



LOUVAIN

School of Management
RESEARCH INSTITUTE

WORKING PAPER

2015/21

Generative Patterns for Cross-
Platform User Interfaces
Engineering: The Case of the
Master Detail Pattern

Thanh-Diane Nguyen

Jean Vanderdonckt

Louvain School of Management

Ahmed Seffah

Lappeenranta University of Technology

Louvain School of Management

Working Paper Series

Editor : Prof. Frank Janssen
(president-ilsm@uclouvain.be)

Generative Patterns for Cross-Platform User Interfaces Engineering: The Case of the Master Detail Pattern

Thanh-Diane Nguyen, Louvain School of Management
Jean Vanderdonckt, Louvain School of Management
Ahmed Seffah, Lappeenranta University of Technology

Summary

HCI design patterns have been recognized to be important mean for gathering and conveying UI designs that have been proved usable, useful, and sometimes enjoyable. To become valuable, such design patterns should encode the structure of a solution and its associated forces, rather than cataloguing just a solution, often for a specific platform. We introduce the generative pattern as a way of both documenting and implementing HCI patterns. A generative pattern not only tells us the rules for implementing a UI design considered as a generic solution to a problem at different levels of abstraction (in the way that a UI could be modelled), but also shows us how to transform these expressions into programmable code for the diverse computing platforms, while being compliant with the style guide rules that may prevail for these platforms. As a case study, the Master-Detail (M-D) pattern, one popular and frequently used HCI design pattern, is developed: displays a master list of items and the details for any selected item. While this MD pattern is documented in very different, possibly inconsistent, ways across various computing platforms, the MD generative pattern consolidate these particular implementations into a high-level pattern description based on design options that are independent of any platform, thus making this pattern ‘cross-platform’. A framework provide developers and designers with a high level UI process to implement this pattern in using different instances and its application in some designated languages. Some examples of applying a MD generative pattern are explained as well as a particular implementation for the Android platform.

Keywords : Cross-context pattern, cross-platform pattern, Generative patterns, Master-Detail pattern, pattern matching,, user interface model

Corresponding author:

Prof Jean Vanderdonckt
Louvain Interaction Laboratory (LILab)
Louvain School of Management
Université Catholique de Louvain
Place des Doyens 1
B-1348 Louvain-la-Neuve, BELGIUM
Email : jean.vanderdonckt@uclouvain.be

Generative Patterns for Cross-Platform User Interfaces Engineering: The Case of the Master Detail Pattern

Thanh-Diane Nguyen, Jean Vanderdonckt and Ahmed Seffah

Abstract: HCI design patterns have been recognized to be important mean for gathering and conveying UI designs that have been proved usable, useful, and sometimes enjoyable. To become valuable, such design patterns should encode the structure of a solution and its associated forces, rather than cataloguing just a solution, often for a specific platform. We introduce the generative pattern as a way of both documenting and implementing HCI patterns. A generative pattern not only tells us the rules for implementing a UI design considered as a generic solution to a problem at different levels of abstraction (in the way that a UI could be modelled), but also shows us how to transform these expressions into programmable code for the diverse computing platforms, while being compliant with the style guide rules that may prevail for these platforms. As a case study, the Master-Detail (M-D) pattern, one popular and frequently used HCI design pattern, is developed: displays a master list of items and the details for any selected item. While this MD pattern is documented in very different, possibly inconsistent, ways across various computing platforms, the MD generative pattern consolidate these particular implementations into a high-level pattern description based on design options that are independent of any platform, thus making this pattern 'cross-platform'. A framework provide developers and designers with a high level UI process to implement this pattern in using different instances and its application in some designated languages. Some examples of applying a MD generative pattern are explained as well as a particular implementation for the Android platform.

Keywords: Cross-context pattern, cross-platform pattern, Generative patterns, Master-Detail pattern, pattern matching, Pattern implementation and representation, user interface model

7.1 Introduction

Similar to Software Engineering (SE-), a Humanc-Computer Interaction (HCI) design pattern consists of a general repeatable solution to a commonly occurring user/usability problem in various User Interface (UI) design projects (Javahery 2002). A HCI pattern is not considered as a finished UI design that can be programmed straightforwardly. Rather, it provides a description or a template to solve a problem that can be used in many different situations. The importance of implementing design patterns has been pointed out since its inception (Alexander 1977). Here, by implementation we mean designing an effective support of applying design patterns and transforming this intention into code. Each time the problem has to be addressed, the pattern application and transformation are applied, thus repeating each time yet another application-specific concrete implementation. Each such implementation often remains specific to one single context of use, i.e. one single user or category of users conducting an interactive task on one dedicated computing platform in a designated location, thus reducing the reusability of this concrete implementation for another context of use. If any dimension of the context of use changes, for instance a new user, a new platform, or a new location changes, a new transformation has to be applied that is not necessarily available for this new context.

Since the emergence of new devices offering a multitude of interaction styles and allowing every user to access to information and services from everywhere and at any time, cross-context patterns are required. Therefore, we hereby define *cross-context patterns* as a general repeatable solution to a commonly occurring task to be conducted in various contexts of use, possibly with different users, platforms, and locations.

Similarly, we hereby define *cross-platform patterns* as a general repeatable solution to a commonly occurring task to be conducted on various computing platforms, independently of user and locations. Consequently, the question arises how to structure and implement such a design pattern on a large myriad of platforms. The goal is to give the opportunity of HCI designers and UI software developers to “switch” patterns when engineering an application for diverse platforms. Because different design patterns may offer different advantages and suffer from different drawbacks depending on the platform, even if they are intended to support the same interactive task of the pattern, another design pattern could become more suitable for a system as it evolves. Or a different behavior observed for the same task that could not be realized by the original patterns might be needed later. However, we cannot switch design patterns using same implementation, as long as these application-specific implementations are derived from design patterns.

Platform capabilities and constraints largely influence the way a cross-platform pattern could be implemented on a target platform: the programming or markup language expressiveness, the operating system, the underlying development toolkit, the constraints imposed by the platform itself such as screen resolution, interaction style, interaction devices.

Rendering a cross-platform pattern could be achieved in two ways: (i) *by code generation*, when UIs are implemented in using a set of instructions of any programming language, whatever programming paradigm it follows, and/or a set of assertions of this language; (ii) *by interpretation*, when the UI is described by a declarative language or a User Interface Description Language (UIDL) to be interpreted at run-time by a rendering engine Typical examples of rendering by code

generation are: direct coding in a programming language such as C, Java, Visual Basic, model-to-code transformation in model-driven engineering such as in JustUI (Molina 2004), code generation techniques such as generative programming, template filling such as Velocity. Typical examples of rendering by interpretation are declarative languages such as HTML, XML, any UIDL or integrated environment like SAP or Oracle that produces their UI internally.

This chapter proposes a technique for describing and implementing a generative UI design pattern in general that improves reusability and effective applicability of HCI design pattern. It instantiates this general definition for a cross-platform pattern for a specific case such the Master-Detail pattern. For structuring and coding, the proposed technique relies on the concept of generative patterns and a set of rules for implementations. These rules can be encapsulated in several programming languages or UI development tools such as task modellers and UIDLs. More particularly, the M/D cross-platform pattern is systematically described according to the formalisms and notations recommended by W3C for model-based UI design. We describe this implementation for two platforms, i.e., a web application and a mobile application, and compare it to other languages and cross-platform environments.

The major contributions of this research are the following:

- A definition of the general concept of generative UI pattern is provided that expresses various aspects to consider when applying the pattern for multiple contexts of use
- An instantiation of this general concept as cross-platform UI pattern for applying it for multiple computing platforms
- An exemplification of a cross-platform UI pattern based on the Master-Detail pattern that is then subsequently detailed at the different levels of abstraction recommended by the Cameleon Reference Framework (CRF) (Calvary, Coutaz, Thevenin, Limbourg, Bouillon and Vanderdonckt, 2003)
- An abstraction of design options found in various computing platforms into a cross-platform M-D pattern, thus offering a wide range of options at once.
- An implementation environment of this cross-platform pattern for two platforms: iOS and Android.

The remainder of this chapter is structured as follows: Section 7.2 reviews various definitions of the M-D pattern found in the literature using classifications and illustrations. Based on shortcomings and requirements identified in this literature, Section 7.3 revisits the definition of the M-D pattern to transform it into a cross-platform pattern as intended. The following sections then respectively examine this pattern more closely at the various levels of abstraction of the Cameleon Reference Framework (CRF): Section 7.4 details the UI development life cycle of the M-D pattern by instantiating it at the task and domain, abstract, concrete and final UI levels respectively, Section 7.5 explains how to generate a UIDL-document to facilitate its implementation on cross-platform context and illustrates this framework on a case study of a “car rental” task; Section 7.6 concludes the chapter by discussing the contributions of our approach comparing it with respect to related work and by presenting some future avenues of this work.

7.2 Related Work

In order to substantiate this research, we decided to focus its discussion on the Master-Detail (M-D) pattern, also known as Master-Slave or Director-Detail pattern (Pastor and Molina 2007). This pattern has been selected for the following reasons: it starts from a domain model, thus offering a data-oriented perspective and a conceptual starting point; it is widely used both in the literature and in practice, by designers, developers, by private ones and software vendors; it is largely considered in systematic development of interactive information systems; previous work do not examine the cross-platform dimension of this pattern in the light of UI implementation and usability concerns; this pattern can be defined as a unified class or can be interpreted such as an aggregation in relationship between two different classes.

This section is divided into five sub-sections regarding five major dimensions of the M/D pattern: the definitions discussed in the literature, its classification in collection patterns, its generative form, its current engineering implementation based on illustrations, and the motivations to present this pattern into a cross-platform one.

7.2.1 Master-Detail Pattern – An Operational Definition

The M-D pattern is typically used in a single scenario where several tasks are performed at the same time, while maintaining the details synchronized related to its master (Pastor and Molina 2007). A M-D pattern is applied, like any pattern, to reflecting on possible changes to a technical space or situation (Nilsson 2009). By relying on the context, patterns can prevent repetitive errors in a cross-platform environment. That also allows understanding better possible impacts of new technologies, the screen resolution being probably one of the most constraining one. This pattern should be prescriptive to promoting creation of new instances in order to help designers in its implementation. The presentation of the M-D pattern for a wide variety of screen types defines how and which elements are suitable. The pattern can then be used to capture essential problems of different “sizes” in using different customizations. Therefore, the using of pattern for documenting design knowledge allows dividing “a large problem area into a structured set of manageable problems” (Nilsson 2009).

