MoCaDiX: Designing Cross-Device User Interfaces of an Information System based on its Class Diagram

JEAN VANDERDONCKT, Université catholique de Louvain, Belgium
THANH-DIANE NGUYEN, ICHEC Brussels Management School, Belgium, Belgium

This paper presents MoCaDiX, a method for designing cross-device graphical user interfaces of an information system based on its UML class diagram, structured as a four-step process: (1) a UML class diagram of the information system is created in a model editor, (2) how the classes, attributes, methods, and relationships of this class diagram are presented across devices is then decided based on user interface patterns with their own parametrization, (3) based on these parameters, a Concrete User Interface model is generated in QuiXML, a lightweight fit-to-purpose User Interface Description Language, and (4) based on this model, HTML5 cross-device user interfaces are semi-automatically generated for four configurations: single/multiple-device single/multiple-display on a smartphone, a tablet, and a desktop. From the practitioners' viewpoint, a first experiment investigates effectiveness, efficiency, and subjective satisfaction of three intermediate and three expert designers, using MoCaDiX on a representative class diagram. From the end user's viewpoint, a second experiment compares subjective satisfaction and preference of twenty end users assessing layout strategies for interfaces generated on two devices.

Additional Key Words and Phrases: Automated generation of user interfaces; cross-device interaction; cross-device user interfaces; design exploration; design option; mapping rule; unified modeling language; class diagram.

ACM Reference Format:
Jean Vanderdonckt and Thanh-Diane Nguyen. 2019. MoCaDiX: Designing Cross-Device User Interfaces of an Information System based on its Class Diagram. *Proc. ACM Hum.-Comput. Interact.* 3, EICS, Article 17 (June 2019), 40 pages. https://doi.org/101145.3331159

1 INTRODUCTION

In today's computing era, carrying out a same task on many computing platforms or devices or across these devices has become more important and prevalent than ever. For some time, *multi-platform user interfaces* (MUIs) [3, 42, 48, 61] have been investigated to enable end users to carry out their tasks on the platform or device of their choice. The assortment of potential devices, ranging from the smallest ones like smart watches to the largest ones, like wall screens, grows everyday, thus keeping MUIs as a key for reigning in demanding end users.

Despite their constant search for consistency, many MUIs deliver different user experiences across different platforms [58]. Therefore, the greatest success is expected to come from creating a cohesive *Cross-Device Interaction* (XDI) through *Cross-Device User Interfaces* (XDUIs) [31]. Instead of carrying out an entire task on a single device, the end user could suspend a task on one device

Authors' addresses: Jean Vanderdonckt, Université catholique de Louvain, LouRIM Institute, Place des Doyens, 1, Louvain-la-Neuve, 1348, Belgium, jean.vanderdonckt@uclouvain.be; Thanh-Diane Nguyen, ICHEC Brussels Management School, Belgium, Rue au Bois, 365a, Brussels, 1150, Belgium, thanh-diane.nguyen@ichec.be.

2573-0142/2019/6-ART17 $15.00
https://doi.org/101145.3331159

and resume it on another. In contrast to MUI, XDI distributes functionalities across multiple devices with the same (*All* in [58]) or with different (*Any* in [58]) methods. During the last decade, there has been an increasing interest in XDI [28, 30, 44, 49, 60] in general, as a MUI extension [23, 26, 53], pushed on one side by technological advances and availability of a ever-growing palette of devices and pulled on the other side by always-demanding end users willing to benefit from these devices.

Instead of studying XDI for any type of interactive application in general, we will focus throughout this paper on *Information Systems* (ISs), which are referred to as the process of and software support for collecting, storing, updating, retrieving of data and communications in any business-oriented organisation [51]. ISs represent a promising field for applying XDI:

- MUI is already in place in many case studies [59].
- ISs belong to the most frequent type of applications largely accessible through the Web [35].
- They share a systematic usage of common, yet simple, management tasks, such as Create-Read-Update-Delete-Search (CRUDS) [43].
- Their UIs tend to be similar in terms of structure and navigation as their interaction style remains consistent [41]: form filling, menu selection, and window-based interaction.
- They require a minimal syntactical and semantic knowledge [22]. Three IS usage patterns have been identified that make IS particularly suitable for XDI [33]: independent usage into resource lending (e.g., one smartphone for one task), unrelated parallel use (e.g., different devices for different tasks), and related parallel use (e.g., multiple devices for the same task).

A *domain model* [16, 25] expresses a conceptual view of a certain domain of human activity [21] with both structural and behavioural characteristics [51]. A domain model could be represented in several ways, like a UML class diagram, an entity-relationship-attribute model, or an object-oriented model. For any IS, a domain model is almost always produced [22, 35], thus offering a permanent starting point for initiating the development, user interface included.

This paper contributes to the XDI research with MoCaDix, a tool-supported method specifically tailored to facilitate the deployment of *Cross-Device User Interfaces* (XDUIs) of ISs over the Internet based solely on a UML class diagram. MoCaDix provides the following original contributions:

- *On the conceptual viewpoint*: a Concrete User Interface (CUI) model capturing a XDUI for IS in terms of QuiXML, a lightweight fit-to-purpose User Interface Description Language (UIDL), a set of device-independent user interface patterns to transform an initial class diagram into a CUI model based on mapping rules, and a definition of design options targeting XDI for four configurations: single vs. multiple devices with single vs. multiple displays.
- *On the methodological viewpoint*: a four-step method for designing XDUI for IS based on the CUI model, the patterns and their design options for governing the XDI.
- *On the implementation viewpoint*: a new, cloud computing-based deployment environment supporting the aforementioned method providing designers and developers with on-line editing and generation capabilities for XDUIs which are accessible by end-users through the same environment via three devices in four configurations.
- *On the experimental viewpoint*: two experiments assess MoCaDix's impact from the designers and developers' viewpoint and from the end user's viewpoint.

The remainder of this paper is structured as follows: Section 2 reports on some selected related work and discusses the background. Section 3 introduces, motivates, defines, and explains the four steps of the MoCaDix method and exemplifies its process based on a running example. Section 4 reports on the results obtained from the first study involving designers. Section 5 reports on the results obtained from the second study involving end users. Section 6 concludes the paper, discusses some threats to validity, and discusses some futures avenues of this work.

2 RELATED WORK AND BACKGROUND

Various definitions [26, 28, 30, 44, 49, 58] exist for XDI, each of them having its own focus, and interests. Among them, we opted for Scharf et al.'s definition [60], a general one suitable to the IS field: "XDI is the type of interaction where end users interact with multiple separate input and output devices, where input devices will be used to manipulate content on output device within a perceived interaction space with immediate and explicit feedback". Based on this definition, XDI could be structured along three axes [60]: *ownership*, which specifies who owns the input/output device; *access*, which specifies who has the right to access to each device; and *distance*, which specifies to what distance the output devices are located.

To support designing XDUIs, different tools range from graphical, direct manipulation authoring environments targeting novice designers to programming frameworks with or without visual editors targeting expert designers and developers. The first category contains representative examples such as: UIDistributor for distributed GUIs across multiple users equipped with multiple devices [26], context-dependent XDUIs [28], and XDStudio [45] for authoring XDUIs visually on emulators or actual devices. The second category also contains representative examples, among them are Conductor [30] and Panelrama [66] which uses a constraint-based approach for specifying XDUIs.

XDBrowser [44] falls between these two categories: it preserves the flexibility of quickly producing XDUIs over the Internet while offering a general purpose XDI programming language for more sophisticated XDI portions, thus marrying both worlds without their respective shortcomings. Paay et al. [50] demonstrate that a XDI could be initiated between a smartphone and a large display by relying on four simple gestures for manipulating objects (i.e., pinching, swiping, swinging and flicking). These gestures are aimed at transferring data from the smartphone to the large display, they are not aimed at covering the traditional methods, such as CRUDS. A property that is common to all these tools is that they target a general-purpose XDUI by direct manipulation. As such, they could be of course applied to any interactive system, such as ISs, the focus of this paper. In this case, the XDUI needs to be created entirely.

On the other, there are several approaches and frameworks that are very effective for producing interactive applications with their UI for the Web: The Google Web Toolkit (GWT) [1], Hop [62], Ocsigen [5], and Ur/Web [10]. But none of them address the specific requirements of XDUI. In their case, even if a single development language is promoted to write the entire application and it UI, everything should be constructed by hand.

In the field of *Model-Based User Interface Design* (MBUID) [19, 54, 57], many models initiate and feed a full UI development life cycle, either in isolation or in combination with another: task, domain, user, abstract UI, concrete UI, device, environment, help, tutorial. Since the UML class diagram is prevalent in IS development, we review only some of the MBUID approaches relying on a similar model, such as: Trident [65], Genius [32], Mecano [55], Janus [6], Mobi-D [56], Huddle [48], ProtoUI [24], AutoGen [16–18], DB-Use [64], Shatnawi [63], Just-UI [43], Seffah & Forbrig [61], OlivaNova [4], UsiMad [46], Modus [41].

Table 1 sorts in chronological order some significant MBUIDs involving a model comparable to the UML Class Diagram according to the following scheme: what kind of input is required to initiate the UI development life cycle?, what are the instruments for ensuring the process?, what is its output?

Regarding the **input**, progressively more complex and expressive models have been considered: first simple data models, without the relationships between data, then the Entity-Relationship-Attribute (ERA) model with its extension almost in parallel with the advent of object-oriented models [11]. The UML class diagram finally imposed its reference status. None of the surveyed

[1] http://www.gwtproject.org/

MBUID	Year	Input	Process	Output
Trident	1993	ERA model	Expert system with ECA	1 GUI for Windows and 1 for DEC OSF/Motif
Genius	1993	Extended ERA model	Rule-based approach	1 GUI in C for OSF/Motif large workstation
Mecano	1994	Data model with inheritance only	Expert system with rule-based approach	1 GUI in C for NextStep workstation
Janus	1995	OO model with inheritance and aggregation	Code generation	1 GUI in C for Windows high-end desktop
Mobi-D	1997	OO model with inheritance	Template-based generation	1 GUI for Windows desktop
Huddle	2006	Specifications	Code generation	n GUIs in Java for desktop and mobile devices
ProtoUI	2006	UML 1.0 scenarios	Code generation	1 GUI for desktop
AutoGen	2009	UML 2.0 Class diagram and use case	Code generation	n GUIs for desktop based on parameters
DB-Use	2010	Task, user, domain models	Java production rules	1 GUI in Java for desktop
Shatnawi	2016	UML 2.0 Class diagram and use cases	Code generation	1 language-independent GUI for desktop
Just-UI	2002	UML 1.0 Class diagram	5 Pattern-based approach	1 GUI in C for Windows desktop
Seffah & Forbrig	2006	Pattern definitions and task model	Template-based generation	n GUIs for desktop and mobile devices
OlivaNova	2007	UML 2.0 Class diagram, presentation model	Model-to-model transformations	3 GUIs for desktop: 1 in Java, 1 in C++ for Windows, and 1 is ASP.Net
Usimad	2016	UML 2.0 Class diagram	Model-to-model transformations	2 GUIs: 1 for desktop and 1 for mobile devices
Modus	2017	UML Class diagram, UI specifications	Architectural patterns, mapping rules	n GUIs for web applications

Table 1. Comparison of some model-based approaches.

work solely exploits the class diagram as a unique starting point: it is always complemented with other UML models [8], such as use case [18], activity diagram [15], scenarios [24], prototypes [17], and UI specifications [41]. For example, DB-USE requires three input models before obtaining any UI, which raises the threshold of workload load needed before getting any result [64]. The UML class diagram still prevails often with other modelling constructs, such as a task model [61], a presentation model [4] or a suite of SE-MDE models [22, 35].

Regarding the **process**, several techniques have been exploited from a simple set of rules, sometimes gathered in a knowledge base or an expert system, to a large set of production rules [55], trigger-action rules [28], or mapping rules [41], all being variants of the general purpose Event-Condition-Action (ECA) production rule. Various code generation techniques have been also successfully investigated, such as template-based, pattern-based, or skeleton-based. With the advent of Model-Driven Engineering (MDE), Model-to-Model (M2M) and Model-to-Code (M2C), transformations [51] replaced previous sets of rules. Virtually any transformation could be defined to support additional design options for different devices, but this flexibility induces some cost [47]: creating, editing, managing, and applying these transformations reveal complexity challenges that

are hard to solve. The challenge of managing a large set of rules or transformations, which requires some prioritization, still remains [28].

Regarding the **output**, more devices are supported with flexibility. MUIs progressively arrive when they were introduced on the market, up to full MUI generation [23, 26, 53].

In conclusion, current technologies do not support adequately XDUIs for ISs per se. We did not identify any work that solely starts from the last version of UML class diagram V2.5 for producing XDUIs to an IS (Table 1).

3 THE MOCADIX METHOD

Since ISs are our primary target, we hereby define a *XDI configuration* as a first-order predicate logical formula `C=InputOutputDevices` $\wedge$ `OutputDevices`, where `InputOutputDevices` expresses any combination of input/output devices S (smartphone), T (tablet), D (desktop) with a subscript indicating i=input, o=output, io=input/output and where `OutputDevices` expresses the combination of output devices in the same way. Based on this formula, we introduce four configurations as a basis for MocaDix (Fig. 1):

(1) Single device-single display (SDSD): only one display is associated with the input device (Fig. 1a): $C = S_{io} \vee T_{io} \vee D_{io}$ (only one device among three serves as input and output). This represents the traditional case of a single-device GUI: everything is concentrated on one device.

(2) Single device-multi-display (SDMD): more than one display is associated to the input device (Fig. 1b): $C = (S_{io} \vee T_{io} \vee D_{io}) \wedge (S_o \vee T_o \vee D_o)$ (at least one device among three serves as input and at least two displays as output). This configuration represents typical cases such as dual screen, multi-monitor: the input CIU is assigned to one input device and the output CIU is directed to the remaining displays.

(3) *Multi-device-single-display* (MDSD): more than one device is involved, each coming with its own single display (Fig. 1c): $C = (S_{io} \vee T_{io} \vee D_{io}) \wedge (S_o \vee T_o \vee D_o)$. This configuration represents the MUI case. For instance, an entirely independent UI is generated for each device.

(4) *Multi-device-multi-display* (MDMD): more than one device is involved with an independent suite of displays, not necessarily with a one-to-one mapping between a device and a display (Fig. 1d): $C = [(S_{io} \wedge T_{io}) XOR (S_{io} \wedge D_{io}) \vee (T_{io} \wedge D_{io}) \vee (S_{io} \wedge T_{io} \wedge D_{io})] \wedge (S_o \vee T_o \vee D_o)$ (at least two devices among three serve as input and at least two devices serve as output).

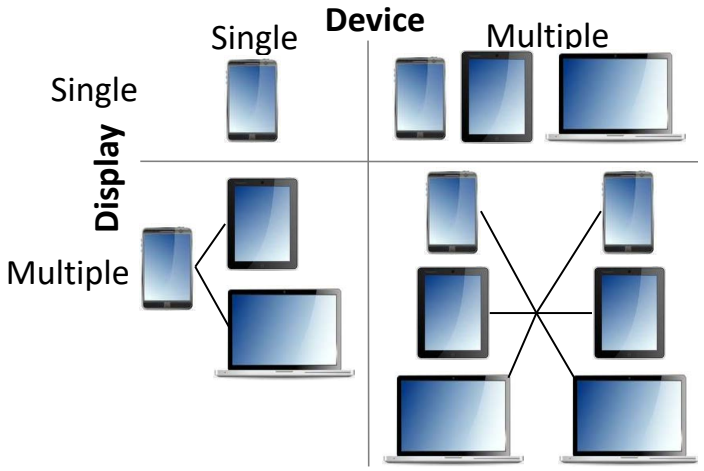

Fig. 1. Four XDI configurations: (a) SDSD, (b) SDMD, (c) MDSD, (d) MDMD.

The MoCaDix method is decomposed into four steps, each step being respectively defined, explained, and illustrated in the next four subsections:

(1) *Model editing*: a designer or a developer edits a UML V2.5 Class Diagram as defined by OMG[2] in an appropriate model editor, which stores this model in a XML file.
(2) *User Interface Pattern Selection and Parametrization*: the resulting class diagram serves as input for the designer to choose user interface patterns that are independent of any device and configuration, along with design options materialized as parameters of these patterns. For each construct of the class diagram, patterns are selected, their parameters are specified, all of them are again stored in a XML file.
(3) *Concrete User Interface Model Generation and Editing*: the class diagram and its pattern application file, serve as input to the Concrete User Interface Model Generator, which apply selected mapping rules governed by the patterns parameters to semi-automatically generate a Concrete User Interface model, stored in QuiXML, a lightweight UIDL for XDUIs at the CUI level. This model can be itself edited in its corresponding editor.
(4) *Cross-Device User Interface Generation*: the CUI model then serves as input for a semi-automated generation of cross-device user interface code in HTML5 for four configurations (Fig. 1), representing each a combination of a smartphone, a tablet, and a desktop.

MoCaDix provides designers and developers with a cloud-computing software architecture, in which all models are maintained based on a Model Manager, with accesses managed by a Access Right Manager. All resulting XDUIs are then deployed over the Internet infrastructure by URLs and Servlets and synchronized via the Model Manager (Fig. 2).

