# Dandelion: Decoupled Distributed User Interfaces in the HI$^3$ Ambient Intelligence Platform

Gervasio Varela, Alejandro Paz-Lopez, Jose A. Becerra, Richard J. Duro

Integrated Group for Engineering Research, University of A Coruña, Ferrol, Spain
`{gervasio.varela, alpaz, ronin, richard}@udc.es`

**Abstract.** Ambient Intelligence and Ubiquitous Computing systems must deal with a wide variety of environments, users and devices. Designing and implementing their interaction systems is quite a complicated task because it is difficult to know in advance the conditions in which the system will be run. This article presents Dandelion, a framework that uses a model-driven approach to support the development of user interfaces for UC and AmI systems by defining a series of high-level declarative models. It decouples the application logic from the interaction elements, which can be physically distributed throughout the environment, and even changed dynamically.

**Keywords.** hci, ambient-intelligence, ubiquitous-computing, distributed-ui

## 1 Introduction

Ubiquitous Computing (UC) and Ambient Intelligence (AmI) systems rely heavily on the information they can extract from the environment and users. This, in turn, makes these systems highly dependent on their capacity to use the available devices to interact with the users and the environment, forcing developers of UC and AmI systems to deal with a wide variety of environments, users and devices in order to cover as many scenarios as possible. This introduces a great level of complexity in AmI and UC systems. For example, it is complicated to predict the kind of devices that will be available for interaction. The environment conditions, like visibility or noise, may also change and they may impact the operation of the interaction system. Furthermore, the users are also an important source of heterogeneity. They have different abilities, different preferences, even the number of users may vary.

During the last few years we have been working in the development of the HI$^3$ architecture [1], which provides a common framework and runtime platform for the development of AmI and UC systems, and on UniDA [2] that provides homogeneous access to a network of heterogeneous devices. Supported by these solutions, we are building Dandelion, a framework for the development of user interfaces for AmI and UC systems. This paper presents a solution for building distributed UIs within Dandelion. It uses a model-driven approach that allows developers to build user interfaces by defining a series of high-level declarative models. These models provide abstract

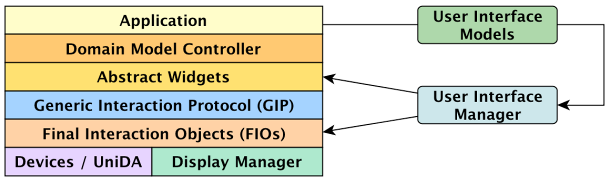

**Fig. 1.** Architecture diagram of the Dandelion distributed-adaptive user interface solution.

components, which, at runtime, are transparently and dynamically connected to end devices and user interaction software.

In the next section, an overview of the Dandelion and its underlying supporting software architecture is given. Section 3 presents a brief analysis of the main contributions of Dandelion. Finally, in section 4 some conclusions are extracted.

## 2 Dandelion decoupled distributed user interfaces

Inside the HI$^3$ platform, Dandelion is integrated within the application layer, and makes use of the capabilities provided by the HI$^3$ low-level layers to free developers from many of the complexities of interacting with the environment and users. It uses a model-driven approach in which a series of models are used to build, at runtime, a user interface that is appropriate for the proposed models and the devices available in each environment.

Fig. 1 shows the main components of the Dandelion system. The system is designed to support a series of user interface models that describe the application domain, its interaction requirements, the user and the environment. The information available in those models is used by the User Interface Manager (UIM) to select the interaction elements (FIOs), among those available in the environment, that better suit the requirements of the application, the user and the environment. The applications are connected to the FIOs through a set of abstract interaction units (Abstract Widgets) that provide a generic vision of any interaction technology. This connection, which is distributed and dynamic, is managed by the UIM, and it can be modified at runtime without impact on the application.

The application and its models are the only elements provided and known by the developers. In the current implementation only the Abstract User Interface model (AUIm) and the domain model are used. The AUIm is implemented using the UsiXML [4] user interface definition language, which introduces the concept of Abstract Interaction Unit (AIU) as an abstract representation of any interaction between a human and a computer. These abstract elements are realized by the Final Interaction Objects, which provide the real user interaction workforce of the system. They are elements capable of interacting with a user, representing physical devices, GUI components, gesture, voice recognition software, etc.