In HCI, a *master-detail interface* displays a master list of items, called *master area*, and the details for any selected item, called the *detail area*. A master area can be a form, a list or a tree of items, and a detail area can be a form, a list or a tree of items typically placed as close as possible to the master area (e.g., below or next to it) in order to satisfy the usability guideline: “Semantically-relation information should be placed close to each other to reflect this link, while unrelated information should be placed far from each other”. Selecting an item from the master area causes the details of that item to be populated in the detail area. A master-detail relationship is a one-to-many type relationship, among which typical examples are: a set of purchase orders and a set of line items belonging to each purchase order, an expense report with a set of expense line items or a department with a list of employees belonging to it. An application can use this master-detail relationship to enable users to navigate through

the purchase order data and see the detail data for line items only related to the master purchase order selected.

Fig.7.1 graphically depicts a master-detail by showing how to display information in soccer regarding teams that plays in a league. Let us assume that `LeagueList` is a collection of `Leagues`. Each `League` has a `Name` and a collection of `Divisions`, and each `Division` has a `name` and a collection of `Teams`. Each `Team` has a `team name`. The `Divisions ListBox` automatically tracks selections in the `Leagues ListBox` and displays the corresponding data. The `Teams ListBox` tracks selections in the other two `ListBox` controls.

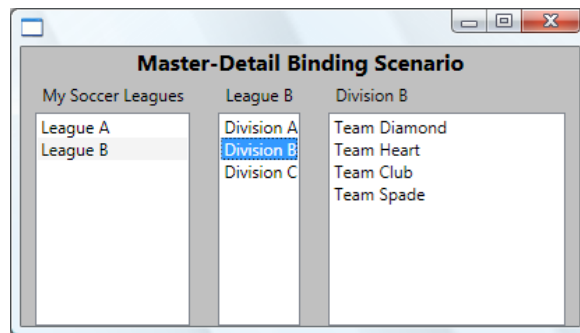


Fig. 7.1 An example of a UI implementation of the M-D pattern on desktop view.

Fig.7.1 also shows that the M-D pattern could be applied recursively (this is sometimes called Master-Detail-MoreDetail pattern):

1. The three `ListBox` controls bind to the same source. You set the `Path` property of the binding to specify which level of data you want the `ListBox` to display
2. You must set the `IsSynchronizedWithCurrentItem` property to true on the `ListBox` controls of which the selection you are tracking. Setting this property ensures that the selected item is always set as the `CurrentItem`. Alternatively, if the `ListBox` gets its data from a `CollectionViewSource`, it synchronizes selection and currency automatically.

7.2.2 The M-D Pattern usage in Pattern Collections

Multiple User Interfaces (MUIs) have to adapt to any variation of the cross-platform context. Using a pattern approach allows to design UIs by a set of models (the model-based UI development) (Seffah and Forbig 2002) and to provide high abstraction elements before coding the software. Several HCI pattern collections were introduced in the literature since Alexander (1977). “Using patterns to clearly and succinctly describe particular workplaces, in order to understand possible impacts of new technologies.” (Bayle, Bellamy, Casaday, Erickson, Fincher and Grinter 1998)

UI patterns are found more descriptive than generative in most pattern collections (Table 1.1): *descriptive patterns* are aimed at maximizing their descriptivity (i.e., the level with which they have described in the collection) and their genericity (i.e., the scope in which they are applicable) (Vanderdonckt and Montero 2010). To become descriptive, a pattern should solve a trade-off: contain enough information to foster its descriptivity, but not too much in order not to constrain its genericity. A *generative pattern* is aimed at maximizing its expressivity (i.e., the capability with which they are expressed in a rigorous way) and their generativity

(i.e., the level with which they could lead to a final user interface, manually or (semi-)automatically. To become generative, a pattern should solve another trade-off: express enough information to foster its expressivity, but not too much or not too informally in order to foster their generativity.

Table 7.1 compares some famous pattern collections according to the aforementioned four properties classified as: ‘+’ if the specification of the language is limited but we have some elements and directives with example to describe UI patterns. ‘++’ if the patterns are described with a specific language for one context and ‘+++’ if their description involving two and more languages for several contexts of use (user, platform or environment). An empty field means that not enough information belonging to this property is available in the publicly accessible literature to assess it.

Table 7.1 Classification of patterns collection according to the four properties: descriptivity, genericity, expressivity, generativity (From ‘=no information, +=low, ++=medium, to +++=high).

Classification of patterns collection				
Pattern collection	Descriptivity		Genericity	
	<i>Descriptivity</i>	<i>Genericity</i>	<i>Expressivity</i>	<i>Generativity</i>
Pattern catalogue				
Tidwell (2010)	+++	+++	+	+
van Welie (2003)	+++	+++	+	
van Duyne <i>et al.</i> (2006)	+++	++	++	
Management patterns				
Borchers (2000)	+++	+++		
Pemberton & Griffiths (1999)	++	++	++	
Coram & Lee (2011)	+++	+++	++	
Pattern based-design tool				
Molina (2002)	+++	++	+++	+++
Henninger (2001)	++	++	++	

Table 7.2 Description of Master-Detail in pattern collections (From ‘=no information, +=low, ++=medium, to +++=high).

Master-Detail Pattern in pattern collections									
General description				Management		Pattern based-design tool		Usability	
Gang of Four (Gamma <i>et al.</i> 1994)	Tidwell (2010)	van Welie (2003)	van Duyne <i>et al.</i> (2006)	Borchers (2000)	Pemberton (1999)	Molina (2002)	Henninger (2003)	Nielsen (2002)	Johnson (2003)
+	+	+	++			+++	++	+	+

Table 7.2 reveals that most pattern collections cover the M-D pattern, but with a limited genericity. Some other collections do not contain the M-D pattern explicitly, but refer to it in a different way. For instance, van Welie (2003) presents the “tab UI element” as a possible master area in a M-D pattern and another UI element called “overview by detail” as a detail area in this pattern. Its description focuses on a single implementation language and usability explanation. Moreover, Nielsen (2002) and Johnson (2003) cover this usability explanation. Globally speaking, the definition of patterns found in these collections are too often oriented towards one single context of use, for instance a particular user, a single computing platform, a specific environment. They do not cover the three aspects together and they do not cover variation within these aspects.

7.2.3 The Master-Detail as a Generative Pattern

JUST-UI, an object-oriented framework for generating UI from object-oriented models, was probably the first to introduce a generative pattern, for instance for the M-D (Molina, Meliá and Pastor 2002). JUST-UI automatically generates interactive application from a series of conceptual models, such as the presentation model (Fig. 7.2), built upon conceptual patterns. The presentation model, also used in OO-Method, is decomposed into three levels (Fig. 7.2):

- Level 1: *Hierarchical Action Tree* (HAT). HAT is also called system access structure. This level solves the user-system interaction issue.
- Level 2: *Interaction Units* (IUs). Each element composing the IUs represents a possible scenario through which users can perform tasks. This middle level is composed of four different types of Interaction Units.
- Level 3: *Elementary Patterns* (EPs). This last level is defined by a large set of basic elements, also named building blocks, from which a variety of scenarios (IUs) are founded.

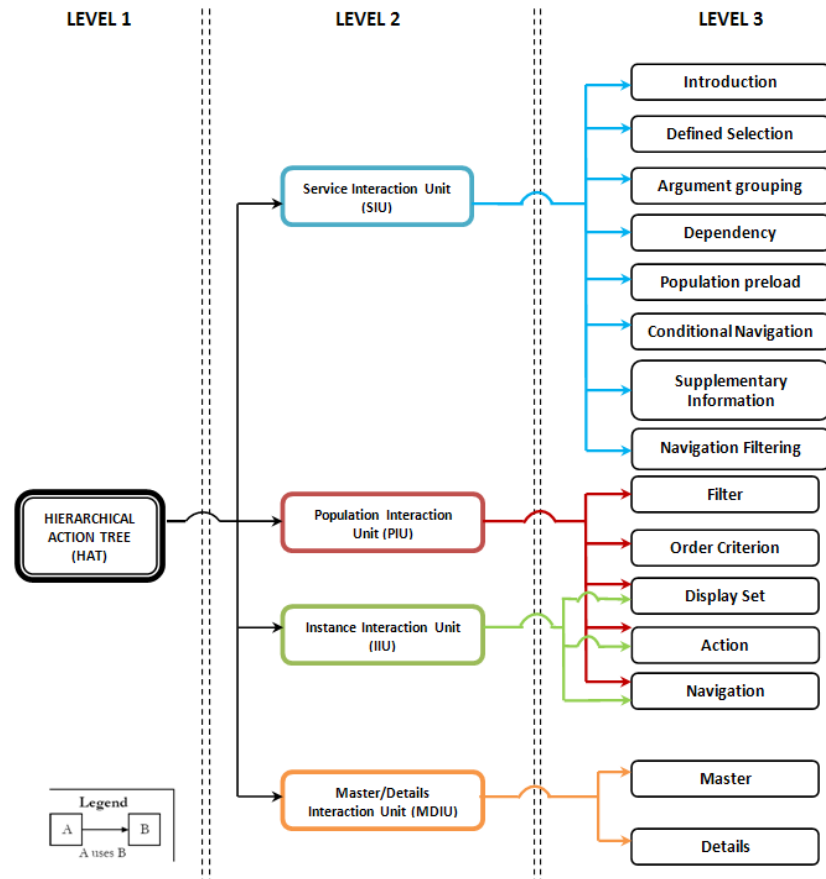


Fig. 7.2 The Presentation Model of OO-Method (Aquino and al. 2010)

The Master-Detail pattern starts at level 2 as a combination of a population interaction unit and an instance interaction unit, which are then automatically generated as a web application (Fig. 7.3), a desktop application for MS Windows (Fig. 7.4a) and for MacOs (Fig. 7.4b) .



Fig. 7.3 Master-Detail pattern for a Web application

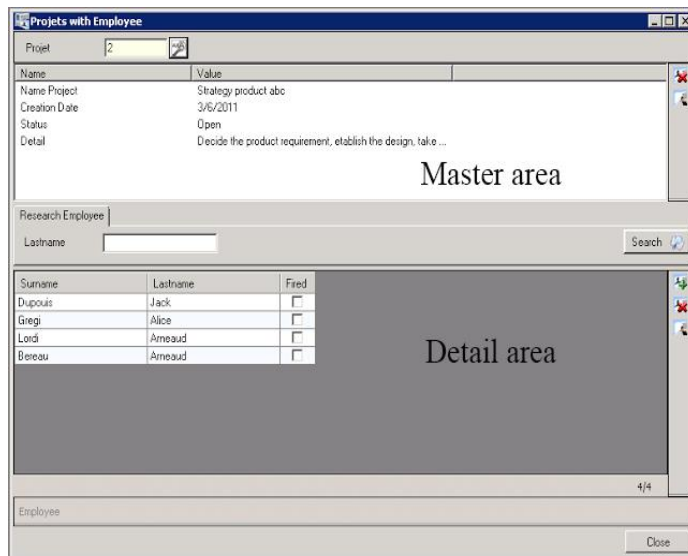


Fig. 7.4 Master-Detail pattern for desktop platforms: (a)Microsoft Windows and (b) MacOS (PJMolina 2013)

7.2.4 Previous Work on M-D Pattern

HCI design patterns have proven their potential as a solution to guide developers in capturing knowledge at a high level of abstraction while facilitating the design of MUIs. But their scattered information is sometimes too complex to understand for some developers, especially when different platform style guides and software manuals address the pattern in different, possibly inconsistent, terms. Fig. 7.5 reproduces the instructions to follow for implementing the M-D pattern in Objective-c, the programming language used by Apple for OS X and iOS operating systems for mobile platforms, while Fig. 7.6 do the same job for desktop, but for Oracle (Fig. 7.6a) and MS Windows again (Fig. 7.6b) and Fig. 7.7 for OpenERP (Fig. 7.7a) and SAP (Fig. 7.7b), which are however two desktop-based Enterprise Resource Planning systems.

