Decoupled Distributed User Interface Development Framework for Ambient Intelligence Systems

G. Varela, A. Paz-Lopez, J. A. Becerra , R. J. Duro
Integrated Group for Engineering Research, University of A Coruña
C/ Mendizábal S/N, 15403, Ferrol, A Coruña, Spain
gervasio.varela@udc.es

ABSTRACT
Ambient Intelligence (AmI) systems are required to be adapted to each scenario and use the available devices to interact with the user. Because of the variability of devices and the diversity of scenarios, they must support a wide number of devices and technologies, thus introducing a lot of complexity in the implementation of the UI.

This paper proposes an AmI UI development framework called Dandelion. It allows the development of highly portable AmI UIs by allowing developers to implement them using high level models that are transformed, at deploy time, into a selection of end devices. This transformation is driven by a novel device abstraction technology that encapsulates devices behind a generic set of distributed user interaction actions, isolating the devices, conceptually and physically, from the system's logic.

Author Keywords
User Interfaces; Ambient Intelligence; Distributed User Interfaces; Hardware Abstraction.

ACM Classification Keywords
D2.2 [**Software Engineering**]: Design Tools and Techniques – *User interfaces.* H5. [**Information interfaces and presentation**]: User Interfaces – *Interaction styles, Input devices and strategies, user interface management system (UIMS).*

General Terms
Human Factors; Design.

INTRODUCTION

Every Ambient Intelligence (AmI) system aims to integrate itself in the daily life of its users, providing its functionality in an unobtrusive and natural way. Two features are critical to achieve this goal [1, 2]: rely on context information to adapt the behavior of the system to the specific characteristics of the environment and users, and enrich the environment with devices, taking advantage of them to interact with the user and the environment itself.

DUI 2013: 3rd Workshop on Distributed User Interfaces: Models, Methods and Tools. In conjunction with ACM EICS 2013
June 24th, 2013. London, UK.
ISBN-10: 84-616-4792-0
ISBN-13: 978-84-616-4792-7

This last feature makes AmI UIs very dependent on their ability to use the sensing, actuation and appliance devices available in an environment [2]. These devices can vary considerably from one scenario to another; they may have very different natures, technologies, and paradigms of interaction with the user. Support this wide variety of devices in a system would be quite complicated. For this reason, many AmI systems are built for a specific set of devices in a particular environment and, therefore, they are very hard to deploy in a different scenario.

The work presented in this paper aims to alleviate this problem by providing an UI development framework that, relying on model-driven engineering techniques and distributed hardware abstraction technologies, allows the development of UIs decoupled from the technologies and locations of the devices chosen to interact with the user. This framework, called Dandelion, has two main components. A model-driven UI management system allows the description of the UI using high level models, whose abstract elements are connected, at deploy time, with a selection of distributed devices. This connection is implemented by the Generic Interaction Protocol (GIP), a device abstraction technology that, implemented as a distributed agent communication protocol, encapsulates the specific behavior of sensor and appliance devices behind a generic interface of distributed interaction actions.

Model-driven approaches have been very successfully within the Human-Computer-Interaction (HCI) community since Thevenin and Coutaz [4] proposed a model-driven approach to support UI adaptation to context changes. Many authors in the field of multiplatform UI development have used similar approaches [5, 6, 7], and it has also been used in Ubiquitous Computing (UC) and Ambient Assisted Living (AAL) systems [7, 8]; which are closely related to the AmI field. Nevertheless, those works have been mainly focused in the adaptation of graphical user interfaces to multiple platforms and displays. The work presented in this paper takes a novel approach by abstracting sensor and appliance devices behind a common set of interaction actions to build AmI UIs that are independent of the specific hardware devices.

This paper is organized as follows. Section 2 provides an overview of the Dandelion system. In section 3, the GIP is presented in detail. Section 4 and 5 are dedicated to the description of other important subsystems of Dandelion.

Section 6 shows a brief use case example. Finally some conclusions are extracted.

OVERVIEW OF DANDELION

Dandelion is being implemented as the UI development framework of the HI^3 general purpose AmI platform [9]. It follows a model-driven approach for distributed UIs inspired by Cameleon-RT [5], and it is integrated vertically in the different layers of the HI^3 architecture, which are described in detail in [9].

Dandelion can be divided into two main blocks. On one side, a model-driven UI management system allows developers to define the UI at an abstract level using the UsiXML abstract UI model [6]. On the other, the GIP provides a distributed communication interface that abstracts devices behind a generic set of interaction actions, allowing a decoupled connection of the abstract UI definition to a set of distributed elements that perform the real interaction with the user. Figure 1 shows a block diagram with the different components implementing these two blocks. The model-driven UI management system is implemented by the abstract UI model and the Application Controller (AC), which manages the connection between the abstract UI definition and the devices. The abstraction of devices is conceptually implemented by the GIP and physically realized by the Final Interaction Objects (FIOs), which provide software abstractions of devices by implementing the GIP interface.

