

# A Design Space for User Interface Composition

Fabio Paternò, Carmen Santoro, and Davide Spano

**Abstract.** Modern user interfaces are highly dynamic and interactive. They often compose in various ways user interface components. Thus, there is a need to understand what can be composed in user interfaces and how. This chapter presents a design space for user interface composition. The design space consists of five dimensions addressing aspects ranging from the UI abstraction level involved by the composition, the granularity of UI elements involved by the composition, the UI aspects that are affected by it, the time when such a composition occurs, and the type of web services involved in the composition. The design space is then analysed with respect to the capabilities of the capabilities of the ConcurTaskTrees and MARIA languages in order to show how it is possible to compose user interfaces at various abstraction levels. In order to provide a deeper insight, in the paper we also present a number of excerpts for several composition examples.

## 1 Introduction

The basic idea of user interface composition is to combine pieces of User Interface (UI) descriptions in order to obtain a new user interface description for a new interactive application. In the context of composition of user interfaces for emerging service-based applications two main cases can be identified. On the one hand, we can compose services that do not have associated any UI. In this case, a strategy could be composing web services (using already existing and well-known techniques for orchestration of Web services) and then derive the UI of the service resulting from such composition. Examples of such approaches are WS-BPEL [9], a notation that is well known and used in the field for representing and composing business processes, BPEL4people [1], which is an extension of BPEL to the standard WS-BPEL elements for modelling human interaction in business processes, and WS-Human Task [2], a notation for the integration of human beings in service-oriented applications. On the other hand, we can have the case when service-based applications have already associated user interfaces. They can be described through

---

Fabio Paternò · Carmen Santoro · Davide Spano  
CNR-ISTI – HIIS Laboratory, Pisa, Italy

annotations (see for example [6]) providing some indications for the corresponding user interface specification or can be complete user interface descriptions composed to create new interactive applications, as it happens with mash-up applications. In this case, the focus is on composing the user interfaces descriptions. In this chapter, we will mainly focus on such aspects.

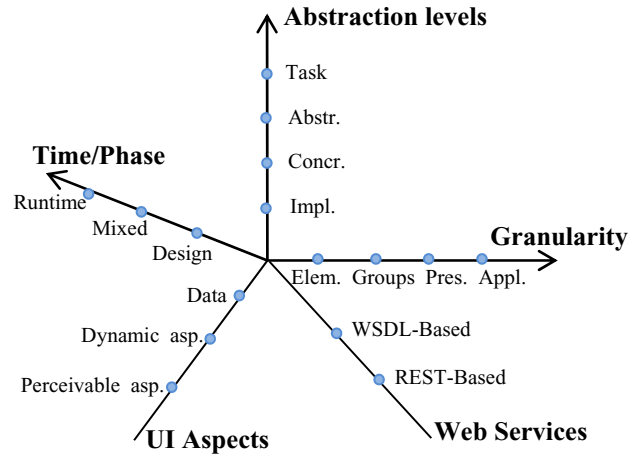
Many UI definition languages are XML-based, thus some work on user interface composition has aimed to apply the tree algebras for XML data system to the XML languages for user interface definition. A tree algebra for XML data systems can be found in [4]. A tree is as a set of nodes, each one with a single parent and many children. The trees have two types of operators: algebraic and relational. Another tree algebra definition can be found in [5], with a similar approach. The main difference is the focus of the definition: Jagadish et al. [5] aim to discuss a solution for creating data structures able to run efficient queries on XML trees and they define operators similar to the usual relational database. A similar tree-based approach, but specific for the concrete level of the UI has been applied for the USIXML Language [7]. In this paper we consider compositions that are mainly driven from the user interface perspective taking into account the support that they have to provide. In order to discuss comprehensively these issues, a systematic approach for analysing this problem is advisable. To this aim, we have identified a design space based on various dimensions on which the composition of the UI can vary.

More specifically, this chapter describes a five-dimensional problem space for UI composition. In particular, in Section 2 we report an overview of the main characteristics of the problem space, showing a number of problem dimensions, and which aspects are modelled by each dimension. Afterwards, Section 3 is dedicated to describing how MARIA and ConcurTaskTrees support the different compositions identified in the problem space. For the various options we provide some examples. Finally, a final section summarises the main points covered.

## 2 The Design Space for Composition of UI

User interfaces can be composed according to a number of aspects. In this section we describe a design space referring to such aspects in order to identify the possible compositions that can occur. Up to now, five dimensions have been identified (see Figure 1):

- **Abstraction level** in which the user interface is described;
- **Granularity** of the user interface elements considered;
- **UI Aspects** addressed;
- **Time/Phase** when the composition occurs;
- The type of **Web Services** involved in the composition.



**Fig. 1** The problem space for composition of UIs.

As we better see in the following sections, only on three (out of five) axis there is an ordering relationships between the contained values. In Figure 1 this has been highlighted by opportunely using arrows or plain lines for representing the different dimensions. In the next sub-sections (Section 2.1 – Section 2.5) we describe more in detail the characteristics of every axis of the problem space. Afterwards, in Section 3 we analyse how MARIA and ConcurTaskTrees support the composition options offered by the problem space.

## 2.1 Abstraction Levels

Since a UI can be described at various abstraction levels (task and objects, abstract, concrete, and implementation level), the user interface composition can occur at each of these levels. The abstraction levels usually considered are those of the CAMELEON Reference Framework [3], starting from the highest abstraction levels to the more concrete ones:

- **Task:** this level refers to activities performed to reach the users' goals;
- **Abstract:** this is the level of a platform-independent interface description
- **Concrete:** an interface description referring to a specified platform, but in a manner that is independent from the implementation framework
- **Implementation:** interface description referring to a specific implementation language.

Please note that with “platform” we mean a group of devices sharing a number of common interaction resources. Examples of different platforms are the graphical desktop platform, the graphical mobile platform, the vocal platform.

Moreover, transformations could be defined between the various user interface descriptions of the different levels. One example of such transformations could be

the mapping that associate abstract interactors to concrete ones (through forward transformations). However, it could also be possible to have backward references (for instance, when it is possible to map abstract interactors to tasks).