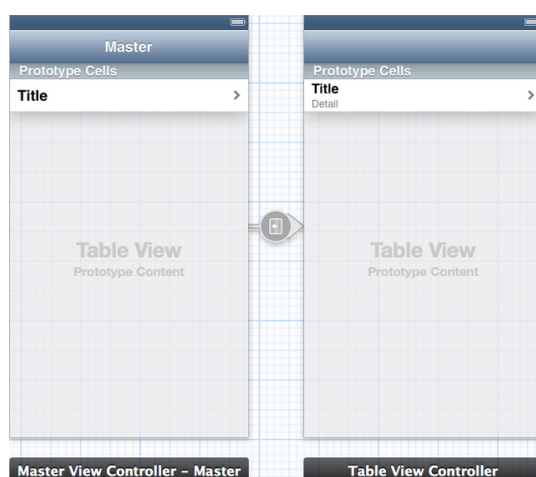
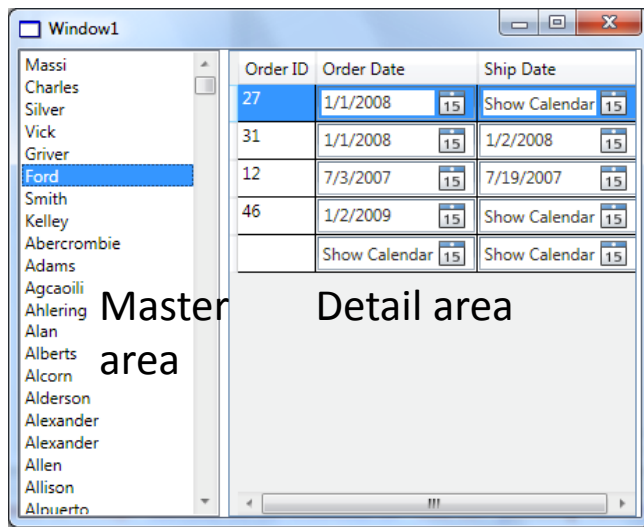


Fig. 7.5 A guidance of M-D pattern implementation based on Objective-C



List of Objects/Hierarchy of Objects
Table/Hgrid



Fig. 7.6 M-D pattern with an expanded List of Detail part on Oracle instance (a) and a list on Windows view (b)

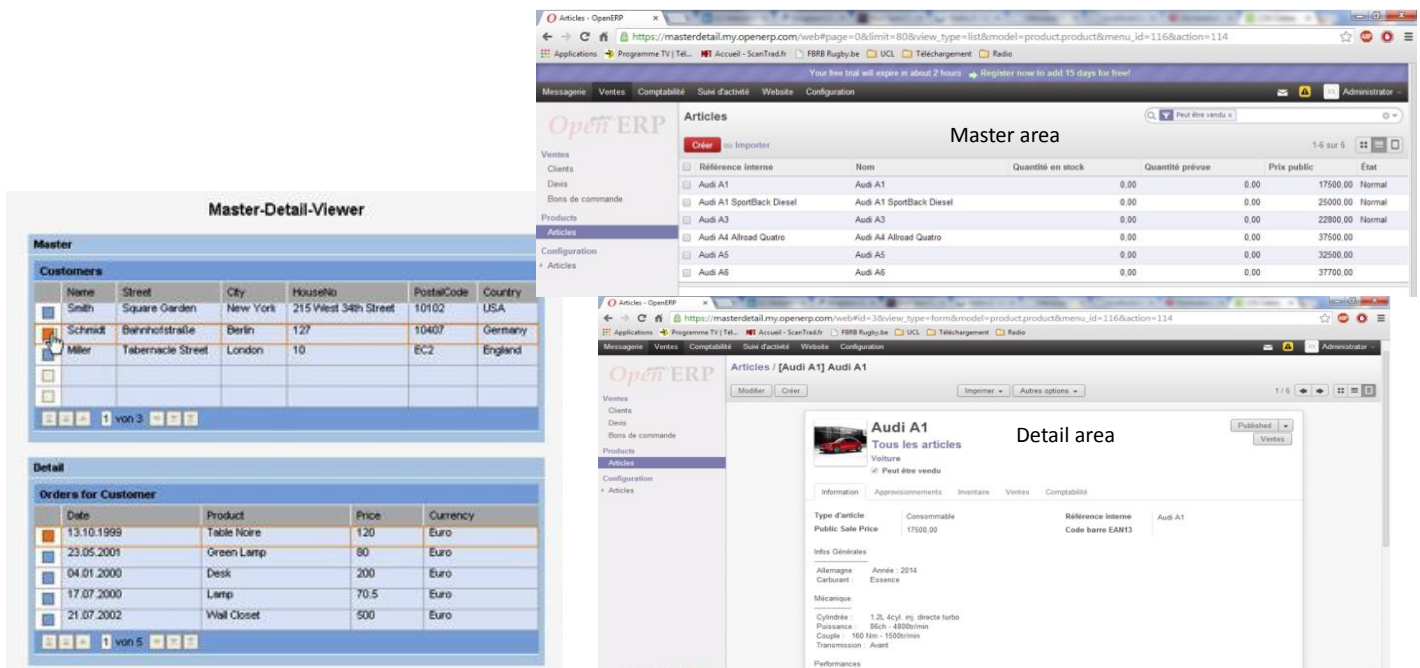


Fig. 7.7 A list of elements to present the Master pattern and a single presentation for Detail pattern in OpenERP (a) and M-D pattern in SAP (b)

Leading to the same conclusion, Table 7.3 suggests that UI toolkits do not frequently support the expression of the M-D pattern at a higher level of abstraction than the code level.

Table 7.3 Toolkits expressing the M-D pattern design at a high level of abstraction (☒=unsupported, ☑ = supported)

To design M/D pattern in high level abstraction					
UI toolkit	Objective-C	Android tool	Visual Studio 2010	OpenERP	Oracle
Task& Domain model	☑ (only domain model)	☑ (only domain model)	☑ (only domain model)	☒	☑ (only domain model)
AUI	☒	☒	☒	☒	☒
CUI	☒	☒	☒	☒	☒
FUI	☑	☑	☑	☑	☑

7.2.5 Shortcomings and Requirements

Based on the examination of M-D literature, the following shortcomings have appeared to be important to address when solving the cross-platform UI design problem:

- s1. **Lack of expression consistency.** Information related to the pattern description and its applicability is fragmented across different attributes. A list of factors or criteria is necessary to validate a pattern.
- s2. **Partial pattern representation.** The current works about patterns are constrained to one or some levels of the UI development life cycle. When they do it, the entire process is not completely addressed.
- s3. **Limitation of technological space.** Most UI patterns available are specific to one platform or at least provides example for one platform, often for the Web and desktop.
- s4. **Lack of usability approach.** Tools to support pattern assisted design and development exist but the way they handle usability knowledge is limited, if not explicitly incorporated.
- s5. **Lack of implementation information.** Only some tools offer effective instructions on how to guide the pattern application and implementation.

In order to address the aforementioned shortcomings, the following requirements are elicited:

- R1. Revisiting the M-D pattern definition with up-to-date information
- R2. Integrate the M-D pattern in the whole UI development process
- R3. Consolidate methods and techniques in using a guidance system
- R4. Design M-D pattern within usability concerns explicitly incorporated
- R5. Structure a M/D pattern based on Dijkstra's principle of separation of concerns (Dijkstra).

In order to satisfy these requirements, the following section revisits the M-D description with a focus on expressivity and generativity, as opposed to descriptivity

and genericity. The usability concern is a large scope. It needs more application and technic. We will just use it in the guidance system.

7.3 Revisiting the M-D Pattern Description

During a period of steady technological growth, a large variety and availability of devices and hardware/software platforms are being developed. Then, the ideal situation for users is to have access information and services on device that they are using in a different context or environment. With the M-D pattern, end-users interact with different scenarios containing several objects aggregated (Pedro, Santiago and Pastor 2002). Usability concerns should be integrated to UI patterns (Folmer and Bosh 2003). M-D patterns are therefore augmented in this work by ergonomic criteria (Scapin and Bastien 1997) such as user guidance, or consistency, or error management which are addressed when applying the pattern. Table 7.4 provides an enriched pattern definition based on the template introduced by the Gang-of-Four (Bayle et al. 1998), such as template attributed found in (Wendler, Ammon Philippow and Streitferdt 2013). Information from (Bayle et al. 1998, Van Welie et al. 2002 and Kruschitz 2009) about the M-D pattern are also included. Elements of consistent template (Engel, Marin, Herbin and Forbrig 2013) are:

- **Pattern Name:** How is the pattern called?
- **Also Known As:** What are the other names for this pattern?
- **Classification:** Is the pattern creational, structural, or behavioral?
- **Motivation or Problem:** What is an example scenario for applying this pattern?
- **Intent or Solution:** What problem does this pattern solve?
- **Restriction:** What restriction does this pattern require? What are its constraints?
- **Forces:** What are advantages and forces to use this pattern?
- **Weakness:** What are disadvantages or limits to use this pattern?
- **Rationale:** Why does this pattern work? What is the history behind the pattern?
- **Applicability or Content:** When does this pattern apply?
- **Context of use:** What are the category of user, environment and platform that this pattern can be applied?
- **Structure:** What are the class hierarchy diagrams for the objects in this pattern?
- **Participants:** What are the objects that participate in this pattern?
- **Collaborations:** How do these objects interoperate?
- **Consequences:** What are the trade-offs of using this pattern?
- **Implementation:** Which techniques or issues arise in applying this pattern?
- **Known Uses:** What are some examples of real systems using this pattern?
- **Related Patterns:** What other patterns from this pattern collection are related to this pattern?

A complete description is necessary to address all parts of the pattern and to show its application:

- Provide a comprehensive and descriptive solution involving the three parameters of the context of problem (user, platforms and environment) by integrating usability knowledge

- Reuse general/standard solution reducing errors and research a complete solution in using an abstract solution in order to implement them in a straightforward way
- Visual explicit view of ergonomic design before implementing.

