A development platform for distributed user interfaces

Anders Larsson, Magnus Ingmarsson, Bo Sun
Department of Computer and Information Science
Linköping University
Linköping, Sweden
andla@ida.liu.se, magin@ida.liu.se, x04bosun@ida.liu.se

Abstract

Developing user interfaces for a heterogeneous environment is a difficult challenge. Partial distribution of the user interface is an even harder one. Specifically providing developers with means of describing and controlling how components move around as devices are included or removed. We present an approach to overcome these challenges, by combining ontologies with reasoning engines. Our tool MaDoE uses Protégé in combination with Jess to exemplify this in a simulated home setting. Our approach allows system developers to take advantage of the formal knowledge in the ontologies as well harnessing the power of rules inside the expert system when they design distributed user interfaces.

1. Introduction

As we move into a more and more distributed and heterogeneous environment, it becomes clear that current development methods are ill suited to handle this new environment. Today's methods lack support for both describing how to divide and distribute user interfaces as well as means for allowing the user interface to reside on multiple devices.

We see a development towards applications using multi-machine user interfaces and distributed user interfaces, where several devices join together to accommodate the user with adequate interaction possibilities. [16, 19]. In a series of projects [8, 4, 20] we have developed such systems. Knowledge from these different systems has provided requirements for a DUI programming framework, called Marve. One essential component within this framework is a reasoning mechanism for describing distribution of UI-components at run-time among a variable set of devices according to a specific application strategy. Providing developers with means of describing and controlling how components move around as devices are included or removed is essential for distributed user interfaces.

This paper presents the MaDoE module which allows for high level of description of controlling component distribution using knowledge engineering tools together with a rule engine to accommodate the changing set of devices the user has to his or her disposal.

While DUIs constitute a more modular and convenient interaction platform for the user, they also constitute a more difficult programming environment for the developers.

There are reasons to assume that tools are required for effective means of development of DUIs, much in the same way as it was for GUI-development [17]. We believe that these tools should provide developers with high level of abstraction to handle the complexity of distributing an application. For instance preparing an application for three devices might in the extreme case results in six different permutations.

2. Background

2.1. Distributed user interfaces

In a mobile and ubiquitous computing world users are constantly moving around, making their possible interaction a function of the available I/O-devices. To illustrate DUI:s we present an example below.

Lisa arrives at Stockholm Arlanda airport, as she passes through the security checkpoint and enters the departure hall she receives a message on her PDA. The message contains information about how she can access the lounge area using a special access code. In order to find the lounge Lisa walks over to one of the wall mounted screens located throughout the airport. As she approaches the screen, a map of the airport is presented on the screen, while Lisa is presented with a search interface on her PDA. She can then search for the lounge and navigate the map on the screen through her PDA. As she starts walking, the map is transferred from the wall mounted screen to her PDA and the search interface is removed. When she arrives at the lounge a small screen tells her to swipe her ticket and to enter the
Applications starts Distribute components over the devices Devices removed/added to application Devices added to application

Described in the following steps. This process can be not only size but also in the set of devices and their capabilities the interface is currently using. This process can be commonly solved by the use of layout managers that re-renders the interface components when the window size is changed. One of the earlier applications that was migration aware was presented in [6]. Their applications could be moved from one platform to another at run-time, provided that the operating system stayed the same and it only allowed the entire application to be moved. The distributed systems area has nearly eliminated the need for for the programmer to do detailed implementations of network connectivity. Modern languages are very easy to use in this respect and provide simple yet powerful ways of handling network connectivity. An example of this is CORBA and Java’s RMI.

However, all are based on the fact that an external process other than the application itself has control over its own migration.

The DUI approach, however, makes moving entire or partial UIs a directly usable asset in application designs. One of the earlier applications that was migration aware was presented in [6]. Their applications could be moved from one platform to another at run-time, provided that the operating system stayed the same and it only allowed the entire application to be moved. The distributed systems area has nearly eliminated the need for for the programmer to do detailed implementations of network connectivity. Modern languages are very easy to use in this respect and provide simple yet powerful ways of handling network connectivity. An example of this is CORBA and Java’s RMI.

As DUIs constitute a richer interaction platform for the user, it also constitutes a more difficult programming environment for the developers. For a traditional GUI application the only outside influences the developer has to be able to handle is the change in size of the window. This is commonly solved by the use of layout managers that re-renders the interface components when the window size is changed.

Whereas in DUIs the developer needs to handle changes not only size but also in the set of devices and their capabilities the interface is currently using. This process can be described in the following steps.