FIOs are implemented as distributed HI$^3$ services that comply with a specific communication interface called Generic Interaction Protocol (GIP). It is an agent

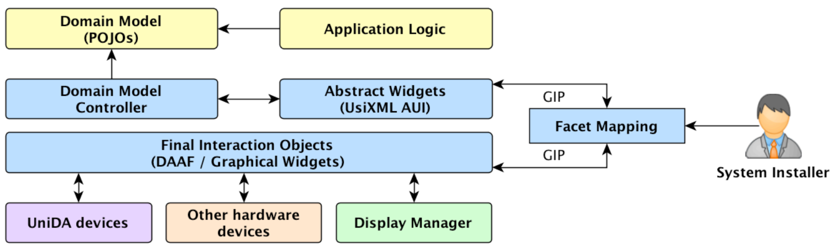

**Fig. 2.** User interface decoupling through facet mapping.

communication protocol that supports the interaction actions proposed by the AUIm model of UsiXML: *Input*, *Output*, *Selection*, *Navigation*, and *Control* triggering.

Looking at Fig. 2 and bearing in mind this architecture, the operation of the Dandelion system can be summarized as:

1. The Developers provide the application logic and a series of models that specify meta-information about the application. Currently two UsiXML models are supported, the Abstract User Interface model (AUIm) and the Domain model.
2. When an application is started, its models are sent to a UIM (Interface Manager) instance. It builds an abstract UI using the Abstract Widgets toolkit, creates a new Domain Model Controller, and connects the application logic to the AUI by linking domain objects with Abstract Widgets defined in the AUIm.
3. The UIM selects among the available interaction resources (FIOs) those that better match the Abstract Widgets used in the AUI. Currently the system installer specifies this selection manually in an XML configuration file.
4. The selected FIOs are connected to their corresponding Abstract Widgets, which can talk remotely to the FIOs using the Generic Interaction Protocol (GIP). This protocol supports the generic interaction actions defined by the AUIm.
5. When a change is made to a domain object, the Domain Model Controller sends the change to its associated Abstract Widget, which redirects it to a FIO using the GIP. The FIO contains the concrete logic necessary to interact with the user through a hardware device or a graphical interface.

As can be seen, the developer is completely decoupled from the final devices and technologies used for the user interface. Moreover, the user interface can be changed dynamically, remotely deployed and physically distributed in the environment.

## 3 Dandelion and model-driven approaches for UI development

The application of model-driven approaches to UI development is an old goal of the UI research community. A prominent example related to distributed UIs is the Cameleon-RT framework [3], which proposes a conceptual reference framework for the development of distributed multimodal UIs. It defines a set of key features that must be supported by this kind of systems, and a set of models required to build multimodal and distributed UIs.

Following the principles of the Cameleon framework, UsiXML [4] provides a user interface definition language that supports the different models proposed by the Cameleon framework. It also provides methods and tools to build user interfaces by progressively transforming models, at design time, from higher levels of abstraction to the final design of the UI.

More specifically related to ubiquitous computing is the MASP [5] project. It follows the specifications of the Cameleon framework, but it uses the models at runtime, using the information to build the UI interface using the local available resources.

Dandelion uses an approach similar to MASP. It follows the guidelines established by the Cameleon framework and uses the models at runtime. Nevertheless MASP requires developers to provide a Concrete User Interface (CUI) specifying the modalities of the user interaction, while in Dandelion, that selection is performed by the system, or, in the current implementation, the system administrator.

The final objective of Dandelion is to provide a UI development solution for AmI and UC system where the developer is abstracted from the final shape of the UI, so that the system can be able to adapt the interaction to any technologies or situations.

The selection of modalities at runtime will allow for more flexible UIs, as they can be more adapted to the current context by selecting those available FIOs which use modalities more adequate to the current users or environment state. In the current implementation, the selection of FIOs, and therefore, the selection of modalities and the final UI, is responsibility of the system administrator. He knows better the users and the environment than the developers and can perform a better selection.

## 4 Conclusions

This article has presented the first stages of development of the Dandelion framework, a system that leverages the advances of model-driven approaches in order to decouple AmI and UC application logic from its user interaction system.

Models allow developers to design the user interaction decoupled from the application logic, the environment state and user's conditions. This characteristic will allow applications to dynamically change and adapt their user interface more easily.

## 5 References

1. Paz-Lopez, A. et al: Some Issues and Extensions of JADE to Cope with Multi-agent Operation in the Context of Ambient Intelligence. PAAMS, 607-614, Salamanca, Spain (2010).
2. Varela G. et al: UniDA: Uniform Device Access Framework for Human Interaction Environments. Sensors 11 (10), 9361–9392, Basel, Switzerland (2011).
3. Balme, L. et al: Cameleon-rt: A software architecture reference model for distributed, migratable, and plastic user interfaces. EUSAI, Eindhoven, The Netherlands (2004).
4. Vanderdonckt, J. et al: UsiXML - User interface extensible markup language. Reference manual. Université catholique de Louvain (2007).
5. Blumendorf, M., Lehmann, G., Albayrak, S.: Bridging models and systems at runtime to build adaptive user interfaces. 2nd ACM SIGCHI symposium on Engineering interactive computing systems - EICS'10, New York, New York, USA: ACM Press (2010).