We can see that implementation attribute is often missing. That requires more details of the pattern application in several programming languages. The pseudo-code aims to facilitate its implementation in different programming languages while bring better understanding for developers. In this pseudo-code, the Master pattern is encapsulated in a table view listing each object and get detail of the object.

Pseudo-code of M-D pattern:

```
do
  insert master record
  for object_1 in tableView_Master
    get
      detailed record (object_1, "details_1" )
    //set foreign key from master record
    loop until end of detailed records
    for object_2 in tableView_Master
      get
        detail record (object_2, "details_2" )
      ...
    ...
  loop until end of master records
```

Table 7.4 Master-Detail Pattern Description

Pattern Name	Master/Details
Also known as	Master/Slave, Director/Details
Classification	Structural/object centric
Problem	The scenario, which the user has to search in a list and select an item to have more details, is frequent. A set of information units linked or not by a relationship have to be presented to users. These last have a scenario that the master interaction unit determines information of details interaction unit will show.
Solution	Perform a composed presentation 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 situation, context. The scenario of this pattern allows to simplify the user task. Indeed, navigation is decreased for getting specific information. Moreover, information is maintained synchronized between the master and details units.
Weakness	The size of screen can discourage the presentation of this pattern. Less information can be shown at the same times on a screen. The details need to have a great navigation and to follow some usability guidelines in order to respect users' requirements and to have a graceful presentation.
Rationale	Provide a presentation to reduce several navigation and to simplify the user 'task. Users need to interact with several objects aggregated or not. The scenario offered by M-D pattern allows to get 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 can get this pattern and adapt it. For instance, we can use this pattern to show the cases Project/Employees or Invoice/Lines. All kind of platforms can adapt this pattern in line with usability studies.
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.
Participants	One or two instances with an aggregated relationship
Collaborations	Objects can operate though their aggregated relationship or attributes.
Consequences	Need to uses usability elements for adapting this pattern on different platform. Knowledge about these devices is required.
Implementation	The issue about using a unity class or aggregated classes is necessary before implementing this pattern. The M-D pattern uses a list of model objects which can be presented in the Table List pattern. Each selected objects are presented in the Display pattern. Therefore, the user navigates through a list with synchronize information.
Known Uses	Commercial system can use this pattern to show the case Invoice/Line.
Related Patterns	Object presentation, Population unit, Instance Interaction Unit, Display Form, Table List Pattern

As we can also see on the current shortcomings in previous section, the major element to take into account in the pattern implementation is the cross-platform context and to improve the software lifecycle development integrating the development of usability studies.

7.4 Integrate the M-D pattern in the whole UI development process

This section presents one possible way of incorporating the M-D pattern usage into the UI development process based on a UIDL that supports the four levels of abstraction of the Cameleon Reference Framework (CRF). Other XML-compliant frameworks are available such as IMG's framework (CIM, PIM, PSM) for developing multi-target UIs. Our choice is motivated by its simplicity of hierarchical structure of each abstract model and its transition of progression levels along the development lifecycle on a variety of devices.

7.4.1 Task Model

A *task model* is a description of tasks that a user will be able to accomplish in interaction with the system. This description is a hierarchical decomposition of a global task, with constraints expressed on and between the subtasks. The UsiXML task model relies on ConcurTaskTree (CTT) notation (Paterno 1999): a hierarchical task structure, with temporal relationships specified between sibling tasks.

Since the M-D pattern is based on a domain model, it needs to be augmented with tasks so as to produce a corresponding task model. Depending on the domain model is interpreted, two task models could result from this process:

- A unity instance expressed as the master element and its object domain attributes could compose the details element (Fig. 7.8). In this task model, an object or a collection of objects is edited by browsing all the attributed belonging to the class object and by invoking typical management methods, such as those found in the CRUDS pattern (Create, Read or Retrieve, Update, Delete, and Search).
- An aggregation representation between two domain objects (Fig. 7.9). In this task model, all object attributes are edited at once or one by one, all object methods are invoked at once or one by one on demand

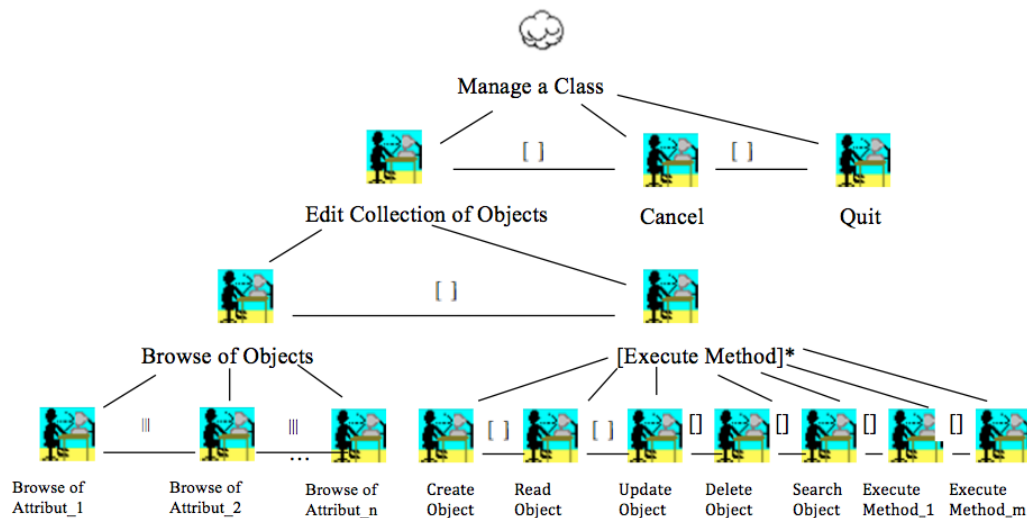


Fig. 7.8 Master-Detail pattern defined by a unity class

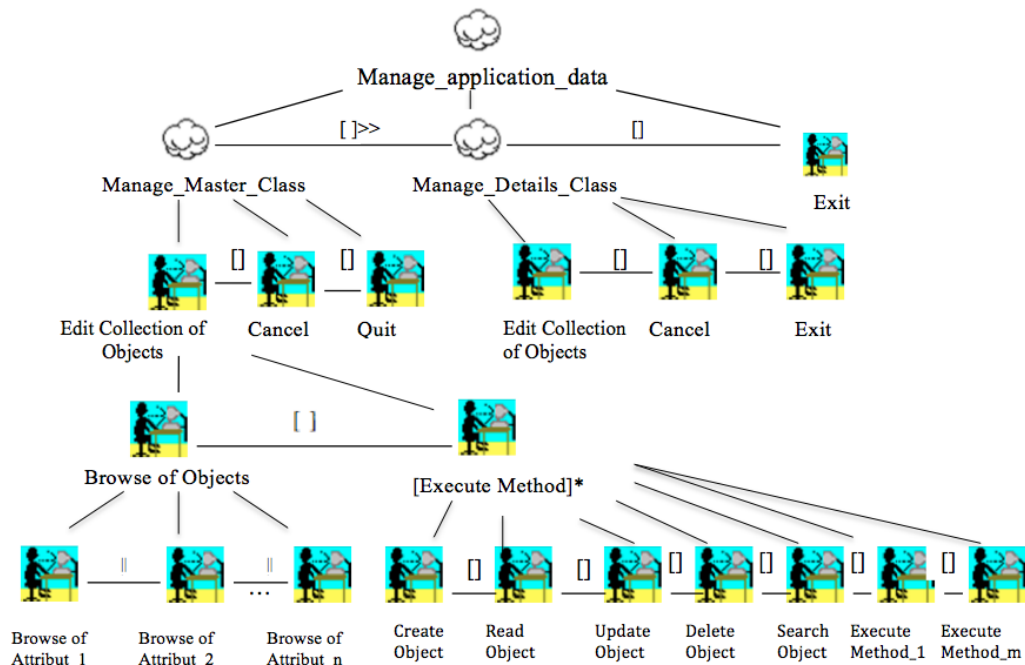


Fig. 7.9 Master-Detail pattern defined by an aggregation relationship

7.4.2 Domain Model

The domain model presents the important entities of the particular application domain together with their attributes, methods and relationship (Sinnig, Forbrig and Seffah 2003). Various notations could be used for this purpose, such as the Entity-Relationship-Attribute (ERA) notation, the Object-Oriented (OO) notation, or more frequent and up-to-date the Unified Modeling Language (UML). Two cases could occur:

1. When the M-D pattern is applied on **a single domain class**. In this case, the domain case merely consists of its usual attributes and methods (Fig. 7.10). Applying the M-D pattern therefore consists of displaying a list of objects belonging to this domain class and displaying its attributes and methods on demand. For instance, the Mac Developer Library (2012) uses this method where the root is the collection of these objects (Fig. 7.10a). Note that the master domain class could refer to several detail

areas that are related to some different areas, such as phone or address (Fig. 7.10 b). Eclipse Documentation (2012) refers to this M-D pattern as a M-D block presented as a list or a tree (master area) with a set of synchronized properties (detail area). Methods of master unit are abstract and must be implemented by the subclass while details unit is created.

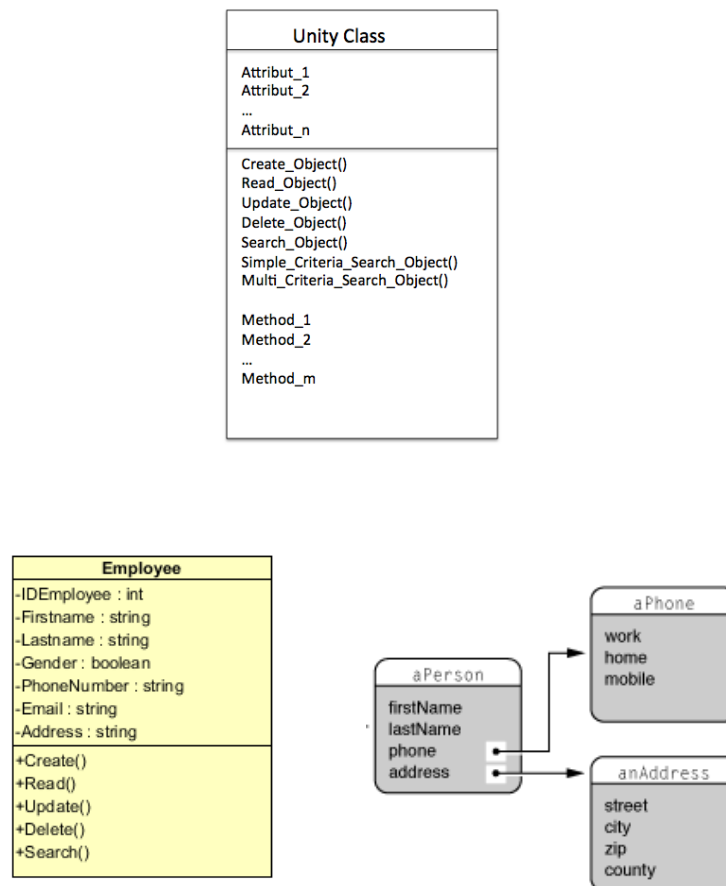


Fig. 7.10 A domain model associated to the Master-Detail pattern (a) and associated to the System Environment (Mac Developer Library 2012) (b)

2. When the M-D pattern is applied on a master domain class associated to a detail domain class **via an aggregation relationship** (Pedro, Santiago and Pastor 2002) see on Fig.7.11. In this case, the details associated to a master are considered conceptually different and important enough to warrant a dedicated handling via a detail area display. “This pattern can be easily mapped to a many-to-one relation schema used within a database design.” (Perrins 2008) Detail role expand objects related to the master object conforming to their aggregated relationship. In this model For example, the **Employee** class contains a sub-group **Address**, which could in turn be decomposed into attributes contained in a separate class: zip code, street, etc. This concept 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. In the **Employee** object, **Address** is considered as a fundamental attribute. Therefore, a mapping from its domain model into two external specific objects could not possible with a 0..n relationship.