1. Applications starts
2. Devices added to application
3. Distribute components over the devices
4. Devices removed/added to application

This can be referred to as the component distribution cycle [7]. In order to allow developers to control the distribution cycle, like developers can control a traditional applications layout with layout managers, new mechanisms are needed. In the Marve platform this is handled by the use of distribution managers which programmers can control. MaDoE adds a formal mechanism which allows for more general means of describing distribution well as a general descriptions of devices used in a DUI system.

2.2. Protégé

In this project, Protégé was used (version 3.0 Beta) to develop the ontology. Protégé is a platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. Protégé can be customized to provide domain-friendly support for creating knowledge models and entering data. First of all, Protégé is designed for system developers and domain experts to develop knowledge-based systems [1]. It is a tool to permit the integration of creating classes and entering instances of classes, as well as making it convenient for domain experts to model concepts in a specific domain and create applications in it to solve problems pertaining to this domain.

2.3. Jess

A rule engine is needed to manage the interaction between service system and mobile devices. In this project, Jess was chosen as rule language to create rule policies. Jess, Java Expert System Shell, is a rule engine and scripting environment written entirely in Sun’s Java language. It is a tool for building a type of intelligent software called Expert Systems. Jess rules can be compared to if ... then statements in procedural language. The rules created in Jess are fired to take actions based on facts which in this project are items related to instances of classes. However, since Jess uses Rete10 algorithm, rules are executed whenever their if parts (their left-hand-sides or LHSs) are satisfied, which makes Jess less deterministic than typical procedural language. This makes Jess rules running faster than normal if ... then statements. An example and description of Jess rules is shown below.

```
Jess> (defrule allowed-person
  "If a person is older than 22, print his (her) name."
?c <- (object (age ?x&:(> ?x 22))) =>
  (printout t (slot-get ?c name))
```

```
4. Devices removed/added to application
```
This rule has two parts, separated by the => symbol (which you can read as "then"). The first part consists of the LHS:

?c <- (object
  (age ?x &:(> ?x 22))).

The second part consists of the corresponding RHS (right-hand-sides) action:

(printout t (slot-get ?c name)
  " is older than 22."
)

This rule means whenever a person (instance) whose age (property) is more than 22, print his (her) name (property).

2.4. JessTab

Since we want to make the Protégé ontology and Jess rules work simultaneously to simulate some interactions under different circumstances, a tool is needed to achieve this goal. JessTab [12] is such a plug-in for Protégé which provides a console window to make Protégé interact with Jess when Protégé is running, see Figure 1 for an example.

Through JessTab, Protégé instances can be mapped into Jess facts and rules can be create which directly operates on Protégé's knowledge base (an aggregation of instances).

![Jess engine](image)

**Figure 1. Jess running an example for a TV application**

3. Related work

Grolaux et al. [13] have with their system Migratable, shown how migratable user interfaces can be used to achieve DUIS, by partial migration of the interface from one platform to another. They illustrate their approach by solving the painter's pallet problem (defined in [3]) by only requiring an extension to the original application with 0.5% of code. The research demonstrates that there is a need for new tools to allow construction of DUIS in a sound and professional manner.

A project that shows a formal way of describing the transformation of the web resources over a set of devices is Web-Splitter. Web-Splitter [14] is a framework for allowing web pages to be split into personalized partial views, depending on users different roles. These partial views can be further divided into xml components that can be transferred to devices located in the users proximity. This is a way of allowing limited mobile devices to handle multimedia by augmenting them with co-located devices. Another project that aims to allow web-systems to be distributed over a set of heterogeneous devices is presented in [22]. Their contribution is twofold it provides means for distributed user interfaces as well as a platform for collaboration.

Vandervelpen and Conix [10] show a system for high level description of user interfaces called Dygimes, which is based on a model-based user interface design approach, that can be extended to support distributed user interfaces. Their approach is a step towards frameworks that aids user interface designers by allowing them to work within a distributed environment where the interface can be partially split over a set of devices.

The Pebbles project [18], where handhelds and PC work together. The project aims to spread computing functions and their related user interfaces over different I/O-devices. In an extension to the Pebbles project the PUC, personal universal controller project shows how a high level description language can be used to describe remote controlling facilities for mobile devices to stationary units, such as VCRs, stereos and TV-sets. The high level description can then be used together with automated user interface engines to generate a limited user interface for the stationary unit on the mobile device. A more theoretical work is presented by De- meure et al in [11] where they present a schema for classification of distributed user interfaces, with the aim to provide designers with a mechanism that can express distribution of interfaces over different devices.

In Computer supported cooperative work a trend has been towards cooperative user interfaces, where a common user interface is shared among the users instead of each user having their separate application running with its own user interface. Pioneering work for cooperative user interfaces was presented by Smith and Rodden [21] when presenting SOL, Shared Object Layer. SOL allows the individual user's interfaces to be projected to devices from a common shared interface definition. This enables users to be presented with only the tools and function that they currently need, or are allowed to use.
4. Marve Distribution Manager Ontology Engine (MaDoE)

MaDoE allows for high-level of descriptions of describing the distributions of user-interface components over a changing set of devices. This is accomplished by the use of an ontology and a rule engine which can operate on the knowledge base described in the ontology. In order to have an easy way of redesign and reorganize the ontology we have used the Protégé ontology editor, and for this implementation we have chosen to use Jess as the rule engine.

