A Model-Based Approach for Dynamically Distributing Graphical User Interfaces Based on their Properties, Graphs, and Scenarios

Jérémie Melchior

Université catholique de Louvain
Ecole polytechnique de Louvain
Computing Science and Engineering Pole

A Model-Based Approach for Dynamically Distributing Graphical User Interfaces Based on their Properties, Graphs, and Scenarios

Jérémie Melchior

Thesis submitted
in partial fulfillment of the requirements
for the Degree of Doctor in Computer Science

Examination committee:

Prof. **J. Vanderdonckt**, Co-supervisor (UCL-Louvain School of Management & ICTeam, Belgium)
Prof. **P. Van Roy**, Co-supervisor (UCL-Computing Science and Engineering Pole & ICTeam, Belgium)
Prof. **K. Mens**, (UCL-Computing Science and Engineering Pole & ICTeam, Belgium)
Prof. **K. Coninx**, (Hasselt University, Hasselt, Belgium)
Prof. **J. Coutaz**, (Université Joseph Fourier, Grenoble, France)
Prof. **C. Pecheur**, President of the jury (UCL-Computing Science and Engineering Pole & ICTeam, Belgium)

January 19, 2016

Acknowledgments

This thesis has been made possible thanks to my advisors Jean Vanderdonckt and Peter Van Roy. I would like to thank them for their motivation, ideas and for all their contributions. I would also like to thank the other members of the jury: Professors Joëlle Coutaz, Karin Coninx and Kim Mens.

Many thanks to all my colleagues at the Operations and Information Management (OI) Department of the Louvain School of Management (LSM) and at the Computing Science and Engineering Pole (INGI) for their help, comments and friendship, especially Boriss Mejías and Ruma Paul for the great help and devotion around my usage of Beernet, Yves Jaradin who ported Mozart to Android and worked on the Mozart 2 DSS, Sébastien Doeraene, Anthony Gégo, Guillaume Derval and Benoit Daloze for the help regarding Mozart2. I am also greatly thankful to Tanguy Lepoutre for his help with the demonstrations and for his support until the end of the thesis.

Finally I would like to thank my friends and my family, especially my dear love Émilie and my daughter Élise.

The thesis has been supported by:

- The UsiXML project standing for USer Interface eXtensible Mark-up Language. ITEA2 European Project ITEA2-2008-0026. January 2010 - March 2013.
- The SELFMAN project standing for Self Management for Large-Scale Distributed Systems based on Structured Overlay Networks and Components. European FP6 (Sixth Framework Programme, Priority 2, Information Society Technologies) Project under convention No. 034080. September 2008 - September 2009.
- The SERENOA project. European FP7 (Seventh Framework Programme) Project under grant agreement No. 258030 (FP7-ICT-2009-5).
- The Usidistrib FIRST SpinOff Project under convention No. 1217843. May 2013 - May 2016.

Contents

List of Figures

List of Publications

Journal article

[MEL12-IJHCI] Melchior, J., Vanderdonckt, J., and Van Roy, P. A Comparative Evaluation of User Preferences for Extra-User Interfaces, *In the International Journal of Human-Computer Interaction (IJHCI)*, 28:11, Taylor & Francis, pp. 760-767, 2012.

[MEL11dui2] Melchior, J., Vanderdonckt, J., and Van Roy, P. Distribution Primitives for Distributed User Interfaces, *In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-Computer Interaction Series*, Springer-Verlag, London, pp. 23-31, 2011.

Conference papers

[MEL09] Melchior, J., Grolaux, D., Vanderdonckt, J., and Van Roy, P. A Toolkit for Peer-to-Peer Distributed User Interfaces: Concepts, Implementation, and Applications, In *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2009)*, ACM Press, New York, pp. 69-78, Pittsburgh, PA, USA, July 15-17, 2009.

[MEL11] Melchior, J., Vanderdonckt, J., Van Roy, P. A Model-Based Approach for Distributed User Interfaces, In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing System (EICS 2011)*, Pisa, Italy, June 13-16, 2011.

[MEL12] Melchior, J., Vanderdonckt, J., Van Roy, P. Modelling and Developing Distributed User Interfaces based on Distribution Graph, In *Proceedings of the Sixth International Conference on Research Challenges in Information Science (RCIS 2012)*, Valencia, Spain, May 16-18, 2012.

Workshop papers

[MEL11dui] Melchior, J., Vanderdonckt, J., Van Roy, P. Distribution Primitives for Distributed User Interfaces, In *Proceedings of the Distributed User Interfaces CHI 2011 Workshop (DUI 2011)*, ACM Press, New York, pp. 29-32, Vancouver, British Columbia, Canada, 2011.

[MEL12dui] Melchior, J., Mejías, B., Jaradin, Y., Van Roy, P., and Vanderdonckt, J. Improving DUIs with a decentralized approach with transactions and feedbacks, In *Proceedings of the Distributed User Interfaces CHI 2011 Workshop (DUI 2012)*, ACM Press, New York, pp. 65-68, Austin, Texas, USA, 2012.

Doctoral consortium

[MEL11dc] Melchior, J. Distributed User Interfaces in Space and Time, In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing System (EICS 2011)*, Pisa, Italy, June 13-16, 2011.