Finally, an ordering relationship exists among the various values of this axis: the task level is the most abstract level, while the implementation level is the most concrete one.

## 2.2 *Granularity*

The granularity refers to the size of elements that can be composed. Indeed, we can compose single user interface elements or groups of objects, and we can join presentations in order to obtain the user interface for an entire application. Since it is also possible to compose user interfaces of applications (e.g., to obtain mash-ups), another value has been added on this dimension: "Applications". Therefore, below you can find the various values on this dimension, as they appear in the corresponding axis, starting from the most elementary ones to more complex objects (for instance, in a presentation we can have compositions of groups, as well as in an application we can have several presentations).

- **Elements:** Composing single user interface elements;
- **Groups:** Composing groups of elements;
- **Presentations:** Composing different presentations. A presentation is a set of UI-elements that are all perceivable at a certain moment. Therefore, a presentation can be seen in some sense as defining the current context where user actions can be performed. For instance, for a Web site, the page the user is currently visiting represents a presentation.
- **Applications:** Composing different applications. For instance, with mash-up applications it is possible to add in a new composed application different (and generally smaller) applications (or 'gadgets'). An example of this is iGoogle (<http://www.google.com/ig>), where the user can create her personalised dashboard by grouping together 'gadgets', such as a weather application, a news feed, etc.

Again, also for this axis, the various values have been placed assuming an implicit underlying order (from the most elementary ones to the most structured/comprehensive ones).

## 2.3 *UI Aspects*

With this dimension we distinguish the different types of compositions depending on the main UI aspects the composition affects.

- **Data:** in this case the composition is carried out on the data manipulated by the UI elements considered;
- **Dynamic aspects:** in this case the composition affects the possible sequencing of user actions and system feedback;

- **Perceivable aspects:** this is the case when the composition acts on perceivable UI aspects.

No specific ordering relationships exists between the various values belonging to this axis.

## 2.4 *Time/Phase*

This dimension specifies when the composition occurs. It can be either a static composition, occurring at design time, in which case the elements we are going to compose are known in advance. It can also be a dynamic composition, occurring at runtime, during the execution of the application.

The latter composition (occurring at runtime) is especially significant in ubiquitous applications, since in such environments the context of use can vary a lot and then it may require different services according to such variations. Another option is the possibility of having types of composition that are a combination of the two possibilities. To summarise:

- **Runtime:** during application execution
- **Design Time:** when designing the application
- **Mixed:** when the composition is a mixed-up of the previous cases (e.g. when the composition occurs partially at design time and partially at runtime)

The time when the composition occurs can be assumed as the underlying order of this axis.

## 2.5 *Web Services*

A "Web service" is defined as "a software system designed to support interoperable machine-to-machine interaction over a network" [13]. Also, two basic classes of Web Services can be identified: REST-compliant Web services, and WSDL- (or SOAP-)based Web Services. On the one hand, SOAP (using WSDL) is a XML standard-based on document passing, with very structured formats for requests and responses. On the other hand, REST is lightweight, basically requiring just HTTP standard to work, and without constraining to a specific required format. In [12] there is an interesting comparison and discussion of strengths and weaknesses of both approaches.

## 3 How MARIA and ConcurtaskTrees fit the Problem Space

In this section we mainly focus on how the MARIA language [11] and Concur-TaskTrees [10] allow the composition mechanisms that are identified through the problem space. In order to concretely describe the support for the problem space, we also provide excerpts for some composition examples analysed. We start our

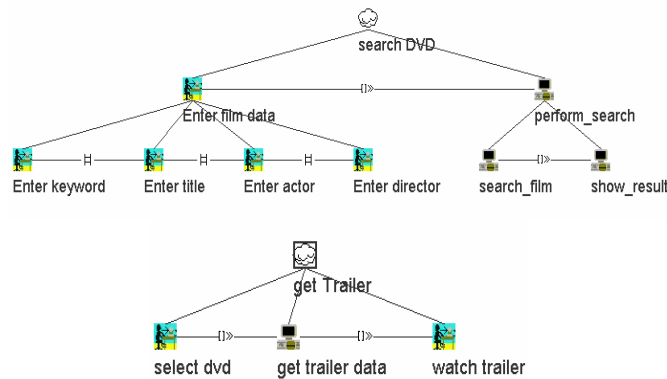
analysis by considering the highest abstraction level (task level, see Section 3.1) and then move to composition options at more concrete levels.

### 3.1 Task Level

In the following subsection we show some examples of composition at the task level by exploiting the ConcurTaskTrees notation.

#### 3.1.1 UI Composition at the Task Level Exploiting the ConcurTaskTrees Language

One example of composition at the task level involves the description of the activities to be performed in order to search for a DVD (using a web service like the Amazon AWSECommerce service) and then obtain the related trailer (using a movie trailer service like YouTube). A possible composition of these two tasks is exploiting the search service in order to find out which DVDs are available in store, allowing the user to watch the trailer of the search results. The starting point is having two separate task models describing the activities for searching for a DVD and getting a trailer (see Figure 2).



**Fig. 2** Task models for accessing the DVD search and watch trailer services.

The composition is performed by specifying the temporal relationship between the two task models: the “*searchDVD*” task enables the “*get Trailer*” task, by passing to it the information associated with the title of the selected DVD. In order to create such composition the designer has to:

1. Define an abstract task (the “*Compos dvd search*” task in Figure 3) that is the root of the composite task model (Step 1);
2. Add the “*search DVD*” task model as sub-task of the root specified at Step 1.
3. Add the “*get Trailer*” model as sub-task of the root specified at Step 1.

4. Connect the two models using the enabling with information passing operator (represented by this symbol: “[>]”)
5. Specify that the object of the search sub-task is used by the “get Trailer” service.

Figure 3 shows the result of the composition.