Following the Figure 1, from top to bottom; the application logic and the UI description models are the only elements provided by the developer. She describes the UI using the UsiXML Abstract UI model, and then connects the application data and actions to that description. The AC manages this connection by storing a set of mappings between the data objects, the elements of the model and the real devices that will be used to interact with the user. Those real devices are represented in the system by the Final Interaction Objects (FIOs). They encapsulate the specific logic, characteristics and particularities of each device behind the GIP interface.

The physical connection between the AC and the FIOs is decoupled by using the GIP. The AC monitors the application, and when it detects a change in the data model associated to the UI model, it sends a GIP event to the FIOs mapped to the corresponding UI element. Conversely, when the AC receives a GIP event from a FIO, it uses the mapping to know which application data object or action needs to be updated. This process is transparent to the application and the developer.

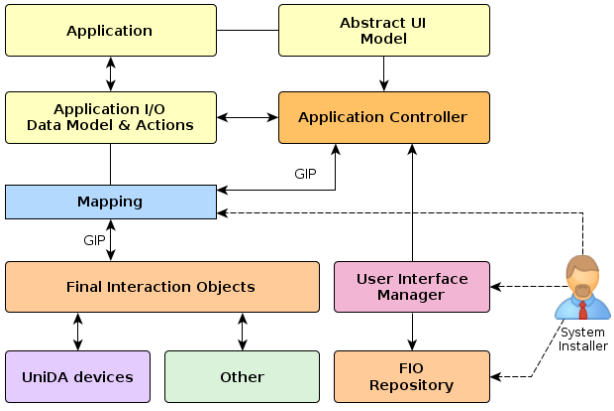

Figure 1. Dandelion system component diagram

The mappings between elements described in the abstract UI model and the FIOs are stored by the AC, but they are managed by the User Interface Manager (UIM). While Dandelion is designed to support the autonomous selection of FIOs at runtime, in the current version this mapping is performed manually by the system installer; who at deployment time selects out of the available FIOs those that better suit the requirements of the application, the user and the environment.

In the next sections the AC, the GIP, the FIOs and their relationships are explored in detail.

GENERIC INTERACTION PROTOCOL

The Generic Interaction Protocol, GIP, is one of the main contributions of this work. It provides a new level of abstraction for distributed hardware devices, encapsulating their specific behaviors behind a generic interface of user interaction operations.

The GIP defines a reduced set of interaction actions which provide a generic remote interface to any kind of interaction device. By implementing this interface, any device can be accessed using the same set of concepts and operations, thus decoupling the application from the underlying interaction technologies and the location of the devices.

The GIP has been designed as an event based distributed protocol following a publisher/subscriber model. A series of publishers, the FIOs, publish events notifying actions that the user has performed (input/selection of data, activation of an action) and a series of subscribers receive those events and react accordingly, the ACs of the different applications. The GIP has been implemented as a distributed agent protocol using the publisher/subscriber capabilities of the HI^3 platform [9]. Figure 2 provides a good overview of the GIP and the relationship to the AC and the FIOs.

The generic interface provided by the GIP defines five events inspired by the set of abstract interaction units (AIUs) used in the UsiXML abstract UI model to describe the UI interaction requirements at an abstract level:

- *output*: an AC requests FIOs to perform data output
- *focus*: an AC requests a given FIO to gain focus over the user attention
- *selection*: an AC requests FIOs to show a selection of data; or a FIO informs an AC about the selection of a user
- *input*: a FIO informs every subscribed AC that the user has performed a data input action
- *action*: a FIO informs every subscribed AC that the user has activated an action

Every GIP event has associated a set of data properties which are the data a FIO is able to either output to the user, or gather from her as input. These data is represented as a string and has an associated basic type (integer, double, byte or string) to it so that Dandelion can know what kind of information a FIO is able to represent.

In order to allow some level of customization of the user interface, GIP events are enhanced with a set of properties called Interaction Hints (IH). They are a set of fixed properties that developers can use to provide indications to the FIOs about an interaction action (ex. priority, size, color). The support for IHs is not mandatory, and each FIO can perform them as it wants.

USER INTERFACE DEFINITION AND MANAGEMENT

Dandelion allows developers to describe the UI of their applications using the abstract UI model of UsiXML [6], a user interface description language (UIDL) for the description of different aspects of an UI.

