Journal of Universal Computer Science, vol. 19, no. 6 (2013), 831-850
submitted: 30/9/12, accepted: 28/1/13, appeared: 28/3/13 © J.UCS

Defining Distribution Constraints in Distributed User Interfaces

Antonio Peñalver, José Juán López, Federico Botella
(Center of Operations Research University Institute
Miguel Hernández University, Elche, Spain
a.penalver@umh.es, jlopez@umh.es, federico@umh.es)

José Antonio Gallud
Interactive Systems Everywhere
Castilla la Mancha University, Albacete, Spain
Jose.Gallud@uclm.es)

Abstract: At present, the spread of hand held devices with growing computing power and functionality allows that different interaction elements can be distributed among a wide range of devices from different platforms, supporting interaction with one or many users. To take advantage of the benefits this kind of devices provides, traditional user interfaces have been evolving towards distributed user ones. The specification of the constraints on the way in which these elements can be distributed is still an open field and further research is needed. In this paper we propose a new schema-based definition of Distributed User Interfaces (DUIs), allowing the specification of the elements to be distributed, defining constraints on the distribution process itself, independently of the language selected to construct the interface. Thus, the interface distribution process becomes the creation of an XML instance from a grammar specified in Schema language. We also introduce two new definitions to complete the formalization of our previous definition of DUI. Our previously defined Abstract User Interface (AUI) model jointly with this new schema-based definition of DUIs will lead us to the full specification of concrete DUIs. We provide some examples of the distribution process using the proposed schema.

Key Words: Distributed User Interfaces, Formal Models, Schemas
Category: H.5.2 H.5.3

1 Introduction

As long as new surprising devices supporting new amazing interaction mechanisms have been introduced, the way people interact with computers (and/or mobile devices) has been changing accordingly. Nowadays, users can perform different tasks through a wide variety of computational devices ranging from mobile phones, Personal Digital Assistants (PDAs), Internet enabled televisions (WebTV), tablet PCs, laptops or notebooks. New mobile devices allow ubiquitous access to information and services as well as the opportunity to accomplish more and more desktop-related tasks with them almost at any time from everywhere [Weiser 1999]. This fact is bringing new challenges to the Human-Computer Interaction (HCI) community [Eisenstein et al. 2001]: (i) applications must be developed and maintained across multiple devices, (ii) consistency between versions for different devices must be guaranteed and (iii) versions must dynamically respond to changes in the environment.

In this context of strong technological growth, the increasing use of different displays managed by several users has improved user interaction. Combining fixed displays with wearable devices allows interaction and collaboration between users when they work together in a common task. This new scenario has been recently defined as Internet of Things [Atzori et al. 2010], referring to the ability to interact with a network of interconnected everyday objects called MDE (Multi-Device Environments).

In order to address these new challenges, user interfaces must be described independently of any concrete device. A User Interface Description Language (UIDL) is a formal language allowing to describe a particular user interface independently of any implementation technology [Guerrero et al. 2009]. A UIDL should be declarative so that it can be edited manually and it should also be formal to be understood and analyzed automatically by specific software.

A common fundamental assumption of most UIDLs is that UIs are modeled as algebraic or model-theoretic structures including a set of interaction objects and the behavior over them. In the last few years, some authors have proposed different UIDLs: UIML [Abrams et al. 1999, Helms et al. 2009], useML [Reuther 2003], MariaXML [Paterno et al. 2009], UsiXml [Limbourg et al. 2004], XIML [Puerta and Eisenstein 2002] or the new XAML [Microsoft 2006] interface description language proposed by Microsoft for developing Metro-based applications. All of them have pros and cons, due to the goals they are mainly intended to achieve. A deep review and a comparative analysis of some of them can be found in [Luyten et al. 2004, Faure and Vanderdonckt 2010, Shaer et al. 2008].

Many UIDLs are based on the XML markup language, rendering and describing the graphical user interface and controls. In order to strictly define the semantics of such UIDLs, various meta-models have been defined. These meta-models adhere to the principle of separation of concerns and could be classified as context of use (user, platform, environment), task, domain, abstract user interface, concrete user interface, usability and accessibility, workflow, organization, evolution, program, transformation and mapping.

Alternatively, user interfaces have been evolving towards “distributed” user interfaces, offering new interaction possibilities in agreement with new technological proposals. Distributed interfaces allow one or more interaction elements to be distributed over different platforms in order to support interaction between one or many users. Throughout the literature, we find several definitions of the concept: a user interface where the components can be distributed across one or more dimensions such as input, output, platform, space and time [Elmqvist 2011]; a user interface with the ability to distribute part or all of its components between multiple monitors, devices, platforms, screens and users [Melchior et al. 2009] or an interface that can be divided into parts and migrate to various devices around the user in order to facilitate user tasks [Vandervelpen et al. 2005]. Before the widespread acceptance of the Distributed User Interface, several terms were used instead, such as: Migration, Migratory User Interfaces, Migratable User Interfaces, Group / Ungroup, Portables, Transportable, Transformable or Reconfigurable [Demeure et al. 2008]. In subsection 2.1.9 we provide our formal definition of a DUI.

Starting from our Abstract User Interface [Gallud et al. 2012] and our formal description technique for developing DUIs presented in [Peñalver et al. 2011], we can produce unambiguous descriptions of complex interactions that occur in DUIs (distribution of elements, including communication and distributed interaction), more accurate and understandable than descriptions using only natural language. This method covered a wide range of descriptions from an abstract point of view but lacks the ability to get an automatic implementation in any of the previously cited UIDLs. Now we have developed a schema-based approach that allows us to automatically construct the concrete distributed user interface (DUI) regardless of the language selected to construct such interface, with the only requirement that it is a language based on XML.