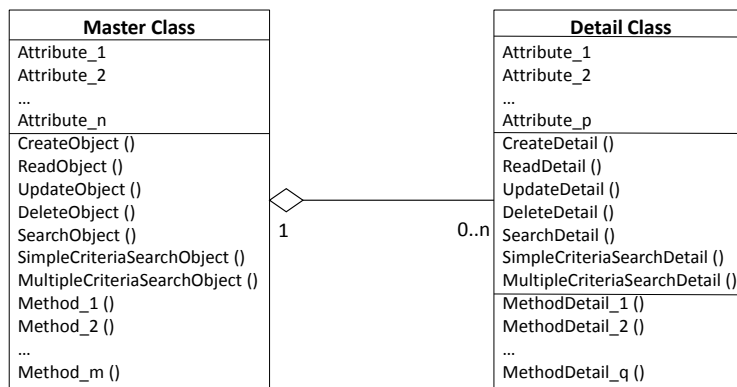


Fig. 7.11 The Master-Detail pattern in an aggregation relationship

7.4.3 Abstract User Interface Model

The Abstract User Interface (AUI) model specifies a UI independently of any interaction modality (we do not know yet whether this UI will be graphical, tactile, gestural, vocal, or multimodal in the future) and any technological space (we do not know which computing platform will be the target). The AUI model is the counterpart of the Platform-Independent Model (PIM) used in Model Driven Engineering (MDE). An AUI consists of a recursive definition of abstract interaction units, each unit could be of input, output, input/output, selection, or trigger, each of them coming with their own event listener. For instance, Fig. 7.12 reproduces a possible AUI for the Employee class of Fig. 7.10 (b), as edited in UsiAbstract, an Eclipse plug-in for editing AUI models.

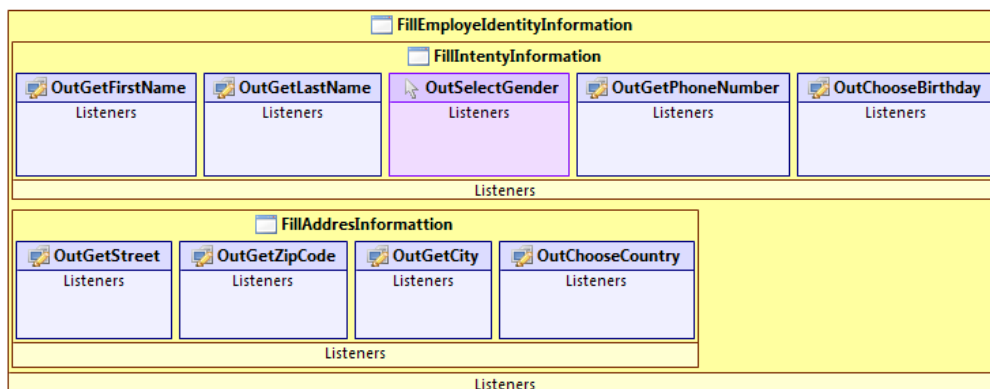


Fig. 7.12 The Master-Detail Pattern leading to a AUI model

7.4.4 Concrete User Interface

The Concrete User Interface (CUI) model specifies a UI independently of any technological space, but for a given interaction modality. The CUI model is the counterpart of the Platform-Specific Model (PSM) in MDE. The benefits consist mainly in improved and expanded definitions of the description of UI elements. The CUI depends on the type of platform and media available. This model allows both the specification of the presentation and the behavior of an UI with elements that can be concretely perceived by end-users. That means to define widgets layout and interface navigation independent of any computing platform.

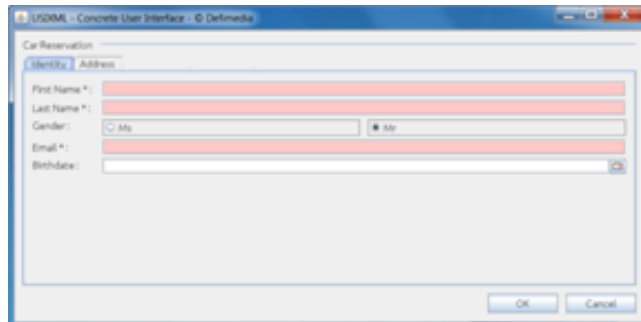
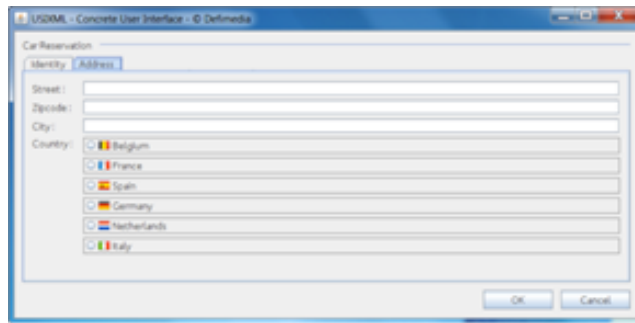


Fig. 7.13 A CUI model of tabbed list presentation for M-D pattern according to van Welie (2000)

Fig.7.12 and Fig.7.13 present merely one possible AUI and CUI respectively for applying a M-D pattern independently of any technological space. We could continue with several final UIs that could cover different contexts of use. But our goal is to show many high level UI models. We want to define different models to specify how sub-tasks of a given task are assembled together for cross-platform environments. Therefore, we begun with the presentation model from Fig.7.2 and we modified its level 3 to adapt with our sub-task presentations of objects. The result in the Fig.7.14 shows different models of possible dynamic presentations from AUI models to FUI models. The contribution of this framework is to offer a great flexibility in implementation of elements, to provide a usable technic by its XML implementation and then an active participation in cross-platform designing. In the next section, we will provide more details about this framework.

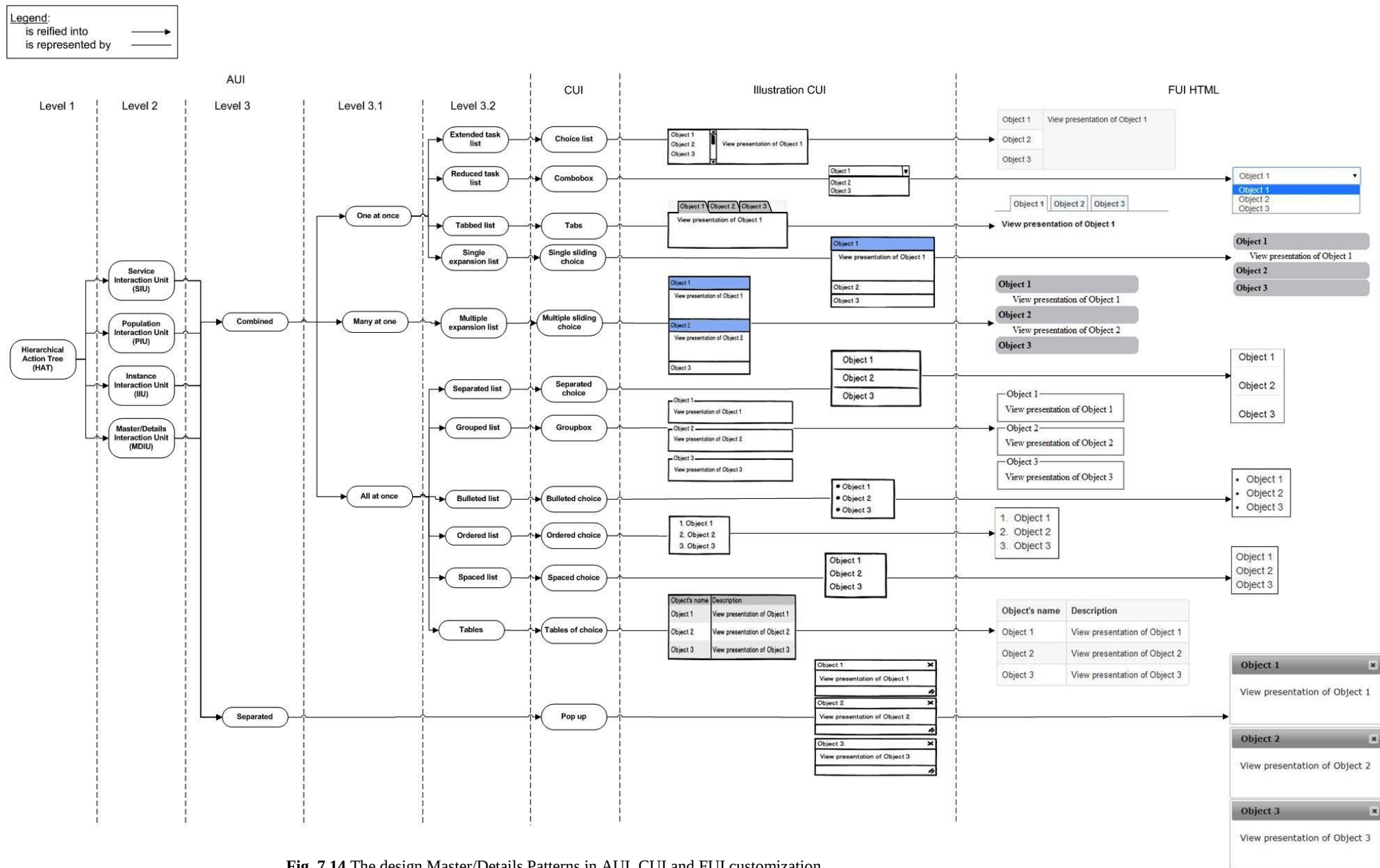


Fig. 7.14 The design Master/Details Patterns in AUI, CUI and FUI customization

7.4.5 The M/D pattern application support toward FUI

To guide the implementation of the M/D pattern is limited as we can see in the scientific literary. Therefore, we created a tool based on the Fig.7.14 applying this UI pattern and at the end generating a XML-document to facilitate its implementation in high level of abstraction UI design. This section is sub-divided in different sections: beginning by a description of the tool, then the presentation options of elements inserting in this tool (how they are created and why).