Recommandation to W3C

MBUI - Abstract User Interface Models,
W3C Working Group Note 08 April 2014,
Vanderdonckt, J., Tesoriero, R., Mezhoudi, N., Motti, V., Beuvens, F., Melchior, J.,
W3C, 08 April 2014,
http://www.w3.org/TR/abstract-ui/

Deliverables

SELFMAN Self Management for Large-Scale Distributed Systems based on Structured Overlay Networks and Components. D5.9: Distributed mobile application on gPhone.

USIXML USer Interface eXtensible Mark-up Language. D1.3: UsiXML definitions.

USIXML USer Interface eXtensible Markup Language (UsiXML), W3C Working Group Submission 1 February 2012, Vanderdonckt, J., Beuvens, F., Melchior, J., Tesoriero, R., W3C, 1 February 2012, http://www.w3.org/wiki/images/5/5d/UsiXML_submission_to_W3C.pdf

Chapter 1

Introduction

The days when the desktop computer was the only computing device used by one single user at a time to work and play are over. The more compact a computing device is, the less capable it will be. A few years ago, the difference in capability between a phone and a PC was very constraining as depicted in the left-hand image of Figure 1.1.

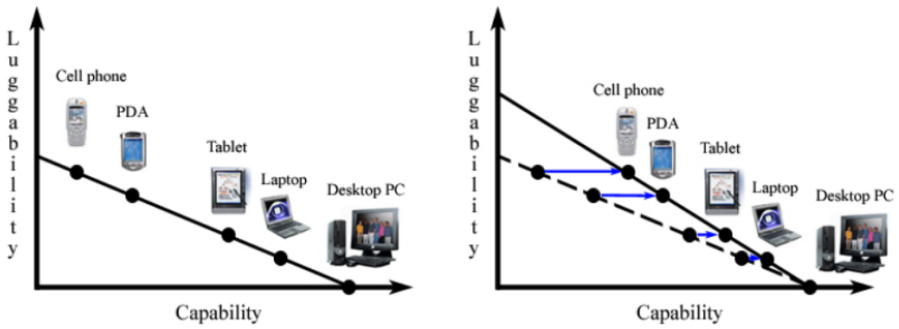


Figure 1.1: Evolution in the capability of computing devices compared to their size.[PIE04]

The use of a cell phone was restricted to very basic games and applications. Nowadays, computing devices offer more capabilities than in the past, even for the most portable ones (Figure 1.1) which are more frequently purchased over time (Figure 1.2).

In the right-hand image of Figure 1.1 we can see that the evolution of computing devices reduces the difference in capability between the more portable computing devices.

In a few years this difference may completely disappear if the trends keep going. The term *luggability* is used in this figure to describe the ease of portability of a computing device. Although these evolutions are considered significant, applications are still designed for a single computing device to be used by only one person at a time.

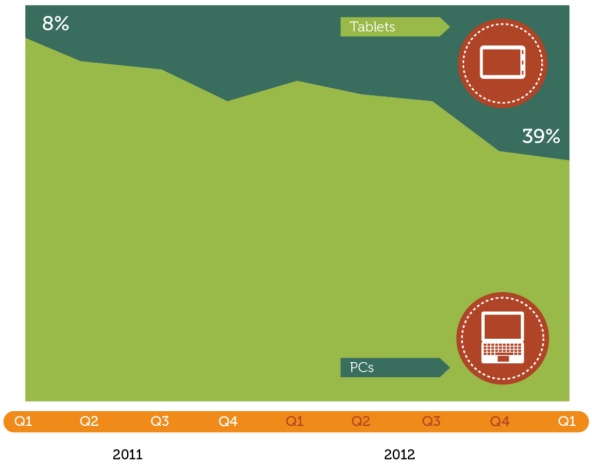


Figure 1.2: Quarterly sales market share in combined tablet and PC category.[VMDE13]

This assumption is becoming less and less true: a single user shares the time across different computing devices (Figure 1.3) and the same computing device can be used by different users. Users also more frequently carry out distributed tasks in many domains of human activity (e.g., management, finance, accounting, learning, gaming, ...).

1.1 Distributed Tasks

Applications created for desktop computers allow a human called the *user* to use a computing device to accomplish a certain goal called a *task*. However certain tasks can be very complex and can require more than one computing device or more than one user. A task that requires or benefits from more than one computing device is called a *distributed task*. The computing devices can either be used at different times, sequentially, or at the same time, concurrently. The definition of a distributed task is now provided.

Definition 1. *A **distributed task [DT]** is a task that is accomplished with the help of more than one computing device.*

To illustrate distributed tasks, we will give three running examples which will help the reader to understand the benefits of using multiple computing devices to accomplish a task. The first use case is a drawing tool that can be distributed in space. The second use case is the Pictionary game, which is naturally distributed in space and among users. And the last use case is a game based on the concept of the Game of the Goose and the Snakes and Ladders because it can show how distribution can be added to a non-naturally distributed game.