The rest of the paper is organized as follows. First we review some formal definitions provided in [Peñalver et al. 2011], and the AUI model proposed in [Gallud et al. 2012]. Then, new definitions required for the schema specification of the interface are included. Next our schema driven distributed user interface model is presented and discussed. Last section provides conclusions and further work.

2 AUI model for DUIs

In this section we first review some formal definitions of distributed user interfaces. In [López-Espín et al. 2011] we defined the specification of distributed user interfaces (DUIs) that now we are going to reformulate partially. This formal view covers a wide range of descriptions from the most abstract model to the implementation-oriented models. Formal description techniques can provide us unambiguous descriptions of the interactions inside a DUI (like distribution of elements, communication and distributed interaction). This description of a DUI will be more precise than a description obtained using only natural language. In addition, formal description techniques provide the foundation for the analysis and verification of descriptions. The analysis and formal verification can be applied to specific or abstract properties. Natural language is a good complement to the formal notation for outlining a first idea of description.

2.1 Formal definitions

We can state that the four essential properties of portability, decomposability (and composability), simultaneity and continuity, will led us to the formal description of a distributed user interface. Next we are going to introduce several definitions to settle the basis of these essential properties:

2.1.1 Definition 1: Interaction Element

An *Interaction Element* $e \in E$ is defined as an element which allows a user u to carry out an interaction through a platform p. We denote an interaction of the user u though a platform p by $\sim^e p$.

2.1.2 Definition 2: Functionality

Two elements of interaction e and e' have the same *functionality* if a user can perform the same action using any of them when he/she interacts with the device. We will denote it by $e =^F e'$. In this sense, a button in a "Graphic Interface Unit" has the same functionality as a hand movement if the computer receives the same order.

2.1.3 Definition 3: Target

A set of interaction elements $E_0 \subset E$ have the same *Target* ($e \in^T E_0$) if $\forall e \in E_0$, a user $u \in U$ obtains, through the functionality of e, an action of the task whose goal is to reach this target. In this paper we use *target* and *goal* indistinctly.

2.1.4 Defnition 4: User Interface

A *User Interface* $i \in UI$ is a set of interaction elements $e \in E$ such as $i = \{e \in E \,/\, e \in^T i\}$, i.e., the user interface i is defined by the target for which these elements were chosen. We can also define a User Interface (UI) as a set of interaction elements which let a user carry out a task in a specific context.

2.1.5 Property 1: Portability

A UI as a whole, or parts of it, can be transferred among platforms and devices by means of easy user actions. For example, a user might be running a graphic editor in his/her desktop computer and then he/she could decide to transfer the color palette panel (UI element) to another platform (a portable device) with a simple action.

Formally we can say that being $p \in P$ and $u \in U$, a user interface $i \in UI$ / $u \sim^i p$ is *portable* if there exists $E_0 = \{e \in E / e \in i\} \subset i$ such as $u \sim^{E_0} p'$ and

$u \sim^{\bar{E}_0} p$ (being $p, p' \in P$) reaching the same target that i. This property can be extended to more than one user. $i \in UI$ has been *ported* if i is portable and this property has been satisfied.

2.1.6 Property 2: Decomposability

A DUI system is decomposable if given a UI composed by a number of elements, one or more elements of that UI can be executed independently without loosing their functionality. For instance, the calculator application could be decomposed in two UI elements, the display and the numeric keyboard. This property can be used together with *Portability* in order to allow that the keyboard is executed in a smartphone, while the display is showed in a public display. These two UI elements can also be joined together in a unique UI (composability).

A user interface $i \in UI$ is *decomposable* if there exists $E_0 \subset i$ such as $E_0 = \{e \in i/\ e \in^{T'} E_0\}$ and $\bar{E}_0 = \{e \in i/\ e \in^{T''} \bar{E}_0\}$ obtaining the same target that i. Thus, if the target T is reached through i, then T' and T'' are two sub-targets of T, and they can be reached through E_0 and $\bar{E}_0$ respectively.

2.1.7 Definition 5: User Sub-interface

Let suppose that $i \in UI$ is a user interface that allows a user $u \in U$ to reach a target T on a platform $p \in P$, i.e. $u \sim^T p$. If T' is a sub-target of T, then the set $i' = \{e \in^T i/e \in^{T'} i'\}$ is a *User Sub-interface* of i, and $u \sim^{T'} p$.

Finally we can say that $i \in UI$ has been *decomposed* if it is decomposable and this property has been fulfilled.

2.1.8 Definition 6: Platform

We can define a *platform* as a physical or logical medium where the user interface can be displayed. Thus, a user interface can be used in a platform if there exists some kind of framework which makes it possible.

An interaction element $e \in E$ exists in a platform $p \in P$ (denoted by $\sim^e p$), if e can be implemented, supported or executed on p. A user interface $i \in UI$ is supported on $p \in P$ (denoted by $u \sim^i p$) if $\forall e \in i$ then $u \sim^e p$ being $u \in U$. In addition, $i \in UI$ is supported on a set of platforms $P_0 \subset P$ ($u \sim^i P_0$) if $\forall e \in i$ then $u \sim^e p\ \forall p \in P_0$, being $u \in U$.

Therefore, a platform is a very general concept which can be particularized according to certain properties. For instance, if Android is considered as a platform, then it could be executed on a mobile smart phone or in a Tablet PC, having each execution different conditions.

2.1.9 Definition 7: Distributed User Interface

A *Distributed User Interface* $di \in DUI$ is defined as a user interface which has been decomposed and ported.

Thus, a distributed user interface is a collection of interaction elements composed of a set of user interfaces, which are also sub-interfaces of the user's distributed user interface. These user sub-interfaces are distributed in platforms without loosing their functionality and their common target.

