Interaction: design centred HCI CHINZ'2006 (Christchurch), New York: ACM Press, pp. 83-90, 2006.

[13] J. Vanderdonckt and F. Simarro, "Generative pattern-based design of user interfaces," in Proc. of the 1st Int. Workshop on Pattern-Driven Engineering of Interactive Computing Systems PEICS'2010 (Berlin, June 20, 2010), New York: ACM Press, pp. 12-19, 2010.

[14] F. Radeke and P. Forbrig, "Patterns in task-based modeling of user interfaces," in Proc. of 6th Int. workshop on task models and diagrams for user interface design TAMODIA'2007 (Toulouse, November 7-9, 2007), M. Winckler, P. Johnson, Ph. Palanque, Eds., Lecture Notes in Computer Science, vol. 4849, Berlin: Springer, pp. 184-197, 2007.

[15] Y. Ormeño, J.I. Panach, N. Condori-Fernández, and O. Pastor, "Towards a proposal to capture usability requirements through guidelines", in Proc. of IEEE 7th Int. Conf. on Research Challenges in Information Science RCIS'2013 (Paris, May 29-31, 2013), IEEE Press, pp. 1-12, 2013.

[16] G. Meixner and G. Calvary, "Introduction to Model-based User Interface," W3C Working Group Note. World Wide Web Consortium, Geneva, 7 January 2014, accessible at https://www.w3.org/TR/mbui-intro/

[17] J. Engel, C. Martin, C. Herdin, and P. Forbrig, "Formal pattern specifications to facilitate semi-automated user interface generation," in Proc. of 15th Int. Conf. on Human-Computer Interaction HCI International'2013 (Las Vegas, July 21-26, 2013), Springer, pp. 300-309, 2013.

[18] P. Forbrig and A. Wolff, "Different kinds of pattern support for interactive systems," in Proc. of the 1st Int. workshop on Pattern-driven Engineering of Interactive Computing Systems PEICS'2010 (Berlin, June 20, 2010), New York: ACM Press, pp. 36-39, 2010.

[19] J. Guerrero, J. Vanderdonckt, and J. Gonzalez, "FlowiXML: a Step towards Designing Workflow Management Systems," Journal of Web Engineering, vol. 4, no. 2, pp. 163-182, 2008.

[20] C. Alexander, "A pattern language: towns, buildings, construction," Oxford: Oxford University Press, 1977.

[21] F. Islam, "On usability pattern documentation: an XML-based approach," Master thesis, Dept. of Computer Science, Concordia University, Montreal, October 2013, http://spectrum.library.concordia.ca/2394/

[22] P. Coad, D. North, and M. Mayfield, "Object models: strategies, patterns, and applications," 2nd ed., New York: Prentice Hall, 1996.

[23] J. Tidwell, Designing interfaces, patterns for effective interaction design, 2nd ed., New York: O'Reilly Media Inc., 2011.

[24] M. van Welie and G. van der Veer, "Pattern languages in interaction design: structure and organization," in Proc. of IFIP TC13 Int. Conf. on Human-Computer Interaction INTERACT'2003 (Zurich, September 1-5, 2003), Zurich: IOS Press, pp. 527-534, 2003.

[25] J.O. Borchers,"A pattern approach to interaction design," in Proc. of 3rd Int. Conf. on Designing Interactive Systems: processes, practices, methods, and techniques DIS'2000 (Brooklyn, August 17-19, 2000), New York: ACM Press, pp. 369-378, 2000.

[26] Å. Granlund, D. Lafrenière, and A. Carr, "A pattern-supported approach to the user interface design process," in Proc. of 9th Int. Conf. on Human-Computer Interaction HCI International'2001 (New Orleans), Mahwah: Lawrence Erlbaum Associates, Pub., 2001.

[27] L. Pemberton and R. Griffiths, "The Brighton usability pattern collection," 1999, http://www.it.bton.ac.uk/Research/patterns/home.html

[28] M. Perrins, "The 12 Patterns for User Interface Design", 12 Dec. 2008.

[29] Portland Pattern Repository, 1995, accessible at http://c2.com/ppr/

[30] T. Coram and J. Lee, "Experiences-a pattern language for user interface design," 2011, http://www.maplefish.com/todd/papers/Experiences.html

[31] A. Wolff and P. Forbrig, "Pattern Catalogs using the Pattern Language Meta Language," Electronic Communication of the European Association of Software Science and Technology, vol. 25, 2010.

[32] H. Javahery, "Usability pattern-oriented design: maximizing reusability of pattern languages over the web," Faculty of engineering and computer science, Concordia University, Canada, 2002.