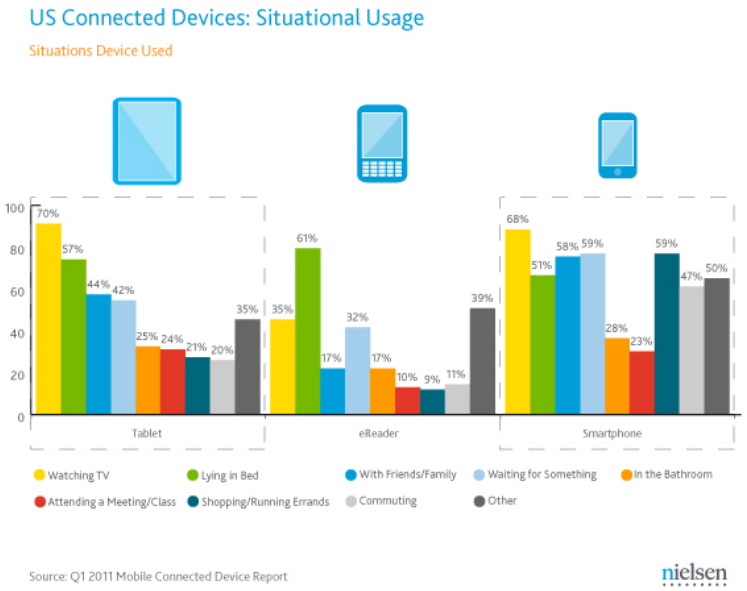


Figure 1.3: Graph of the situation in which a tablet, an eReader and a smartphone are used by US inhabitants.[NIE11]

Use Case 1: the Painter's Palette

Here is a very basic drawing tool such as Microsoft Paint. Let us call it: the Painter's Palette. Such applications are almost always designed for a single user with only one computing device: a desktop computer or a laptop. The common interaction for such applications is with a mouse: to choose the type and color of the pen, and to choose a shape (e.g., rectangle, ellipse, square, ...). However in the natural world we need several objects in order to create a painting. Figure 1.4 compares how to paint in the natural world and on a computing device.

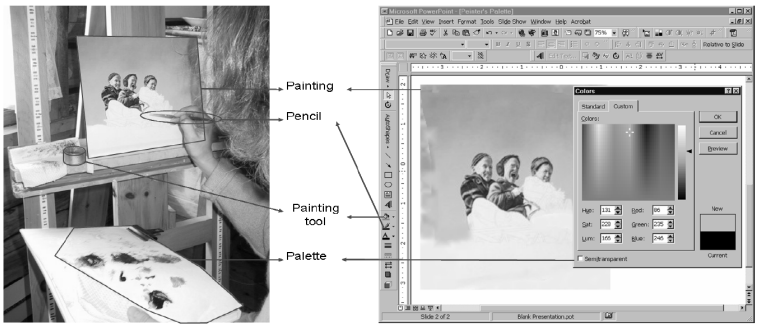


Figure 1.4: Natural world vs. digital world [GRO05]

Someone who has several computing devices may want to separate the palette from the painting itself. There are two ways to distribute the application: on only one or across several computing devices. An example is to display the toolbars on a smartphone while drawing with a pen on a tablet. There are currently no drawing applications that can use several computing devices at the same time. An example of this separation is depicted in Figure 1.5.

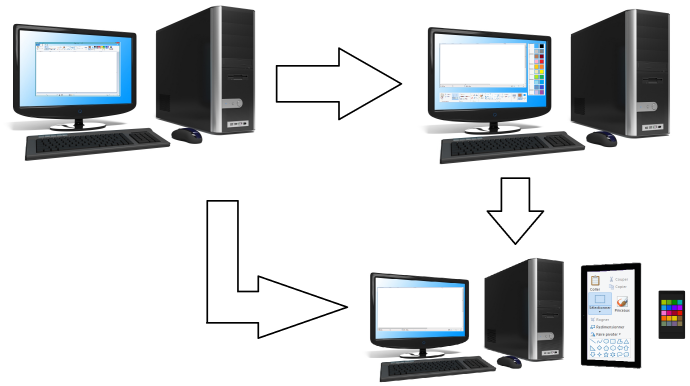


Figure 1.5: An example of application molding

Definition 2. ***Distributed User Interfaces [DUIs]*** *enable end users to distribute any user interface element at design- and/or run-time across different users, across different computing platforms, and across different physical environments.[VDD10]*

There are very few solutions allowing an application to use several computing devices simultaneously. Most of these solutions have not been released for developers to create their own applications. The current situation is to create several applications (one per computing device) and allow them to communicate together. Even such inter-applications communication is not easy to implement for a developer. So today, we think that there is no tool that allows developers to create distributed graphical user interfaces (DGUI) in a straightforward manner. Here is a definition of a DGUI as a specific DUI.

Definition 3. *A* ***Distributed Graphical User Interface [DGUI]*** *is a DUI in which all components are only graphical components.*

DGUIs and other DDUIs are said to be dynamic if the distribution occurs at run-time, and static if the distribution occurs at design-time. We will refer to Dynamic Distributed Graphical User Interfaces (DDGUI) if it is

dynamically distributed as opposed to Static Distributed Graphical User Interfaces (SDGUI). The thesis focuses on DDGUIs.

Definition 4. *A **Dynamic Distributed Graphical User Interface [DDGUI]** is a DGUI that can be dynamically distributed at run-time. The whole distribution is not established at design-time.*

Definition 5. *A **Static Distributed Graphical User Interface [SDGUI]** is a DGUI where the whole distribution is established at design-time and cannot be changed at run-time.*

To allow this kind of distribution we would like to create a software support that will connect computing devices together, and support distribution across them, e.g., to allow the UI to move from one computing device to another.