Using this new notation it is possible to express the user interaction through traditional UIs as $u \sim^{i} p$, being $i \in UI$, the interaction of a user through DUIs as $u \sim^{di} p$, being $di \in DUI$, and the interaction of some users through some platforms through DUIs as $\{u/u \in U\} \sim^{di} \{p/p \in P\}$.

2.1.10 Definition 8: State of a User Interface:

The *State* of a user interface $i \in UI$, denoted by $S(i)$, is defined as the temporal point in which i lies after the user has used part of his/her elements with the goal of reaching the target associated to i. The State of i is the *Initial State* ($S_0(i)$) if none of the elements have been used or if they have been used without contributing any step to reach the target of i. The *Final State* of i ($S_F(i)$) is reached when the target of i is reached. It is said that this target is achieved in n steps or states, if through the sequence $S_0(i), ..., S_n(i)$, the target of i is reached and this target is achieved without any of the referred states.

Note that moving from the state $S_j(i)$ to $S_{j+1}(i)$ requires to use the appropriate interaction element $e \in i$. Others used elements do not change the state. There exists some elements which move from state $S_j(i)$ to $S_{j-1}(i)$, to $S_F(i)$ or to $S_0(i)$.

2.1.11 Definition 9: State of a Distributed User Interface

The *State* of a Distributed User Interface $di \in DUI$, denoted by $\mathbf{S}(di) = (S(i_1), ..., S(i_n))$, is defined as a n-tuple where each element corresponds to the state of each user interface in which di has been decomposed. Note that $\mathbf{S}(di)$ depends on the decomposition in sub-interfaces of di and those which have been ported to different platforms.

We say that a DUI di is in the initial state if $\mathbf{S_0}(di) = (S_0(i_1), ..., S_0(i_n))$, and di is in the final state if $\mathbf{S_F}(di) = (S_F(i_1), ..., S_F(i_n))$. The number of states required to reach the target of di is the product of the number of states required to reach each sub-target in each ported user sub-interface.

2.1.12 Property 3: Simultaneity

A DUI system is said to be simultaneous if different UI elements of the same DUI system can be managed at the same instant of time on different platforms. For instance, two or more users interacting simultaneously with the same DUI from different platforms. This fact does not imply that all DUI systems are multiuser as we will see later.

A distributed user interface di is *simultaneous* in $p_0, p_1, \ldots p_n \in P$ with $n > 1$ for $u_k \in U$ with $k = 1, ..., n_u$ $(n_u \geq 1)$ users, if $di = \bigcup_{j=1}^{N} i_j$ with $i_j \in UI$, and $u_k \sim^{i_j} p_s$ in the same temporal point, with $j = 1 \ldots N$ and $s = 1 \ldots n$ and $k = 1 \ldots n$.

2.1.13 Property 4: Continuity

A DUI system is said to be continuous if an element of the DUI system can be transferred to another platform of the same DUI system maintaining its state. For instance, a user could be on a call in his/her mobile phone walking on the street and then transfer the call to the TV when he/she arrives at home without interruption.

A distributed user interface $di \in DUI$ is *continuous* in $p_0, p_1 \in P$ if $\forall e \in di$, $u \sim^e p_0$ and $u \sim^e p_1$, the state of di is maintained, i.e., if $S_j(di)$ is the state of di, then $S_t(i)$ is reached in both cases (being able to be $t = 0, j, j+1, j-1, F$).

A DUI which satisfies the simultaneity property, having all the targets associated to the user sub-interfaces with a common purpose can raise interaction problems. For instance, a user might employ an interaction element which causes a setback in the state of another user interface. Thus, it is required to provide some kind of control between the user sub-interfaces in a concrete DUI.

2.1.14 Definition 10: Requirements function:

The *Requirements function* of a DUI obtains the necessary requirements of the state of a sub-interface to reach the next state. This function has two parameters: first, the sub-interface whose state is evaluated, and second, the point of the state which is going to be reached.

A user sub-interface can move back from a state to the previous state if the current state is not a requirement of any reached state in another sub-interface.

Let $di \in DUI$ and $\mathbf{E}(di) = (E(i_1), ..., E(i_n))$ the state of di, we define the requirements of the state of i_j to move from $m-1$ to m with $E_0(i_j) \leq m \leq E_F(i_j)$, as $R(i_j, m) = (E_1, ..., E_n)$, being E_k the minimum state required in the sub-interface i_k $\forall k = 1, ..., n$. If the sub-interface i_j is independent of i_k then the value of E_k in $R(i_j, m)$ is $E_0(i_k)$.

2.1.15 Definition 11: Concurrency Restriction

If the state of i_j is r (denoted by $E_r(i_j)$) and $E_s(i_k)$ is the value of the k-th position of $R(i_j, m)$, i.e., it is necessary to get $E_r(i_j)$ so that i_k reaches the state s, because it is a requirement of i_j. Then, it is not allowed to use an element which changes from $E_s(i_k)$ to $E_{s-1}(i_k)$.

2.2 The AUI model

The concept of Distributed User Interface is based on the concept of User Interface. In fact, we can consider that a DUI is a set of interaction elements distributed across different platforms. We can consider a set of interaction elements as a UI, if all these interaction elements have a common target. This common target is connected with the user's task, i.e. a UI is the way the user achieves the result of the task. This result appears in the model as *target.* Figure 1 shows the abstract user interface (AUI) model used in this paper.

A User Interface is composed of an abstract interaction object (AIO) that can be either an interaction element or a user sub-interface (uSubI). The interaction element can be used to express input, output, navigation or control actions. In this model two entities called "target" and "subtarget" can be found. These entities are present in the model to remark the importance of the common target that is inherent to the concept of user interface.