To illustrate the functionality of MaDoE a small ontology, an overview of the ontology can be seen in Figure 2. This ontology both describes the different types of devices (Devices) that will be available as well as the properties (Features) of those devices. The domain of this model is oriented to mobile units which include Cellphone, PDA (Personal Digital Assistant), and Laptop. The Features included in this ontology are communication schemes (Bluetooth, infrared, etc.), operating system, and interaction capabilities. The ontology also includes information about the different type of applications (ServiceSystem) that will be used, in this small example the applications are oriented in services that might be found with in a home.

4.1. Classes

This small ontology was implemented using Protégé. Figure 3 shows classes hierarchy inside Protégé. The major classes defined in the ontology are ServiceSystem and Device which has subclasses Laptop, PDA, Cellphone. The Service System class represent the different applications available to the end-users.

Some other classes are needed to represent the different features of the Devices. The class Features which consists of subclasses Communication, OperatingSystem, Pad, and Screen. The class Communication represents the wireless communication schemes (WiFi, Bluetooth, Infrared), used by devices. The class Pad represents the platforms (Keypad, Keyboard, Touchpad) on which users could operate devices. Class Screen represents some parameters related to the screen of devices, like screen size and color-depth. For each service in the ontology a set of interaction-components are defined, called sub-service in the ontology, these components are then available for distribution among the different devices.

These features are then later used by the rule engine to decide what components can be used on what device. More complex system like MagUbi containing world knowledge can take advantage of this to reason on why certain components should be placed on certain devices.

From Figure 4, it can be seen that all properties of Device are defined as Instance type and ranged specifically to each subclass of Features and class ServiceSystem. Then cardinality restrictions are defined to properties of Device. Some assumptions about the mobile devices in the ontology:

1. Each mobile device has only one operating system while running.

2. Some mobile devices have been equipped with more than one pad, for example, Sony PEG-UX50 (PDA) has both Touchpad and Keyboard.
Some mobile devices have more than one communication port, for example, Acer TravelMate 661xvi (Laptop) has connectivity through Wi-Fi, Bluetooth and Infrared. But in this example, it is assumed that a device can connect service system through only one communication method at one time.

One single device can run only one application provided by ServiceSystem.

### 4.2. Programming Rules

From this simple ontology we can provide a developer with a high-level mechanism for describing how the applications in the ontology should distribute their different user interface components as devices are connected and disconnected from the application. This is accomplished in MaDoE by the use of Jess and the JessTab plug-in for Protégé. The JessTab provides automatic translation between Protégé classes to Jess Facts. MaDoE includes built-in rules for assuring that UI components are not transferred to devices where they can not be used.

For example MaDoE ensures that a drop-down menu component is not transferred to a device with no interaction capabilities like a TV, without simultaneous connection to for instance a cellphone, from which the users makes the selection. In this example the TV acts as a output device and the cellphone acts as the input device.

MaDoE also includes a set of rules that can be compared to layout mangers in traditional user interfaces. The MaDoE border distribution rule set, compared to Java Swing BorderLayout, ensures that a component named CENTER always is displayed on the largest screen available to the user. Formal and sound ways of describe application specific rules for controlling the behavior is also available in MaDoE through Jess rules. MaDoE acts as a plugin to the Marve system. As devices are added or removed from an application in the Marve runtime Jess rules are triggered and the underlying platform redistributes the component accordingly to the information from the rule engine.

### 5. Discussion

Allowing developers to control UI-component distribution through the use of ontologies and rule engines, provides a sound and formal way of expressing application behavior as devices are added or removed. A rule engine can ensure that components are not transferred to devices where the components can not be used. In order to allow for application specific behavior developers also needs to be able to describe own rules for how an applications user interface should change over time as the number of devices connected to an application change. As the rules supplied by programmer might contradict the built in rules in MaDoE, a set priorities is required governing rule precedences. This is not available today in MaDoE but is needed.

In MaDoE knowledge about devices and applications is described in a ontology and then transformed into facts that can be used by the rule engine. This in order to allow a higher abstraction level for describing how an application should change as the number of devices are changing. The output from the rule engine is then interpreted by the Marve platform. In the future there might be a way of letting the device ontology be available in the Marve framework and then have a rule system in the framework itself. Today Marve has a small set of distribution managers which can distribute components according to a predefined schema.