The abstract UI model introduces the concept of Abstract Interaction Unit (AIU) as a representation of the typical widgets found in graphical user interface toolkits. In Dandelion this concept represents any kind of interaction element, such as physical devices like appliances or sensors, or even gestures or GUIs. Using the abstract UI model, a developer describes the application UI as a collection of interrelated AIUs representing the different high-level user interactions (input, output, action, and selection) required by an application and how they relate to each other (containers). The GIP interface is designed to match this set of generic interactions so that it is easier to connect an AIU to a FIO.

Apart from specifying the UI at an abstract level, the developer must provide a set of data and action objects that will be used by Dandelion as the I/O interface between the application logic and the UI. The developer associates every one of these objects to an AIU and, as shown in Figure 2, the AC uses the observer pattern to monitor those objects, detect changes, and send the data to the FIOs, and vice versa. When a GIP event is received by the AC, it looks for its associated AIU, and uses a callback to notify the change to the application logic. This last association is

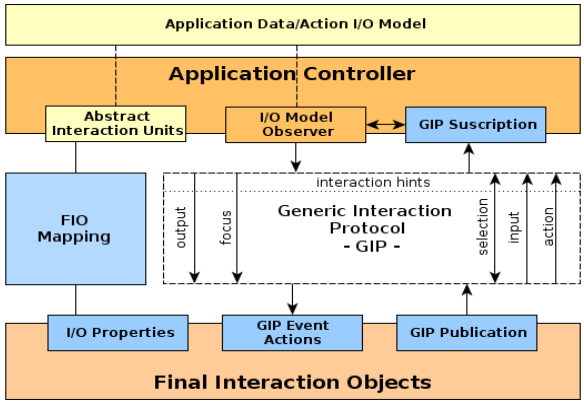

Figure 2. Relationships between the AC, GIP and FIOs

managed by the system installer, who manually connects the AIUs to the GIP properties of the concrete FIOs.

The combination of the abstract UI model with the usage of the observer pattern to drive the connection between the application and the UI, allows a great level of decoupling between the application business logic and the user interface. Nevertheless, this generic description of the UI and its behavior can impact the capacity of UI customization that can be achieved, which is alleviated to some extend by using the GIP interaction hints.

FINAL INTERACTION OBJECTS

The Final Interaction Objects are the end elements in charge of physically interacting with the user. They are software abstractions of heterogeneous interaction resources that could be either hardware (appliances, sensors, etc.) or software (GUIs, voice recognition, etc.).

FIOs are implemented as agents realizing the GIP interface in the sensing/actuation layer of HI^3. They assume the role of GIP publishers, abstracting the behavior of a device as a set of events notifying the different actions the user is performing. Each FIO implementation is only required to support a subset of GIP events because there can be devices that only support input or output. The supported GIP events and the data properties a FIO can input/out are specified in a FIO description, which can be consulted by the system installer using the FIO repository. See Figure 1.

For every different kind of device or interaction software used by an application, a FIO must exist that abstracts them using the GIP. Obviously, a key point is that developers should not need to develop their FIOs, or at least, not many of them. They should be provided by Dandelion itself or by the manufacturers of the interaction resources.

In order to alleviate the problem of developing FIOs for the wide number of different devices and technologies available, Dandelion makes use of the hardware abstraction layer of HI^3, UniDA [4], which provides a generic interface to remotely access and use any kind of hardware device. In UniDA every device is accessed using the same generic operations and concepts, and each type of device is reduced to a set of common operations. It is consequently, possible to use similar devices from different manufacturers or technologies using the same exact API. This way, one FIO can support a wide number of physical devices.

EXAMPLE USE CASE

To briefly illustrate how Dandelion is used to build AmI UIs, a small real world example has been implemented. It is a notification system for a home assistant application, which would be in charge of notifying events, alarms or messages to the users.

The UI of the system is a fairly simple one. It only requires an output element to show the message or notification, and an input one to discard the notification. In a classical PC system it could be a simple notification dialog with a label

and a button. Figure 4 shows how this simple interface could be described using UsiXML. It only requires an output AIU and a trigger AIU that are related inside a container using a compound AIU.

The association between this abstract UI and the AC is achieved by associating a string object to the AIU with id '2' and a callback action to the AIU with id '3'. This is done programmatically in the application.

Our system needs now some FIOs that implement the real interaction with the user. In order to illustrate how this UI could be realized in different environments, a small set of different FIOs has been implemented:

- *output*: colored lights using UniDA
- *output*: An overlay GUI display in a TV set
- *output*: voice synthesizing using Festival
- *input*: a light switch connected to a KNX home automation network using UniDA
- *input*: a hand gesture in a Kinnect like device

The three output FIOs provide one I/O string property indicating that they are able to present a string to the user. Obviously, they will transmit the message in different ways. The colored light device will only notify the presence of a message by powering on a colored light, while the TV display will show the complete message. The input FIOs only provide Action GIP events to notify when the user activates the switch or makes a specified gesture with a hand.