We can find a significative difference between this AUI model and the model defined in [Limbourg et al. 2004] as we introduce *target* and *subTarget* entities in the model. A DUI is a set of distributed interaction elements supporting a common target. Before distributing a UI, we can check if a set of interaction elements can be split and transferred to another platform with a simple verification. We have to test if the selected interaction elements have associated a sub-target or not.

Let's see the example of the calculator to explain the AUI model. The calculator as a whole is a UI so that the user can perform mathematical calculations (his/her final target). The calculator consists of an abstract interaction object which also contains an interaction element (the display) and a user sub-interface (the keyboard). The display has a subtask related to the main target: to show the calculations to the user. The sub-interface keypad consists of a set of interaction elements (buttons) with a common sub-objective: to allow user input. The display is framed within the output interaction elements, while each of the keys can be considered as input elements. The "key" interaction elements share a common sub-target: to allow data input, so it makes no sense to be distributed separately, but framed within its sub-interface, since they share the same input sub-target. To split or to distribute interaction elements that do not share a sub-objective linked to the main purpose of the UI can jeopardize the attainment of this objective.

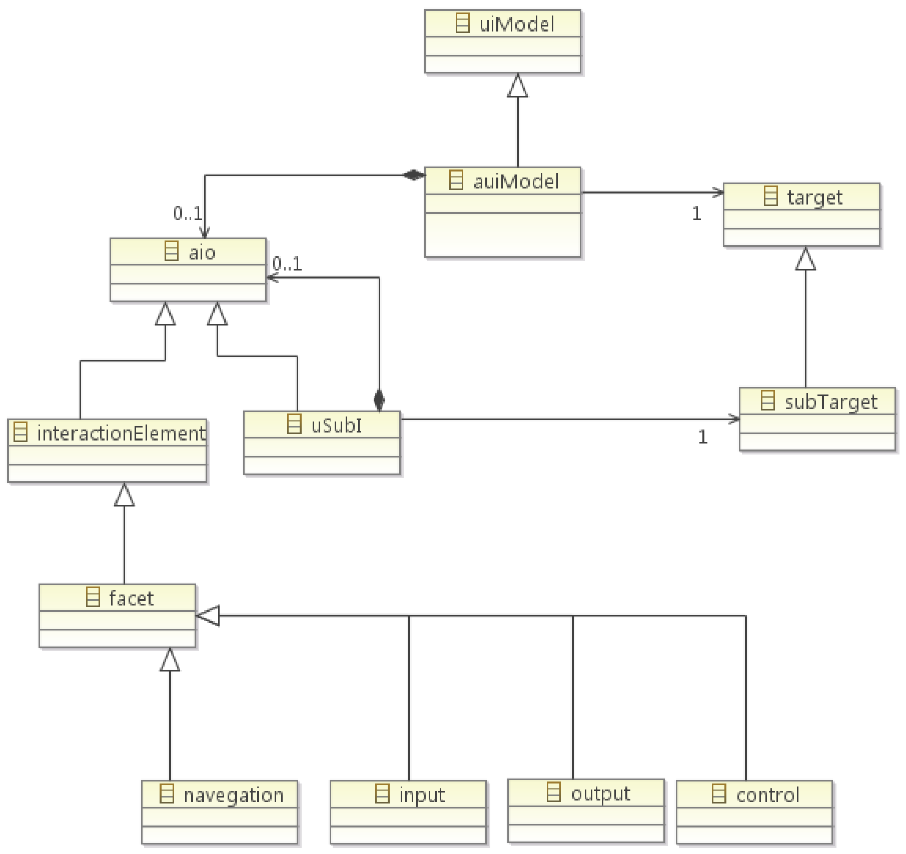

Figure 1: *Abstract User Interface with the DUI perspective*

3 Schema Driven DUI Generation

Now we are going to outline our approach for generating a concrete DUI that will be rendered on different devices. We get an AUI model in which the different sub-interfaces have been defined considering that all the associated interaction elements have a sub-target compatible with the common target of the user interface. There are important considerations related to the specification of the process: (i) the method should be independent of the selected UIDL and (ii) the syntax must be easy to learn so that the developer can design the DUI with minimal effort.

Another option may be to integrate the distribution properties in the UIDL directly. if we consider UIML, we could introduce the tags <distributable> </distributable> to indicate that the sub-interfaces defined between these tags can be split and thus distributed to other devices [Luyten and Coninx 2004]. However, this does not follow the current UIML specification and also imply that UIML renderers should understand these new tags. We rather aim for a more general solution to capture the distribution feature that is independent of the used description language, assuming that it is an XML-based one.

According to our previous definition, a DUI is a set of distributed interaction elements supporting a common target. As we said before, the process of distributing a UI, requires to check if a set of interaction elements can be split and transferred to other platform (portability and decomposability properties) and then, testing if the selected interaction elements have associated a sub-target or not. We first design the interface as a whole to define a distributable interface and then we define the distributable parts, taking into account if a sub-target is associated.

In a context of a multimedia player, these parts could include for example a “main” part, a “playlist” part and a “settings” part. We also need to impose some constraints on the parts in order to introduce some information about the structure of the interface. As all UIDLs considered are XML-based languages, we can constrain the structure of the UI using some XML schema language.

Previous work in the development of distributed interfaces required new methodologies [Bandelloni and Paterno 2004, Larsson and Berglund 2004] or ontologies [Balme et al. 2004]. We have discarded these approaches as we are looking for a method independent of the UIDL language and a syntax easy to learn.