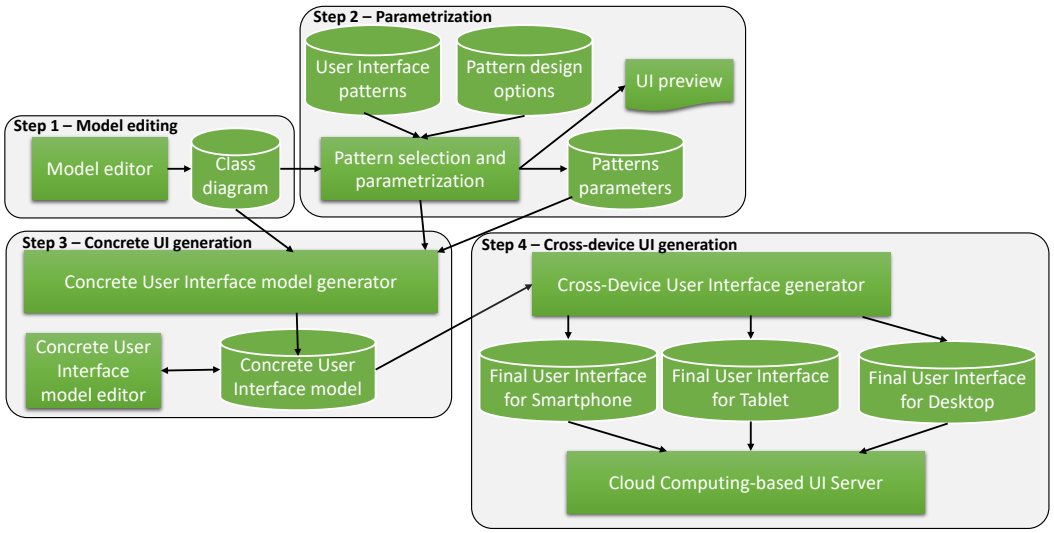

Fig. 2. Overview of the MoCaDix Method.

3.1 Step 1. Model Editing

The UML Class Diagram has nowadays become the most recent and frequent model representing a domain: this structure diagram describes a domain of interest by specifying its classes, their attributes and methods, and the relationships between these classes. Since 2000, the class diagram is the most favoured conceptual model by designers [8]. More than two-thirds of the time, the

[2]See http://www.omg.org/spec/UML/2.5/

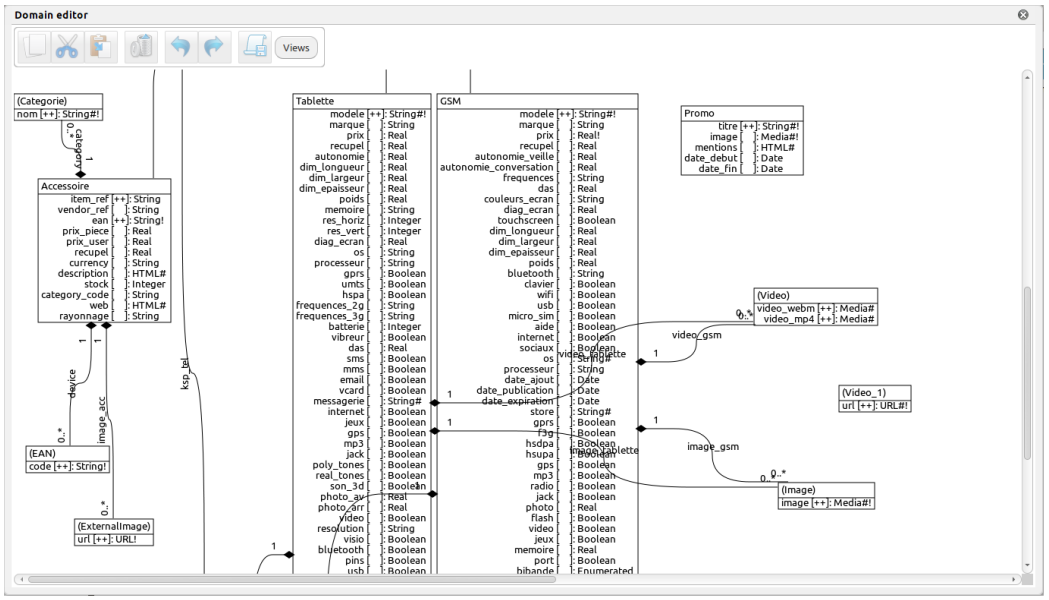

Fig. 3. The MoCaDix cloud-based model editor.

designers are engaged in specifying a class diagram, with classes, methods, and attributes as the most frequently used modelling constructs. In a dataset of 907 class diagrams studies, 423 used classes, 171 used attributes, 296 used methods, and 71 used associations. In an analysis of 121 open UML projects, all of them (100%) use class diagrams and half of them use cases (47%) [35]. One third of these projects only used one model (34%), 17% used two models, and 22% used three models. When a class diagram is used in combination with another UML model, classes and use case diagrams were used in the first preferred combination (71%). The most frequently used modelling constructs are respectively: class, attribute, method, inheritance, and association.

Fig. 3 reproduces a screenshot of MoCaDix's cloud-based model editor where a real-world large-scale class diagram is edited for an Internet Service Provider: each class is described by a class name and type (*concrete*, when the class is instantiated into immediate objects, *abstract*, when no such instantiation is meaningful), and a series of attributes, also specified with their status (*intrinsic*, when the attribute is a semantic property, *calculated*, when the attribute is derived from the values of other attributes). Each attribute is specified by various parameters (Fig. 4): an internal name, a type (*simple* vs *compound*), a data type (e.g., Boolean, hour, date, char, URL, hashtag, media, string, integer, real, or enumerated), an importance (expressing the need to be displayed when screen constraints apply, ranging from 1-could be skipped to 5-always visible), and a label (e.g., for field label to be displayed). Methods are similarly declared with a unique name, a type (*intrinsic* vs *calculated*), and a signature. The model editor enables designers and developers to create, retrieve, update, delete, and manage models on any HTML5-compliant platform. Any other class diagram editor, e.g., Visual Paradigm, could be used provided that XMI is supported to ensure interoperability.

Running example. In order to progressively illustrate MoCaDix's method, a running example is introduced from a simple class diagram: the Collection-Worker pattern from the Coad, North, and Mayfield collection [12], which is considered as a fundamental Master-Detail pattern [46]. More complex examples will be addressed in Section 4. Fig. 5 provides an overview of the full process resulting from applying the method. Each step will be explained in the next subsections.

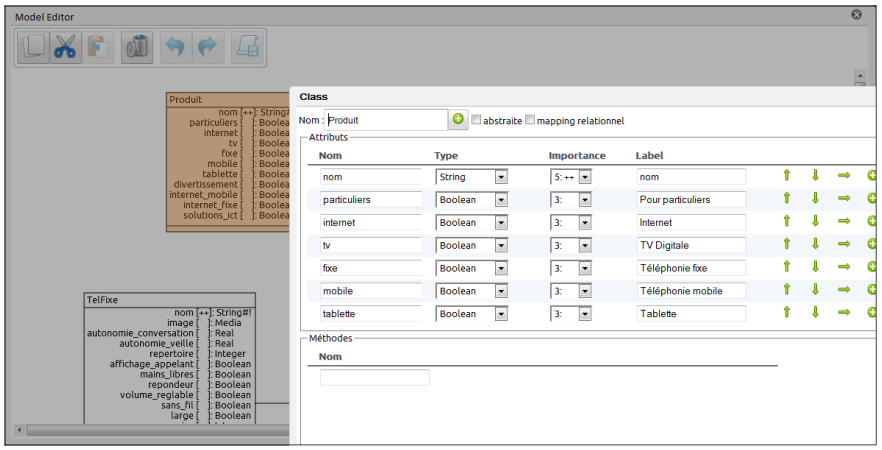

Fig. 4. Definition of attributes: name, data type, importance, and label.

3.2 Step 2: User Interface Pattern Selection and Parametrization

For each model construct present in the class diagram resulting from Step 1, device-independent graphical user interfaces patterns are selected in a pattern selector, and their corresponding parameters are decided according to the design options decided by the development team. These patterns are manipulated at the level of Concrete User Interfaces (CUI) [9], i.e., the abstraction level where user interfaces are defined for a particular interaction modality, here Graphical User Interfaces (GUIs), but in a way that is independent of any target device. These patterns, along with their parameters, are consequently expressed in terms of GUI design options, but without any reference to any implementation (e.g., in terms of a programming or declarative language) and software/hardware device (e.g., in terms of physical properties of the devices).

For this purpose, several families of patterns selection rules are defined, each family addressing a sub-problem of the overall problem of XDUI construction. A special attention is paid to those pattern parameters that are directly relevant to XDI, independently of the devices involved in any future XDI. A mapping rule consists of a configuration directive that causes a model construct in the Model Manager to be processed in some way, passed to an origin server, redirected, or rejected. Setting mapping rules correctly is important to the proper functioning of the overall process. Mapping rules affect the following: model transformation, basic proxy function, access to the browser-based configuration, and ability to cache Servlet results and other dynamically generated content. Mapping rule directives use the following format:

```
rule id template target [address | host_name]:[port]
```

where *id* represents an identifier of the mapping rule, template denotes a filtering condition applied on the source model, target denotes the target model on which the mapping rule will be applied, `address`, `host name`, and `port` respectively specify the path where the mapping rule will store its results. Only requests that match the specified template and address-port combination are subject to that rule. A template can contain wildcards, for example, https://**/*.asp. The order in which the rules appear in the family is significant, for which the id explicitly represents the order according to which a mapping rule will be considered. As soon as the mapping rule is matched to a template, it is executed and subsequent mapping rules are not evaluated. A `Map` directive could replace the URL in the request, which then continues to be compared to the remaining mapping rules.

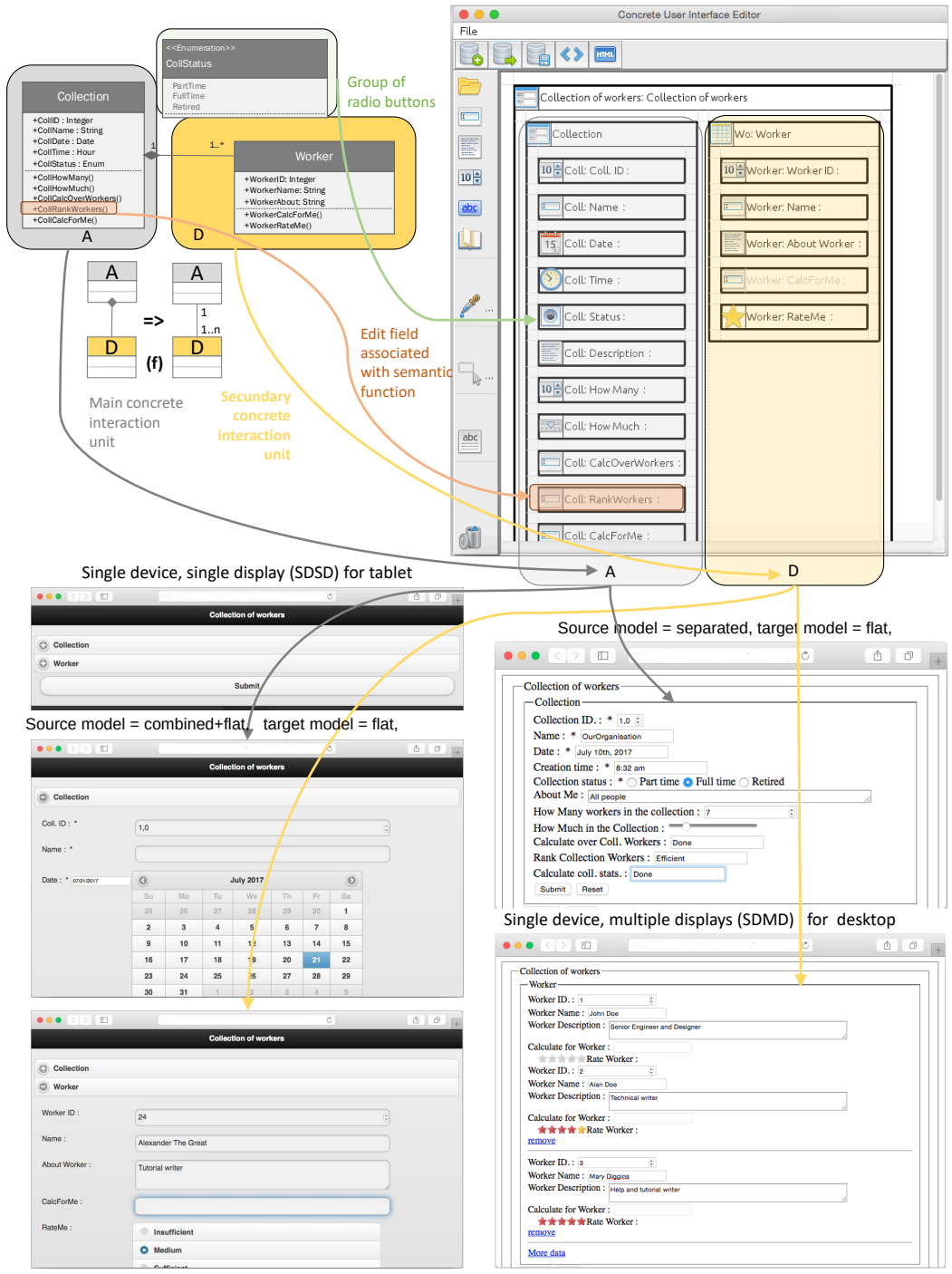

Fig. 5. The running example starting from the Collection-Worker model.

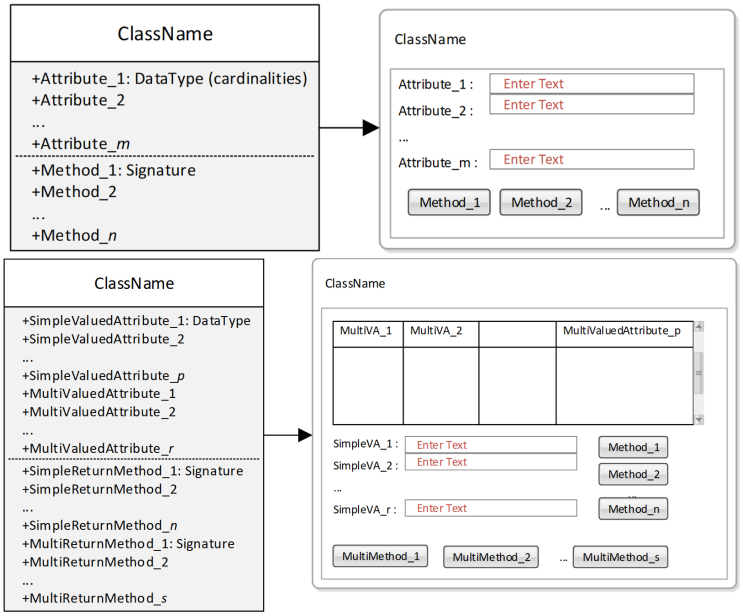

Fig. 6. Class Mapping Rules.

3.2.1 Class Mapping Rules. This family of rules is responsible for mapping any class in the diagram into a *Concrete Interaction Unit* (CIU) [57], an interaction unit defined for a graphical modality, but independently of any implementation technology. An interaction unit is defined by W3C as "any entity that supports a task or a set of sub-tasks that are logically related to achieve a goal or a set of goals" [57]. Fig. 6 summarizes how a CIU is formed for each class containing simple or repeated attributes: a CIU is created for any class and populated by the CUI widgets resulting from the previous selection. CIUs are represented as generic windows, and will be later on mapped to any type of container, such as a web page, a section in a web page, a window, a dialog box, a tabbed dialog box, or a group box.

3.2.2 Attribute Mapping Rules. This family of rules is responsible for mapping the meta-data of any class attribute into a Concrete Interaction Unit (CIU), which does not refer to any particular device. Table 2 summarizes the mapping between data type and cardinality and widgets. For instance, a single-line edit field will be selected for any string of up to 30 characters, a two-line edit field up to 60 characters, and a multi-line edit field beyond. These mapping rules can be edited so as to create new entries for specifying more appropriate or advanced widgets for any model construct. Alternate widgets will be also offered in the Step 2, but consistency will therefore no longer be guaranteed, unless a round trip engineering process takes place.

Running example. Table 3 shows how attribute mapping rules have been executed for the `Collection` class of the running example (Fig. 5). Methods are mapped onto an edit field containing the results of the executed method, which are all calculating methods here. The edit field is itself resulting from the Integer data type of Table 2. Attributes from the `Worker` class can be obtained by analogy.

3.2.3 Relationship Mapping Rules. This family of mapping rules is responsible for mapping any relationship declared in the class diagram into one or many CIUs that are structured according to design options. While attributes and class mapping rules basically define the basic blocks in the

Attribute data type	Property	CIU
Boolean		Check button
Hour		Profiled edit field with regular expression
Date		Profiled edit field with regular expression
Char		Alphanumeric edit field
URL		Link
Hashtag		Profiled edit field with regular expression
Media		Browse push button linked to media manager
String	30	Single-line edit field
	60	Two-line edit field
		Multi-line edit field
Integer	2	Slider
		Profiled edit field
Real		Profiled edit field
Enumeration	3	Group of radio buttons
	7	List box
	30	Combo box
	>30	Accumulator
Method	direct	Push button
	indirect	Edit field associated with semantic function

Table 2. Attribute Mapping Rules.

Attribute and data type	Property	CIU
`CollID`, Integer		Profiled edit field
`CollName`, String	30	Single-line edit field
`CollDate`, Date		Profiled edit field with regular expression
`CollTime`, Hour		Profiled edit field with regular expression
`CollStatus`, Enumerated	3	Group of radio buttons
`CollDesc`, String	50	Two-line edit field
`CollHowMany`		Edit field associated with calculating function
`CollHowMuch`		Edit field associated with calculating function
`CalcOverWorkers`		Edit field associated with calculating function
`RankWorkers`		Edit field associated with calculating function
`CalcForMe`		Edit field associated with calculating function

Table 3. Attributes mapped in the Collection-Worker model.

Concrete User Interface (CUI), and as such they do not directly affect whether they will be presented in a normal GUI or a XDUI, on the contrary, relationship mapping rules will directly affect how CIUs of the whole CUI will be presented, ranging from a presentation completely concentrated on one device to a presentation that is fully-distributed across devices. Domain relationships should be exploited to offer a wide range of XDUIs depending on design options chosen by the designer, developer. For this purpose, **three design options** are offered:

(1) **Source Class Presentation**: specifies whether the source class of any relationship should be generated in a separate way from the class targeted by the relationship (*separate*) or combined with the class targeted either at the same structural level in the CIU as the target class (*flat*) or be nested within each other to reflect the semantic structure of the class (*nested*). This design option applies to any source class and its immediate children. Consequently, the CIU presenting the source class is either kept separate or combined with the CIU presenting

any target class (subsequently, at the same level of the source CIU or nested to reflect the semantic structure). A typical application of this option is to specify whether a master is presented on a source device (e.g., a smartwatch) and its details are presented together or on another device (e.g., such as a desktop). Each object selected on the input device results in displaying its details on the output device, thus focusing selection of one small device and leaving ample screen real estate for details on a larger device.