7.4.5.1 Description of the framework supporting the M/D pattern in abstract high level UI models

Advice-giving system and guidance tool, MDPAG (The Master Details Pattern Application Guide): M/D patterns are used, as other patterns, reflecting on possible changes to a technical space or situation. In using the cross-platform context, patterns can prevent repetitive errors in a changing platform. That also allows understanding better possible impacts of new technologies such the size of the screen. Therefore, patterns are prescriptive and promote creation of new instance in order to help designers. The presentation of the M/D patterns in a variety of type of screens defines how and which elements are suitable. Design patterns can be used to capture essential problems of different “sizes”. Moreover, the use of pattern for documenting design knowledge “divides a large problem area into a structured set of manageable problems”. Alexander’s patterns are defined to be pleasant to humans. Usability concept should be integrated to UI patterns. M/D patterns in this work are based on ergonomic principles such as user guidance, or consistency, or error management. Previous works used M/D patterns in the UI development process but they are limited on cross-platform and ergonomic contexts. In (Molina, Santiago and Pastor 2002), its tool is limited in standard view with a limitation of customization and guidance for developer. Indeed, the tool generates a FUI with few customizations. MDPAG is a tool focused on abstract UI design pattern with usability integration and high level of abstraction to guide developer/designers. Its offers the possibility to choice parameters and have a dynamic abstract representation related to this choice.

Indeed, MDPAG is still a work-in progress support, structured like a tree (see on Fig.7.14). The conceptual model from (Molina, Santiago and Pastor 2002) is extended to show the possible UI design elements to guide implementation. Each AUI elements are following with CUI elements and possible FUI illustrations in the case study. The representation is dynamically related to parameter choices. Each representation offers the possibility to see a complete description based on context and implementation of the UI element. Moreover, the application generates a XML document at the end of the representation to facilitate its flexibility and implementation according the cross-platform context. Currently, FUI are generated with illustrations in HTML and Balsamiq Morkup tool (2014) which is more abstract and then it facilitates elements mapping understanding between CUI and FUI units.

Therefore, this guide support using a high level of abstraction of implementation based on the possibility to choose parameters to draw dynamic view of abstract object models. The choice is flexible and not restraints to two only choices like yes or no but in exhaustive possibility. Abstract Views dynamic change related to parameter choices. Each design parameters are defined in using a complete description with illustrations and case study like the M/D pattern.

7.4.5.2 Possible Presentation Options toward FUI model

The M/D pattern presentations have some usability restrictions. Indeed, they need to have a correct layout to present information in order to perform a task. The scenario of tasks for this pattern needs to get synchronized information between the master and details units. The layout of some devices can limit their presentation. Consistency and usability are essential characteristics for presentations. In this section, we can find different presentation of the master unit. In the Fig. 7.15, you can see a suggestion of presentation using List or Table of objects in starting point. When an item is selected, its details are presented in the display form pattern. We have two ways: one column list or multi-column list. In the first case, master unit can be shown by a simple list, a drop-down list or a fish eye menu. In the multi-column table, attributes of the master unit are presented by prioritized criteria: the most frequent, critical or mandatory.



Fig. 7.15 An example of M/D pattern presentation (Perrins 2008)

To build MDPAG, we need to draw different presentation options for the M/D patterns. The Fig. 7.16, different presentation specifies how objects of a given task are assembled together. Instances can be presented in direct mode or progressive presentation. In the first format, each object is shown one by one. It is a list that includes all attributes and methods in one view for each object. The vertical and horizontal scroll can be heavy for the users. In the progressive mode, methods and attributes can be selected for each object.

The Fig. 7.17 suggests a presentation of an instance in the M/D pattern case for desktop view. The attributes and methods of objects are presented by specific criteria: the most importance, recurrence of use, critical characteristic. Therefore, attributes and methods are characterized by multiple criteria: simple/repetitive, elementary/decomposable and optional/mandatory. Attributes can have the label presentation or optional checkbox. For methods, we can find the traditional CRUD Method (Create, Read, Update and Delete Method) in the unit. Other methods are obviously possible. Each method can have a specific view: a textual, graphical or both presentations on their button.

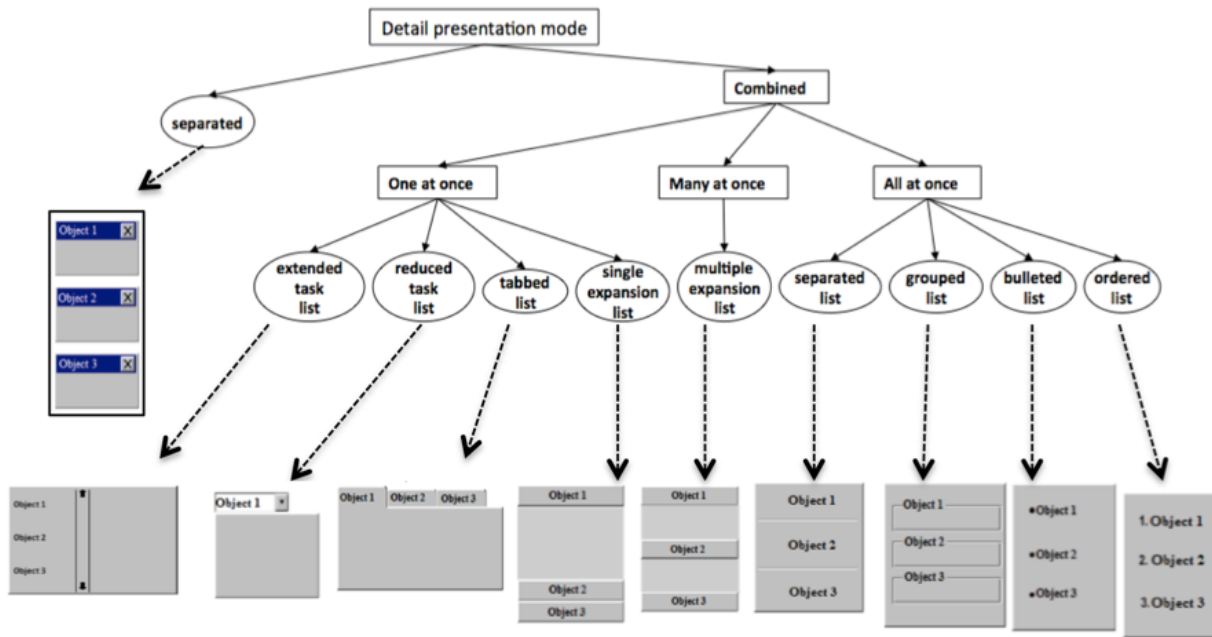


Fig. 7.16 Object Presentation in M/D pattern

In the design option presentation of the M/D pattern, we explain possible representation of objects. In Fig. 7.17, a unity class is represented in specific desktop window. Attributes can be edited by fields or in using checkboxes. Methods are action buttons.

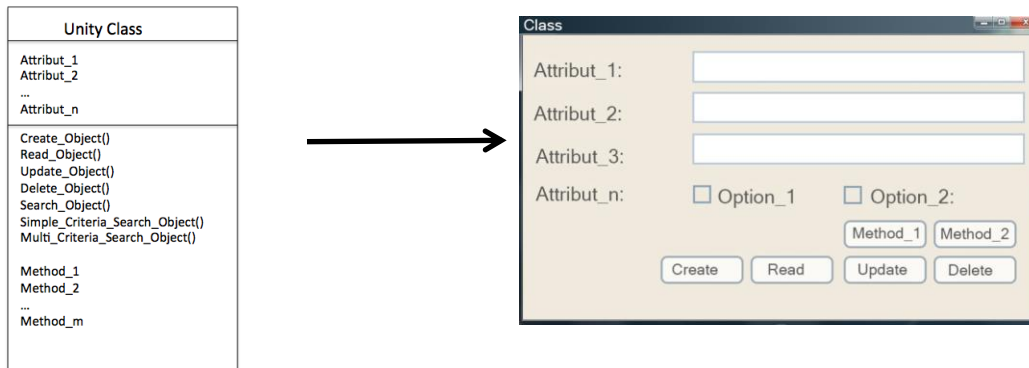


Fig. 7.17 Instance presentation of M/D pattern in desktop view

This standard representation can be completed by graphical or/and textual option for buttons. Standard buttons for commons methods such as add, create or delete an item can use frequent icons with extra information. User experiences allow to act quickly and to reduce stress. On mobile platform, a listview is preferred to use. That allows a great structure and improves the usability. Another usability guideline specifies to use button in correct and understanding context. The button cannot appear in confuse situation.

In Fig. 7.18, we can observe two same situations: we want to add a new contact. In the figure on the left, we can see this situation on android 1.1. That does not respect usability guidelines described above. On the right, we can observe a listview of android 1.5 using both graphical presentations: textual and graphical. Usability graphic is better to understand the meaning of the application.

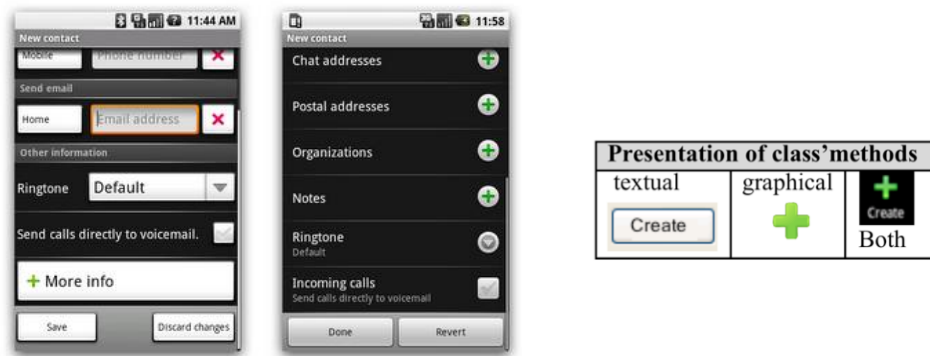


Fig. 7.18 Graphical Presentation of Class Methods on Android OS (Android 2012)

The technique of M/D information used in progressive or direct display, are presented in a specific public support on-line to guide developers/designers in using the question how sub-tasks of a given task could assemble together. By a set dynamic choice, different sub-task presentations in AUI, CUI and FUI models are displayed. The next section presents the application of M/D pattern in a specific tool with a case study: the renting car.