Here we consider that an XML document can be validated against a schema to check if it accomplishes certain constraints. The process of validating an XML document consists of several stages: analysis of the structure of the document, analysis of the content of each node and its attributes and analysis of constraints of relations between different nodes. These stages can be performed through some schema language like Document Type Definitions (DTDs), W3C XML Schema [W3C] or RelaxNG [Clark and Murata 2001]. There are many tools allowing us to check if a concrete XML document matches a specific schema. Furthermore, we could develop an algorithm that provides valid instances (XML documents) from a schema. As schema languages allow “choices”, there are an infinite number of instances that matches a concrete schema, so such an algorithm could not deterministically generate a single instance based on random choices.

In [Fitch 2002], author introduces the use of a schema language for dynamically generating HTML forms by means of a self-defined schema language instead of any of the well-known standard ones. Rather than hard-coding forms, the approach dynamically generates an HTML form using an XML document representing the application data and an XML schema document describing the

```
<body>
...
   <div>
      This is
      <a href="#">an example</a> of mixed content
   </div>
...
</body>
```

Figure 2: *XML node with mixed content that cannot be constrained with RelaxNG*

data and the way it may be operated upon by the user. The schema describes the document so that an HTML form can be generated using Javascript DOM API.

In [Vandervelpen et al. 2005] a framework for dynamically distribute application UI's among several devices is developed. In the approach, RelaxNG schema language is used to describe the XHTML interface, defining constraints for each element, attribute and text value in the XML file. They create a RelaxNG schema file for each service an application provides. Then, a RelaxNG-based schema to instance algorithm is developed in order to generate a concrete XHTML user interface.

Although the RelaxNG schema language is simple and flexible, it has some drawbacks in order to be used for defining a DUI: (i) it does not allow to constrain text values in mixed content nodes and (ii) it does not support datatype validation, so it needs a datatype library to extend the RelaxNG functionality, like W3C XML Schema Type Library.

Figure 2 shows an instance of an XML document with a <div> node containing the texts "This is" and "of mixed content" and a <a> element that is so called "mixed content". Although it is just a simple example, the above situation is very common and there are many situations that require multiple content in a web interface. Both text values cannot be constrained with RelaxNG. Therefore, we select XML Schema in order to define in a more accurate way the constraints related to the distributions of the different elements of the interface in a more accurate way.

The key idea is to use the XML Schema patterns like <sequence>, <choice> and <all> to define constraints about the sub-interfaces that can be distributed between different devices. We use the following XML Schema order container indicators:

- <*sequence*>: Each child node must occur in the order specified in the XML file.
- <*all*>: Specifies that the child elements can appear in any order, and that each child element must occur only once.
- <*choice*>: One of the child nodes of a choice node must be chosen. Thus, there are n possible distribution options where n is the number of child nodes.

Notice that the introduction of "choice" container indicators provides different paths that can be followed while generating an actual instance from the schema. Each path results in a specific user distributed interface that incorporates a number of sub-interfaces. Combining the containers allows us to define different constraints between sub-interfaces: we could say that sub-interfaces SI_1 and SI_2 must appear in the specified order (<sequence> container), or that sub-interfaces SI_1 and SI_2 must appear in any order (<all> container) or that either sub-interface SI_1 or sub-interface SI_2 should be included in the final instance (<choice> container)...

Also we can constraint the minimum and maximum number of appearances of an element in the container using the occurrence indicators. If no indicators are defined then the element is required and must appear just once:

- *minOccurs*: Specifies the minimum number of times an element can occur. If the value is "0" then the element is optional.
- *maxOccurs*: Specifies the maximum number of times an element can occur. To allow an element to appear an unlimited number of times we use the "unbounded" value.

3.1 Distributed Interfaces Examples

In order to test our proposal, some interfaces have been designed and their sub-interfaces described using an XML schema. Here we provide a distributed calculator, based on classic XHTML and a distributed drawing tool, based on HTML5 that makes use of the advanced features of this new version of the standard for managing graphical interfaces using the new <canvas> tag and Javascript. We provide some screenshots of the final interfaces generated and displayed by the Distributed User Interface Framework we are currently developing.

Figure 3 lists an XML Schema definition for DPaint, our distributed drawing tool, with three available distributable parts: "canvas" sub-interface, "colors" sub-interface and "sizes" sub-interface. Last lines include references to the

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!– definition of "html" node –>
  <xs:element name="html">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="head"/>
        <xs:element ref="body"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="head">
  <!– definition of "head" node –>
  </xs:element>
  <xs:element name="body">
  <!– definition of "body" node –>
    <xs:complexType>
      <!– UI parts –>
      <xs:sequence>
        <xs:sequence minOccurs="0" maxOccurs="1" >
          <xs:element ref="canvas"/>
        </xs:sequence>
        <xs:choice minOccurs="0" maxOccurs="1" >
          <xs:element ref="colors"/>
          <xs:element ref="sizes"/>
        </xs:choice>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!– definition of "canvas" sub-interface –>
  <xs:include schemaLocation="canvas.xsd"/>
  <!– definition of "colors" sub-interface –>
  <xs:include schemaLocation="colors.xsd"/>
  <!– definition of "sizes" sub-interface –>
  <xs:include schemaLocation="sizes.xsd"/>
</xs:schema>
```

Figure 3: *XML Schema for an HTML 5 distributed paint application*