[33] J. Engel, Ch. Märtin, and P. Forbrig, "A Concerted Model-driven and Pattern-based Framework for Developing User Interfaces of Interactive Ubiquitous Applications," in Proc. of 1st Int. Workshop on Large-scale and Model-based Interactive Systems: Approaches and Challenges, LMIS'2015 (Duisburg, June 23, 2015), vol. 1380, 2015, pp. 35-41.

[34] D. van Duyne, J. Landay, and J. Hong, "The design of sites, patterns for creating winning websites," NY: Prentice Hall International, 2006.

[35] P.J. Molina, M. Santiago, and O. Pastor, "User interface conceptual patterns," in Proc. of the 9th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2002 (Rostock, June 12-14, 2002), Lecture Notes in Comp. Science, vol. 2545, pp. 159-172, 2002.

[36] S. Henninger, "An Organizational Learning Method For Applying Usability Guidelines and Patterns," in Proc. of 8th IFIP Int. Conf. on Engineering for Human-Computer Interaction EHCI'2001 (Toronto, May 11-13, 2001), LNCS, vol. 2254. Berlin: Springer, pp. 141-156, 2001.

[37] E. Gamma, R. Helm, R. Johnson and J. Vlissides, "Design patterns: elements of reusable object-oriented software," Addison Wesley, 1995.

[38] J. Coplien and D.C. Schmidt, "Pattern language of program design," New York: Addison-Wesley, 1995.

[39] S. Wendler, D. Ammon, I. Philippow, and D. Streitferdt, "A factor model capturing requirements for generative user interface patterns," in Proc. of 5th Int. Conf. on Pervasive Patterns and Applications PATTERNS'2013 (Valencia, May 27-June 1, 2013), Wilmington: Int. Acad., Research, and Industry Association, pp. 34-43, 2013.

[40] PLMLx: Extended Pattern Language Markup Language, May 2003, accessible at https://www.cs.kent.ac.uk/people/staff/saf/patterns/diet helm/plmlx_doc/plml_doc.dtd.html

[41] J. Engel and C. Märtin, "PaMGIS: a framework for pattern-based modeling and generation of interactive systems," in Proc. of 13th Int. Conf. on Human-Computer Interaction HCI International'2009 (San Diego, 19-24, 2009), LNCS, vol. 5610, Berlin: Springer, pp. 826-835, 2009.

[42] D.L Scapin and J.M.C Bastien, "Ergonomic criteria for evaluating the ergonomic quality of interactive systems," Behaviour & Information Technology, vol. 16, no. 4/5, pp. 220-231, 1997.

[43] C. Kruschitz, "XPLML - a HCI pattern formalizing and unifying approach," in Proc. of ACM Int. Conf. on Human Aspects in Computing Systems CHI'2009 (Boston, April 4-9, 2009), Extended Abstracts, New York: ACM Press, pp. 4117-4122, 2009.

APPENDIX 1. ATTRIBUTES OF UI PATTERN COLLECTIONS

Elements		Attributes to define UI patterns in catalogues						Attributes from UI pattern framework									
		Gang of Four [37]	**Tidwell [23]**	**van Welie [24]**	**van Duyne [34]**	**Coplien [38]**	**Port-land [29]**	**PLML 1.1 [10]**	**PLML 1.2**	**XPLML [43]**	**PLMLx [40]**	**UPLML [21]**	**UIML 4.0**	**UsiPXML [14]**	**PPSL [33]**	**Minig *et al.* 2015**	**UIPLML**
PatternID		☒	☒	☒	Pattern ID	☒	☒	☑	☑	☑	☑	☑		☑	☑	☒	☑
Name		Name	Name	Name	Name	Name	Name	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
Alias		Know use	☒	Use When	Figure	☒	☒	☑	☑	☑	☑	☑		☑	☑	☑	☑
Problem		Intent	What	☑	Problem	☒	User decision	☑	☑	☑	☑	☑		☑	☑	☑	☑
Context		Applica-bility	Use when	Use When	Background	Context	User decision	☑	☑	☑	Resul-ting con-text	☑		☑	☑	☑	☑
Rational		Intent	Why	Why	☒	Resulting context	User decision	☑	☑	☑	☑	☑		☒	☑		☑
Imple-mentation	Type	Implementa-tion	How	How	☒	☒	Task	☑	☑					☑	De-ploye-ment	☒	☑
	Sample code	Sample code	☒	☒	☒	☒	☒							☑	☑		☑
Illustration		Motivation	Examples	More Ex-amples	Figure	☒	Task and Task Window	☑	☑	☑	☑	☑		☑	☑	☑	☑
Diagram (UML)		Structure	☒	How				☑	☑	☑	☑	☑		☑	☑	☒	☑
Evidence		☒	☑	☑	Problem			☑	☑	☑	☑	☑		☑	☒		☑
Confidence		☒	☑	☑				☑	☑	☑	☑	☑			☑	☒	
Solution		☒	☒	Solution	Solution	Solution	Task	☑	☑	☑	☑	☑		☑	☑	☑	☑
Forces		Consequence	☒	☒	☒	Forces	Task								☑	☒	☑
Weakneasses		☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒		☑	☒	☒	☑
Threats		☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒		☒	☒	☒	☑
Opportunities		☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒		☒	☒	☒	☑
Categorization		☒	☑	☑	☑	☑	☑	☒	☒	☒	☒	☒		collec-tion	☒	☒	☑
Pattern-Link	Type	Related pattern	In other librar-ies, How, why	Use when, How, why	Other patterns to consider, problem, background	☒	Task and Task Window	☑	☑			Associa-tion Mod-ule		Related pattern	Rela-tion-ship	☑	☑
	PatternID	☒	☒	☒	☑	☒		☑	☑	metadata		Associa-tion Mod-ule		Pattern link	Rela-tion ID	☒	☑
	Collection ID	☒	☒	☒	☒	☒		☑	☑		Organi-sation	Associa-tion Mod-ule		☒	Related UPID	☒	☑
	Label	☒	☒	☒	☒	☒		☑	☑			Associa-tion Mod-ule		☒	☑	☒	☑