### 6. Conclusion

To provide developers with the right tools to build and maintain systems that support distributed user interfaces, it is essential to have formal and sound way of describing how the user interfaces of an application should change over time when devices are added or removed. The MaDoE module for the Marve framework also allows developers to use ontologies to describe the properties of both devices and information services. If combined with a system with world knowledge such as Magubi [15], proper DUls can be constructed as well as making sure that mistakes such as displaying the UI on an inaccessible display do not occur.

Users are today equipped with more and more devices for accessing different information systems. Devices ranging from traditional personal computers, PDAs, cellphones, or hybrids like smart-phones. For the developer it is important to accommodate users with adequate interaction capabilities. This project provides a model of mobile devices as well as a service system. The description of this model (ontology) combined with some rules is helpful in...
designing a service system. This is especially true for with
regards to the rule engine that offloads the burden of de-
ciding what goes where and when in this highly dynamic
environment.

A clear hierarchy of defined classes with related infor-
mation (properties, instances, comments) is illustrated. This
model is created by Protégé. It aims to give designers of ser-
vice system a rudimental prototype about different mobile
devices, their features, internal structures, etc.

Jess rules created for this model gives a way to simul-
tate the interaction between mobile units and service system.
They provide some basic thoughts of how to design rule poli-
cies for the rule engine of a service system. Even though
these rules have only been applied in a laboratory setting so
far, they could still be incorporated into research proto-
types of service systems to test how they are going to work in
actual circumstances.

This project shows how developers and system designers
can take advantage of such a model in both designing as
well as deploying information systems for mobile devices.
The rule engine also gives developers a high level way of
defining requirements as well as behavior of the system as
a whole. The project also demonstrates how services can be
split up into sub-services (components), to allow applica-
tion supporting distributed user interfaces to be developed.

Referencer

stanford.edu/doc/users_guide/index.html.
overview/index.html.
a hybrid computing environment for small computing de-
vices. In CHI ’00: CHI ’00 extended abstracts on Human
factors in computing systems, pages 133–134, New York,
[4] A. Berglund, E. Berglund, A. Larsson, and M. Bång. The
paper remote: An augmented tv guide and remote control.
[5] E. Berglund and M. Bång. Requirements for distrib-
ted user-interface in ubiquitous computing networks. In
MUM2002, Mobile and Ubiquitous MultiMedia confe-
rence, 2002.
UIST ’95: Proceedings of the 8th annual ACM symposium
on User interface and software technology, pages 132–142,
[7] M. Bång, A. Larsson, E. Berglund, and H. Eriksson. Distrib-
uted user interfaces for clinical ubiquitous computing ap-
lications. International Journal of Medical Informatics,
based ubiquitous computing healthcare environment to sup-
port data capture and collaboration. In Proceedings of
the 2003 American Medical Informatics Association Annual
for ubiquitous computing environments for healthcare pro-
[10] K. Coninx, K. Luyten, C. Vandervelpen, J. V. den Bergh, and
B. Creemers. Dygimes: Dynamically generating interfaces
for mobile computing devices and embedded systems. In
reference model for distributed user interfaces. In TAMO-
DIA ’05: Proceedings of the 4th international workshop on
Task models and diagrams, pages 79–86, New York, NY,
USA, 2005. ACM Press.
[12] H. Eriksson. The jesstab approach to protégé and jess integ-
ration. In Proceedings of Intelligent Information Processing
interfaces: beyond migratory interfaces. In Proceedings of
MOBIQUITOUS 2004. The First Annual International Con-
fere nce on Mobile and Ubiquitous Systems: Networking and
[14] R. Han, V. Perret, and M. Naghshineh. Websplitter: a unifi-
ed xml framework for multi-device collaborative web brows-
ing. In CSCW ’00: Proceedings of the 2000 ACM conference
on Computer supported co operative work, pages 221–230,
Service Discovery in Ubiquitous Computing, Number 1305
in Linköping Studies in Science and Technology. Linkö-
[16] A. Larsson and E. Berglund. Programming ubiquitous soft-
ware applications: requirments for distributed user interface.
In Proceedings of The Sixteenth International Conference on
Software Engineering and Knowledge Engineering (SEKE
04), 2004.
[17] B. A. Myers. User interface software tools. ACM Transac-
[18] B. A. Myers. Using handholds and pcs together. Communi-
[19] B. A. Myers, J. Nichols, J. O. Wobbrock, and R. C. Mil-
er. Taking handheld devices to the next level. Computer,
[20] M. Sjölund, A. Larsson, and E. Berglund. The walk-away gui:
Interface distribution to mobile devices. In IASTED-
Light-weight distributed web interfaces: Preparing the web
for heterogeneous environmen. In 5th International Con-
fere nce, ICWE 2005, Sydney, Australia, volume 3579/2005,