```
...
<xs:element name="body">
<!– definition of "body" node –>
   <xs:complexType>
      <!– UI parts –>
      <xs:all>
         < xs:element ref="canvas" minOccurs="0" maxOccurs="1" />
         < xs:element ref="colors" minOccurs="0" maxOccurs="1" />
         < xs:element ref="sizes" minOccurs = "0" maxOccurs="1" />
      </xs:all>
   </xs:complexType>
</xs:element>
...
```

Figure 4: *XML Schema for a less constrained distribution of sub-interfaces for the distributed paint application.*

schema definitions for every sub-interface. The lines inside the “body” element constrain the appearance of the sub-interfaces. The lines between the “html” element define the standard structure of a standard XHTML-based document with <head> and <body> tags.

The combination of <sequence> and <choice> elements with “minOccurs” and “maxOccurs” properties specify that the “canvas” sub-interface is optional and may appear just once and that the “colors” or the “sizes” sub-interfaces may appear once in a valid distributed user interface instance, but if one of them appears, then the other cannot. Moreover, if the “canvas” sub-interface is selected, it will be placed before the “colors” and “sizes” sub-interfaces.

This schema is quite restrictive, since we have established that the “colors” and “sizes” sub-interfaces cannot appear together in any generated DUI. In general, DUIs should be more flexible, i.e. users may wish to select both, the “colors” and "sizes" sub-interfaces, which is not allowed according to the schema, so it must be considered just as an instance of the power and flexibility of our approach. Figure 4 shows a different schema specifying that each sub-interface can appear (or not) in any position and combination just one time.

This strategy allows us to establish different constraints on the referred sub-interfaces by changing the XML Schema elements and using different combinations. It could be defined that a sub-interface must appear close to each other

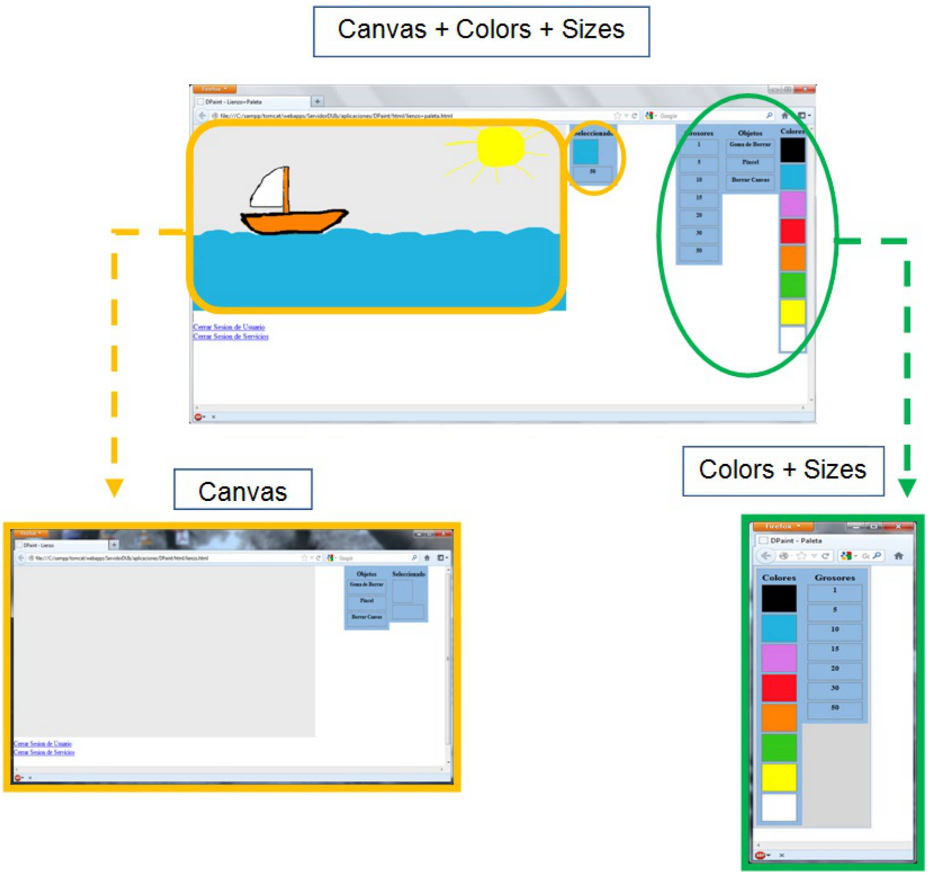

Figure 5: *Distributed Paint interface rendered in three different browsers*

by placing them in an XHTML <table> or <div> tag, and so on. Although we have selected an XHTML-based example, the method could be easily extended to other UIDLs, just by changing the tags used in XML Schema to instance algorithm.

Once the DUI has been defined, a schema to instance algorithm processes the XML Schema files and generates a new XML instance encapsulating each user sub-interface before sending it to each device. As we use choices in the definition of the DUI, a potentially infinite number of instances could be generated, so the algorithm should be able to decide the most suitable path to follow and hence, the concrete instance to generate. On one hand we can generate a single instance based on random selection. But also it could be better to define a metric