There are indeed applications such as Paint.NET [Paint.NET], Microsoft Visual Studio [VS], TeXnicCenter [TeXnicCenter] making it possible to separate toolbars from the main window and to move them within the desktop, which itself could be decomposed into several viewpoints, such as in Compiz Fusion for Linux[Compiz], nVidia nView for Windows [nView], AMD/ATI Hydravision which is now AMD Eyefinity[HydraVision, Eyefinity]

These applications force the distributed elements to stay on the same computing devices and do not allow these toolbars to move from one computing device to another.

HyperPalette [AYA00] connects a small computing device which is a pointing device acting like a gesture command related to a real world action such as copy. The Scoop-and-Spread technique (Figure 1.6) allows users to cut some elements from a virtual drawing board and paste them somewhere else.

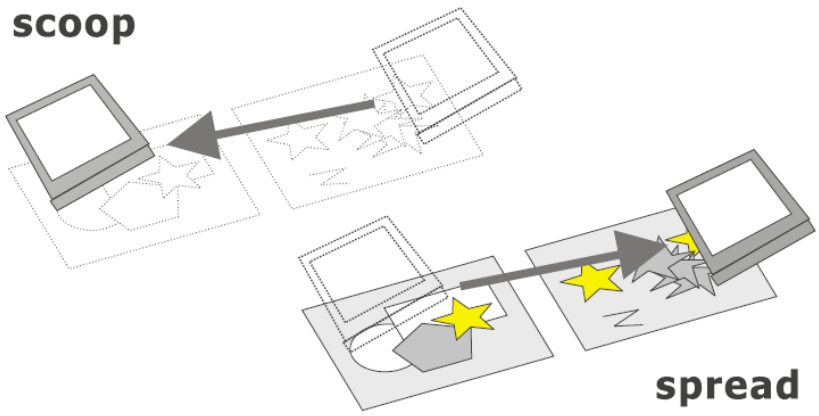


Figure 1.6: The Scoop-and-Spread technique of HyperPalette[AYA00]

WinCuts [TAN04] is a novel interaction technique which allows users to replicate regions of windows into independent windows called WinCuts. The new windows are live views of the corresponding regions (Figure 1.7).

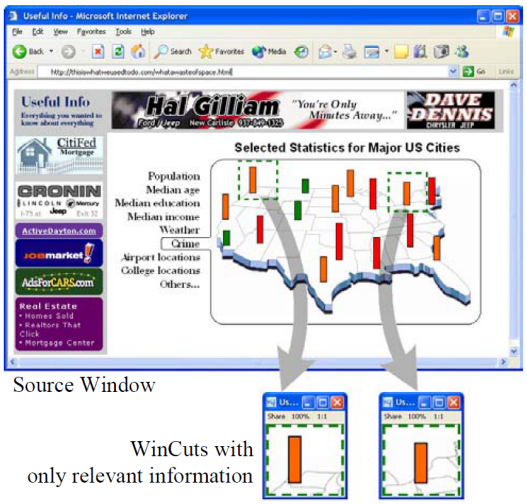


Figure 1.7: WinCuts allows users to only display the areas they want to see.

Use Case 2: Pictionary

Pictionary[Pictionary] is a guessing word game invented in 1985 by Robert Angel. An illustration of people playing Pictionary is depicted in Figure 1.8.

Figure 1.8: Children playing pictionary on a white board.

The game is designed for at least 4 players and can be played in teams. A player receives a word which the other players have to guess. The drawer can use a pen to draw sketches on paper. The other players attempt to say the correct word. The first player to guess the word gets a point for the team and the turn is over.

The board game

Pictionary was released as a board game and is sold by Hasbro and Mattel. The board is constituted of sequence of squares. Each square has a letter identifying the type of picture to be drawn (Figure 1.9).

Figure 1.9: A picture of the board of Pictionary.

Each team or player gets a piece that is placed on the first square of the board.

The objective of the game is to reach the last square of the board. At each turn the players from one team must guess the word or phrase being drawn by their current player. The role rotates with each word.

The drawer gets a card out of a deck. and draw sketches that help other players to guess the word that is on the card. The drawings cannot contain any numbers or letters, and no verbal clues can be given to help the other players guess. Their teammates try to guess the word the pictures are intended to represent and shout their proposals out loud.

There are five types of squares: P, O, A, D and AP.

- P: draw a person, a place or an animal
- O: draw an object
- A: draw an action
- D: draw a word that is difficult to represent in a drawing

- AP: one player from each team attempts to illustrate the same concept simultaneously

In P, O, A and D, there is only one player drawing.

In AP (All Play), one player from each team plays simultaneously. The first player to guess correctly wins the turn.

A one-minute timer, usually a sand timer, prevents a turn from lasting too long. If the timer runs out, then no point is given and the teams rotate.

Mobictionary

We have decided to choose this case study because the game is naturally distributed across these players. Depending on the role, the player is given a task to accomplish. A player is either the drawer or the guess player.

Each player has a different UI depending on the role. There are three roles: game manager, drawer and observer.

The game manager gives the word to the drawer and starts the timer. The drawers see the word which they need to make other players guess. They can use a pen on a drawing area. The other players can see the current drawing without the ability to draw anything.

There can be observers to the game. They can also see the drawing but they cannot play.

A timer is available on any UI for players and observers to see the time remaining before the end of the turn.

Use Case 3: Game of the Goose

Finally we will introduce another use case based on the Game of the Goose.