(2) **Target Class Presentation**: specifies whether any class targeted by the relationship should be generated in a separate way from the other classes targeted by the relationship (*separate*) or combined either at the same structural level in the CIU (*flat*) or be nested within each other to reflect the semantic structure of the class (*nested*). This design option applies to any target class of a relationship and its immediate children when any. Consequently, the CIUs presenting the target classes are either kept separate or preferred combined with the CIU presenting any class (subsequently, at the same level of the source CIU or nested to reflect the semantic structure). A typical application of this option is to design a multi-level Master-Detail pattern, where a master is located on one device such as a (input) smartphone, a first-level detail is displayed on a second device such as a tablet (input/output), and the last-level detail is displayed on a third device such as a (output) desktop.

(3) **Target Class Collection**: specifies whether classes targeted by a relationship should be generated so as to form a semantic collection or not. The different values of this design option are: none when no class should be collected, one-to-one when classes targeted by any relationship with a one-to-one cardinality should be collected, one-to-many when classes targeted by any relationship with a one-to-many cardinality should be collected, both when classes targeted by any relationship with a one-to-one and one-to-many cardinalities should be collected, and all when classes targeted by any relation-ship with all cardinalities should be collected. This design option is expected to reflect the cardinality of any relationship into a structure of CIUs depending on its value, regardless how they can be distributed later on across one or many input/output devices. There is no value for zero-to-one and zero-to-many cardinalities because these situations could arise when no object is still created. Therefore, the presentation is replaced by a push button labelled "Create Object" to edit its first instances.

To understand the impact of these design options, let us define a hypothetical class diagram linking a source class A with six target classes $B - G$ (Fig. 7) with two cardinalities (one-to-one in left part and one-to-many in right part) and with single level vs. multi-level relationships, which results into four possible configurations (Fig. 1):

(1) Target class C is linked from source class A by a one-to-one relationship only.
(2) Target classes B and F are linked from source class A by a one-to-one relationship at one vs two levels. Hence, a first relationship exists between A and B and a second between B and F.
(3) Target class D is linked from source class A by a one-to-many relationship only.
(4) Target classes E and G are linked from source class A by a one-to-many relationship at one vs two levels. Hence, a first relationship exists between A and E and a second between E and G.

CIUs are rendered as rectangular containers in Fig. 8 and 9 to avoid referring to any specific implementation. Similarly, this rendering does not assume any particular physical layout of the containers, apart from their logical ordering. For example, the top left cell in Fig. 8 (source=target=`Flat`) represents a sequential ordering of six CIUs corresponding respectively to A, B, F, C, D, E, and G without assuming any associated layout. Later on, this ordering will be mapped to appropriate containers, such as subsequent regions in a `FlowLayout` or a tree in a `GridBagLayout` in Java terms. Different layout renderings and managers can be alternatively employed without the need to specify them at this sub-step. These design options fulfill two external IFIP quality properties [29]:

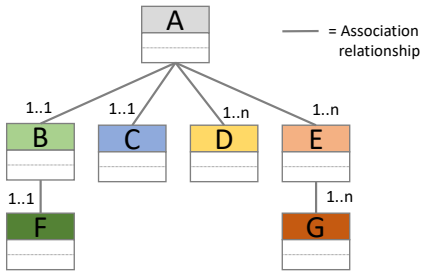

Fig. 7. Generic UML 2.5 class diagram.

observability (when attributes and methods are directly manipulable in the UI) and *browsability* (when attributes and methods are accessible on-demand: everything that is not observable is made browsable [29]). When target classes are presented together with the source class, observability is ensured. When target classes are presented into collections, browsability is obtained instead.

Twenty-four mapping rules operationalize the three aforementioned design options (18 in Fig. 8 and 6 in 9). They are valid for any relationship: association, dependency, inheritance, aggregation and composition. Consequently, they are estimated more generic than any other set of mapping rules considered in the related work: for all related work starting from a domain model, separate rules were defined for each specific relationship and only some of them were supported (e.g., JANUS [6] supports five mapping rules for inheritance only). In contrast, MoCaDix's mapping rules are the most generic, yet covering XDI. When a specific relationship is involved, only some predefined values are acceptable for the three design options, thus leading to restrictions (see Fig. 12):

- *Simple inheritance*: a simple inheritance is structurally equivalent to a one-to-many association (Fig. 12a): a series of transformations could transform a simple inheritance into this corresponding association without any loss of information and vice versa), therefore restricting the coverage of Fig. 8 to only *A* and *D* classes with a one-to-many relationship between. This simplification leads to Fig. 10.
- *Multi-level inheritance*: a multi-level inheritance is structurally equivalent to two subsequent one-to-many associations (Fig. 12b), therefore restricting the coverage of Fig. 8 to only *A*, *E*, and *G* classes.
- *Multiple inheritance*: a multiple inheritance, which typically represents polymorphism, is structurally equivalent to two simple inheritances having two super-classes and one sub-class sharing the two super-classes (Fig. 12c), therefore restricting the coverage of Fig. 8 to *A*, *D*, and *H* classes. This simplification leads to Fig. 11.
- *Multi-level multiple inheritance*: if a multiple inheritance is combined with a multi-level inheritance, the class diagram of Fig. 12d is obtained, which is structurally equivalent to two one-to-many association starting from the super class *A* to two sub-classes *B* and *C*, both of them being themselves super-classes of a common sub-class *D*.
- *Complex inheritance*: if a multiple inheritance is combined with a simple-level inheritance, the class diagram of in Fig. 12e is obtained, which is structurally equivalent to a super-class *A* having two sub-classes, *B* and *C*, in which *C* is itself a sub-class of another super-class *D*.
- *Composition*: a composition is structurally equivalent to a one-to-many association (Fig. 12f), therefore restricting the coverage of Fig. 8 to only *A* and *D* classes with a one-to-many relationship between.
- *Aggregation*: an aggregation is structurally equivalent to a zero-to-many association (Fig. 12g), therefore restricting the coverage of Fig. 8 to only *A* and *D* classes with a one-to-many relationship between and a zero-object case.

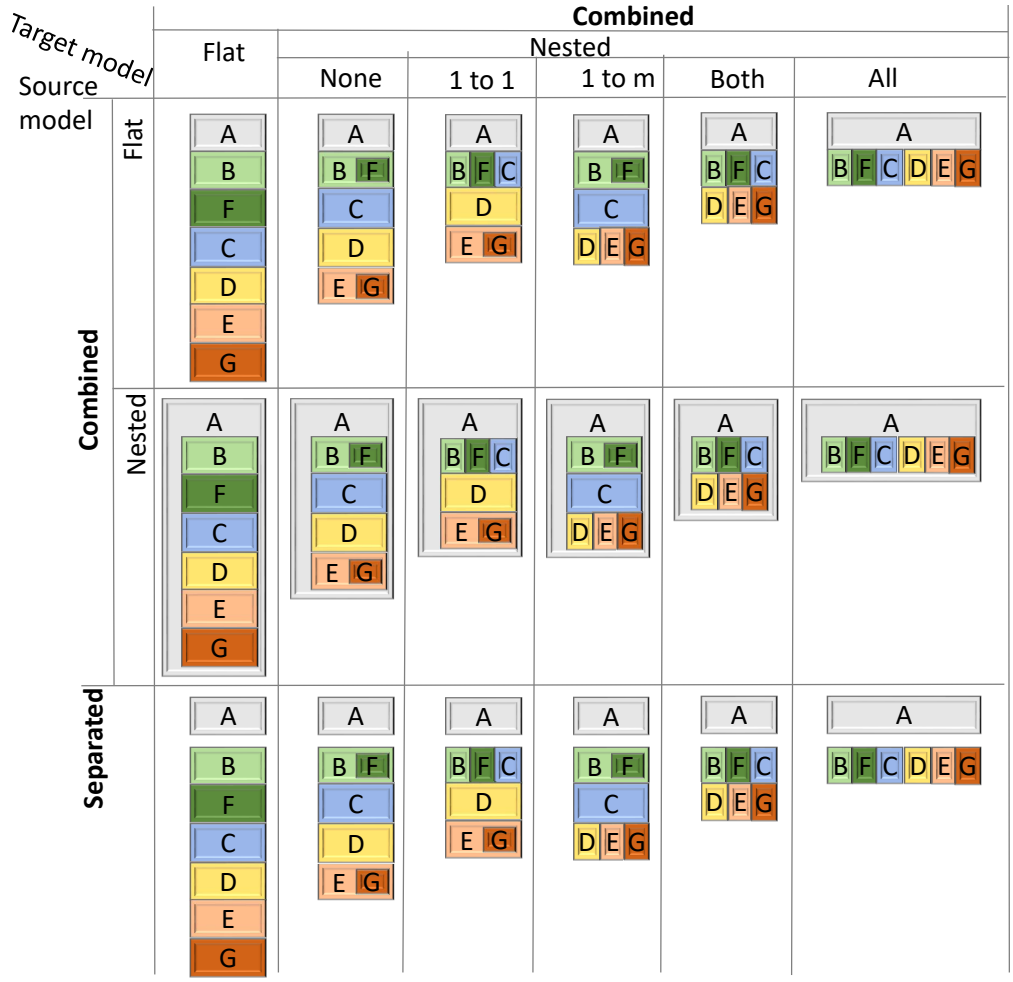

Fig. 8. Source/target class presentation and collection.

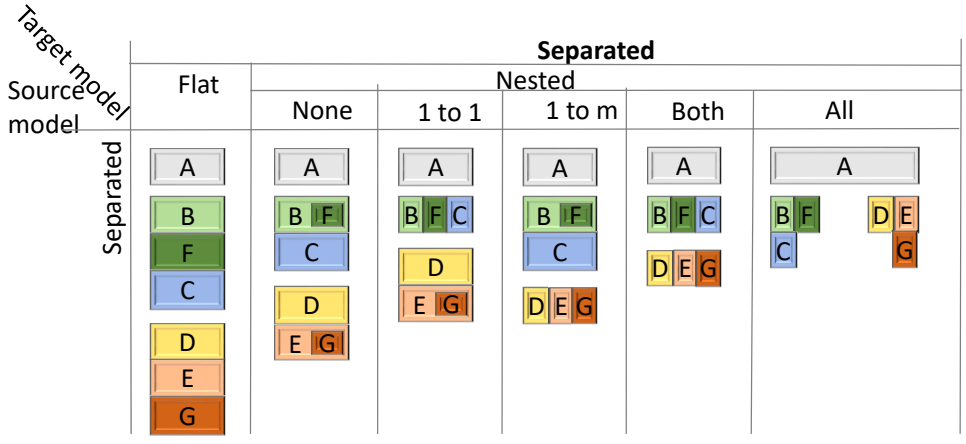

Fig. 9. Source/target class presentation and separated distribution.

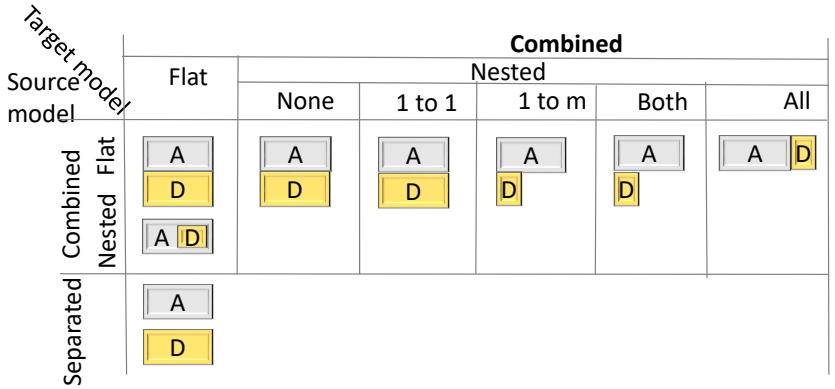

Fig. 10. Source/target class presentation for a simple inheritance.

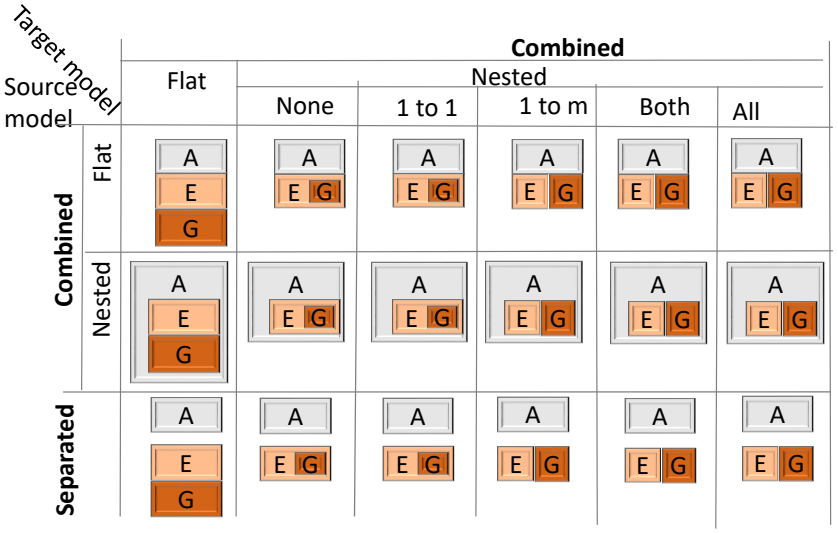

Fig. 11. Source/target class presentation for a multiple inheritance.

- *Dependency*: a dependency is structurally equivalent to a one-to-many association (Fig. 12h), therefore restricting the coverage of Fig. 8 to only *A* and *D* classes with a one-to-many relationship between.

Running example. Top right part of Fig. 5 shows how the mapping rules have been applied to the Collection-Worker pattern. Since a composition is defined between the source class `Collection` and the target class `Worker`, the CUI basically consists of two Concrete Interaction Units (CIUs), one for each class with the following design options: source class presentation=`Combined+flat`, target class presentation=`Combined+flat`, which corresponds to the top left cell in Table 5 for the *A* and *D* classes only. Note again that the physical positioning of the two CIUs side-by-side does not induce any particular layout in the Final UI: they are only placed for visualization as they only reflect a structure.

3.3 Step 3. Concrete User Interface Model Generation and Editing

The application of mapping rules devised in Step 2 leads to populating a CUI model with its appropriate elements stored in a XML-compliant format, which is typical of the many User Interface

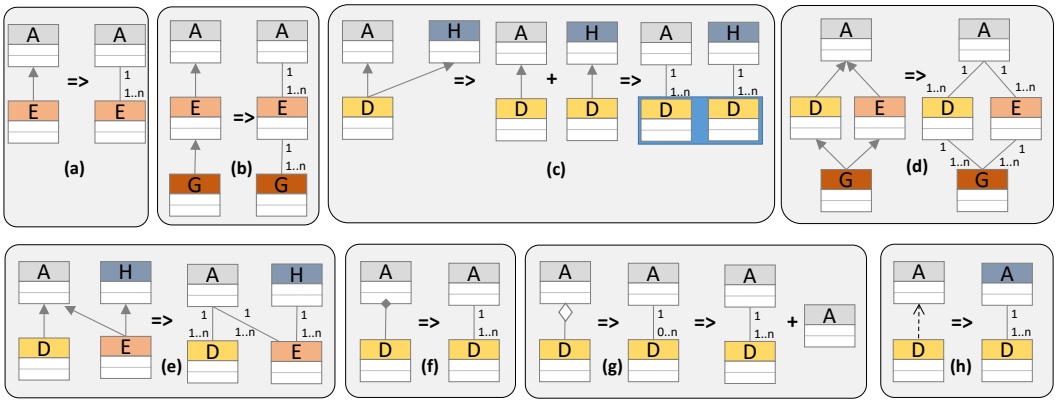

Fig. 12. Special relationships: (a) simple inheritance, (b) multi-level inheritance, (c) multiple inheritance, (d) multi-level multiple inheritance, (e) complex inheritance, (f) composition, (g) aggregation, and (h) dependency.

Description Languages (UIDLs) [27][3] introduced insofar. For expressing a CUI model in MoCaDix, QuiXML (Quick User Interface XML), a new UIDL, has been defined to fulfill these requirements:

- *CUI coverage*: although existing UIDLs cover many models at different stages of the UI development life cycle, only the CUI should be covered here in order to avoid the proliferation of models and their inherent cost. Only the graphical modality is hereby retained since information systems primarily rely on this modality, even when they are accessed via different devices.
- *Lightweight expressivity*: for specifying all the widgets that are relevant to CUI, we face the union vs intersection perpetual dilemma. A union approach assumes that the expressivity should cover all possible widgets, which is impossible because devices support these widgets in a very heterogeneous way: some suffer from inconsistent rendering, some others are only available on a particular device, some others are available through libraries (e.g., jQuery). An intersection approach subsumes determining the lowest common denominator between all widgets available on all target devices, which may result in a very small amount of widgets expressed. QUIXML seeks to find a middle ground between these two approaches: preserving the facility of specifying the most frequent widgets should be balanced with enabling external widget references to another XML-compliant file.
- *Application of the Don't Repeat Yourself (DRY) principle*: in contrast to UIDLs repeating the same concept at different levels of abstraction, QuiXML should express any concept one time and only one.
- *W3C compliance*: the modelling concepts should be termed according to the glossary recommended by the W3C Charter Group on Model-Based User Interfaces [4] and its related concepts [9].

Based on these requirements, QuiXML semantics have been defined as a UML V2.5 class diagram, which has been transformed by Model-to-Model (M2M) transformation into a XML Schema that governs any CUI specified in QuiXML. This UIDL is not intended to be actually read by human designers and developers, but to be efficiently processed by machine algorithms. Anyway, it still remains readable part by part. Once generated, the CUI model could be edited within the appropriate editor as follows, from to gross-grained to fine-grained modifications:

[3] Also see https://www.w3.org/TR/mbui-intro/
[4] https://www.w3.org/TR/mbui-glossary/

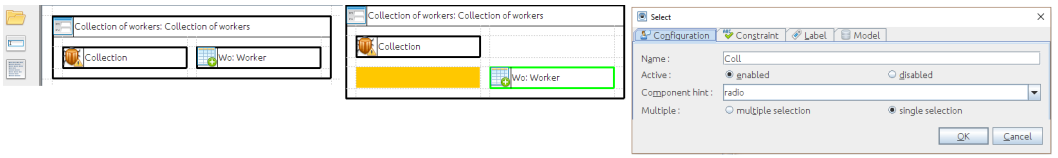

Fig. 13. CUI modifications: (a) collapsing an element, (b) moving it in the hierarchy, (c) configuring an element.