As shown in Figure 3 two different scenarios have been contemplated and, depending on the scenario, the system installer would associate the AIUs to each of the different available FIOs. For example, in the home of a deaf user, the output AIU could be connected simultaneously to colored light FIOs and to a TV FIO. When the application output a notification, the lights will power on, so that the user knows that there is a notification, and can go to the living room to read it on the TV and use the light switch to discard it.

CONCLUSIONS

This paper has shown how the combination of model-driven engineering techniques with distributed device abstraction technologies like the GIP and UniDA establishes a promising approach to allow the development of AmI systems decoupled from the physical location and particular complexities of interaction devices.

```
<AbstractUIModel>
  <AbstractCompoundIU id="1" shortLabel="Notification Dialog">
    <AbstractDataIU id="2" shortLabel="Notification Label">
        <AbstractDataItem\>
        <AbstactOutputIU\>
    <\AbstractDataIU>
    <AbstractTriggeIU id="3" shortLabel="Discard action">
        <AbstractOperationIU\>
    </AbstractTriggerIU>
  </AbstractCompoundIU>
</AbstractUIModel>
```

Figure 4. Description of the example UI with UsiXML

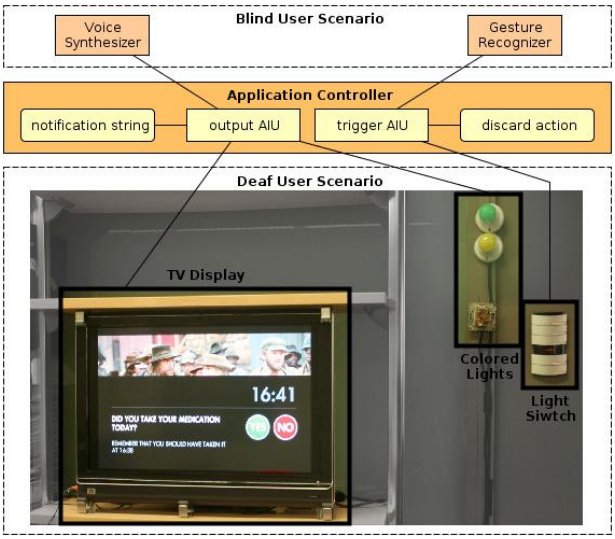

Figure 3. Use case example block diagram and photo

Using Dandelion, developers are able to build AmI UIs declaratively and isolated from the final implementation, thus allowing more portable AmI systems. Furthermore, the UIs are described using machine readable models which in the future could be used to autonomously manage the adaptation of the UI to specific scenarios. Nevertheless, the generic character of the GIP and the abstract UI model restricts the customization capability of the UI, which should be one prominent area of future work.

REFERENCES

1. Augusto, J. C., & McCullagh, P. Ambient Intelligence: Concepts and Applications. Int'l J. Computer Science and Information Systems, vol. 4, number 1, 1–28.
2. Dadlani, P, Peregrin Emparanza, J, & Markopoulos, P. Distributed User Interfaces in Ambient Intelligent Environments: A Tale of Three Studies. *Proc. 1st DUI*, University of Castilla-La Mancha (2011), 101-104.
3. Varela, G., et al. UniDA: Uniform Device Access Framework for Human Interaction Environments. Sensors 11 (10), MDPI (2011), 9361–9392.
4. Thevenin, D., & Coutaz, J. Plasticity of user interfaces: Framework and research agenda. *Proc. INTERACT'99*, IOS Press (1999), 110-117.
5. Balme, L., et al. Cameleon-rt: A software architecture reference model for distributed, migratable, and plastic user interfaces. *Proc. EUSAI 2004*, Springer-Verlang (2004), 291-302.
6. UsiXML. http://www.usixml.org, http://www.usixml.eu
7. Blumendorf, M., Lehmann, G., & Albayrak, S. Bridging models and systems at runtime to build adaptive user interfaces. *Proc. 2nd EICS 2010*, ACM (2010), *9-18.*
8. Abascal, J., & Castro, I. F. de. Adaptive interfaces for supportive ambient intelligence environments. *Proc. 11th ICCHP, Springer (2009),* 30–37.
9. Paz-Lopez, A., et al. Some Issues and Extensions of JADE to Cope with Multi-agent Operation in the Context of Ambient Intelligence. *Proc. PAAMS*, Springer (2010), 607-614