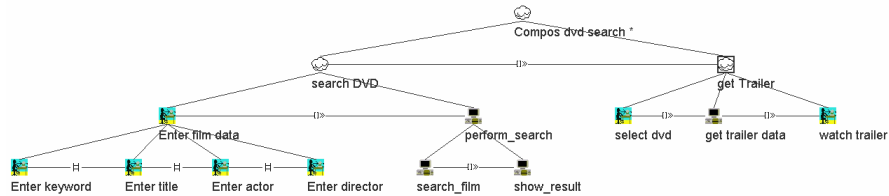


Fig. 3 The composition result.

In the five-dimensional space identified, this example can be represented as shown in Figure 4 (Abstraction level: task; Granularity: groups; UI Aspects: Dynamic aspects; Time/Phase: design time; Web Services: WSDL-based). Indeed, the abstraction level is the task because we are considering tasks for composition, the granularity is at the group level group level because we are composing parts of user interfaces associated with given services (we can suppose that the presentation might contain additional UI elements). Regarding the UI aspects, the composition in this example basically intervenes on dynamic aspects since it indicates that *after* executing the first service, the second one is enabled by receiving the result by the first one. In addition, through the CTT language it is possible to refer to

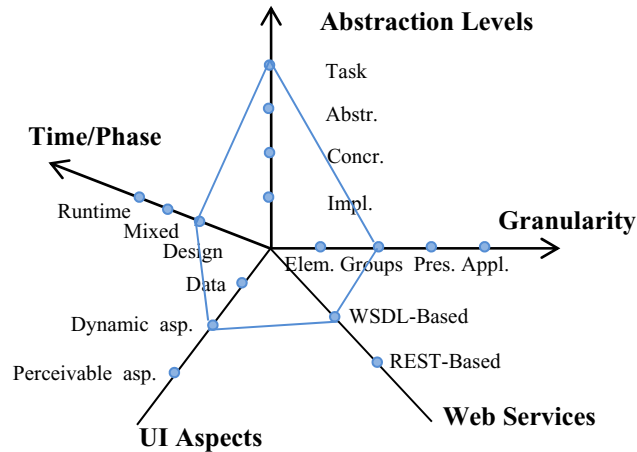


Fig. 4 The considered composition example referred to the problem space.

web services specified through WSDL descriptions, as it has been considered in this composition example. Finally, in the considered example, the composition occurs at design time.

### 3.2 Abstract Level

Figure 5 depicts how MARIA supports the problem space. In terms of abstractions it covers both the abstract and the concrete level. At the abstract level, the UI is described using abstract elements and relationships, independent from a specific platform. Such relationships are *abstract composition operators*, which are provided by the language itself to combine together different interactors or, in turn, other composed expressions. Also the mechanisms used for composing together abstract presentations, by means of *abstract connections*, can be used for composition at this level. In addition, *mechanisms specified in the dialog model (at the abstract level)* can be used to specify compositions occurring within a single abstract presentation. While the first one (composition operators) is a composition that defines a static organization of the interactors, the other two mechanisms define dynamic compositions, which evolve over time. In the following sections we better detail all such different cases, by devoting to each a separate section.

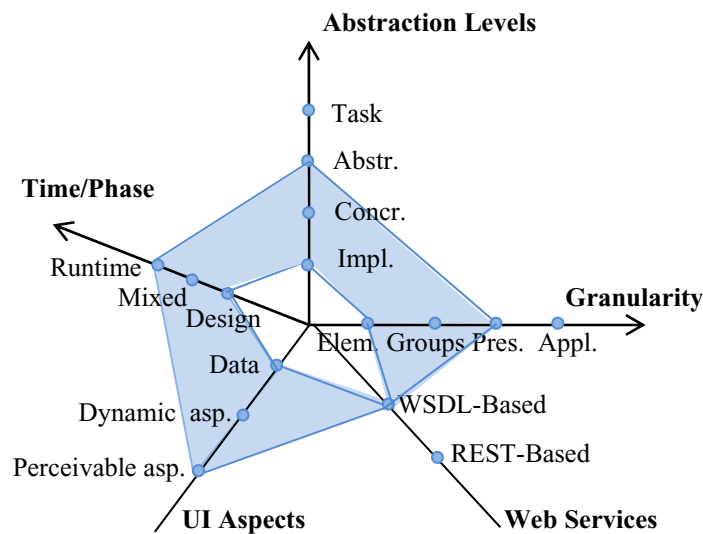


Fig. 5 The problem space supported by MARIA.

#### 3.2.1 Abstract Composition Operators

The composition expressed through the abstract composition operators is a composition that generally affects the presentation aspects of an interface. In MARIA, two composition operators have been identified: relation and grouping.



The composition operators allow for composing together (composition of) interactors. As it is indicated by the name, the objective of the grouping operator is to indicate that some interactors are logically grouped together: this information should be appropriately rendered in the UI. The definition of the Grouping operator in MARIA requires the specification of a number of attributes:

- **continuous\_update.** Boolean type, optional. It is used to specify a continuous update of the content. The technique used for implementing such an update will depend on the specific technology that will be used at the implementation level.
- **data\_list.** String type, optional. In the grouping definition, if the repeat\_contentattribute is true, this attribute is a reference to a list of elements in the data model. For each element in the list, the interactors specified into the grouping will be replicated and bounded to the element fields (according to the data\_reference attribute in the interactors).
- **hidden.** Boolean type, optional. It is True if the interactor is not displayed to the user.
- **hierarchy.** Boolean type, optional. If true the grouped interactors have a hierarchical relation.
- **hierarchy\_value.** String type, optional. Value for hierarchical presentation of the content.
- **id.** NMTOKEN type, required. It is the interactor composition identifier
- **ordering.** Boolean type, optional. If true, the grouped interactors have an ordering relation.
- **update\_function.** String type, optional. The name of the function for a continuous\_update.