Game of the Goose

The *Game of the Goose*[GotG] is a board game created during the 16th century as family entertainment. The board consists of a sequence of 63 consecutively numbered squares (Figure 1.10).

They are usually arranged in a spiral. There are one or two dice. The current player throws the dice to move forward on the board.

Some squares have a goose depicted on them. The player who lands on a square with a goose is allowed to move again by the same distance. Some other squares have a different action associated to them: a bridge or a penalty. A bridge moves the player forward to another square. A penalty brings the player back to a previous square, or makes the player skip one or more turns.

Figure 1.10: An example of board for the Game of the Goose[GotG].

The distributed game

For our case study we would like to get inspiration from the Game of the Goose.

In our *distributed game*, every square of the board is a different game.

We have added one role: the manager of the board. The manager can change the game affected to a square during the game itself. We think that this will create more fun to keep the competition alive.

1.2 Models, Approaches, Software supports

There are initiatives in which a software application allows some particular distribution. However, there is no way, either synthetic or organized, to specify and design a DDGUI

In order to foster an approach that is not tied to a particular distributed task or to a particular distribution, let us introduce three dimensions to help us organize the research and analyze related work: The first dimension is the **models** that describe and define the concepts of interest. Then there is the **approach** which can be followed to use the models efficiently. And finally, comes the **software support** which can be either a toolkit or a software application.

The software support uses the approach which is itself based on the models (Figure 1.11).

These dimensions are investigated further in this section.

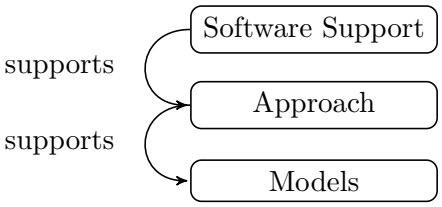


Figure 1.11: Schema of the dimensions.

1.2.1 Models

There are several entities that are part of a computing system and can communicate with each other. In order to understand, use, and support the whole computing system, we need to describe these entities to reason about them.

The main entities of a computing system are the computing devices. The evolution goes from one to many computing devices, leading to new capabilities. We need to model the different kinds of computing devices according to their size, their weight, their format, their capabilities and other aspects. Or, in short, any physical property of interest.

Computing devices range from a digital alarm clock or household appliances to cell phones and computers. Any domestic element could virtually be controlled from a smartphone or a computer (e.g., a television, some light, an audio player, a fridge, a microwave oven or the curtains of a window). According to research there are now more than one computing device per person[DEA08]. Instead of a single personal computer, people use several computing devices including desktop computers, laptops, mobile phones, digital cameras and media players. Applications could be made aware of all these computing devices and be able to manage information and activities across them. The next evolution could be the interconnection of all these computing devices wherever they are, in private, public or work spaces.

A closer look at the evolution of the market shows that the world tends to a full-interconnection of any physical objects, which is promoted by the concept of Internet Of Things (IOT). This makes it possible to create smart homes where people can listen to music and change songs, print a document, and control lights and room temperatures, from any remote computing device: physical (e.g., a switch) or digital (e.g., a software application) devices.

We also need to model the communication mechanisms that can be used along with computing devices and to describe how they work. This will allow us to choose what kind of communication mechanism we are going to use and support, with their advantages and drawbacks.

There is a plethora of communication mechanisms that allow these computing devices to inter-operate. Most of them rely on two types of addresses: IP and MAC addresses.

IPv6's onset brought more than 300 undecillion ($300 * 10^{36}$) addresses, which at the moment, seems to be enough to address the whole world of computing devices.

The primary communication mechanisms are Ethernet, WiFi and Bluetooth. But there are lots of other communication protocols and they all have their specific properties. Ethernet and WiFi allow devices to access routers and internet with or without wires. Bluetooth allows the transfer of information (e.g., sound, files, ...) from one computing device to another. NFC [NFC] is a near-field communication mechanism that can exchange a small amount of information when two computing devices are put against or close to each other. The primary use of NFC is to establish a Bluetooth connection between two computing devices but it can also send commands (e.g., with NFC tags) from one computing device to another. Today cloud computing appears as a solution to exchange or synchronize data across all the computing devices that are connected to it [cloud computing]. However, only data can be stored in the cloud and there is no direct interaction between computing devices.

We will use the term *distributed system* to refer to a system of interconnected computing devices that will appear to the users as a coherent whole to accomplish the tasks that users want to carry out. The main reasons for using this term are that it is largely used and commonly accepted in the field of *Distributed computing* [AND00, PEL00], it is sufficiently expressive, and it encompasses all other possible terms [EMM98, 1].

Definition 6. *A **distributed system [DS]** consists of a collection of autonomous computers, connected through a network and distribution middleware, which enables computers to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility [EMM98].*

In the Painter's Palette example, the distributed system contains all the computing devices that will be used by the drawing application and the user that will interact with the application and with these computing devices.

Distribution mechanisms provided by the domain of Distributed Computing can be a solution for managing the complexity of a DS. There are several DS properties to consider [EMM98]: the physical distribution of the computing devices and users, the problem of tasks running in parallel, the failure of a computing device or the failure of the communication between two computing devices, the lack of global knowledge and the dynamic aspect of computing devices joining and leaving the network. Such complex algorithms would allow developers to support the distribution of the UIs.