7.5 The M/D pattern application support

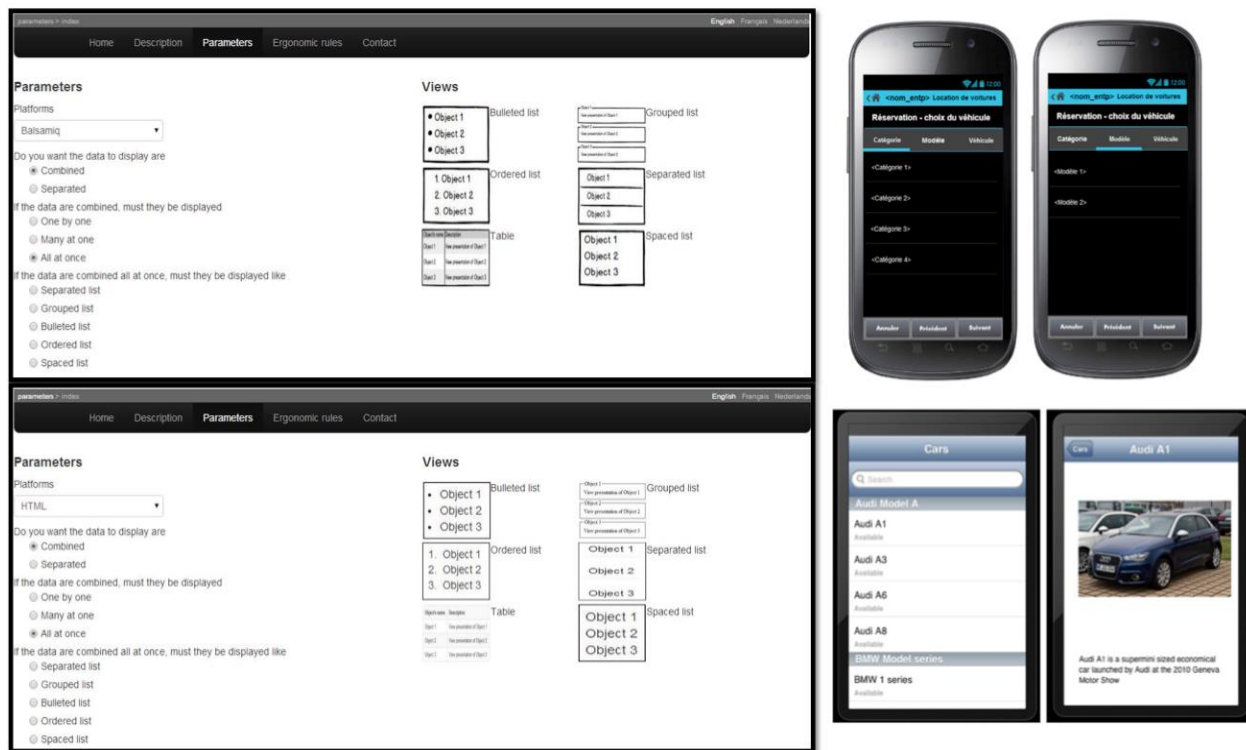


Fig. 7.19 A tool of guidance for M/D pattern implementation with Balsamiq FUI model (left above), with HTML FUI model (left below), final CUI model (right above), FUI model (right below)

These options are presented in the tree (cf. Fig. 7.14) and applied in a specific support called MDPAG. The translation from the Task/Domain model, AUI, CUI and FUI are based on study case. For example with the renting car study case, the CUI of the M/D pattern is illustrated in Fig.7.19. The CUI of the M/D pattern is presented with the following selection of parameters: The master interaction unit

is reified into object-combined presentation in AUI model. Its objects are shown one at once in using a tabbed list. In CUI, its representation in using tab object is illustrated in the guide support and in a FUI instance. The detail interaction unit is shown in each object of the tab list of its master in using AUI presentation based on all at once with a separated list. In its CUI model, objects are represented by a separated choice element.

rules > Index												English	Français	Nederlands
Ergonomic rules														
Rule	Extended task list	Reduced task list	Tabbed list	Single expansion list	Multiple expansion list	Separated list	Grouped list	Bulleted list	Ordered list	Spaced list	Table	Pop up		
Elements of a window have to be align	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Minimize scrolling	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
A text should not be written entirely in capital letters.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Put symbol (puce) for a better structure and visibility	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗

Fig. 7.20 Renting Car application validated with ergonomic guidelines in MDPAG

These generated dynamically applications are validated by different guidelines (Fig. 7.20). We can observe what guidelines are validated for each illustration. At the end of preview illustration and selected parameters, the end-user can also generate a XML-document related to dynamic choice for the implementation. Other illustration of the application based on other operating system and platform integrated in this support is presented in the following section.

7.5.1 Support for M-D Pattern Application

The application is developed in order to adapt the M/D pattern in mobile devices. We decided to focus on Android-based mobile systems. The motivations are the free accessibility of Android-Framework and its code language, Java that is a widespread programming language known by all developers.

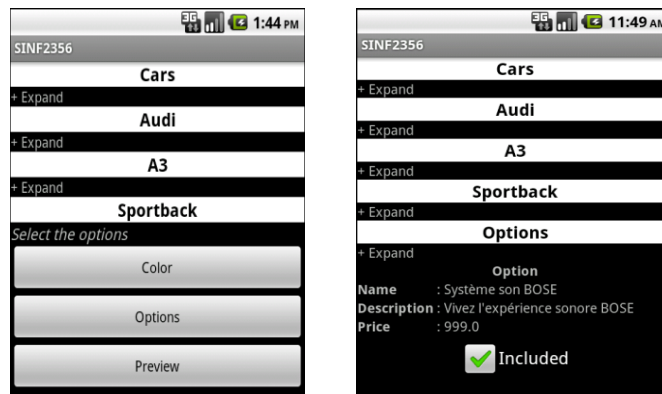


Fig. 7.21 Master/Details pattern in Android System

The usability concern is the screen size limitations of general mobile devices. Therefore, a minimal set of information is available at any time. A possible situation is to minimize the accessible information set thanks to an adequate use of “reducing” and “expanding” controls of the list, so that the user keeps the focus on the part of the application that he is using (see on Fig.7.21 on the right).

‘Cars’ is the class ‘title. All attributes of this class are included in details presentation in using a combined way. In this way, three details views are possible of attributes: one at once, many at once or all at once. It is a basic possible presentation and using of Master/Details pattern combined with the population pattern and other auxiliary popular patterns such as filters, order criterions, selection and display sets. All used patterns in Mandroid are presented. All attributes of Mandroid are viewed many at once or all at once. That depends on auxiliary patterns used.

On the Fig. 7.22, the relevant patterns are the Master/Detail and the Order ones. This model selection allows sorting alphabetically the brands and, secondly, when a brand is selected, the detail (i.e. the next step of the car configuration) appear. Then, the user has to select a model of car represented by standard button of Android System. Basically, this step is implemented the same way as the previous one, using Master/Detail and ordering, but it also contains the Filter pattern. Once the model is selected, the resulting detail concerns the selection of the body style of the car. This step uses a nested Master/Detail pattern.

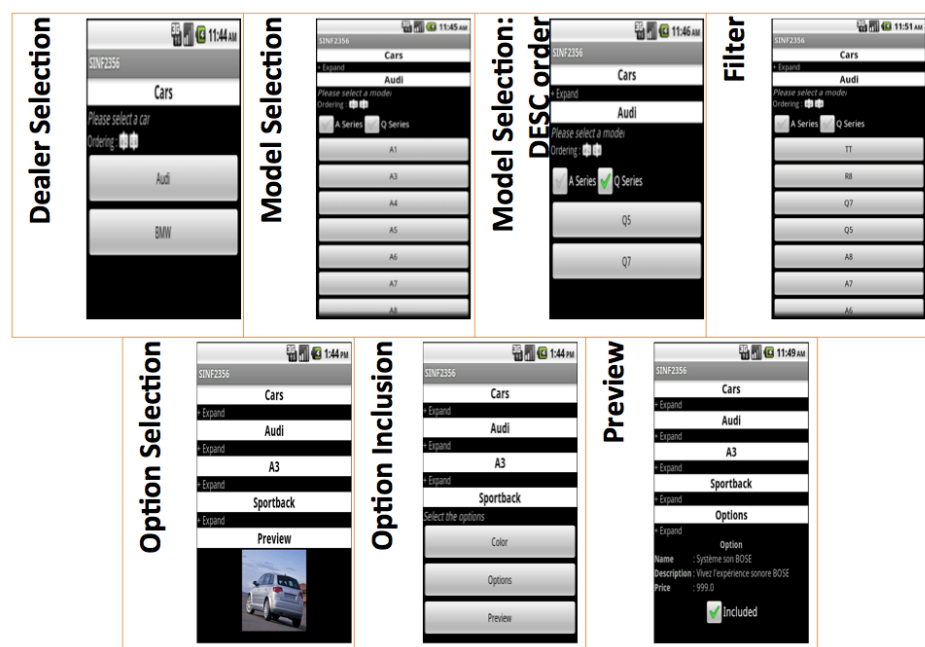


Fig. 7.22 Mandroid Application

Next, the user can specify the options and the color that he wants. So, we only focus on the “Options” one. Typically, the detail of this button is a list of options, which, once again, use the Master/Detail pattern. When an option is selected, a screen allowing the user to select it appears. To get back to the options list, the “+ Expand” link can be clicked. This link is present each time the Master/Detail pattern is used in order to get back to the master. Finally, a preview of the car is available.

On a technical point of view, the filling of the application is done automatically thanks to our XML parser compatible with Android. Thanks to the developed tool, the data is fetched from a XML-file and then presented on the user interface. This strategy enables to update the data about cars and even add new models and/or brands (without having to recompile the application). The idea behind the algorithm is the following: each time we meet a node in the XML-file we check its value and create the corresponding elements with the attributes specified in the XML-file. Example: a node with value “model” causes the creation of a Master element. Every node that follows and whose value is different from “model” concerns the model previously created (we go through the XML-file line by line). Then, depending on the values of the next nodes, masters and details elements are created and added to

previous elements. If the value is equal to “ordering” or “filter”, the corresponding patterns are initialized on the population of the appropriate master. This XML parser helped us to maintain our application clean, well structured, to enforce the quality of the user interface and to efficiently work in team.

Another illustration of application is the Fig.7.7 in the Oracle instance, the Master is also table of objects. When an object in the master table is selected from the single select column, the details section below draws with label/data layout of object details. Multiple selections of objects is not allowed in the Master Objects Table. If multiple selections is functionally required and there are drill down actions for the objects, then the actions will have to be performed in separate pages, such as an Object Template. To access full object details, the user must select the master object from the single select column, then select the "Advance Update" button in the detail section. Details are represented as a single object (label/data layout) based on the selection of a master object.

7.5.2 M/D pattern presentation for tabbed list presentation in mobile application

The combination of the two interactions units (Master and Details) is shown in two tabs on the mobile view (cf. Fig.7.23). We take our example with employee class in aggregated relationship with project class. In our instance, we have in the master role: the interface interaction unit of project class. In the Detail role is the population interaction unit of employee class. The detail role means to show only employee from the selected project in the master part. It is the same with the mobile view, the second tab show all employees from the project in the first tab. Therefore, we can say that a dependence part is defined.