A simplified example of grouping expression in the MARIA language is in the following XML-based excerpt in which three UI elements are grouped together: an interactor used for selecting a customer, an activator element for searching the quotations associated with the selected customer, and a description interactor showing the results. A more structured example of use of MARIA XML-based language will be presented later on in the Chapter.

```
<grouping id="Select_customer" hierarchy="false"
[...other attributes...]>
  <single_choice id="Specify_customer_ID" [...]>
    <choice_element value="item/id"/>
  </single_choice>
  <activator id="Search_customer_quotations" [...] />
  <description id="P3_Show_result" [...] />
</grouping>
```

Differently from the Grouping operator, which models a relationship between *several* abstract UI objects (or compositions of them), the *Relation* operator models a relationships between only two expressions of interactors. A typical example of Relation abstract operator is translated at the concrete level (for a graphical platform) in a UI *form*. In this case the two expressions of interactors involved by the Relation are on the one hand, the N fields to be filled by the users, and, on the

other hand, the button (generally labelled with “Send”) for transmitting the corresponding values to the application server. So, the Relation operator supports a N:1 relationship between the N elements appearing in the first expression, and the second expression involved in the composition.

### 3.2.2 Connections

The *connections* are mechanisms for composing together presentations, where a *presentation* is a set of UI elements that can be perceived at a certain time. Differently from what happens for the composition operators presented in the previous sections (which combine elements belonging to the same presentation), the compositions supported by the connections involve different presentations of the UI and generally affect the dynamic behaviour of the UI. A connection is generally defined by specifying a presentation, which is the current presentation, and an interaction element, which is generally a navigator element triggering the activation of another presentation (i.e., the target presentation) when it is selected. At the abstract level, the connections are defined in terms of abstract presentations and abstract interactors.

At the abstract level three types of connections can be specified in MARIA:

- **Elementary connections:** connections linking together two presentations through one abstract interactor (e.g., by activating an interactor of type navigator). Thus, it is possible to move from a source presentation to a target presentation through an interactor.
- **Complex connections:** a more structured connection in which it is possible to specify a set of interactors composed by Boolean expressions able to activate the connection.
- **Conditional connections:** a more flexible mechanisms for describing connections, in which it is possible to specify that, depending on the value currently assumed by a certain parameter, it is possible to move to different presentations. Therefore, a "conditional connection" is a connection that can activate more than one presentation, and the presentation that is finally activated depends on a specific value.

A simplified example of conditional connection in MARIA has the following specification:

```
<conditional_conn id="" interactor_id="" parameter_name="">
  <cond parameter_value="" presentation_to_load=""
    target_composition_operator=""
    target_presentation_name="">{1,unbounded}</cond>
</conditional_conn>
```

As you can see from the above excerpt, a conditional connection can have one or multiple child elements of type *cond*, whose definition has a number of attributes. In addition, the definition of the conditional connection includes some attributes, which are detailed below:

- **id.** Required, it is the unique identifier of the conditional connection

- **interactor\_id.** Required, it is the identifier of the interactor which connects the current presentation with another one (generally refers to a navigator or an activator)
- **parameter\_name.** Required, it is a string with the name of the variable to test for selecting the target presentation

In order to better illustrate how connections work, in the continuation of the Chapter we will provide a more structured example in which they are exploited.

### 3.2.3 Dialog Model –Related Composition Mechanism (Abstract Level)

Another mechanism that can be used for composing UIs is the mechanism defined in the dialog model, which is an abstract model that is used to specify the structure of the dialogue between a user and an interactive computer system. The composition technique associated with the dialogue model involves dynamic aspects of a UI and it is generally used for specifying the dynamic behaviour occurring within a single presentation. Indeed, a presentation does not have just one single state, but it can move between different states depending on a number of events, which represents its dynamic behaviour. Therefore, the compositions intervening at the level of the dialog model allows for composing together over time the interactors that belong to the same presentation, by identifying different *states* within the same presentation.

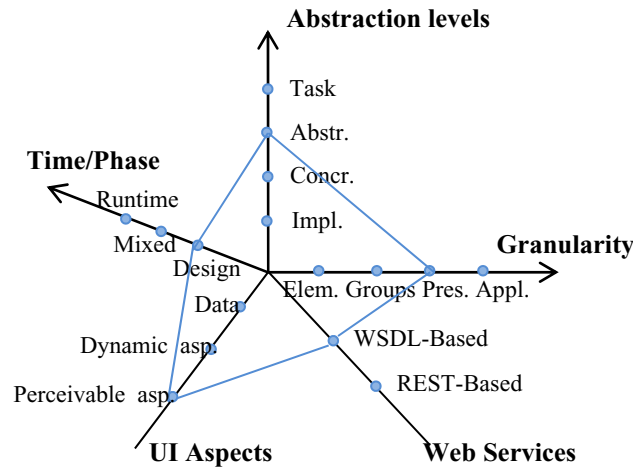
For example, as a consequence of selecting a specific item in a list of cities in a part of a presentation, it can happen that some other UI elements in the presentation are disabled, since they are not possible for the specific item selected, while other elements are enabled. In this case, we have identified a first *state* of the presentation as the one in which a certain subset of interactors are disabled, and another state of the presentation as the one in which the same subset of interactors are enabled as a consequence of performing a certain selection in the UI (city selection). So, in this case the mechanism for dynamically moving from the first state to the second state (in other terms: to compose together the two states or groups of interactors) is the UI element that supports the selection of a specific city. Another simple example would be a touristic web page with a list of cities, where the user selects a city in the countryside, and therefore, in the presentation, the part dedicated to reach this location by boat should be dynamically disabled.

The mechanism modelled by the dialog model is a consequence of the fact that the dynamic behaviour of an abstract presentation can evolve over time depending on some events that occur in the UI. In MARIA the dialogue model is basically an event-based model in which the CTT temporal operators can be used for defining complex dialog expressions. As for the type of actions that can be performed in reaction to a change of state (the so-called *event handlers*), they can be i.e. to enable (or to disable) UI objects, as well as to support further, more flexible behaviour. To this aim, in MARIA some more flexible constructs have been identified as useful for modelling the different activities that can be performed as a consequence of a change of state. For instance, a value change of an interactor attribute, can be supported in MARIA through the *change\_property* attribute. Other activities are: change of values of some elements in the data model, invocation of

external functions, usage of typical programmatic operations on Strings, Numbers, and Boolean values, and the support for well-known programming language constructs like if-then-else and while. Finally, it is worth pointing out that at this level the composition will involve *abstract* events and *abstract* event handlers, (which are associated to *abstract* interactors).