- The ordering and the structure of CUI elements is modifiable by direct manipulation without changing the rest of the specifications. In this way, alternate structures could be quickly explored without altering their individual contents. Any CUI element can be dragged and dropped at any level of the hierarchy. A compound element can be expanded to reveal its constitution or collapsed into its container with label and dragged/dropped in the hierarchy following a yellow placeholder (Fig. 13).
- The ordering and the structure of any element within a CUI: any terminal CUI element, whether a CIU or a leaf widget, could be relocated to any level in the hierarchy to reflect its semantics.
- Widgets are selected in two ways: automatically based on the attribute data type imported from the class diagram (an alternate widget can be selected provided that it accommodates the same data type by relaxing or imposing some constraint) or manually (the automatic selection can be overwritten at the risk of destroying the adequacy of widgets). For instance, an integer is automatically mapped to a slider or a profiled edit field (Table 2), but also manually re-mapped to a spin button, a radio button, a list box or a combo box. Similarly, a `Date` attribute is automatically mapped by MoCaDix to a profiled edit field or manually to a calendar. This double procedure maintains easiness of mapping rules management. A new mapping rule for selecting another widget could be incorporated, but multiplying such rules may lead to an unmanageable set of potentially conflicting rules.

Any CUI model that is automatically generated by MoCaDix. Consequently, any manual modification no longer preserves consistency with the initial class diagram and design options prevailing at generation time. When a CUI is re-generated, regulated by the same design options or not, all previously applied modifications will be lost, automatically propagated in the QuiXML code.

Running example. Fig. 14 shows an excerpt of QuiXML specifications related to the CUI. The total amount of lines for specifying this simple case is 86 QuiXML LOC (lines of code) with 357 language words (tags and delimiters included), which represents probably one of the most compact UIDLs available today, without any significant loss of information. Row and col tags specify the CUI hierarchy in terms of CIUs (e.g., lines 3,6) recursively decomposed into sub-CIUs to end up with widgets (e.g., line 14). These do not capture any physical layout to preserve its CUI level of abstraction. The `hint` tag holds the widget resulting from applying the mapping rules for widget selection. Changing a widget for another only affects this tag.

3.4 Step 4. Cross-Device User Interface Generation

Once the CUI model is finished in the CUI editor, an automatic XDUI generation is initiated: the whole CUI or some designated parts of it are assigned to one input/output device, automatically generates responsive HTML5 code by template matching for three possible devices: a smartphone (HTML5 optimised for smartphones with vertical layout and jQuery), a tablet (HTML5 optimised for tablets with horizontal layout), or a desktop (advanced HTML5). To generate a XDUI, any CUI fragment can be assigned to one device according to any configuration (Fig. 1). No device model is involved to reduce the development cost as much as possible. Taking into account the constraints imposed by a device specified in a device model would certainly make sense and improve the

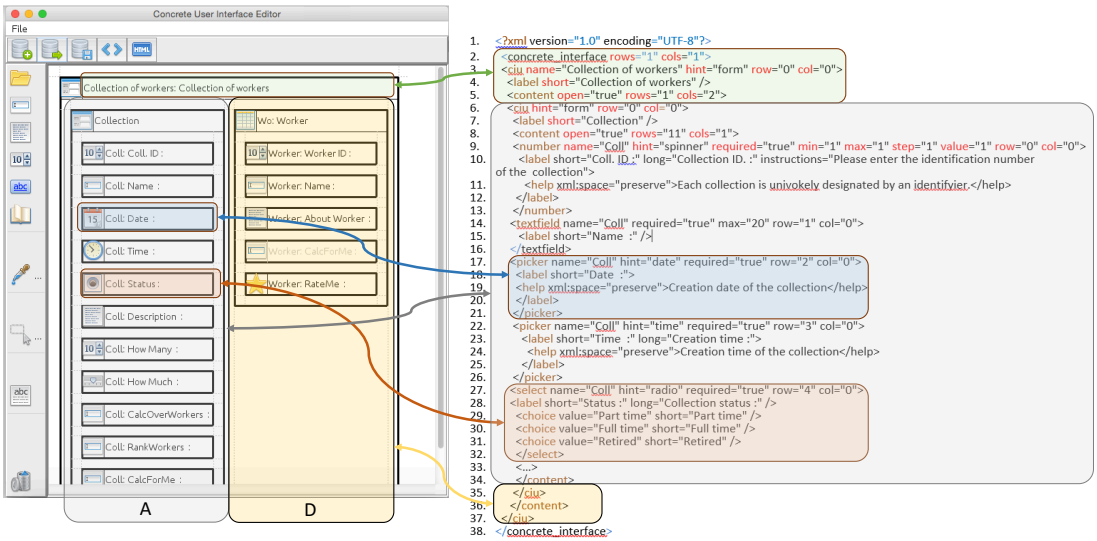

Fig. 14. QuiXML specifications of the Collection-Worker user interface (excerpt).

flexibility of the approach, but designers tend to prefer a straightforward process. Although the design options are defined in a device-independent fashion, the values assigned to each individual design option and their combination may induce a configuration that is more or less adapted to the screen real estate (e.g., mainly the screen resolution, the screen size, and the points per inch-PPI) and interaction capabilities of some particular device. Fig. 15 recommends design options values for the four configurations from the point of view of device: design options suitable for a large device (left column in Fig. 15) would probably go for `Flat` and `None` values. Moderately structured and moderately constrained devices opt for `1..1` or `1..m` (middle column), whereas highly constrained devices prefer `Both` and `All` values in theory (right column).

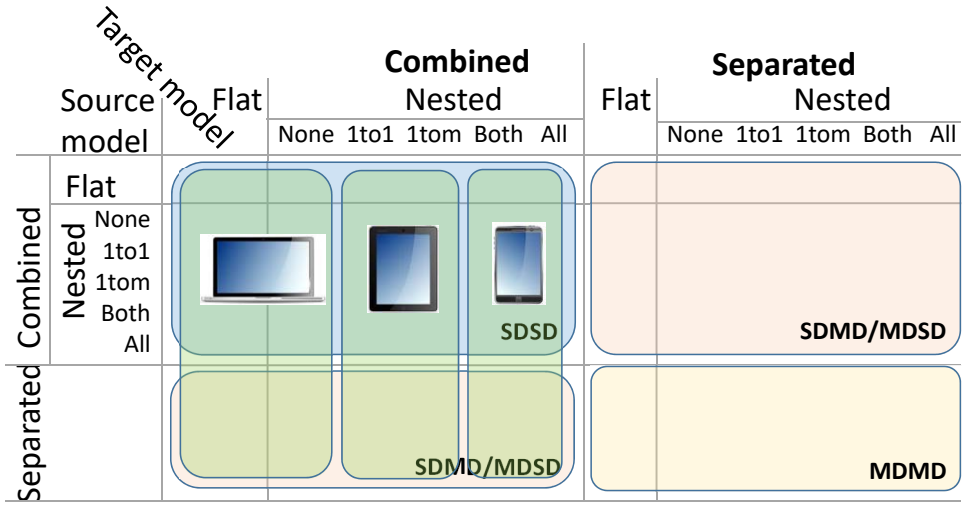

Fig. 15. Suitability of design options for the four XDI configurations.

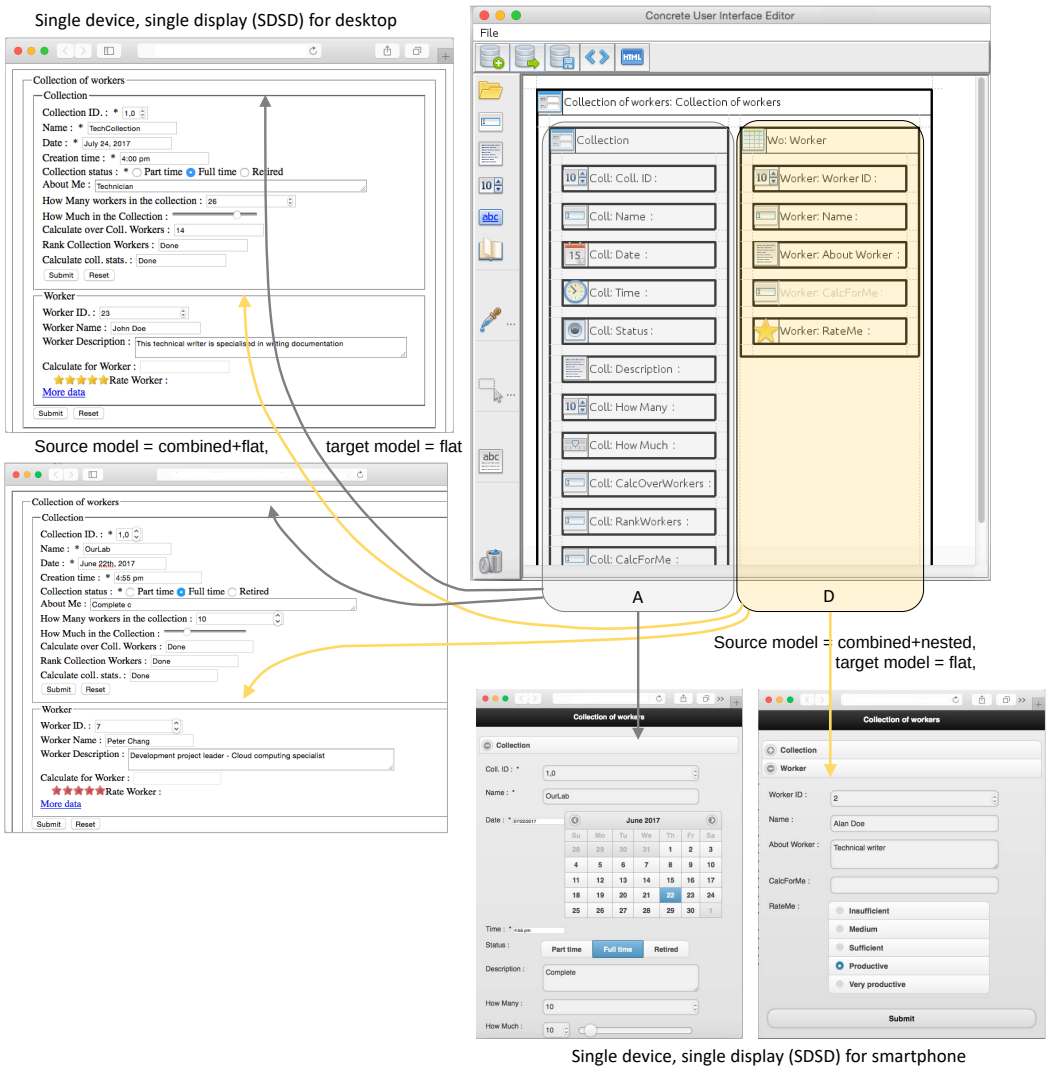

Fig. 16. XDUIs generated for 2 configurations: single device, single display (SDSD) for desktop (left) and smartphone (right).

Running example. Figs. 16 and 17 reproduce some XDUI screenshots obtained by MoCaDix for the four configurations: (a) SDSD for a tablet device displaying the collection of workers, (b) SDMD for a desktop (displaying the collection) equipped with a dual monitor (displaying the workers), (c) MDSD with versions for smartphone and desktop, and (d) MDMD with an input smartphone for selecting any Worker in the Collection and both a tablet and a desktop input/output devices for editing Workers.

4 FIRST EXPERIMENT

This section aims at conducting an experiment from the viewpoint of designers and developers: to test MoCaDix's method for designing a XDUI for a given class diagram. More specifically, the main goal of this experiment is now turned into a specific research question. For *effectiveness*, to what

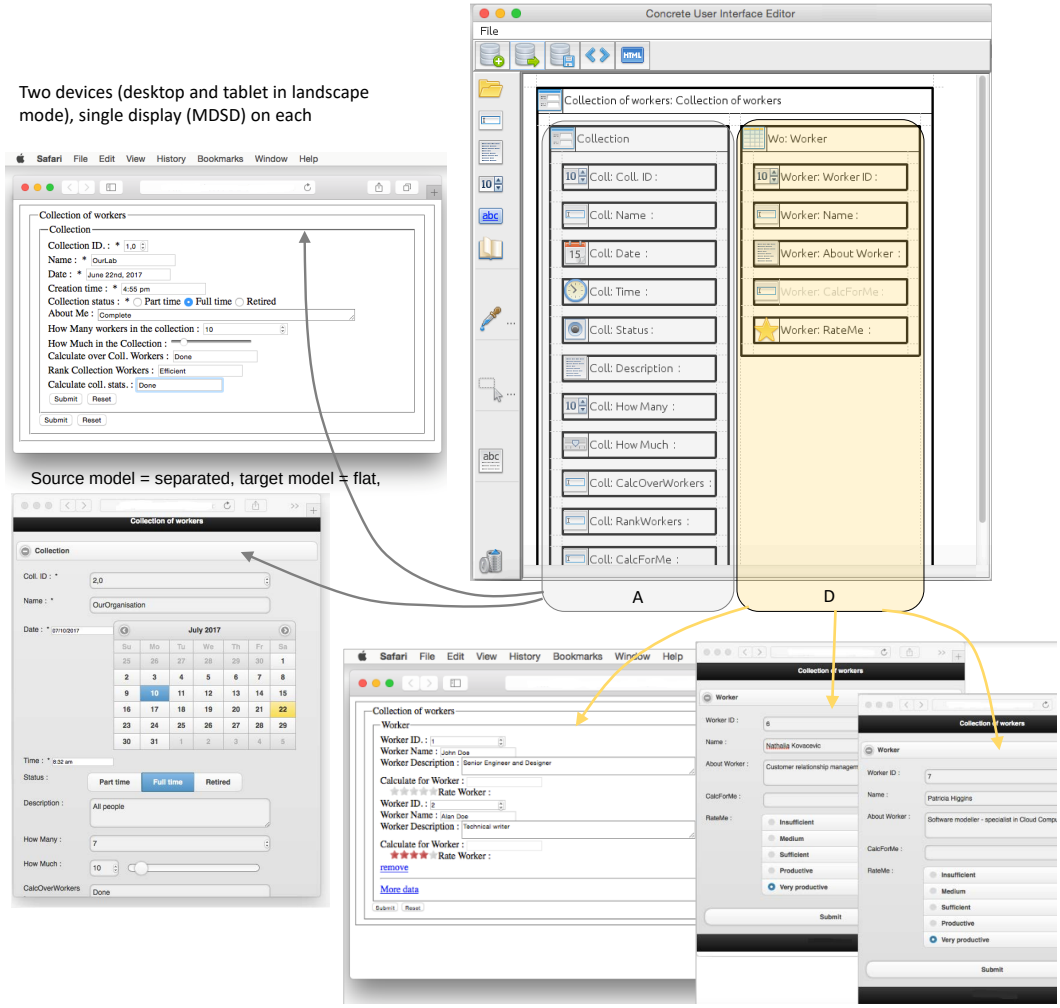

Fig. 17. XDUIs generated for 2 configurations: MDSD for smartphone+desktop and MDMD for smartphone+desktop.

extent would designers and developers with various levels of experience be able to produce a XDUI for a given class diagram based on the MoCaDix method introduced. For *efficiency*, how much time would they need. For *subjective satisfaction*, to what extent does this approach matches their expectations? These three factors are selected because they are part of the ISO 9241-11 definition of usability [1]: "the extent to which a product can be used by specified users to achieve specified goals with *effectiveness*, *efficiency* and *satisfaction* in a specified context of use". We are now describing this experiment according to the structure defined in Ko et al. [34].

4.1 Method

4.1.1 Recruitment. Participants were recruited from a mailing list of about 200 volunteers maintained at Anonymous. An email was sent to the mailing list asking for designers and/or developers willing to conduct an experiment on XDUI. No compensation was offered to volunteers. Any respondent was then submitted to a selection procedure.

4.1.2 Selection. In order to select participants with a representative coverage, the nine development expertise levels defined in [14] were considered (Fig. 18). On the left side of this continuum is Internet surfing, where no genuine development actually occurs. On the right side of this continuum is professional application development. Each level in between defines both the activities a designer or a developer is able to perform and corresponding development skills required to reach this level. The levels are defined as follows:

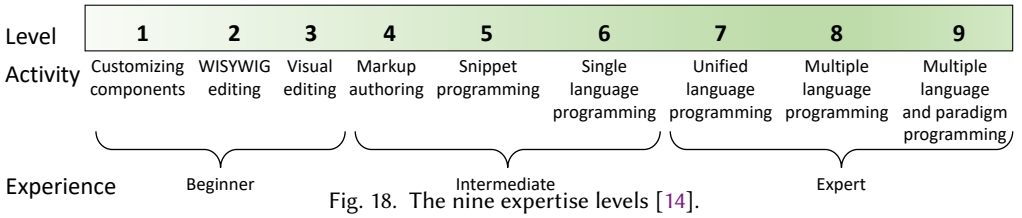

Fig. 18. The nine expertise levels [14].

(1) *Customizing components*: developers adjust existing UIs by setting values to parameters, such as customizing service preferences, adaptation, and display.
(2) *WYSIWYG editing*: developers edit the UI presentation in a WYSIWYG editor.
(3) *Visual editing*: developers create UIs visually by adding, customizing, and connecting components, such as in mashup editors and component-based editors. Visual editing is possible without having learned programming, but often requires a basic understanding of the programming paradigms (e.g., imperative, declarative, functional).
(4) *Markup authoring*: developers produce UIs using plain markup languages. This is a code editing with a simplified markup language, which involves some minimal understanding of declarative programming.
(5) *Snippet programming*: developers copy, paste, manually edit existing source code for templates called *snippets*. This is performed in a dedicated editor which automatically generates source code to be edited afterwards.
(6) *Single language programming*: developers have a sufficient knowledge of a single programming language and they are able to develop a piece of functionality from scratch.
(7) *Unified language programming*: developers have sufficient knowledge of a single programming language, which can be used to create all necessary components of an entire interactive application.
(8) *Multiple language programming*: developers have a thorough command of several programming languages and combine them. The difference with respect to the previous level is that developers need to master and integrate several languages together to produce an interactive application.
(9) *Multiple language and paradigm programming*: developers are professional programmers as they combine several different technologies to entirely build an interactive application. Programming paradigms can be declarative, imperative, or multi-paradigm (e.g., declarative for the interface and imperative for the controller).