Because of all these aspects, it is not possible to exactly know what happens in the system. Some computing devices are in a certain state while giving information, but as soon as the information goes through the network, it is not sure how long this information will stay valid or persistent.

Some delays can lead to weird situations like turning on a light and wait for two minutes before the light is really turned on. There is a need to synchronize information between the computing devices for consistency.

These problems have already been addressed in the domain of Distributed Computing , therefore inviting us to re-use their underlying algorithms. The whole complexity of the DS would then be encapsulated into these algorithms and not in the applications.

In the remainder of the thesis we will refer to a computing device as a *device*.

The models should help us describe the devices that are part of the DS. It should also be possible to see and understand the relationships and communications between devices, and how to distribute a DDGUI across these devices.

1.2.2 Approach

We have been looking for any form of methodological guidance such as guides, methods or approaches that define the aspects to consider. However these have proven to be very rare.

What we have found was either unavailable, or not sufficiently documented to enable us to use it for our approach.

1.2.3 Software support

The last step is to create any kinds of software support for the method and the models.

Apple, Google and Microsoft have recently released services to allow users to display photos or to play music and videos on any connected device, using technologies like AirPlay[AirPlay], AirPrint for Apple and Xbox SmartGlass[Xbox SmartGlass] for Microsoft. Lately, Apple has also introduced Continuity[Continuity] which allows people to connect an iPhone o an iPad to a MacBook in order to execute a few basic operations. For instance, an Apple TV can be controlled from any device such as a remote control, a smartphone, a tablet, or any compatible laptop or desktop.

Recent technologies like Miracast [Miracast], Air Display [Air Display] and Project My Screen [Project My Screen] allow people to display the screen of a smartphone, a tablet or a computer wirelessly to a compatible device, e.g., a T.V., an external screen or a compatible computer.

There are also a few brands that work together to simplify connection between a smartphone and their devices for home automation. This is a short-term solution to this problem, but they will never support all possible devices. They also use closed protocols which prevent them from extending their solution with others.

All these solutions have been created by different groups of people and these efforts have not been integrated into a single development tool. There is therefore a need for a software support that would allow developers to benefit from these results without having to learn how to use all these solutions separately.

1.3 Thesis Statement

The problem to be addressed in this thesis is to enable designers and developers to create applications that support dynamic distributed graphical user interfaces (DDGUIs) and that can be used on all the available devices. DDGUIs enable end users to distribute any user interface element, ranging from the largest to the smallest, across one or many devices at both design-time and run-time. Using this world of fully interconnected devices will allow people to arrange and mold the applications according to their needs. In brief to support Distributed User Interfaces.

To create the software support that will allow us to create such applications we first need to define conceptual models and to create our own approach based on these models. With this approach we want to hide the complexity of a distributed system inside distribution mechanisms.

Here is the thesis that we want to address:

> In order to provide designers and developers with a model, an approach and a toolkit to support dynamic distributed graphical user interfaces of interactive applications, we introduce the concepts of distribution graphs, distribution scenarios, in a model-based approach that supports the properties of distributed systems and is implemented by a toolkit.

A DDGUI is not just the ability to move the UI from one device to another (migration). It also allows the use of several devices, the exploitation of their different sizes and characteristics, and their integration. Several users can fully interact together thanks to the support of distribution.

Along with DDGUI we want to support the properties of a DS such as the observation of devices joining and leaving, delays, and failures.

Facing a lack of definitions and models to support the creation of a DDGUI, we have decided to introduce some concepts that allow us to model and manage the distribution of UIs in a DS. Upon these concepts we have created a toolkit that demonstrates the possibilities offered by DDGUIs.

1.3.1 Single VS Multiple distributions

There are two ways to distribute the user interface (UI) across devices: single and multiple distributions.

A single distribution of the UI means that the features offered by an application can be migrated to other devices. It does not matter if the

devices are used by the same user or not. It means that a feature is only instantiated once. If the feature is available on a device, it is not available on other devices. There is no concurrency between devices as they cannot use the same feature at the same time. An example of single distribution is an application that controls lights in a house and that would not let two computing devices control the same light.

A multiple distribution of the UI allows the distribution of all the features on all the devices at the same time. This can lead to multiple instantiations of the same features. Indeed a feature can be reproduced on several devices. When there are multiple instances of a feature, we need to handle the concurrency between them. An example of multiple distribution is an application that controls room temperatures in a house. If two devices attempt to change room temperatures at the same time, this leads to a conflicting situation. How can the system know which of the change should take place?

1.3.2 Scope

Although DUIs could be applied to many domains of human activity and various contexts of use, this thesis states a series of assumptions (A_i) to focus on a specific scope and leave other problems for future work. Let us start by introducing the general assumptions (GA_i):

- GA_1**: DDGUI only: no other modality of interaction**
  The thesis only focuses on visual modality. Vocal, taptic, haptic, or any multimodal systems are not addressed directly in the thesis.

- GA_2**: No complex or safety critical system**
  The thesis will only focus on interactive applications which can be represented as a simple system. These applications are not safety critical (e.g., neither an unmanned aerial system, a train control/management system, nor health critical applications, ...).

Here are the assumptions regarding the model dimension (MA_i):

- MA_1**: No coverage of DDGUI usability ergonomic aspects**
  Since this thesis is intended to introduce a principle-based way to design DDGUI, we do not assume that any GUI resulting from a distribution issued by the method is usable. Further references on DDGUI usability include [DEE10].