Refe-rence	referenceID	☒	☒	Literature +Comments	☒	☒	Task	☒	☒			Associa-tion Mod-ule		litera-ture	☒	☒	☑
	referenceType	☒	☒	☒	☒			☒	☒					☒	☑	☒	☑
	authors	☑	☑	☑	☑			☒	☒					☒	☑	☑	☑
	referenceTitle	☑	☑	☑	☑			☑	☑					☒	☑	☑	☑
	Publishing date	☑	☑	☑	☑			☒	☒					☒	☒	☑	☑
	URL_HTML	☒	☒	☒	☒			☒	☒					☒	☒	☑	☑
	URL_PDF	☒	☒	☒	☒			☒	☒					☒	☒	☑	☑
	URL_abstract	☒	☒	☒	☒			☒	☒					☒	☒	☑	☑
	RefComments	☒	☒	☒	☒			☒	☒					☒	☑	☒	☑
Example	exampleID	☒	☑	☑	☑			evi-dence	evi-dence	☑	☑	☑		☒	☒		☑
	exampleName	☑	☑	☑	☑			☑	☑			☑		☒	☑	☑	☑
	exampleType	☒	☑	☑	☑			☒	☒			☑		☒	☑		☑
	exampleDescription	☒	☑	☑	☑			☒	☒			☑		☒	☑		☑
	ExampleRationale	☒	☑	☑	☑			☑	☑		☑			☒	☑		☑
UIModel	Activity Domain	☒	☒	☒	☒	☒	☒	☒	☒					☒	☒	☒	☑
	Authors	☒	☒	☒	☒	☒	☒	☒	☒		Man-age-ment				☑		☑
	Version: VersionID	☒	☒	☒	☒	☒	☒	Revi-sion num-ber	Revi-sion num-ber	☑	Revi-sion number			☑	☑		☑
	Version: Creation-Date	☒	☒	☒	☒	☒	☒	☑	☑					☑	☑		☑
	Version: Copyright	☒	☒	☒	☒	☒	☒	☒	☒								☑
	Version: licence	☒	☒	☒	☒	☒	☒	☒	☒								☑
	Version: ModifDate	☒	☒	☒	☒	☒	☒	Last modi-fied	Last modi-fied								☑
	Transform model	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒		☑		☒	☑
	Task model		☒	☒	☒	☒	☒	☒	☒	☒	☒	☒		☑	☑	☒	☑
	Domain model		☒	☒	☒	☒	☒	☒	☒	☒	☒	☒		☑	☑	☒	☑
	AbstractUI-model: AUI	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☑	☑	☑	☑		☑
	AbstractUI-model: Com-posi-tion	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☑	☑	☑			☑
	AbstractUI-model: Presen-tation-Sup-port	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☑	☑	☑			☑
	ConcreteUImodel	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒		☑	☑	☒	☑
	FUI model	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒		☑	☑	☑	☑
	Mapping model	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒		☑		☒	☑
	ContextModel	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒	☒		☑		☒	☑
		Gang of Four [37]	Tidwell [23]	van Welie [24]	van Duyne [34]	Coplien [38]	Portland [29]	PLML 1.1 [10]	PLML 1.2	XPLML [43]	PLMLx [40]	UPLML [21]	UIML 4.0	UsiPXML [14]	PPSL [33]	Minirg *et al.* 2015	UIPLML