Since MoCaDix targets the six last levels (Fig. 18), respondents from level 4 to level 9 were considered.

4.1.3 Group assignment. A stratified sampling was performed to randomly select one participant belonging to each level: *intermediate* (from level 4 to 6) and *expert* (from level 7 to 9). Randomly selected candidates were then contacted on a one-to-one basis to explain the study to be conducted and to check that their expertise matches the required level. The selection finished when one participant was found for each level.

4.1.4 Consent. Each participant performed the experiment in a controlled environment, which was our usability laboratory. Prior to the task, each participant was welcomed, had the process explained to them, signed a GDPR-compliant consent form, and filled in a questionnaire on their background.

4.1.5 Procedure. Accessing a large pool of publicly-available class diagrams remains hard to identify, to gather, and to control with respect to conceptual coverage, construct variety, complexity, and frequency of usage. So, instead of considering such a pool, we considered the Coad, North & Mayfield (CNM) collection of model patterns [12] based on the following rationale:

(1) *Conceptual coverage.* The CNM collection represents a structured library of five integrated families of model patterns that provide an extensive conceptual coverage in its definition:
   - A family of fundamental patterns: the Master-Detail is a classical design pattern [46] that is largely used and that leads to several instantiations, such as filter patterns, navigation patterns. JUST-UI [43] and OlivaNova [2] hols such instantiations: Master-Detail pattern, Population Interaction Unit, Instance Interaction Unit, and Service Interaction Unit.
   - A family of transaction patterns: they concern a bidirectional exchange of attributes and methods between two agents, such as Actor-Participant, Participant-Transaction, Place-Transaction, Specific Item-Transaction.
   - A family of part-of patterns: they concern methods applicable to a large group of objects based on individual methods executed on sub-groups of this large group, such as Container-Content, Container-Container Line Item, Group- Member, Assembly-Part. The Part-of concept itself gives rise to several particular relationships, such as aggregation, composition, dependency, materialization and metaclass [20].
   - A family of plan patterns: they concern the planning of activities in time and space, such as Plan-Step, Plan-Plan execution, Step-Step execution.
   - A family of interaction patterns: they concern any type of interaction between two agents, such as Peer-to-Peer, Proxy-Specific Item, Publisher-Subscriber.

(2) *Construct variety.* The CNM collection displays a wide variety of classes, attributes (e.g., with sufficiently different data types with different cardinalities), methods (e.g., returning different data types such as integer, real), and relationships (e.g., different types such as association, composition, aggregation, and inheritance with various cardinalities).

(3) *Complexity.* The CNM collection also exhibits a varying level of complexity in the relationships, the most complex association being a fourth-level descending relationship from a top class.

(4) *Frequency of usage.* The CNM collection becomes one of the most frequently and largely used collection of patterns [51], namely because they are independent of any structural model. Five information systems [11] were built using the 31 design patterns (fundamental patterns) contained in the collection and 148 strategies, which can be viewed as plans of actions on how to build models. They present guidelines on how to identify system purpose and features (12 strategy patterns), selecting classes (37 strategy patterns), establishing responsibilities (76 strategy patterns), working out dynamics with scenarios (14 strategy patterns).

In order to preserve the aforementioned CNM quality properties without affecting the duration and the complexity of the setup, a subset of the 31 design patterns is considered. Pattern #1 of the fundamental pattern category describes a collection of workers (Fig. 5) which serves as a starting point for the next twelve patterns consolidated into a single model. Each pattern is a list with detail elements and each detail element becomes a master element from another transaction pattern in relationship with a detail class. For instance, if we want to see more details about an actor (master class), we select this one and we can see its relation with an associated participant

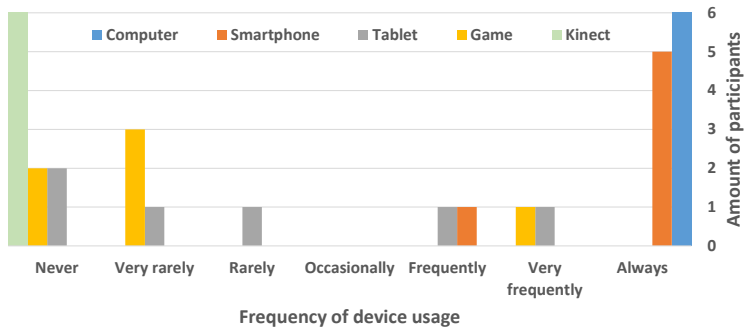

Fig. 19. Frequency of usage for the participants by device: computer, smartphone, talet, game console, and Kinect-like device.

(details) implied in a specific transaction. And if we want to know all transactions performed by this participant (master), we select this one and show more details related to transactions, which mimic the Overview+Details metaphor [13]. A UML class diagram is therefore created with the first fundamental pattern and the 12 subsequent transaction patterns: Actor-Participant, Participant-Transaction, Place-Transaction, Specific Item-Transaction, Transaction-Transaction Line Item, Transaction-Subsequent Transaction, Transaction Line Item-Subsequent Transaction Line Item, Item-Line Item, Specific Item-Line Item, Item-Specific Item, Associate-Other Associate, and Specific Item-Hierarchical Item (Fig. 21).

4.1.6 Demographic measurements. Through the questionnaire, the participants were asked to provide some personal information for statistics such as age at the time of the experiment, gender, profession, domain of activity, their background. Six participants (2 female, 4 male) conducted the experiment from three different countries speaking two different languages (i.e., English, French). Their age varied from 29 to 52 ($M = 34.17$, $SD = 10.46$). Five people were employees while one was working as an independent consultant. Each participant was also asked to self-report her familiarity with five devices (1=minimum, 7=maximum): computer, smartphone, tablet, game console, Kinect-like device. All participants revealed a frequency of device usage (Fig. 19) matching the desired profile: they are all sufficiently frequent computer users, including for other devices such as respectively smartphone, tablet, game console, and Kinect-like device. Participants also reported the amount of years of experience in modelling since they started developing. The distribution is as follows: L4=1/7 (this participant benefits from one year of experience in modelling over 7 years of development), L5=2/2, L6=3/3, L7=6/6, L8=14/14, and L9=30/35. Most of them started using modeling at the same time that started developing, but not all of them.

4.1.7 Group assignment. Due to the aforementioned selection, only one participant was assigned to each group representing one level of experience.

4.1.8 Training. After the questionnaire was completed, the experimenter demonstrated on screen a simple scenario executed according to the MoCaDix method. A familiarization period was then included: a 10-min. tutorial was delivered as a demonstration of the MoCaDix approach and its capabilities, followed by another 10-min period for freely playing with the software. Participants could finish this last part early if they wanted.

4.1.9 Task. Designers and developers were instructed to enter the resulting UML class diagram in the model editor (Step 1) and to produce a XDUI for both mobile and desktop devices by adopting a "one pattern at a time" approach (Step 2 to 4), then by integrating them all into a single model and XDUI. The task comprised three steps:

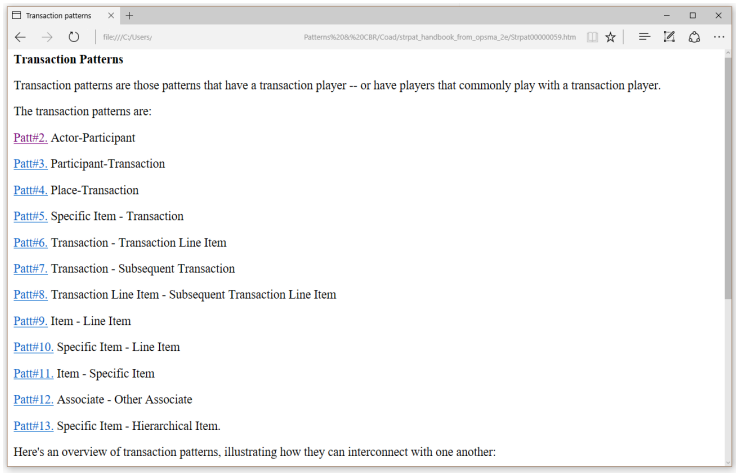

Fig. 20. On-line hypermedia with list of patterns used in the experiment.

(1) *A repeated test for each pattern*: the participant receives access to an on-line hypermedia (Fig. 20) documenting one model pattern at a time and is instructed to proceed as follows without any pressure or time limit, but trying to mimic real-world project conditions:
   - Select the link for each pattern for starting the test to discover the domain pattern and understand it.
   - Create the corresponding model in the editor with the following constraints: all attributes and methods should be declared by their data type and length, a short name and a long name should be provided for each entry, all relationships should be entered and the model should be saved. An internal small test checks whether these specifications are indeed complete and displays a ”bug” icon when some data are missing. The participant is allowed to reuse any previously defined model construct, such as a class, and change it on demand.
   - Generate a XDUI with the following parameters: devices = `SmartphoneAndDesktop`, source class presentation = `CombinedAndFlat`, target class presentation = `CombinedAnd1..m`.
   - Strive for consistency between patterns as much as possible but without any checking in order not to break the procedure.
   - Test the results and save them into a QuiXML file. Since each pattern holds two classes, the participant is asked to save the results into two steps: a first save when the first class is finished (which then records an intermediate completion time) and a final save when the second class was finished (which then records the final task completion time).

(2) *A single integrated test*: the participant is asked to create a brand new project from scratch, to adopt the same procedure as for the repeated test. No intermediary time was recorded, only the task completion time and the task completion rate. The participant is allowed to reuse any previously defined construct from any previously created pattern by copy and paste. No quality check is performed to ensure the fluidity since the design options are predefined and to keep the control of the experiment.

(3) *A IBM CSUQ form filling*: this Computer Satisfaction Usability Questionnaire [37] enables participants to express their level of satisfaction with the usability of the setup and the testing process. This 19-question questionnaire has been empirically validated with a large number of participants on a significant set of stimuli [39], it is widely applicable for any interactive system [38], and it benefits from a proved $\alpha = 0.89$ reliability coefficient between its results and the perceived system usability [37].

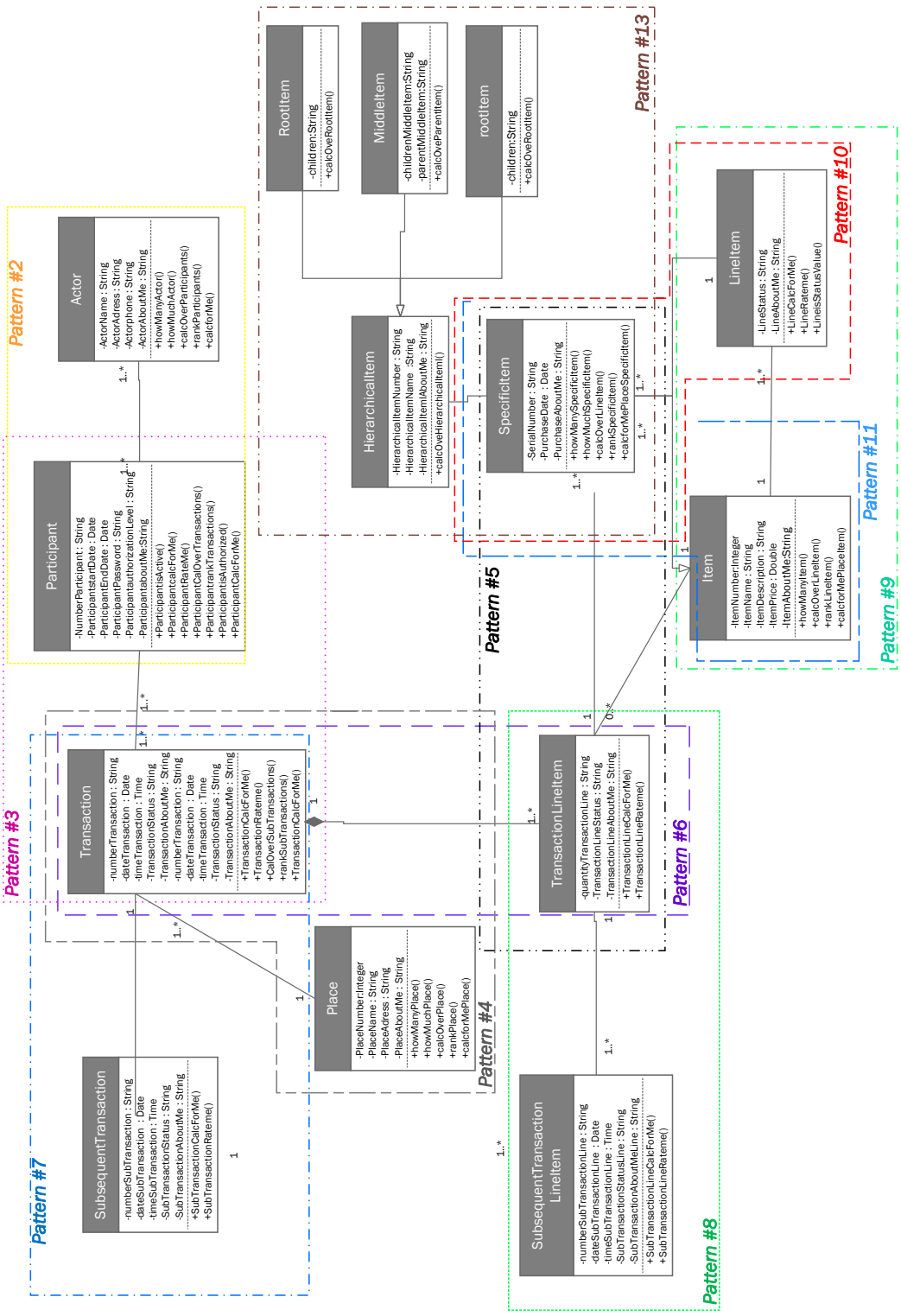

Fig. 21. Coad, North & Mayfield 13 model patterns [12].

Pattern	Markup (L4)	Snippet (L5)	One PL (L6)	Unified PL (L7)	Many PL (L8)	Many PL+P (L9)	*M*	*SD*
P2	1139	937	897	805	720	953	909	131
P3	637	554	541	630	660	531	592	51
P4	310	224	289	355	420	251	308	65
P5	327	233	293	289	360	267	295	41
P6	668	594	475	422	120	548	471	176
P7	334	336	367	302	240	278	310	42
P8	457	456	411	429	240	356	392	76
P9	440	478	398	374	230	362	380	78
P10	484	459	360	301	180	395	363	102
P11	489	375	264	227	240	408	334	97
P12	497	326	293	271	200	397	331	95
P13	333	367	355	329	480	278	357	62
Total	6115	5339	4943	4734	4090	5024		
M	510	445	412	395	341	419		
SD	220	184	166	159	185	186		

Table 4. Thinking times for all 13 patterns applied by the six participants (*M* =average, *SD* =standard deviation).

Pattern	Markup (L4)	Snippet (L5)	One PL (L6)	Unified PL (L7)	Many PL (L8)	Many PL+P. (L9)	*M*	*SD*
P2	145	137	109	92	64	82	105	29
P3	112	101	97	82	52	74	86	20
P4	94	93	77	55	34	60	69	21
P5	45	50	53	51	56	47	50	4
P6	42	57	55	39	33	30	43	10
P7	44	49	39	47	44	35	43	5
P8	35	33	30	36	34	31	33	2
P9	51	52	48	38	66	35	48	10
P10	52	48	43	35	20	50	41	11
P11	29	22	25	29	25	62	32	14
P12	37	25	22	26	32	44	31	8
P13	45	39	21	31	40	23	33	9
Total	731	706	619	561	500	573		
M	61	59	52	47	42	48		
SD	35	33	28	20	14	18		

Table 5. Modelling Times for all 13 patterns applied by the six participants (*M* =average, *SD* =standard deviation).

4.1.10 Outcome measurements. Each session is observed by a researcher equipped with a chronometer to note the THINKING TIME (i.e., time elapsed from selecting the link in the hypermedia to start using the software) and MODELLING TIME (i.e., time elapsed from the beginning to the end of using the software). All times are captured in minutes and seconds and subsequently converted into seconds only. Individual times produced by participants are averaged by pattern and by level. The task completion rate is measured in percentage (%) with respect to the goal of creating a XDUI for all patterns. Each IBM CSUQ closed question is measured using a 7-point Likert scale [40] (1=strongly disagree, 2=largely disagree, 3=disagree, 4=neutral, 5=agree, 6=largely agree, 7=strongly agree) and the following measures are computed: system usefulness (SysUse: Items 1-8), quality of the information (InfoQual: Items 9-15), quality of the interaction (InterQual: Items 16-18), and system quality (Overall: Item 19). The answers provided by participants are entered into a MS Excel Spreadsheet.

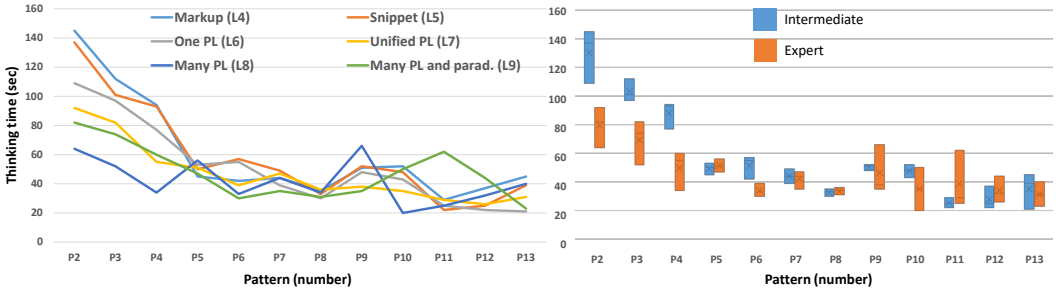

Fig. 22. Thinking time per pattern from P2 to P13 (in sec.): time per level (left) and distribution by experience (right).