A navigation bar on the top is not visually correct because the title is reproduced twice as we can observe on the right of this figure. Moreover, that is not in line with the specific guideline: Indicate the position of the user only once. Without a navigation bar, the user cannot come back to the menu. The solution can be to integrate this in the bottom menu bar which defines the action patterns. The problem with this solution is that it reduces the options of action set and information quantity.



Fig. 7.23 M/D pattern using a tabbed list presentation of Mobile Platform

7.5.3 M/D pattern in grouped, ordered or structured list presentation

The most majority of tool and M/D pattern presentation is defined by grouped, ordered and structured element presentation in its Master part and by a large information content in its details part. For instance, in the SAP FUI (cf. Fig. 7.7), the two tables in the Master-Detail Viewer is be filled with data records that are saved in the context of the view controller. The (upper) Master table displays a row for each customer, containing his or her name and address. The (lower) Detail table displays the order records for the currently selected customer. There are presented in ordered date (old date to new date). On another illustration, on Oracle system (cf. Fig. 7.6) represents hierarchy of objects; and single object details (from list.). Master is a hierarchic of objects. When one of the objects in the tree is selected, the details section draws with selected objects details. Depending on the detail contents for each item in the Tree, it is possible to show a different master/detail template depending on what object type is selected. For instance, if the master tree is a hierarchy of banks, branches and accounts, and the user selects an account. Single Object Details are represented as a single object (label/data layout) based on the selection of a master.

7.6 Conclusion

On the one hand, the importance of HCI design patterns is broadly recognized as they clearly impact the overall quality of a User Interfaces. On the other hand, several problems of implementing software design patterns have been pointed out, for example, ordinary object-oriented style implementations reduce the traceability of design patterns and the reusability of the implementation of design patterns. Our approach is one of the solutions for these problems.

Therefore, the key contribution of this chapter is the meta-model of Master/Details pattern structured into four levels of abstraction to foster portability and reusability. The proposed model aims also to support user interfaces that are or have to be easily customizable. Another contribution of this work is a systematic review and analysis of the recent scientific literature regarding M/D pattern description and implementation. We revisited its description and implementation for better understanding, to facilitate its integration and to demonstrate how the proposed method works. Indeed, current tools offer the possibility to generate FUI in standard representation in involving the context of uses and some usability guidelines. But they are limited in the customization of design UI patterns. Our support MDPAG offers the observation of possible design options of the Master/Details pattern on different platforms at different high abstraction levels. The case study focuses on heterogeneous context and design M-D pattern presentation in an attempt to validate the outcomes. We show, for example how the M-D pattern could be represented at the bottom level of UI development process (such as task model) independently of platform implementation. Then, different abstract models of M-D based on a real life case are dynamically generated in integrating usability concerns. At the end, the developer can generate a XML document to facilitate the pattern implementation.

Between discussions and the case study presented here, the advantages and disadvantages of these definitions and methods have been highlighted. The main advantage of these methods is the ability to identify the origin of the M-D pattern and to revisit it in including the current innovation, cross-platform environment. “Experience” and “usability” are both defined in its meta-model, illustrations and applications. The main disadvantage is the measure of performance and reliability of these methods. Developers understand the definition and capability of these methods but it need more quantify evaluations. Therefore, the futures works are to evaluate them in long term in integrating more parameters based on the context of uses and usability points. In addition to the user and developer experiments, another future work is to quantify the gain of times of its implementation on cross-platforms.

Acknowledgements

The authors would like to thank the Computer Science Master student Tristan Dorange for his participation in the study to build the framework MDPAG. We also acknowledge the support of this work by the DESTINE (Design & Evaluation Studio For Intent-Based Ergonomic Web Sites) project, funded by DGO6 of Walloon Region, under convention #315577 in «WIST» Wallonie Information Science & Technology research program

References

1. Aquino, N., Vanderdonckt, J., Condori-Fernández, N., Dieste, Ó., and Pastor, Ó. (2010). Usability Evaluation of Multi-Device/Platform User Interfaces Generated by Model-Driven Engineering, Proc. of ESEM'2010.ACM Press, New York, Article #30.
2. Alexander, C. (1977). A pattern language, Oxford University Press, New York.
3. Android Developers (2012). UI Framework Changes in Android 1.5 accessible at <http://jayxie.com/mirrors/android-sdk/resources/articles/ui-1.5.html>.
4. Balsamiq Morkup support (February 2014). Accessible at <http://balsamiq.com/>
5. Bayle, E., Bellamy, R., Casaday, G., Erickson, T., Fincher, S., and Grinter, B. (1998) Putting it all together: towards a Pattern Language for Interaction Design. SIGCHI Bulletin 30, 1,17-24.
6. Brochers, J.O. (2000). A pattern approach to interaction design. Proc. of ACM Conf. on Disiging Interactive Systems DIS 2000. ACM Press, New York, pp. 369-378.
7. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. (2003). A Unifying Reference Framework for Multi-Target User Interfaces, Interacting with Computers, 15, 3, pp. 289–308.
8. Coram T. and Lee J. (2011). Experiences-a pattern language for user interface design. Accessible at <http://www.maplefish.com/todd/papers/Experiences.html>
9. Dijkstra, E. W. (1959), Numerische Mathematik In Numerische Mathematik, Vol. 1, No. 1. pp. 269-271, doi:10.1007/bf01386390
10. Eclipse Documentation (September 2012), Master/Details block, . Accessible at http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Fforms_master_details.htm
11. Engel, J., Märtin, C., Herdin, C. and Forbrig, P. (2013). Formal pattern specifications to facilitate semi-automated user interface generation. Proc. of HCI International'2013, M. Kurosu (ed.), LNCS, vol. 8004. Springer, Heidelberg, pp. 300–309
12. Folmer, E. and Bosch, J. (2003). Usability Patterns in Software Architecture. Proc. of Workshop on SE-HCI, Greece, pp. 61–68.
13. Forbrig, P. and Lämmel, R. (2000). Programming with Patterns, Proc. of 34th TOOLS 2000, Santa Barbara, pp.159-170.
14. Forbrig, P. and Wolff, A. Different kinds of pattern support for interactive systems. Proc. of the 1st International workshop on pattern-driven engineering of interactive computing systems PEICS '10. ACM Press, New York, pp. 36-39.
15. Gamma, E., Helm, R., Jonhson, R., and Vlissides, J.,(1994). Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, New York.
16. Henninger, S. (2001). An Organizational Learning Method For Applying Usability Guidelines and Patterns. Proc. of EHCI'2001, pp. 141-156.

17. Henninger, S., Keshk, M., and Kinworthy, R. (2003). Capturing and disseminating usability patterns with semantic web technology. CHI 2003 workshop.
18. Javahery, H. (2002). Usability Pattern-Oriented Design: Maximizing Reusability of Pattern Languages over the web, Faculty of Engineering and Computer Science Concordia University. Accessible at <http://www.cusec.net/archives/2002/javahery.pdf>
19. Johnson, J. (2003) Web bloopers, 60 Common Web Design Mistakes and How to avoid them. Morgan Kaufman, San Francisco
20. Kruschitz, C., XPLML - a HCI pattern formalizing and unifying approach. Proc. Of CHI EA2009.
21. Mac Developer Library (2012). Cocoa Bindings Programming Topics, Creating a Master-Detail Interface. Accessible at <https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/CocoaBindings/Tasks/masterdetail.html>
22. Molina, P., Meliá, S. and Pastor, O. (2002) JUST-UI: A User Interface Specification Model. Proc. of CADUI 2002. Kluwer Academics, Dordrecht.
23. Molina, P.J. (2004). User interface generation with OlivaNova model execution system. Proc. of IUI'2004. ACM Press, New York, pp. 358-359.
24. Nielsen, J., Lorange, H., and Shade, A. (2002). Usability of Rich Internet Applications and Web-Based Tools. Nielsen Norman Group.
25. Nilsson, E.G. (2009). Design patterns for user interface for mobile applications, Advances in engineering software, vol. 40,12, Oxford, pp. 1318-1328.
26. Pastor, O. and Molina, J.-C. (2007). Model-Driven Architecture in Practice. Springer, Berlin.
27. Paterno, F. (1999). Model-based design and evaluation of interactive applications. Springer, Berlin.
28. Pribeanu, C. (2006). Task Modeling for User Interface Design: A Layered Approach. International Journal of Information Technology vol.3, 2, pp. 86-90.
29. Pemberton, L. and Griffiths, R. (1999). The Brighton usability pattern collection. Accessible at <http://www.it.bton.ac.uk/Research/patterns/home.html>
30. Perrins, M. (2008). The 12 Patterns for User Interface Design. Accessible at <http://mattperrins.wordpress.com/2008/12/21/the-12-patterns-for-user-interface-design/>
31. PJMolina (2013), Conceptual User Interfaces Patter, Master/Detail Interaction Unit <http://pjmolina.com/cuip/node-MasterDetailIU>
32. Radeke, F. and Forbrig, P. (2007). Patterns in task-based modeling of user interfaces, Proc. of 6th Inter-national workshop on task models and diagrams for user interface design, TAMODIA 2007. Springer, Berlin, LNCS, vol. 4849.
33. Seffah, A. and Forbrig, P. (2002). Multiple user interfaces: towards a task-driven and patterns-oriented design model, Proceedings of DSV-IS 2002.
34. Sinnig, D., Forbrig, P., and Seffah, A. (2003). Patterns in Model-Based Development, INTERACT 03 Workshop Software and Usability Cross-Pollination: The Role of Usability Patterns, September 2003.
35. Scapin, D.L. and Bastien, J.M.C (1997), Ergonomic criteria for evaluating the ergonomic quality of interactive systems. Behavior & Information Technology, vol 16, 4/5, pp. 220–231.
36. Tidwell, J. (2010). Designing interfaces, patterns for effective interaction design, 2nd ed., O'Reilly Media Inc.
37. Vanderdonckt, J. and Montero, F. (2010). Generative Pattern-Based Design of User Interfaces, Proc. of 1st International Workshop on Pattern-Driven Engineering of Interactive Computing PEICS'2010. ACM Press, New York.
38. van Duyne, J., Landay, J., and Hong, J. (2006). The design of sites, patterns for creating winning websites. 2nd ed., Prentice Hall International.

39. van Welie M. and van Der Veer, G.C. (2003). Pattern Languages in Interaction Design: Structure and Organization. Proc. INTERACT '03, Zurich, Switzerland, pp. 527-534.
40. van Welie, M., van der Veer, G.C. , and Eliens, A. (2000). Patterns as Tools for User Interface Design. Proc. of International Workshop on Tools for Working with Guidelines TFWWG'2000. Springer-Verlag, Berlin, pp. 313-324.
41. Wendler, S., Ammon, D., Philippow, I. and Streitferdt, D. (2013).A factor model capturing requirements for generative user interface patterns. Proc. of Patterns 2013, IARIA.