```
...
<xs:element name="body">
<!– definition of "body" node –>
    <xs:complexType>
        <!– UI parts –>
        <xs:sequence>
            < xs:element ref="display" />
            < xs:element ref="keyboard" minOccurs="0" maxOccurs="1" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
...
```

Figure 6: *XML Schema for a classic calculator application with two different sub-interfaces: the display and the keyboard.*

that allows the algorithm to decide what sub-interface are distributed to each device based on some distribution criteria. Furthermore, the schema does not include any information about the number of simultaneous devices in which a sub-interface can be distributed nor constraints related to concurrent access (as it was defined in the formal definitions section) to the same sub-interface from different users. Both aspects are out of the scope of this work and they will be the focus of further research.

Figure 5 shows three different interfaces generated from the schema of figure 4. On the top we can see an instance of the schema including the three available sub-interfaces. On the left, the instance only includes the “canvas” sub-interface. On the right we can see the “colors” and “sizes” sub-interfaces rendered together in a browser. All of them fulfill the constraints specified in the schema.

The well-known calculator program as a whole has a UI devoted to allow users to perform mathematical calculations. According to our previous formal definitions of Finality and User sub-interface, we could split the original interface into two sub-interfaces: the display and the keyboard. In this easy example, the keyboard and display sub-interfaces may be distributed together to a user, but the display sub-interface could also be distributed alone to another user. Figure 6 shows an XML Schema definition of the distributed calculator where the “display” sub-interface must be always included and the “keyboard” sub-interface is optional, but, if it is selected it must appear just once. Last, figure 7

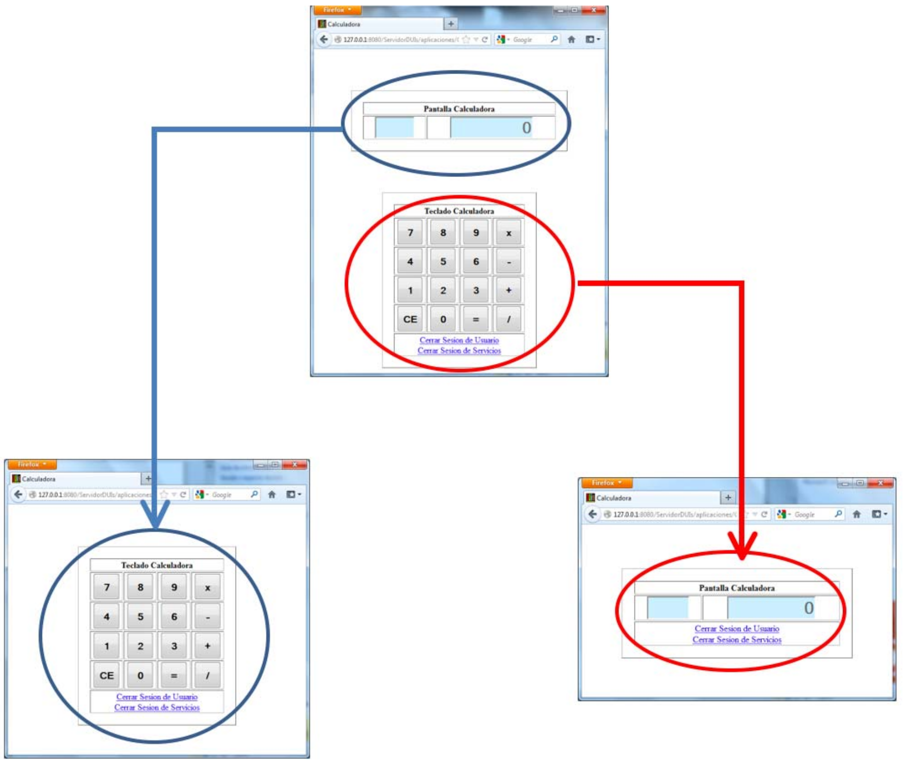

Figure 7: *Distributed calculator interface with full interface (top), "keyboard" sub-interface (bottom-left) and "display" interface (bottom-right)*

shows the full interface with both sub-interfaces on the top and only the display sub-interface blue line) on the bottom-right image. The bottom-left image, including only the "keyboard" sub-interface (red line), cannot be distributed with the specified schema.

4 Conclusions and further work

In this paper, we have proposed a new W3C XML Schema-based definition for user interfaces allowing different sub-interfaces being distributed between heterogeneous devices. We defined a DUI through a formal notation and after that an Abstract User Interface (AUI) model as previous steps before the specification of a concrete DUI. It is worth noting that the schema does not depend on the UIDL selected and encapsulates constraints related to the distribution process itself.

After the DUI has been defined, an XML instance generator algorithm generates a new XML instance (concrete DUI) in some UIDL, taking into account the constraints specified in the schema. The use of a formal notation to characterize Distributed User Interfaces is necessary to understand the essential foundations of DUIs and a previous stage to analyze different distribution mechanisms. The AUI model introduces the target hierarchy associated to every user interface and supports the distribution of user interface elements across different devices, maintaining the coherence with the user task target. The model is based on the characterization of the essential properties of Distributed User Interfaces: decomposability, portability, simultaneity and continuity. We have employed a formal notation to describe these properties.

Our future work includes the finalization of a user friendly model-based framework software that using the AUI model as input, allows as to automatically generate the XML Schema associated to the DUI. We are also working on the definition of a metric in order to decide the most suitable distribution scheme when the <choice> container is used. This metric will require the use of device profiles and the formal definition of the "optimal distribution" concept.

Acknowledgments

This research is partially funded by the project 11859/2011 from Bancaja-UMH of Miguel Hernández University of Elche.

References

[Abrams et al. 1999] Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., Shuster, J.: "Uiml: An appliance-independent xml user interface language"; Computer Networks; 31 (1999), 1695–1708.

[Atzori et al. 2010] Atzori, L., Iera, A., Morabito, G.: "The internet of things: A survey"; Comput. Netw.; 54 (2010), 15, 2787–2805.

[Balme et al. 2004] Balme, L., Demeure, A., Barralon, N., Coutaz, J., Calvary, G.: "Cameleon-rt: A software architecture reference model for distributed, migratable, and plastic user interfaces"; Lecture Notes in Computer Science; 3295 (2004), 291–202.

[Bandelloni and Paterno 2004] Bandelloni, R., Paterno, F.: "Flexible interface migration"; Intelligent User Interface 2004 (IUI 04); 148–155; 2004.