4.2 Results and Discussion

4.2.1 Task Completion Time. The integrated class diagram resulting from the consolidation consists of 14 classes with a grand total of 54 attributes and 46 methods, 2 compositions, 10 associations, and 3 inheritance relationships to declare (Fig. 21). Fig. 22, respectively Fig. 23, graphically depict the thinking time, respectively the modelling time aggregated for all participants per pattern, corresponding to data in Table 4, respectively to Table 5. Left part of Fig. 22 depicts the time curve per participant while the right part depicts the distribution of this time for all participants on the same pattern. Pattern #2 is the first pattern considered in the experiment since the first one is already included. This pattern requires the largest amount of time, both thinking and modelling, since each participant discovers how to manage a model in the editor (Fig. 3). This time is still important even after a period of familiarisation. For Pattern #3, participants made little reuse of previously modelled construct, even if the Participant class was almost the same. It turns out that participants figure out how to reuse previously defined model constructs at Pattern #4: while the source class is new, the second is merely copied and pasted from the previous pattern, thus explaining why only less time is needed. The effect is the same for Pattern #5. Pattern #6 has different variations inducing changes and incompleteness, thus inviting the participant to review each item individually. Participants then devote more time than for a simple copy-paste, but not much more. The time then decreases again in Pattern #7, probably due to some learning effect. From now on until Pattern #12, the results suggest that the thinking time remains stable (between 20 and 40 sec.) as well as the modelling time (between 300 and 500 sec.) after some initial patterns. Fig. 24 graphically depicts the total thinking time (left) and the total modelling time (right) per participant for all patterns cumulated.

Regarding the **thinking time**, Table 4 details the thinking time for all thirteen patterns required by all participants, along with their average (M) and standard deviation (SD). we first computed Levene's test [36] to determine whether the six levels in Fig. 24 have significantly different population variances. Since the various p-values are all above $\alpha = .05$ (by means = .0976, by medians = .6736, by trimmed mean = .0976), we cannot reject the null hypothesis and conclude there is no significant difference between the 6 group means. So, the ANOVA test satisfies the homogenity of variances assumption. This is also confirmed by a Welch's test (F-stat=.92, df1= 5, df2= 30.30): $p = .47 > \alpha = .05$. Given this assumption, we computed a series of Student's t-tests with two paired samples (since each pair member considered the same pattern) represented by two consecutive levels (i.e., L4 with respect to L5, L5 with L6, and so forth) to detect whether any significant difference exists between the various levels. Only the Snippet (L5)-One PL(L6) appears as significant, all the others being not significant. There was a highly significant difference in the thinking times for Snippet (L5) (M=58.83, SD=34.32) and One PL(L6) (M=51.58, SD=28.99) conditions; $df = 11$, t= 2.73,

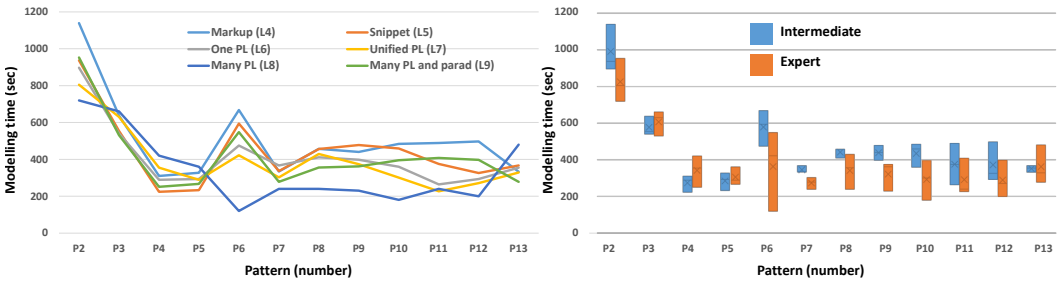

Fig. 23. Modelling time per pattern from P2 to P13 (in sec.): value per level (left) and distribution by experience (right).

p^{**}<.01, Pearson's ρ=.63, Cohen's d = .79. These results suggest that having some experience in at least one programming language has a positive effect on thinking time. For other consecutive levels, there is no significant difference, thus suggesting that thinking about the problem of turning the pattern into a model does not really depends on the level of experience. We tested all combinations between all levels, the significant ones being represented in Fig. 24.

There was a significant difference in the thinking times for Markup (L4) (M=60.92, SD=36.13) and One PL(L6) (M=51, 58, SD=28, 99) conditions; df= 11, t= 2.43, $p^{*} \leq .05$, Pearson's ρ=.59, Cohen's d = .70. These results confirm that having some experience in at least one programming language has a positive effect on thinking time since having experience using markup language or snippet represents more declarative activities than programming. The L4 is outperformed by all other levels: by the L7 (M=46.75, SD=20.80) condition; df=11, t=2.69, $p^{*} \leq .05$, Pearson's ρ=.63, Cohen's d = .78; by L8 ($p^{*} \leq .05$) and L9 ($p^{*} \leq .05$).

There was also a significant difference in the thinking times for Snippet (L5) and Unified Programming (L7) conditions; df=11, t=2.58, $p^{*} \leq .05$, Pearson's ρ=.61, Cohen's d = .74. There was a significant difference in the L5 and L8 conditions; df=11, t=2.04, $p^{*} \leq .05$, but with a smaller magnitude though (Pearson's ρ=.52, Cohen's d = .52). Finally, since L4 to L6 cover the intermediate level and L7 to L9 cover the expert level according to the scale (Fig. 18), we also grouped all results for these two groups (represented in blue and orange respectively in Fig. 24). We performed a last t-Test with two paired samples to find out a very highly significant difference between the intermediate (M = 57.11,SD = 32.58) and expert (M = 45.39, SD = 17.91) conditions; df= 35, t=3.07, $p^{***} \leq .001$, Pearson's ρ=.46, Cohen's d = .51).

Regarding the **modelling time** (Fig. 23), similar tests were conducted and are summarized as follows. Levene's test gave a positive result of homogeneity of variance through all measures: by means (.9543), by medians (.9808), and by trimmed means (.9543). There was a highly significant difference in the modelling times for Markup (L4) (M=509.53, SD=229.93) and Snippet (L5) (M=444.92, SD=192.68) conditions; df= 11, t= 2.92, p^{**}<.01, Pearson's ρ=.66, Cohen's d = .84. Same for Snippet (L5) and One PL (L6) (M=411.91, SD = 173.03) conditions; df=11, t=1.82, p^{**}<.01, Pearson's ρ=.48, Cohen's d = .52. Having some better experience in programming languages over snippet and over markup languages does have a positive effect on modelling time. After these levels, participants seem to spend the same amount of time for all patterns in MoCaDix. This is also confirmed by the t-Test between the intermediate and expert groups in modelling: $p^{**} \leq .01$, Pearson's ρ=.43, Cohen's d = .47). So, the modelling time progressively decreases from one level of experience to a subsequent one to almost asymptotically converge towards the average time of the last level. Both intermediate and expert participants were efficient enough, but experts converge more rapidly towards a constant time per pattern. A carry-over effect from one pattern to another improves the overall efficiency.

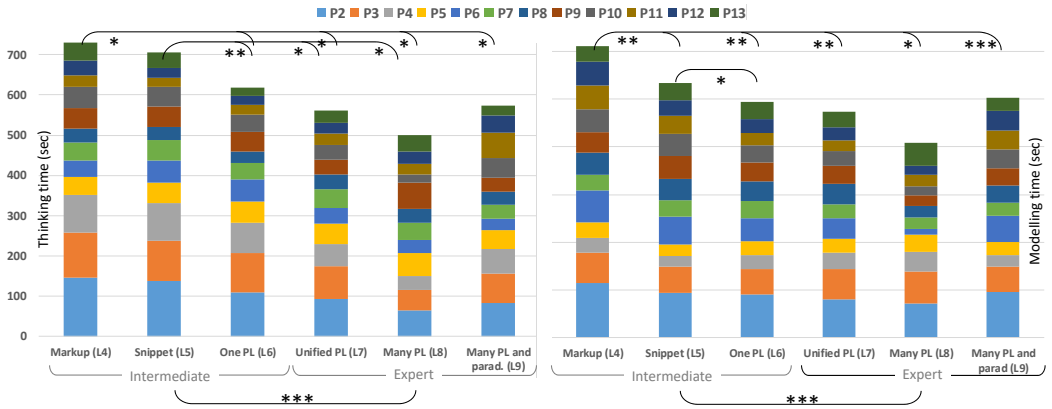

Fig. 24. Thinking and Modelling times for all patterns cumulated (in sec.). No significance: *n.s.*=$p > .05$, $p^{*} \leq .05$, $p^{**} \leq .01$, $= p^{***} \leq .001$.

4.2.2 Task Completion Rate. All participants succeeded to complete all repeated tasks and the integrated one. So, the task completion rate is effective enough to justify its adoption from this viewpoint. As an example, Fig. 31 reproduces some screen shots of the XDUI created by the L9-participant for desktop and smartphone.

4.2.3 Subjective Satisfaction. Fig. 25 shows the distribution of participants' answers to the 19 open questions of the IBM CSUQ questionnaire per question, ranging from Q1 to Q19. The average values of these questions are then aggregated into the four CSUQ metrics, which are computed in the right part: SysUse ($M = 4.73$, $SD = 1.24$), InfoQual ($M = 4.27$, $SD = 1.31$), InterQual ($M = 4.50$, $SD = 1.12$), and Overall ($M = 4.5$, $SD = .96$), all computed with a 95% interval of confidence ($\alpha = .05$). All values are beyond the threshold of 4 (which represents a good value), but below 5 (which represents an excellent value in average). Apart from the L8-participant, the more advanced the level of expertise of the participant is, the lower score they give (a linear regression suggests that the overall score is decreasing with this level: Fig. 26). Overall, it seems that advanced participants (L7-L9) are more demanding in terms of functions and control than intermediate participants (L4-L6), who seem to be relatively satisfied by the effectiveness and efficiency offered by the software.

Fig. 25. Distribution of IBM CSUQ answers and related metrics.

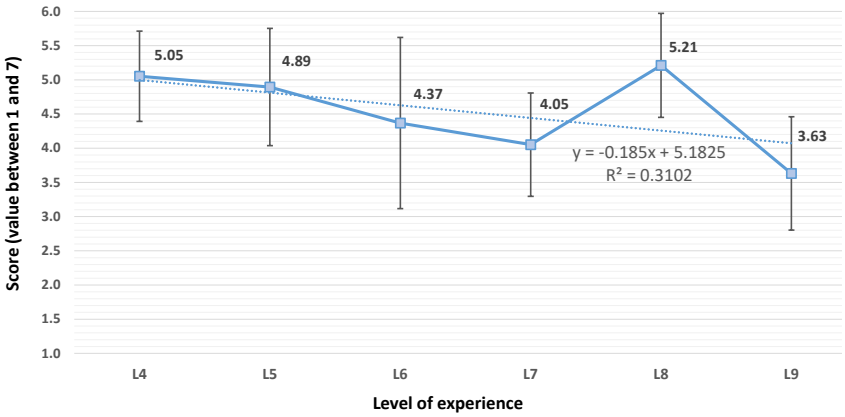

Fig. 26. Global scores for the IBM CSUQ by participant.

5 SECOND EXPERIMENT

This section aims at conducting a preference analysis of results generated by MocaDix's approach by end users with the research question: which design option do you prefer for each XDUI generated for each of the three devices? Fig. 15 synthesized the theoretical suitability of design options for four configurations involving the supported devices. In contrast, this experiment investigates how end users express their preference for some design option. We performed an exploratory user study to evaluate end users' reactions with respect to the various XDUIs generated based on the design options. We were interested in what people would think of the different options governing the different layouts and which they would prefer. However, the multiplicity of design options and their parameters combined all together would produce a design space that is too large to systematically investigate. Therefore, we concentrated the study on the following parameters, which seek to represent the most significant changes on XDUIs: source class presentation = `CombinedAndFlat`, target class presentation = `CombinedAndNested` (none, 1..1, 1..m, both, all) for three devices: smartphone, tablet, and desktop. Since this value range covers multiple presentations (Figs. 8 and 9), it was decided to cover them all for a simple case: only 4 CNM classes from Fig. 21 were kept with `Actor`, `Participant`, `Transaction`, and `SubsequentTransaction`. When no nesting was selected, target classes were grouped into a group box included in the source class space. When `All nesting` was selected, target classes were synthesized into a series of tabs in a dialog box included in the source class space. We therefore built a data set of 3 devices x 6 options = 18 samples.

5.1 Method

Each participant performed the task in a controlled environment. Prior to the task, each participant was welcomed, had the process explained to them (including signing a consent form) and filled in a short questionnaire on their background and to check whether they had prior experience in using information system on the three platforms. After the questionnaire was completed, the researcher demonstrated the 18 samples printed for one device at a time. The ordering of the devices and the samples by device were randomly selected by a pseudo-random generator. The participants were then given 5 minutes to familiarize themselves with the software and ask any questions. If desired, the participant could finish this part early. The participants were then given 5 minutes per device to give two dependent variables: (i) their overall subjective satisfaction using a five-point Likert scale [40] (1= strongly unsatisfied to 5 = strongly satisfied) for each sample, (ii) their ranking of the samples in order from most preferred to least preferred.

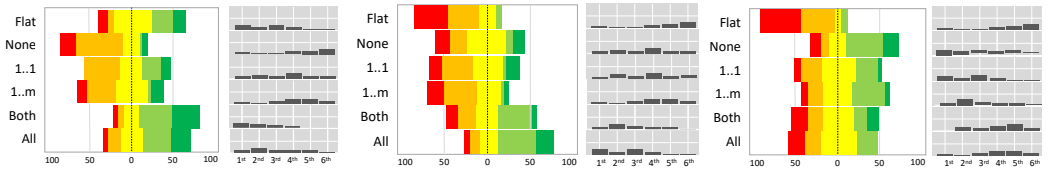

Fig. 27. Overall satisfaction rating and preference ranking: (a) for smartphone, (b) for tablet, (c) for desktop.

5.2 Results and Discussion

A total of twenty (7 female, 13 male) participants ($M = 32$ years, $SD = 6$) participated in this experiment recruited from a end-user mailing list maintained at Anonymous. All participants reported themselves as regular computer and IS users. They were recruited in our organization through a mailing list. They have different backgrounds such as: accounting, finance, information systems, management, marketing, and human resources. All the participants in this study were volunteers and they were not given any remuneration (financial or otherwise) for their time. After each participant, the questionnaire, the subjective satisfaction and ranking data was added into an Excel Sheet. The data was entered in an anonymous format so the participants could not be identified.

Fig. 27 shows the correspondence between the distribution of subjective satisfaction in terms of percentage (represented as a divergent horizontal stacked bar) and the distribution of ranking (represented as vertical bars), respectively for smartphone (Fig. 27a), tablet (Fig. 27b), and desktop (Fig. 27c). Fig. 15 was supposed to represent the potential suitability of design options depending on the surface provided by each device, also respectively smartphone, tablet, and desktop. The results of this experiment suggest another interpretation: for the smartphone which is recognized as the most constrained device, the two first designs emerging from both measures are `Both` and `All`, mainly because tabs were used. Even on smartphones, participants opt for tabs delineating target classes: tabs properly divide content into meaningful sections which consume less screen real estate, they are visually connected with the content of a view, they were acceptable because up to 4 tabs were displayed at most (`All`) or 2 (`Both`). Tabs were preferred over sequence (`Flat` arrives in third position) and over group boxes (used for `None`, `1..1`, and `1..m`).

Although group boxes draw a bevelled line around a group of widgets for a particular class, inside another one, they create visually distinct groups. But they also increase the level of hierarchy with many embedded lines, which may explain why `None` option was the least desired. This is probably the same explanation for desktop, where the `None` option was almost the least preferred. Two embedded group boxes is already a maximum beyond which distinguishing between visual groups become harder to achieve. The tab based presentation was also the most satisfying for desktop even if screen space was not a concern. Then having target classes presented again as tabbed dialog boxes are almost equally rated for `1..1` and `1..m`, as well as between `Both` and `All`.

6 CONCLUSION AND FUTURE WORK

This paper presented MocaDix, a new cloud computing-based user interface development specifically tailored to facilitate the deployment of XDUIs of ISs according based on its UML class diagram. MocaDix supports designers and developers in applying a four-step method: (1) a class diagram of the information system is created, (2) patterns are selected and parameterized to decide design options regulating how the model will be presented, (3) a CUI model is generated from both the class diagram and the patterns that have been selected and parameterized and stored in QuiXML, a lightweight fit-to-purpose UIDL, and (4) cross-device user interfaces are automatically generated from the CUI Model for four ranges of configuration: single-device single-display, single-device

multi-display, multi-device-single display, and multi-device multi-display, where supported devices include a smartphone, a tablet, and a desktop, the three most frequently used devices. All XDUIs were generated in HTML5 with design options for each device, such as class presentation.

A first experiment demonstrated that a significant collection of model patterns can be effectively, efficiently, and positively covered by MocaDix. A second experiment compared the satisfaction and preferences of end users for various design options related two target devices. The cloud computing based availability of MocaDix enables the quick deployment of a service of a new kind: Cross-Device User Interface-as-a-Service (XaaS), where XDUIs are directly offered from a UML V2.5 class diagram without any extra effort apart from choosing design options. Since the generation process contains complex steps, pushing this generation process at run-time according to the principle of Models@run-time [7] would pose performance challenges, unless a caching mechanism is added. We finish this paper by discussing some potential avenues of this work with respect to the various main contributions brought by this paper in the four-step method.

Step 1. Mapping rules were defined for a UML V2.5 class diagram, but should be widely applicable in principle to previous versions of the class diagram metamodel and to any other structurally equivalent model, such as an object-oriented model or an entity-relationship-attribute (ERA) model, considered having comparable expressiveness. For any other structural model with less expressiveness, mapping rules will give rise to a sub-set of restricted rules. For the moment, all classical modelling constructs of the class diagram are taken into account, but new constructs and relationships could be added similarly, such as the materialization [20] and metonymic relationships, the difference between a meta-class and a simple class, or new aggregated constructs.