- MA_2**: No coverage of DDGUI security**
  The security of the whole DS and its applications is left for future work. However it is possible to add this concept to the distribution method introduced. Anyway, the security for the DDGUI is less critical than for the functional core. An example of security issue is someone pretending to be a user who he is not , also known as identity theft.

To cover this issue, we need to guess who is using a device with some recognition (e.g., face, voice recognition, biometrics, ...).

And regarding the approach (AA_i):

- AA_1**: handling 2 reference use cases**
  The concepts have been evaluated on a small number of case studies. We wanted to validate our concepts and approach on real case studies. The same reasoning can be applied on other case studies. We want to select a few case studies that will help us understand the concepts and the way we can apply them to the case studies. Any kind of applications could have been used as a case study. The first case study we have selected is Pictionary because this game is naturally distributed across several players that have different roles and these roles change depending on who wins. The second case study selected is an adaptation of an existing application: Transdraw, which is a drawing tool using transactions. This will prove how easy it is to adapt an existing application in order to support DDGUI.

- AA_2**: focus on the distribution of the UI part of an application**
  We have focused the research on user interface (UI). The logic part of an application is already widely covered in distributed computing and their solutions still work with our conceptual solutions.

- AA_3**: the logic part is supposed to be always running, active and reliable**
  This logic part is supposed to be always running, active and reliable. If an application needs some warranty about the reliability of the core (where the logic is), it can still distribute the logic part using methods of distributed computing. Thus, if the logic is always running, the UI of a device can always be recreated in case of a crash.

Finally we also have assumptions for our software support (SA_i)):

- SA_1**: handling a set of supported devices**
  We support a subset of operating systems. The reason is that with a small number of operating systems we can cover more than 95 percent of the common smartphones, tablets, laptops and desktop computers. For this we have selected the most commonly used operating system. The choice of the operating systems supported does not influence the solutions proposed in this thesis. The potential candidates are all the operating systems. For computers we will support: Linux, Mac OS X and Windows. We also target mobile devices through the main operating systems on tablets and smartphones: Android, iOS and Windows Phone.

- SA_2: **focus on the visual modality**
  In Human-Computer Interaction domain, there are several modalities: vocal, visual, tactile and haptic. The thesis only focus on the main modality for software applications: visual. The main aspects of other modalities are already covered in other papers. Thus the distribution will only be for Graphical User Interfaces (GUI). Note that our research may probably be adaptable to multimodality.

1.3.3 Contributions

There are several contributions that are brought by the thesis. Let us introduce them according to each dimension.

Models

Regarding the models, we introduce the concept of *Distribution Graph* to model a DS. It is a graph where all the computing devices and users are represented as vertices, and their connections as arcs. Then we use the EBNF grammar to define formally the language expressing distribution primitives.

Approach

We have also created a **model-based approach** based on the concepts of distribution graphs, distribution scenarios and distribution primitive.

Software support

One of the results of this thesis is a software support in the form of a toolkit that allows the creation of distributed user interfaces. We have called this toolkit JayTk. It implements the concepts introduced during the thesis and is built on top of Beernet as depicted in Figure 1.12.

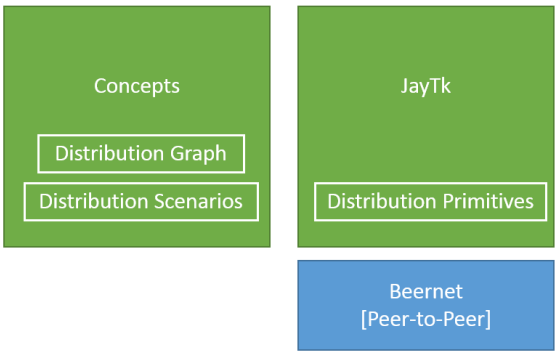


Figure 1.12: JayTk based on Beernet and implementing the concepts of the thesis.

JayTk is the main contribution brought by the thesis. It allows developers to create applications with DDGUIs and to support the main properties of a DS. There are two different kinds of applications: *distribution unaware* and *distribution-aware* applications.

- Distribution unaware applications do not need any information about distribution. They can use the toolkit with no or little modification of the code. They get the power of distribution through the toolkit which manages the distribution automatically or manually through an additional interface provided.
- Distribution-aware applications take full power of the toolkit and the distribution mechanisms. These applications can be notified if one of the devices crash and react to it. They can also manage how the GUI is distributed when a device joins the network.

1.4 Support for mobile devices

The way our toolkit will allow applications to be distributed across mobile devices is different from how they are distributed across computers. Indeed, mobile devices will be used as a destination of distribution but will not create applications. The main reason is because Mozart and Beernet are not yet available on the operating systems that run on these devices. To our knowledge, there is no standard or well known way of supporting distribution mechanisms on these devices either.

However these mobiles devices can be used as weak-node in the peer-to-peer network. This means that they are not responsible and part of the distribution but they can receive and interact with applications that are running in the network. This is how the toolkit supports mobile devices without offering a full compatibility. In the future, this limitation will be removed because these devices will become as efficient and powerful as computers.

1.5 Organization of the Thesis

Figure 1.13 describes the structure of the research. It clarifies the relations between the concerns, the shortcomings and the requirements of the thesis. The concerns have been defined in this chapter. The shortcomings and the requirements will later be derived from the concerns.