[Clark and Murata 2001] Clark, J., Murata, M.: "Relax ng specification. available at http://www.oasis-open.org/committees/relax-ng/spec-20011203.html"; (2001).

[Demeure et al. 2008] Demeure, A., Sottet, J.-S., Calvary, G., Coutaz, J., Ganneau, V., Vanderdonckt, J.: "The 4c reference model for distributed user interfaces"; Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems; ICAS '08; 61–69; IEEE Computer Society, Washington, DC, USA, 2008.

[Eisenstein et al. 2001] Eisenstein, J., Vanderdonckt, J., Puerta, A.: "Model-based user-interface development techniques for mobile computing"; J. Lester (Ed.), Proceedings of ACM International Conference on Intelligent User Interfaces IUI 2001, Santa Fe; 69–76; ACM Press, New York, 2001.

[Elmqvist 2011] Elmqvist, N.: “Distributed user interfaces: State of the art”; Distributed User Interfaces; Human–Computer Interaction Series; 1–12; Springer London, 2011.

[Faure and Vanderdonckt 2010] Faure, D., Vanderdonckt, J.: “User interface extensible markup language”; 2nd ACM SIGCHI symposium on Engineering interactive computing systems(EICS ’10). ACM, New York, NY, USA; 361–362; 2010.

[Fitch 2002] Fitch, K.: “Schema driven user interface generation. available on-line at: http://ausweb.scu.edu.au/ aw02/papers/refereed/fitch/paper.html”; (2002).

[Gallud et al. 2012] Gallud, J. A., Peñalver, A., López-Espín, J., Lazcorreta, E., Botella, F., Fardoun, H. M., Sebastián, G.: “A proposal to validate the user’s goal in distributed user interfaces”; International Journal of Human Computer Interaction; (To appear in 2012).

[Guerrero et al. 2009] Guerrero, J., González, J., Vanderdonckt, J., Muñoz, J. A.: “Theoretical survey of user interface description languages: Prelimi-nary results”; LA-Web/CLIHC’2009; 36–43; IEEE Computer Society Press, Los Alamitos, 2009.

[Helms et al. 2009] Helms, J., Schaefer, R., Luyten, K., Vermeulen, J., Abrams, M., Coyette, A., Vanderdonckt, J.: “Human-centered engineering with the user interface markup language”; Seffah, A., Vanderdonckt, J., Desmarais, M. (eds.), Human-Centered Software Engineering; 141–173; Springer, 2009.

[Larsson and Berglund 2004] Larsson, A., Berglund, E.: “Programming ubiquitous software applications: requirements for distributed user interfaces”; 16th International Conference on Software Engineering and Knowledge Engineering (SEKE4); 2004.

[Limbourg et al. 2004] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V.: “Usixml: a language supporting multi-path development of user interfaces”; 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS’2004; 11–13; Kluwer Academic Press, Dordrecht, 2004.

[López-Espín et al. 2011] López-Espín, J., Lazcorreta, E., Gallud, J., Peñalver, A., Botella, F.: “Formal specification of distributed user interfaces”; In Proc. of DUI 2011 Workshop; University of Castilla-La Mancha, 2011.

[Luyten et al. 2004] Luyten, K., Abrams, M., Vanderdonckt, J., Limbourg, Q.: “Developing user interfaces with xml: Advances on user interface description languages, advanced visual interfaces, galipoli”; (2004).

[Luyten and Coninx 2004] Luyten, K., Coninx, K.: “Uiml.net: An open uiml renderer for the .net framework. technical report, limburgs universitair centrum”; (2004).

[Melchior et al. 2009] Melchior, J., Grolaux, D., Vanderdonckt, J., Van Roy, P.: “A toolkit for peer-to-peer distributed user interfaces: concepts, implementation, and applications”; Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems; EICS ’09; 69–78; ACM, New York, NY, USA, 2009.

[Microsoft 2006] Microsoft: “Xaml language specification. available on-line at: http://download.microsoft.com/download/0/a/6/0a6f7755-9af5-448b-907d-13985accf53e/%5bms-xaml%5d.pdf”; (2006).

[Paterno et al. 2009] Paterno, F., Santoro, C., Spano, L.: “Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments”; ACM Trans. Computer-Hum. Interaction; 16 (2009), 4.

[Peñalver et al. 2011] Peñalver, A., López-Espín, J., Gallud, J., Lazcorreta, E., Botella, F.: “Distributed user interfaces: Specification of essential properties”; J. A. Gallud, R. Tesoriero, V. M. Penichet, eds., Distributed User Interfaces; Human-Computer Interaction Series; 13–21; Springer London, 2011.

[Puerta and Eisenstein 2002] Puerta, A., Eisenstein, J.: “Ximl: a common representation for interaction data”; 7th international conference on Intelligent user interfaces (IUI ’02). ACM, New York, NY, USA; 214–215; 2002.

[Reuther 2003] Reuther, A.: "useml - systematische entwicklung von maschinenbediensystemen mit xml. fortschritt-berichte pak, band 8, kaiserslautern: Technische universitt kaiserslautern"; (2003).

[Shaer et al. 2008] Shaer, O., Green, M., Jacob, R., Luyten, K.: "User interface description languages for next generation user interfaces"; Proc. of Extended Abstracts of CHI'08; 3949–3952; ACM Press, New York, 2008.

[Vandervelpen et al. 2005] Vandervelpen, C., Vanderhulst, G., Luyten, K., Coninx, K.: "Light-weight distributed web interfaces: Preparing the web for heterogeneous environments"; The 5th International Conference on Web Engineering (ICWE '05); 2005.

[W3C] W3C: "World wide web consortium. xml schema. http://www.w3.org/xml/schema.".

[Weiser 1999] Weiser, M.: "The computer for the 21st century"; SIGMOBILE Mob. Comput. Commun. Rev.; 3 (1999), 3, 3–11.