Step 2. Mapping rules defined for attributes, methods, and relationships are always subject to improvement, but at the risk of complexification: the more mapping rules are added, the more complex they become to manage and they reach a 'plateau' effect when adding another mapping rule no longer significantly improves the usability of the resulting UI. Mapping rules for selecting widgets can be certainly improved by mapping fine tunes data types onto richer widgets (e.g., carousel, alpha slider, toggle button, time lines, tables), but choosing between alternate widgets then becomes harder and the usability of some widgets is debatable.

Step 3. For the moment, the generated CUI model is editable within the CUI editor, thus potentially destroying the consistency between the initial domain model and the generated CUI model. Re-establishing this consistency after any manual modification involves round-trip engineering, which is a hard to solve. However, manual operations can be saved in a configuration file so as to apply them again on-demand, with the risk that some operations are irrelevant when they operate on inexistent widgets. This is the problem of *UI beautification* [52].

Step 4. Expanding presentation styles is certainly desirable. For instance, considering Google Material[5] should improve the rendering of two options, i.e. source and target class presentations, by emphasising or de-emphasising sections in the structure by applying lighting and shadows or other effects such as grid-based layouts, animated transitions, or padding. Since XDUIs are resulting from the MocaDix's approach, it would be very interesting to conduct a usability evaluation of the XDUIs produced. Abrahao et al. [2], as well as Aquino et al. [4], already conducted such a comparative analysis for multi-device UIs generated by OlivaNova, respectively JustUI [4], according to Model-Driven Engineering [51] showing the interest not only for individual UIs generated, but also to compare the respective UIs generated for the three platforms considered. Conducting the same method would be certainly interesting but a new method for XDUI evaluation would be even more interesting, in particular to compare related effects, such as consistency, switching cost between devices, device paths, and transition.

[5] http:www.material.io

REFERENCES

[1] 2018. *Ergonomics of human-system interaction – Part 11: Usability: Definitions and concepts, ISO/TC 159/SC 4.* https://www.iso.org/standard/63500.html

[2] Silvia Abrahão, Emilio Iborra, and Jean Vanderdonckt. 2008. *Usability Evaluation of User Interfaces Generated with a Model-Driven Architecture Tool.* Springer London, London, 3–32. https://doi.org/10.1007/978-1-84628-941-5_1

[3] Mir Farooq Ali, Manuel A. Pérez-Quiñones, Marc Abrams, and Eric Shell. 2002. Building Multi-Platform User Interfaces with UIML. In *Computer-Aided Design of User Interfaces III, Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, May, 15-17, 2002, Valenciennes, France*, Christophe Kolski and Jean Vanderdonckt (Eds.). Kluwer, 255–266.

[4] Nathalie Aquino, Jean Vanderdonckt, Nelly Condori-Fernández, Óscar Dieste, and Óscar Pastor. 2010. Usability Evaluation of Multi-device/Platform User Interfaces Generated by Model-driven Engineering. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '10).* ACM, New York, NY, USA, Article 30, 10 pages. https://doi.org/10.1145/1852786.1852826

[5] Vincent Balat. 2013. Client-server web applications widgets. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013, Companion Volume*, Leslie Carr, Alberto H. F. Laender, Bernadette Farias Lóscio, Irwin King, Marcus Fontoura, Denny Vrandecic, Lora Aroyo, José Palazzo M. de Oliveira, Fernanda Lima, and Erik Wilde (Eds.). International World Wide Web Conferences Steering Committee / ACM, 19–22. https://doi.org/10.1145/2487788.2487795

[6] Helmut Balzert. 1995. From OOA to GUIs - the JANUS System. In *Human-Computer Interaction, INTERACT '95, IFIP TC13 Interantional Conference on Human-Computer Interaction, 27-29 June 1995, Lillehammer, Norway (IFIP Conference Proceedings)*, Knut Nordby, Per H. Helmersen, David J. Gilmore, and Svein A. Arnesen (Eds.). Chapman & Hall, 319–324.

[7] Gordon Blair, Nelly Bencomo, and Robert B. France. 2009. Models@ Run.Time. *Computer* 42, 10 (Oct. 2009), 22–27. https://doi.org/10.1109/MC.2009.326

[8] Ashley Bush and Sandeep Purao. 2011. Mapping UML Techniques to Design Activities. In *Information Modeling in the New Millennium*, Matti Rossi and Keng Siau (Eds.). IDEA Publishing Group, Chapter 12, 199–217.

[9] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. 2003. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers* 15, 3 (2003), 289–308. https://doi.org/10.1016/S0953-5438(03)00010-9

[10] Adam Chlipala. 2016. Ur/Web: A Simple Model for Programming the Web. *Commun. ACM* 59, 8 (July 2016), 93–100. https://doi.org/10.1145/2958736

[11] Peter Coad. 1992. Object-oriented Patterns. *Commun. ACM* 35, 9 (Sept. 1992), 152–159. https://doi.org/10.1145/130994.131006

[12] Peter Coad, David North, , and Mark Mayfield. 1997. *Object Models: Strategies, Patterns, and Applications. 2nd Edition.* Prentice Hall, Upper Saddle River, New Jersey.

[13] Andy Cockburn, Amy Karlson, and Benjamin B. Bederson. 2009. A Review of Overview+Detail, Zooming, and Focus+Context Interfaces. *ACM Comput. Surv.* 41, 1, Article 2 (Jan. 2009), 31 pages. https://doi.org/10.1145/1456650.1456652

[14] Maria Francesca Costabile, Piero Mussio, Loredana Parasiliti Provenza, and Antonio Piccinno. 2008. End Users As Unwitting Software Developers. In *Proceedings of the 4th International Workshop on End-user Software Engineering (WEUSE '08).* ACM, New York, NY, USA, 6–10. https://doi.org/10.1145/1370847.1370849

[15] António Miguel Rosado da Cruz. 2014. Use Case and User Interface Patterns for Data Oriented Applications. In *Model-Driven Engineering and Software Development - Second International Conference, MODELSWARD 2014, Lisbon, Portugal, January 7-9, 2014, Revised Selected Papers (Communications in Computer and Information Science)*, Slimane Hammoudi, Luís Ferreira Pires, Joaquim Filipe, and Rui César das Neves (Eds.), Vol. 506. Springer, 117–133. https://doi.org/10.1007/978-3-319-25156-1_8

[16] António Miguel Rosado da Cruz and João Pascoal Faria. 2007. Automatic Generation of User Interfaces from Domain and Use Case Models. In *Quality of Information and Communications Technology, 6th International Conference on the Quality of Information and Communications Technology, QUATIC 2007, Lisbon, Portugal, September 12-14, 2007, Proceedings*, Ricardo Jorge Machado, Fernando Brito e Abreu, and Paulo Rupino da Cunha (Eds.). IEEE Computer Society, 208–212. https://doi.org/10.1109/QUATIC.2007.19

[17] António Miguel Rosado da Cruz and João Pascoal Faria. 2009. Automatic Generation of user Interface Models and Prototypes from Domain and Use Case Models. In *ICSOFT 2009 - Proceedings of the 4th International Conference on Software and Data Technologies, Volume 1, Sofia, Bulgaria, July 26-29, 2009*, Boris Shishkov, José Cordeiro, and Alpesh Ranchordas (Eds.). INSTICC Press, 169–176.

[18] António Miguel Rosado da Cruz and João Pascoal Faria. 2010. A Metamodel-Based Approach for Automatic User Interface Generation. In *Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I (Lecture Notes in Computer Science)*, Dorina C. Petriu, Nicolas

Rouquette, and Øystein Haugen (Eds.), Vol. 6394. Springer, 256–270. https://doi.org/10.1007/978-3-642-16145-2_18

[19] Paulo Pinheiro da Silva. 2000. User Interface Declarative Models and Development Environments: A Survey. In *Interactive Systems: Design, Specification, and Verification, 7th International Workshop DSV-IS, Limerick, Ireland, June 5-6, 2000, Proceedings (Lecture Notes in Computer Science)*, Philippe A. Palanque and Fabio Paternò (Eds.), Vol. 1946. Springer, 207–226. https://doi.org/10.1007/3-540-44675-3_13

[20] Mohamed Dahchour, Alain Pirotte, and Esteban Zimányi. 2002. Materialization and Its Metaclass Implementation. *IEEE Trans. Knowl. Data Eng.* 14, 5 (2002), 1078–1094. https://doi.org/10.1109/TKDE.2002.1033775

[21] Anke Dittmar, Mathias Kühn, and Peter Forbrig. 2014. A Domain-specific Model-based Design Approach for End-user Developers. In *Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '14)*. ACM, New York, NY, USA, 161–166. https://doi.org/10.1145/2607023.2610275

[22] Brian Dobing and Jeffrey Parsons. 2008. Dimensions of UML Diagram Use: A Survey of Practitioners. *J. Database Manag.* 19, 1 (2008), 1–18. https://doi.org/10.4018/jdm.2008010101

[23] Tao Dong, Elizabeth F. Churchill, and Jeffrey Nichols. 2016. Understanding the Challenges of Designing and Developing Multi-Device Experiences. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems (DIS '16)*. ACM, New York, NY, USA, 62–72. https://doi.org/10.1145/2901790.2901851

[24] Mohammed Elkoutbi, Ismaïl Khriss, and Rudolf K. Keller. 2006. Automated Prototyping of User Interfaces Based on UML Scenarios. *Automated Software Engineering* 13, 1 (01 Jan 2006), 5–40. https://doi.org/10.1007/s10515-006-5465-5

[25] Henrik Eriksson, Angel R. Puerta, and Mark A. Musen. 1994. Generation of knowledge-acquisition tools from domain ontologies. *Int. J. Hum.-Comput. Stud.* 41, 3 (1994), 425–453. https://doi.org/10.1006/ijhc.1994.1067

[26] Luca Frosini and Fabio Paternò. 2014. User Interface Distribution in Multi-device and Multi-user Environments with Dynamically Migrating Engines. In *Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '14)*. ACM, New York, NY, USA, 55–64. https://doi.org/10.1145/2607023.2607032

[27] Josefina Guerrero García, Juan Manuel González-Calleros, Jean Vanderdonckt, and Jaime Muñoz Arteaga. 2009. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *2009 Latin American Web Congress, Joint LA-WEB/CLIHC Conference, Merida, Yucatan, Mexico, 9-11 November 2009*, Edgar Chávez, Elizabeth Furtado, and Alberto L. Morán (Eds.). IEEE Computer Society, 36–43. https://doi.org/10.1109/LA-WEB.2009.40

[28] Giuseppe Ghiani, Marco Manca, and Fabio Paternò. 2015. Authoring Context-dependent Cross-device User Interfaces Based on Trigger/Action Rules. In *Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia (MUM '15)*. ACM, New York, NY, USA, 313–322. https://doi.org/10.1145/2836041.2836073

[29] Christian Gram and Gilbert Cockton. 1996. *Design Principles for Interactive Software*. Chapman Hall, London.

[30] Peter Hamilton and Daniel J. Wigdor. 2014. Conductor: Enabling and Understanding Cross-device Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 2773–2782. https://doi.org/10.1145/2556288.2557170

[31] Steven Houben, Nicolai Marquardt, Jo Vermeulen, Clemens Klokmose, Johannes Schöning, Harald Reiterer, and Christian Holz. 2017. Opportunities and Challenges for Cross-device Interactions in the Wild. *interactions* 24, 5 (Aug. 2017), 58–63. https://doi.org/10.1145/3121348

[32] Christian Janssen, Anette Weisbecker, and Jürgen Ziegler. 1993. Generating User Interfaces from Data Models and Dialogue Net Specifications. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)*. ACM, New York, NY, USA, 418–423. https://doi.org/10.1145/169059.169335

[33] Tero Jokela, Jarno Ojala, and Thomas Olsson. 2015. A Diary Study on Combining Multiple Information Devices in Everyday Activities and Tasks. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 3903–3912. https://doi.org/10.1145/2702123.2702211

[34] Andrew J. Ko, Thomas D. LaToza, and Margaret M. Burnett. 2015. A practical guide to controlled experiments of software engineering tools with human participants. *Empirical Software Engineering* 20, 1 (01 Feb 2015), 110–141. https://doi.org/10.1007/s10664-013-9279-3

[35] Philip Langer, Tanja Mayerhofer, Manuel Wimmer, and Gerti Kappel. 2014. On the Usage of UML: Initial Results of Analyzing Open UML Models. In *Modellierung 2014, 19.-21. März 2014, Wien, Österreich (LNI)*, Hans-Georg Fill, Dimitris Karagiannis, and Ulrich Reimer (Eds.), Vol. 225. GI, 289–304. http://subs.emis.de/LNI/Proceedings/Proceedings225/article21.html

[36] Howard Levene. 1960. Robust tests for equality of variances. In *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*, Ingram Olkin and Harold Hotelling et al. (Eds.). Stanford University Press, Palo Alto, CA, USA, 278–292.

[37] James R. Lewis. 1995. IBM computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use. *International Journal of HumanâĂŞComputer Interaction* 7, 1 (1995), 57–78. https://doi.org/10.1080/10447319509526110 arXiv:https://doi.org/10.1080/10447319509526110

[38] James R. Lewis. 2002. Psychometric Evaluation of the PSSUQ Using Data from Five Years of Usability Studies. *International Journal of HumanâĂŞComputer Interaction* 14, 3-4 (2002), 463–488. https://doi.org/10.1080/10447318.2002.

9669130 arXiv:https://doi.org/10.1080/10447318.2002.9669130
[39] James R. Lewis. 2006. Sample Sizes for Usability Tests: Mostly Math, Not Magic. *interactions* 13, 6 (Nov. 2006), 29–33. https://doi.org/10.1145/1167948.1167973
[40] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of Psychology* 22, 140 (1932), 55–. http://psycnet.apa.org/record/1933-01885-001
[41] Marina Machado, Rui Couto, and José Creisssac Campos. 2017. MODUS: Model-based User Interfaces Prototyping. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '17)*. ACM, New York, NY, USA, 111–116. https://doi.org/10.1145/3102113.3102146
[42] Jan Meskens, Jo Vermeulen, Kris Luyten, and Karin Coninx. 2008. Gummy for multi-platform user interface designs: shape me, multiply me, fix me, use me. In *Proceedings of the working conference on Advanced Visual Interfaces, AVI 2008, Napoli, Italy, May 28-30, 2008*, Stefano Levialdi (Ed.). ACM Press, 233–240. https://doi.org/10.1145/1385569.1385607
[43] Pedro J. Molina, Santiago Meliá, and Oscar Pastor. 2002. *JUST-UI: A User Interface Specification Model.* Springer Netherlands, Dordrecht, 63–74. https://doi.org/10.1007/978-94-010-0421-3_5
[44] Michael Nebeling. 2017. XDBrowser 2.0: Semi-Automatic Generation of Cross-Device Interfaces. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, May 06-11, 2017.*, Gloria Mark, Susan R. Fussell, Cliff Lampe, m. c. schraefel, Juan Pablo Hourcade, Caroline Appert, and Daniel Wigdor (Eds.). ACM, 4574–4584. https://doi.org/10.1145/3025453.3025547
[45] Michael Nebeling, Theano Mintsi, Maria Husmann, and Moira Norrie. 2014. Interactive Development of Cross-device User Interfaces. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 2793–2802. https://doi.org/10.1145/2556288.2556980
[46] Thanh-Diane Nguyen, Jean Vanderdonckt, and Ahmed Seffah. 2016. Generative Patterns for Designing Multiple User Interfaces. In *Proceedings of the International Conference on Mobile Software Engineering and Systems (MOBILESoft '16)*. ACM, New York, NY, USA, 151–159. https://doi.org/10.1145/2897073.2897084
[47] Jeffrey Nichols, Duen Horng Chau, and Brad A. Myers. 2007. Demonstrating the Viability of Automatically Generated User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. ACM, New York, NY, USA, 1283–1292. https://doi.org/10.1145/1240624.1240819
[48] Jeffrey Nichols, Brandon Rothrock, Duen Horng Chau, and Brad A. Myers. 2006. Huddle: Automatically Generating Interfaces for Systems of Multiple Connected Appliances. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST '06)*. ACM, New York, NY, USA, 279–288. https://doi.org/10.1145/1166253.1166298
[49] Jeni Paay, Dimitrios Raptis, Jesper Kjeldskov, Bjarke M. Lauridsen, Ivan S. Penchev, Elias Ringhauge, and Eric V. Ruder. 2016. A Comparison of Techniques for Cross-Device Interaction from Mobile Devices to Large Displays. In *Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media (MoMM '16)*. ACM, New York, NY, USA, 137–146. https://doi.org/10.1145/3007120.3007140
[50] Jeni Paay, Dimitrios Raptis, Jesper Kjeldskov, Mikael B. Skov, Eric V. Ruder, and Bjarke M. Lauridsen. 2017. Investigating Cross-Device Interaction Between a Handheld Device and a Large Display. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 6608–6619. https://doi.org/10.1145/3025453.3025724
[51] Oscar Pastor and Juan Carlos Molina. 2007. *Model-driven architecture in practice - a software production environment based on conceptual modeling.* Springer, Berlin.
[52] Inés Pederiva, Jean Vanderdonckt, Sergio España, José Ignacio Panach, and Oscar Pastor. 2007. The Beautification Process in Model-Driven Engineering of User Interfaces. In *Human-Computer Interaction - INTERACT 2007, 11th IFIP TC 13 International Conference, Rio de Janeiro, Brazil, September 10-14, 2007, Proceedings, Part I (Lecture Notes in Computer Science)*, Maria Cecília Calani Baranauskas, Philippe A. Palanque, Julio Abascal, and Simone Diniz Junqueira Barbosa (Eds.), Vol. 4662. Springer, 411–425. https://doi.org/10.1007/978-3-540-74796-3_39
[53] Christian Prehofer, Andreas Wagner, and Yucheng Jin. 2016. A Model-based Approach for Multi-device User Interactions. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS '16)*. ACM, New York, NY, USA, 13–23. https://doi.org/10.1145/2976767.2976776
[54] Angel R. Puerta. 1997. A model-based interface development environment. *IEEE Software* 14, 4 (Jul 1997), 40–47. https://doi.org/10.1109/52.595902
[55] Angel R. Puerta, Henrik Eriksson, John H. Gennari, and Mark A. Musen. 1994. Beyond Data Models for Automated User Interface Generation. In *People and Computers IX, Proceedings of HCI '94, Glasgow, UK, August 1994*, Gilbert Cockton, Stephen W. Draper, and George R. S. Weir (Eds.). Cambridge University Press, 353–366.
[56] Angel R. Puerta and David Maulsby. 1997. Management of Interface Design Knowledge with MOBI-D. In *Proceedings of the 2nd International Conference on Intelligent User Interfaces, IUI 1997, Orlando, Florida, USA, January 6-9, 1997*, Johanna D. Moore, Ernest A. Edmonds, and Angel R. Puerta (Eds.). ACM, 249–252. https://doi.org/10.1145/238218.238337

[57] Jaroslav Pullmann. 2014. *Model-Based User Interface Charter - Glossary. W3C Working Group Note.* https://www.w3.org/TR/mbui-glossary/

[58] Claire Rowland, Elizabeth Goodman, Martin Charlier, Ann Light, and Alfred Lui. 2015. *Designing Connected Products âĂŞ UX for the Consumer Internet of Things.* O'Reilly, Cambridge.