### 3.2.4 Abstract Composition Example

In this section we focus on one example of composition at the abstract level and involving perceivable aspects of interactors belonging to a single presentation. In addition, the example considers composition occurring at design time. Therefore, the five-dimensional element identified by this example (see Figure 6) is: Abstraction level: abstract; Granularity: presentation; UI Aspects: perceivable aspects; Time/Phase: design time; Web Services: WSDL-Based.



**Fig. 6** The composition considered in the example, and referred to the problem space.

In order to provide an example of this kind of composition, we can consider a simple application with which the user interacts in order to search for a DVD within an online DVD database.

The application has three presentations. The first presentation "Presentation\_1" (see excerpt below) provides the UI for specifying the data on which the search will be performed. As you will see in the MARIA excerpt below, in this presentation there are various interactors of type `text_edit` for specifying such information (e.g., title, actor, director, etc.):

```
<presentation name="Presentation_1">
  <grouping id="Specify dvd_search">
    <grouping id="Enter_film_data">
      <text_edit id="Enter_keyword"/>
      <text_edit id="Enter_title"/>
    </grouping>
  </grouping>
</presentation>
```

```

        <text_edit id="Enter_actor"/>
        <text_edit id="Enter_director"/>
        <activator id="search_film_activator"/>
    </grouping>
    <grouping id="perform_search">
        <activator id="show_result_activator"/>
    </grouping>
</grouping>
</presentation>

```

The second presentation shows the result of the search (see the description interactor "show\_result" in the excerpt below). This result shows a list of possible movies satisfying the constraints specified in the search: from this list of results the user can then select (see the single\_choice interactor) a specific movie:

```

<presentation name="Presentation_2">
    <grouping id="Perform_dvd_search">
        <grouping id="perform_search">
            <description id="show_result"/>
            <navigator id="perform_search_navigator"/>
        </grouping>
        <grouping id="choose_dvd">
            <single_choice id="select_dvd"
cardinality="4"/>
        </grouping>
    </grouping>
</presentation>

```

Once the user has selected a specific trailer, in the third presentation the user can obtain the data about the trailer that has been selected by interacting with a navigator interactor.

```

<presentation name="Presentation_3">
    <grouping id="Get_Trailer">
        <navigator id="get_trailer_data_navigator"/>
    </grouping>
</presentation>

```

An example of composition for such UIs is to merge the three presentations into only one presentation. In such combined presentation there will be one part dedicated to entering the search keywords, one part for showing the results, and a remaining part for watching the selected trailer. In terms of the MARIA language, the resulting composition at the abstract level will produce the following result:

```

<presentation name="Composed_presentation">
    <grouping id="Composed_dvd_search">
        <grouping id="Specify_dvd_search">
            <grouping id="Enter_film_data">
                <text_edit id="Enter_keyword"/>
                <text_edit id="Enter_title"/>
            </grouping>
        </grouping>
    </grouping>
</presentation>

```

```

        <text_edit id="Enter_actor"/>
        <text_edit id="Enter_director"/>
        <activator id="search_film_activator"/>
    </grouping>
    <grouping id="perform_search">
        <activator id="show_result_activator"/>
    </grouping>
</grouping>
<grouping id="Perform_dvd_search">
    <grouping id="perform_search">
        <description id="show_result"/>
        <navigator id="perform_search_navigator"/>
    </grouping>
    <grouping id="choose_dvd">
        <single_choice id="select_dvd" cardinality="4"/>
    </grouping>
</grouping>
<grouping id="get_Trailer">
    <navigator id="get_trailer_data_navigator"/>
</grouping>
</grouping>
<presentation>

```

The composition in this case has been exploited through a grouping operator composing the content of the involved presentations.

### 3.3 Concrete Level (*Graphical Desktop Platform*)

At the concrete level, the UI is expressed by referring to concrete elements and relationships. Differently from the abstract level, at this level the UI description refers to a specific *platform*. Therefore, at this level we refer to a number of interaction techniques that are *dependent* on a platform, but are *independent* of a specific implementation language. As an example, while on a vocal platform we use specific sounds for grouping together vocally rendered elements, on a graphical platform we can use a shared background colour for rendering the fact that some elements are grouped together. Then, the concrete composition mechanisms that have been identified are basically the concrete composition operators (which are refinements of grouping and relation abstract operators), the dialog model (defining the dynamic behaviour associated within a single concrete presentation), and the composition mechanisms that can be specified through the connections (expressing the dynamic behaviour between different concrete presentations).

#### 3.3.1 Concrete Composition Operators

At this level the composition mechanisms identified are refinements of the composition techniques identified at the abstract level, using references to a specific platform/modality. For instance, while at the abstract level we refer to abstract

grouping relation between UI-elements, at this level we have to specify which concrete techniques we use on a specific platform/modality in order to render such a grouping relation between the various elements. At the concrete level, the composition operators inherit the attributes that they have at the abstract level, adding further concrete details. In particular, at the concrete level (e.g., the desktop platform) the composition operators have some default\_settings.

For the grouping operator they are the following, modelled as attributes of the grouping\_settings element, as described in the excerpt below:

```
<grouping_settings bullet="" fieldset="" position="">
  <hierarchy_properties
visualization="">{0,1}</hierarchy_properties>
  <ordering_properties position=""
visualization="">{0,1}</ordering_properties>
  <background>{0,1}</background>
</grouping_settings>
```

- **bullet.** Optional, if the grouping is carried out by using bullets
- **fieldset.** Optional, for grouping the elements with a fieldset (yes/no)
- **position.** Required, the element position settings

As for the Relation operator, since this operator is generally used for modelling the relationships occurring between a set of interactors (possibly composed with some composition operators) and one interactor, it has the form as its default supporting mechanism at the concrete desktop level.

### 3.3.2 Connections

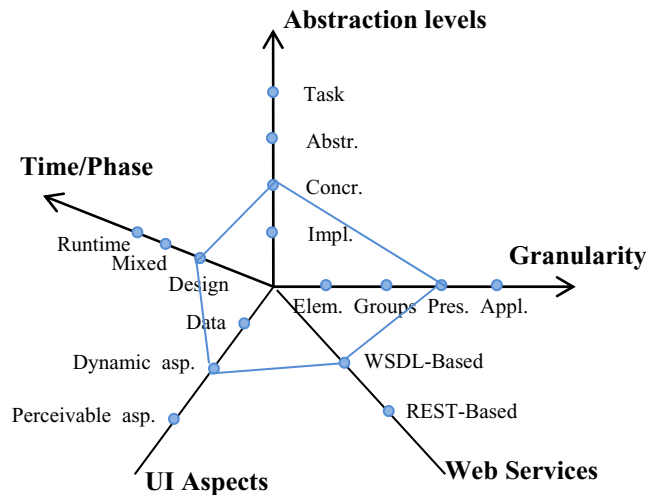
At the concrete level, the connections are defined in the same way as at the abstract level. The only difference is that, at this (concrete) level the specification is done by referring to *concrete* interactors (and concrete presentations), since they are a mechanism that just refines the corresponding elements introduced at the abstract level (abstract connections) by just adding platform-specific details.

### 3.3.3 Dialog Model –Related Mechanism (Concrete Level)

As it happened at the abstract level, also at the concrete level, it is possible to use composition techniques based on the dialogue model, in order to model a dynamic behaviour affecting a single concrete presentation. In order to do this, we use complex expressions of concrete events (associated with concrete interactors), which might trigger the activation of concrete event handlers. The main difference from the dialog model-related mechanisms in the abstract case is that now the composition (occurring in terms of dynamic behaviour) will be defined by means of concrete events (associated with concrete interactors), which trigger the activation of concrete event handlers. For example, in a graphical platform a click event is a refinement of the abstract select event.

### 3.3.4 Concrete Composition Example (Graphical Desktop Platform)

At this level we show an example of composition that involves dynamic aspects. In particular, we show how it is possible to group together three different presentations in such a way that, depending on the value held by a specific parameter that can be selected in the first presentation, different elements are dynamically activated in the other two presentations.



**Fig. 7** The composition considered in the example, and referred to the problem space (concrete level, dynamic aspects, design time, WSDL-based services).

We show how this behaviour can be modelled by using the "conditional connection" structure of MARIA language. Therefore, the five-dimensional element identified by this example is: Abstraction level: concrete; Granularity: presentations; UI Aspects: Dynamic aspects; Time/Phase: Design; Web Services: WSDL-based (see Figure 7). The considered example is drawn from an application that allows the user to present and edit home features. We can consider a first presentation "presentation\_1" (see excerpt below), in which there is a grouping (implemented, in concrete terms, by a fieldset) of some pieces of information (textual elements) referring to a specific apartment are presented. In addition, within this presentation, it is also possible to select the various rooms composing the flat.

```
<presentation name="presentation_1">
  ....
  <grouping id="deviceListGrouping">
    <properties position="row" bullet="false" fieldset="true">
      <background>
        <background_color>#D3D3D3</background_color>
      </background>
    </properties>
    <descriptionid="textRooms">
```



```

<text> <string>Room List:</string>
      <font_settings style="" align="Center" size="20pt"
        color="#FF0000" name="Verdana"/>
</text>
<description id="RoomDescription"><text>
  <string>The apartment is located in a seven-storey
  building situated in the immediate outskirts of the town. It
  consists of five rooms.</string></text>
  <font_settings style="" align="Center" size="20pt"
    color="#FF0000" name="Verdana"/>
</description>
<single_choicedata_reference="ArrayOfRooms"
id="roomSelection" selected="Bedroom" cardinality="5">
  <drop_down_list label="Room selection">
    <choice_element selected="false" value="Dining_room"
      label="Dining room"/>
    <choice_element selected="false" value="Living_room"
      label="Living room"/>
    <choice_element selected="false" value="Kitchen"
      label="Kitchen"/>
    <choice_element selected="true" value="Bedroom"
      label="Bedroom"/>
    <choice_element selected="false" value="Bathroom"
      label="Bathroom"/>
  </drop_down_list>
</single_choice>
</grouping>
  ....
</presentation>

```

In addition to “presentation\_1”, we have other presentations that are devoted to describe the characteristics of each room in the apartment. More specifically, in the example presented, we have a second presentation (“presentation\_2” in the excerpt below) dedicated to showing details of the bedroom, while the third presentation shows details of the bathroom (we do not include its XML specification for sake of brevity).

```

<presentation name="presentation_2">
  ...
<grouping id="bedroomGrouping">
  <properties position="column">
    <background>
      <background_color>#004466</background_color>
      <background_image></background_image>
    </background>
  </properties>
<description id="title Image">
  <image width="100" height="100" horizontal_align="Left"
    alt="Bedroom image" source="img/bedroom.png"/>
</description>
<description id="titleText">
  <text><string>Bedroom</string>

```

```

    <font_settings style="italic" align="Left" size="30pt"
    color="white" name="Arial"/>
    </text>
  </description>
</grouping>
  . . . .
</presentation>

```

The idea is to have a new combined presentation in which, depending on the value that has been chosen between the set of rooms composing the flat, different presentations are activated. In the MARIA language, if we want to compose such presentations, in order to have a new UI with a dynamic behaviour that allows the activation of a specific presentation, depending on the value that is assumed by a specific interactor attribute, we can use the "conditional connection" construct. A "conditional connection" is a connection that can activate more than one presentation, and the presentation that is finally activated depends on a specific value. In the example it is the value assumed by the drop-down list "roomSelection". Thus, if the value, currently selected, is "Bedroom", the "presentation\_2" will be activated, and if the selected value is "Bathroom", the "presentation\_3" will be activated.