[59] Stephanie Santosa and Daniel Wigdor. 2013. A Field Study of Multi-device Workflows in Distributed Workspaces. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '13).* ACM, New York, NY, USA, 63–72. https://doi.org/10.1145/2493432.2493476

[60] Florian Scharf, Christian Wolters, Michael Herczeg, and Jörg Cassens. 2013. Cross-Device Interaction : Definition, Taxonomy and Application. In *AMBIENT 2013 : The Third International Conference on Ambient Computing, Applications, Services and Technologies*, Maarten Weyn (Ed.). IARIA, IARIA, Porto, Portugal, 35–41. http://www.imis.uni-luebeck.de/publikationen/scharf-et-al-IARIA_Ambient-2013-dl.pdf

[61] Ahmed Seffah and Peter Forbrig. 2002. Multiple User Interfaces: Towards a Task-Driven and Patterns-Oriented Design Model. In *Interactive Systems:Design, Specification, and Verification*, Peter Forbrig, Quentin Limbourg, Jean Vanderdonckt, and Bodo Urban (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 118–132.

[62] Manuel Serrano and Gérard Berry. 2012. Multitier Programming in Hop. *Commun. ACM* 55, 8 (Aug. 2012), 53–59. https://doi.org/10.1145/2240236.2240253

[63] Amany Shatnawi and Raed Shatnawi. 2016. Generating a language-independent graphical user interfaces from UML models. *Int. Arab J. Inf. Technol.* 13, 6B (2016), 1039–1044. http://ccis2k.org/iajit/?option=com_content&task=blogcategory&id=104&Itemid=387

[64] Vi Tran. 2010. UI generation from task, domain and user models: the DB-USE approach. In *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing System, EICS 2010, Berlin, Germany, June 19-23, 2010*, Noi Sukaviriya, Jean Vanderdonckt, and Michael Harrison (Eds.). ACM, 353–356. https://doi.org/10.1145/1822018.1822079

[65] Jean M. Vanderdonckt and François Bodart. 1993. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93).* ACM, New York, NY, USA, 424–429. https://doi.org/10.1145/169059.169340

[66] Jishuo Yang and Daniel Wigdor. 2014. Panelrama: Enabling Easy Specification of Cross-device Web Applications. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems (CHI '14).* ACM, New York, NY, USA, 2783–2792. https://doi.org/10.1145/2556288.2557199

A CASE STUDY USED FOR FAMILIARIZATION IN THE FIRST EXPERIMENT

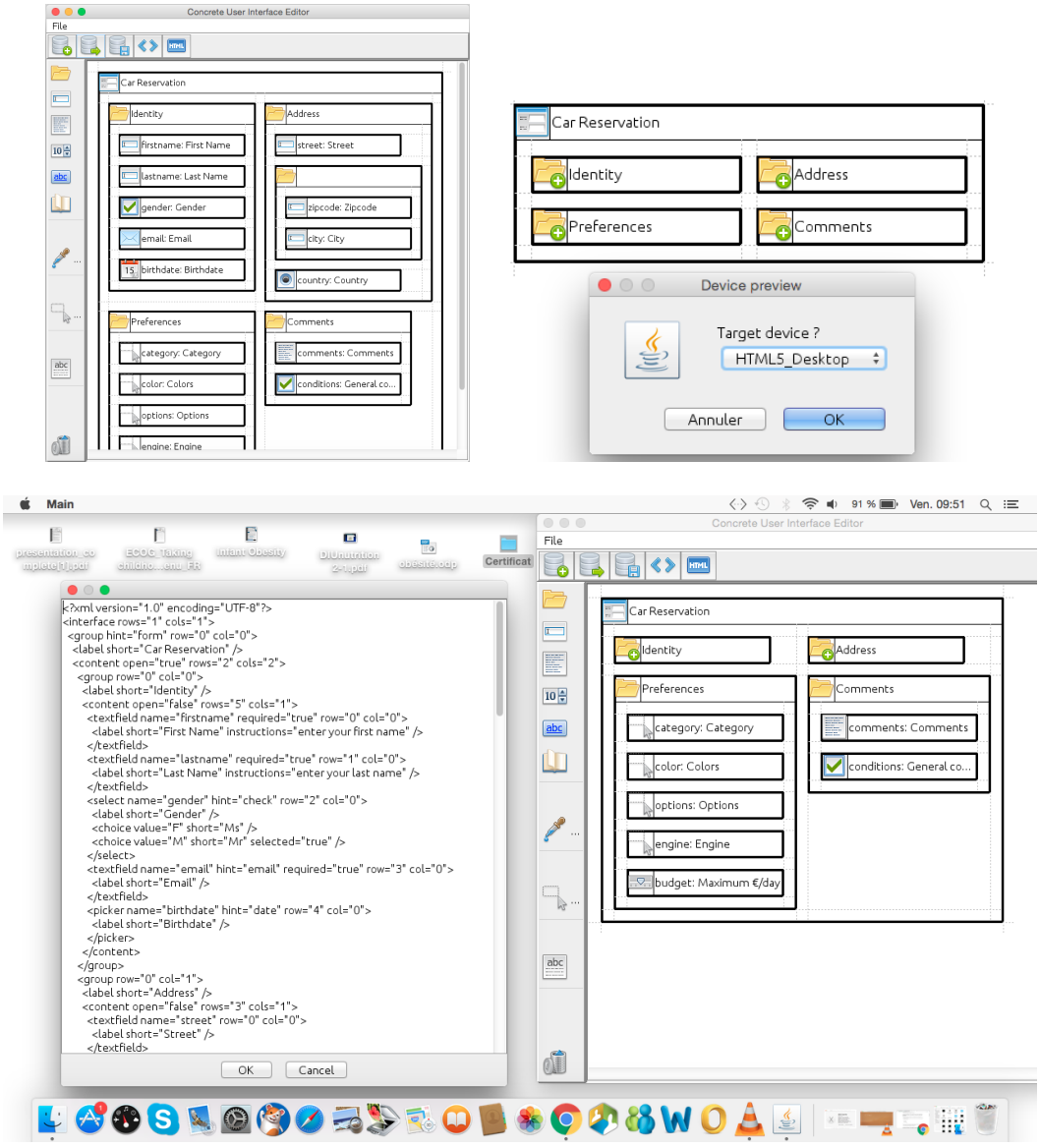

Fig. 28. Car rental case study: part 1/3.

B XDUIS USED FOR THE REPEATED AND INTEGRATED TESTS

Received July 24th, 2017; revised October 26th, 2018; revised February 21st, 2019; revised April 3rd, 2019; accepted April 24th, 2019

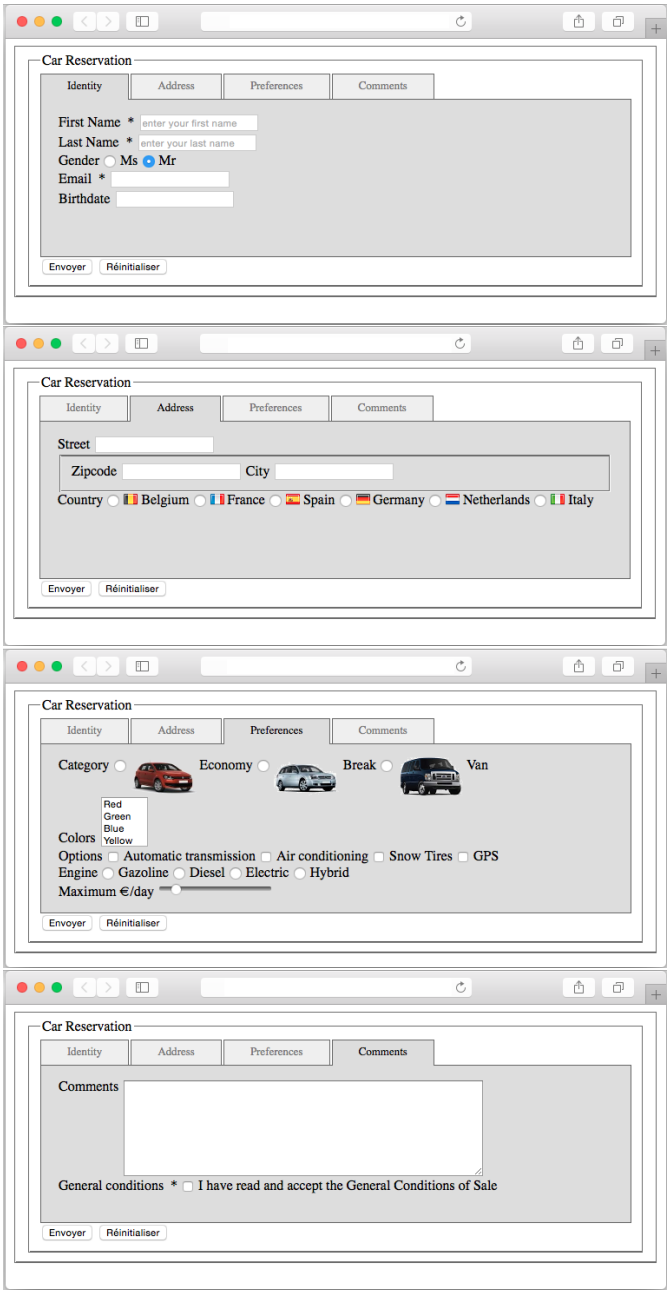

Fig. 29. Car rental case study: part 2/3.

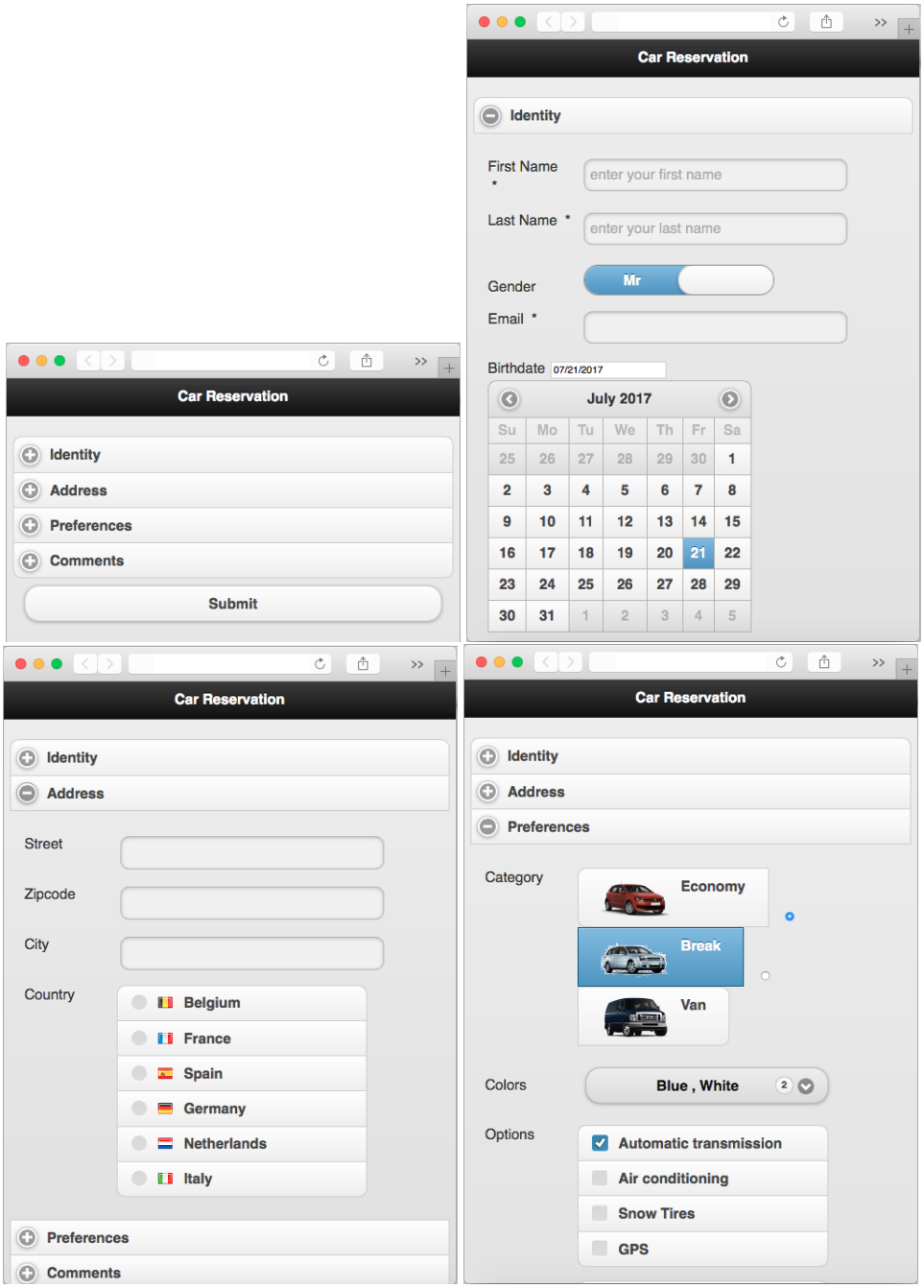

Fig. 30. Car rental case study: part 3/3.

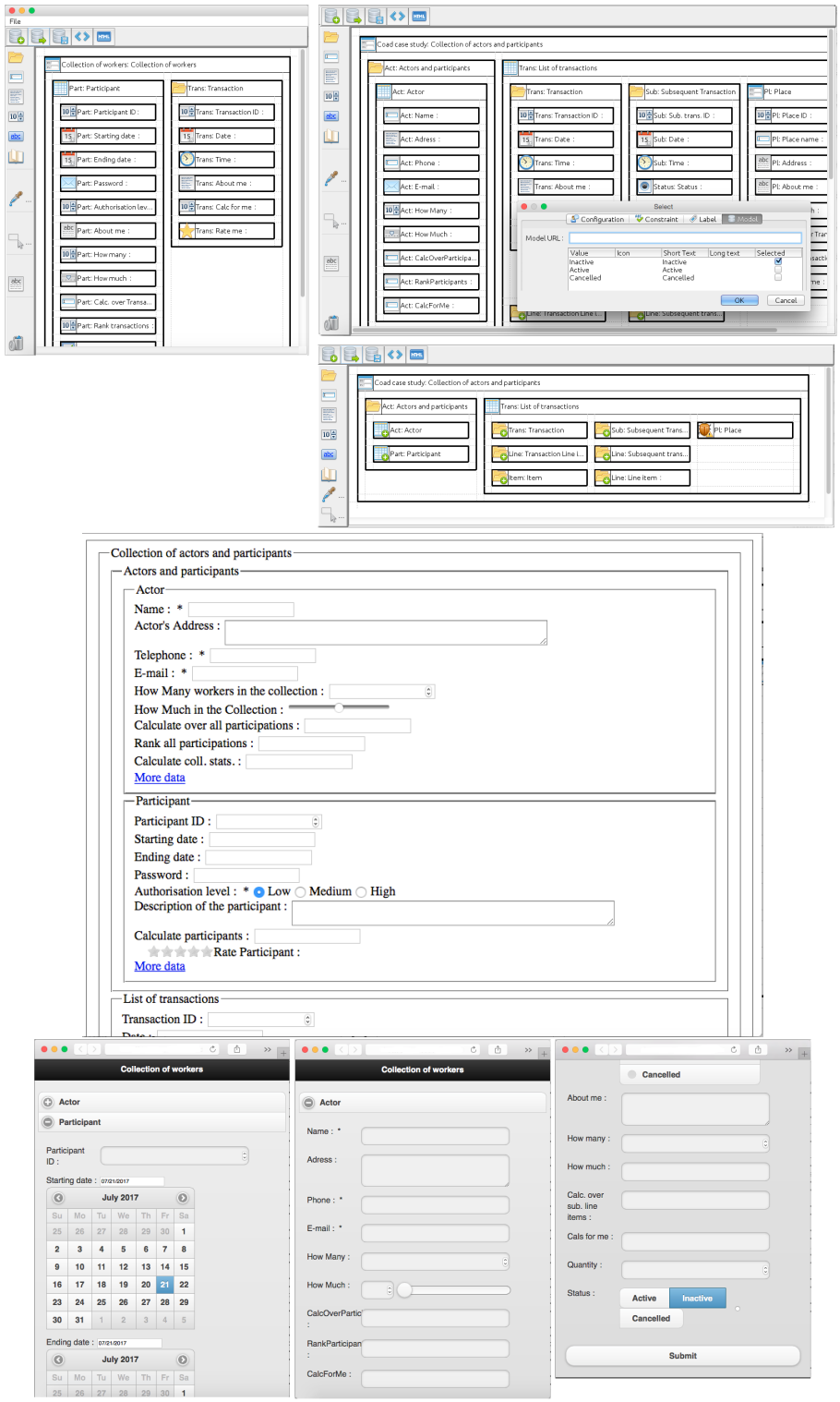

Fig. 31. Example of XDUIs involved in repeated and integrated tests.