Below is the related MARIA excerpt specifying the result of composition through conditional connection in "presentation\_1" (only the part related to the connection specification has been described):

```

<presentation name="presentation_1">
  <connections>
    <conditional_conn id="id1"interactor_id="roomSelection"
    parameter_name="roomSelection/selected">
      <cond parameter_value="Bedroom"
        target_presentation_name="presentation_2"
        target_composition_operator="target_composition_operator1"
        presentation_to_load="presentation_1"/>
      <cond parameter_value="Bathroom"
        target_presentation_name="presentation_3"/>
    </conditional_conn>
  </connections>
<grouping id="deviceListGrouping">
  [ definition of deviceListGrouping here ]
</grouping>


```

### 3.3.5 Example of Composition of Services at the Concrete Level

In this section we show a possible composition of services at the concrete level. In particular, we consider the case when there is a first Web service used by a last minute tour agency, which, depending on the current time and location, delivers a list of cheap return flights (from the current location). We suppose that the user is currently in the area of London (Gatwick) and the service displays the flights currently available and departing from that airport from that moment onward, showing them in ascending order, from the cheapest flight to the most expensive one. A visualisation of this service (for the desktop platform) is shown in Figure 8.

Price	Time/Date	To	Flight Number	Class
£ 291	09:35 9 Oct	Madrid	OP1456	Economy
	6:55 15 Oct	London Gatwick	OP1453	Economy
£ 295	12:30 9 Oct	Barcelona	TR1235	Economy
	6:55 15 Oct	London Gatwick	TR1665	Economy
£ 300	22:40 9 Oct	Barcelona	AA1466	Economy
	6:55 14 Oct	London Gatwick	AA1216	Economy
£ 350	10:35 8 Oct	Paris	PP4452	Economy
	6:55 15 Oct	London Gatwick	PP4452	Economy

**Fig. 8** A possible graphical rendering of the UI for the first service providing information on flights (desktop platform).




**Hotel Rex** ★★★

*Gran Via, 43, 01\_Centro, Madrid*

Centrally located, this elegant, traditional-style hotel provides a great base to visit Madrid's many cultural sights. It lies on Gran Vía, between the main squares of España and Callao.... [More](#)

[Show rates](#)

---




**Il Castillas Madrid** ★★★

*Abada, 7, 01\_Centro, Madrid*

The Hotel Il Castillas has a fantastic central location, just 330 yards from Madrid's Puerta del Sol. It offers traditional style with modern facilities, including free Wi-Fi access.... [More](#)

[Show rates](#)

---




**Ganivet** ★★★

*Toledo, 111, 01\_Centro, Madrid*

Hotel Ganivet has an ideal location in Madrid's La Latina district, a 10-minute walk from the Plaza Mayor. It offers good-value accommodation with free Wi-Fi access.... [More](#)

[Show rates](#)

---



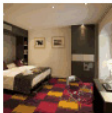
**Medium Cortezo** ★★★

*Doctor Cortezo, 3, 01\_Centro, Madrid*

The Medium Cortezo is set in the heart of Madrid, 800 metres from Atocha Train Station. It offers free Wi-Fi and 24-hour reception.... [More](#)

[Show rates](#)

---




**Petit Palace Arenal Sol** ★★★

*Arenal, 16, 01\_Centro, Madrid*

Petit Palace Arenal is set in a pedestrian street by the Puerta del Sol. All rooms come with a flat-screen TV, a hydromassage shower and free Wi-Fi access.... [More](#)

[Show rates](#)

---



**Suites Kris Aeropuerto** ★★★

*Campezo, 8, 20\_San Blas, Madrid*

The Suites Kris offers a 24-hour free shuttle service to nearby Barajas Airport and free transfer to IFEMA. It has a garden and a swimming pool, open seasonally.... [More](#)

[Show rates](#)

**Fig. 9** A possible graphical rendering of the UI for the second service providing information on hotels (desktop platform).

The area given in input to this first service can vary, and can be specified, e.g., depending on the current position of the mobile user. In this case, a GPS receiver could provide this information. Then, the first service might be defined likewise:  $flight\_info=getFlightInfo(area)$ , where  $getFlightInfo$  is the name of the service,  $area$  is the input parameter passed to the service, and  $flight\_info$  is the output value returned by the service itself (it includes the city destination and departure/arrival times). Another service is also available, which, having in input a destination city and two dates (arrival/departure date), offers a list of available hotels for the considered period, also providing a visualization of each available hotel in a graphical map. The functionality of this second service can then be summarised with the following function:  $hotels\_info=provide\_hotels\_info(destination\_city, arrival\_date, departure\_date)$ , where  $destination\_city$  is the city where to find an accommodation, while  $arrival\_date$  and  $departure\_date$  are the dates that identify the period during which the user will stay in that city.

This service delivers in output  $hotels\_info$ , which is information about available hotels including name, address, rates, and also it provides a visualization of this hotel in a graphical map. A possible graphical rendering for this second service (for the graphical desktop platform) is visualised in Figure 9. These two UI services can be composed together on temporal aspects. Indeed, after the user selects a particular return flight in which s/he is interested from the currently available ones (depending on associated dates and destination selected), the second service will show the list of possible accommodation options in a graphical map. The composition of such services in the mobile device produces two presentations, which are displayed in Figures 10-11 below.

Price	Time/Date	To	Flight Number	Class
£ 291 ●	09:35 9 Oct	Madrid	OP1456	Economy
	6:55 15 Oct	London Gatwick	OP1453	Economy
£ 295 ●	12:30 9 Oct	Barcelona	TR1235	Economy
	6:55 15 Oct	London Gatwick	TR1665	Economy
£ 300 ●	22:40 9 Oct	Barcelona	AA1466	Economy
	6:55 14 Oct	London Gatwick	AA1216	Economy
£ 350 ●	10:35 8 Oct	Paris	PP4452	Economy
	6:55 15 Oct	London Gatwick	PP4452	Economy

**Fig. 10** First presentation (mobile device) of the UI resulting from composing the two services.

More specifically, Figure 10 shows how the user can select the flight in which s/he is interested. Additionally, as soon as the user selects a particular hotel, this action triggers the visualization of a second presentation (visualized in Figure 11) in which the possible accommodation options are visualized. Depending on the

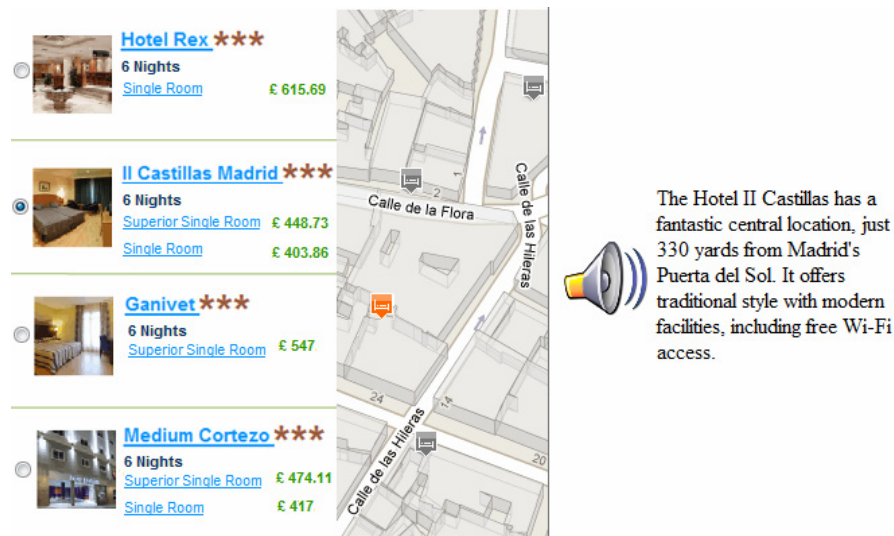


Fig. 11 Second presentation (mobile device) of the UI resulting from the composition

option selected by the user, the corresponding hotel is visualised in the graphical map, and its position will be highlighted in it by using an icon with a colour different from the one used for the other icons, to distinguish it from the other hotels in the same area. Also, in the mobile device, an adaptation step will generate an UI (see Figure 11) in which only a part of the information available for each hotel is immediately visible on the screen. Indeed, when the user selects a specific hotel, the application presents the overall description by displaying the picture and a part of the information in a textual manner, while the remaining information is vocally rendered.

As far as our problem space is concerned, in this case the composition is carried out at the concrete level, involving groups of UI objects. Indeed, in this case we are composing services, which can even provide data for only some parts of a presentation, and for this reason we consider it as a composition involving groups of objects. In addition, the composition affects temporal aspects, since, after the user selects a particular item in the list different events are triggered in the other parts of the UI. The map shown in the right part of the UI changes its appearance since the selected item is highlighted with a different colour, and a vocal rendering of the remaining information about the selected hotel starts.

Therefore, the five-dimensional element identified by this example (see Figure 12) is: Abstraction level: Concrete; Granularity: groups of UI objects; UI Aspects: Dynamic aspects; Time/Phase: Runtime; Web Services: WSDL-Based.

The composition is done at runtime since we suppose that the service that provides information on hotels is not statically determined but it dynamically changes depending on the current position of the user. Regarding the level of composition, it occurs at the concrete level; the granularity is that of groups of UI objects, and



Future work will be dedicated to further analysis of the proposed design space and to evaluating its validity and generality in identifying the possible situations that can occur in user interface composition.

**Acknowledgments.** This work has been supported by the ICT EU ServFace STREP project (<http://www.servface.eu>).

## References

- [1] Agrawal, A., Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., König, D., Leymann, F., Müller, R., Pfau, G., Plösser, K., Rangaswamy, R., Rickayzen, A., Rowley, M., Schmidt, P., Trickovic, I., Yiu, A., Zeller, M.: Web Services Extension for People (BPEL4People), Version 1.0 (June 2007)
- [2] Agrawal, A., Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., König, D., Leymann, F., Müller, R., Pfau, G., Plösser, K., Rangaswamy, R., Rickayzen, A., Rowley, M., Schmidt, P., Trickovic, I., Yiu, A., Zeller, M.: Web Services Human Task (WS-HumanTask), Version 1.0 (June 2007)
- [3] Calvary, G., Coutaz, J., Bouillon, L., Florins, M., Limbourg, Q., Marucci, L., Paternò, F., Santoro, C., Souchon, N., Thevenin, D., Vanderdonckt, J.: The CAMELEON reference framework. CAMELEON Project.Deliverable 1.1 (2002), <http://giove.isti.cnr.it/projects/cameleon/pdf/CAMELEON%20D1.1RefFramework.pdf>
- [4] El bekai, A., Rossiter, N.: A Tree Based Algebra Framework for XML Data Systems. In: Proceedings of the Seventh International Conference on Enterprise Information Systems, ICEIS 2005, Miami, USA, May 25-28 (2005)
- [5] Jagadish, H.V., Laks, V., Lakshmanan, S., Srivastava, D., Thompson, K.: TAX A Tree Algebra for XML. In: Proceedings of DBPL Conf. (2001)
- [6] Janeiro, J., Preußner, A., Springer, T., Schill, A., Wauer, M.: Improving the Development of Service-Based Applications through Service Annotations. In: Proceedings of IADIS WWW/Internet (2009)
- [7] Lepreux, S., Vanderdonckt, J., Michotte, B.: Visual Design of User Interfaces by (De)composition. In: DSV-IS 2006, pp. 157–170 (2006)
- [8] Mori, G., Paternò, F., Santoro, C.: CTTE: Support for Developing and Analysing Task Models for Interactive System Design. *IEEE Transactions on Software Engineering* 28(8), 797–813 (2002)
- [9] Oasis standard: Web Service Business Process Execution Language (April 2007)
- [10] Paternò, F.: Model-Based Design and Evaluation of Inter-active Applications. Springer, Heidelberg (2000)
- [11] Paternò, S.C., Spano, L.D.: MARIA: A Universal Language for Service-Oriented Applications in Ubiquitous Environment. *ACM Transactions on Computer-Human Interaction* 16(4), 19:1–19:30 (2009)
- [12] Pautasso, C., Zimmermann, O., Leymann, F.: RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision. In: Proc. of the 17th International World Wide Web Conference (WWW 2008), Beijing, China (April 2008)
- [13] W3C, Web Services Glossary (2004), <http://www.w3.org/TR/ws-gloss/>