This chapter introduced the thesis topic by explaining the motivations. It describes what a DS is and all the important concepts that come with it. Then the scope of the thesis is set. And a summary of the main contributions is given.

In Chapter 2, we establish the related work of research related to the thesis. One of the aspects in the comparison is how a DS is modeled. Then

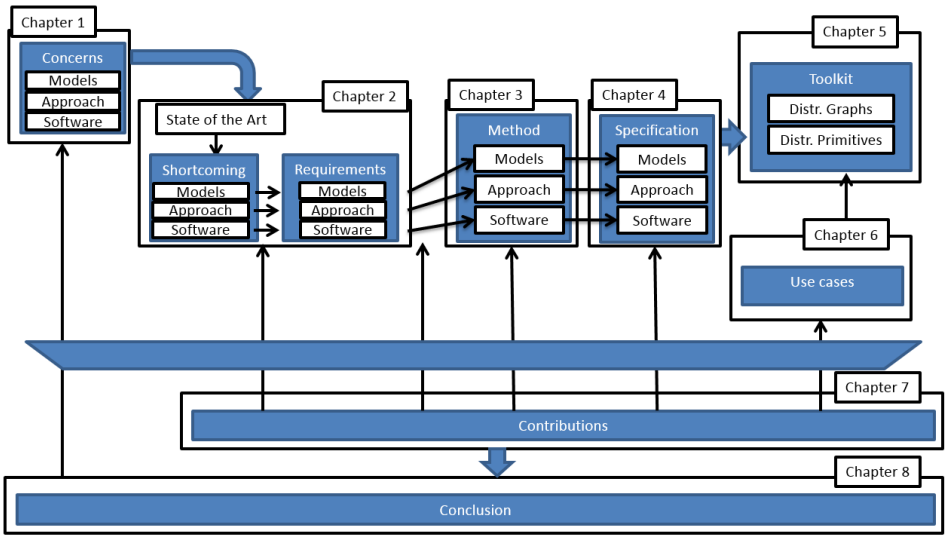


Figure 1.13: Outline of the thesis

we explore the existing tools that support the distribution of the application UI. Finally, we discuss other methods and studies on specific aspects covered here.

The method is then described in detail in Chapter 3. We first formally define the concept of distribution graph. Then, we select some operations that provide a sufficiently representative set of the possibilities offered by the distribution. The last part is the description of the method to create, support and manage DDGUIs in distribution scenarios.

The implementation of the toolkit with all the questions and choices that have been raised by using the method are provided in Chapter 4.

In Chapter 5, we use the toolkit to create a solution for several case studies. We also demonstrate a solution for some case studies: a Pictionary with three devices, a distributed transactional drawing tool using more than three devices and running on Android devices, and other small examples.

Chapter 6 evaluates and compares the results of the method with some important results in the related work. It is also the validation of the toolkit built during this thesis.

The last chapter concludes the thesis with a list of all the contributions. A list of the ongoing work is also provided.

The structure of the whole system is summarized on top of JayTk's architecture (Figure 1.14).

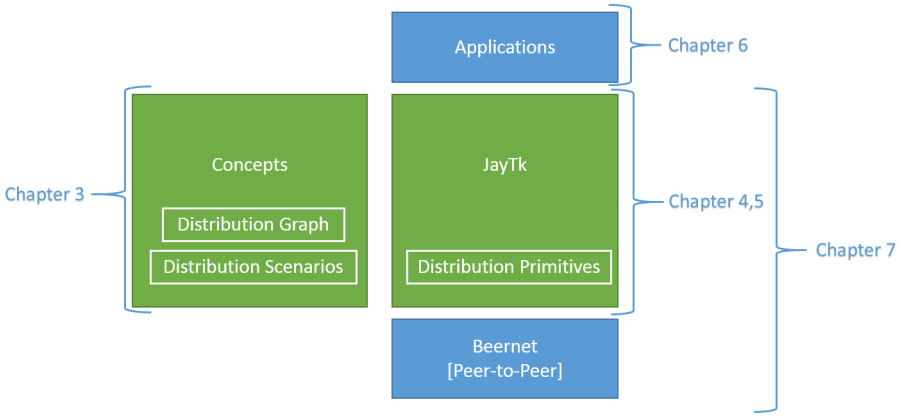


Figure 1.14: Thesis roadmap

Chapter 2

Related Work

In this chapter we compiled the related work by first listing them all and then we evaluate their impacts on each dimension.

In order to create a list of the related work we first started from a paper classifying several papers on DUIs[ROU06]. We have been through its references and iterated in each paper's references when we found them as interesting for the thesis. We also included papers that have been submitted at the DUI workshop of the CHI conferences. The main topics used as criteria for the selection of a paper are UI and DUI, distributed computing, models, approach and software supports. We will use these criteria as categories to discuss about these paper's contributions through this chapter.

Here is the expanded list of the related work we have built. The most important contributions will be detailed after the list. Other references can be described by analogy to them.

Models

There have been lots of papers on models. Software engineering is an important domain[DAM05, DAM06]. Mandviwalla[MAND94] has worked on requirements for groupware systems. Letier[LET01] has written a thesis about agents in goal-oriented requirements engineering. Puerta and Eisenstein[EIS01, PUE02] have worked on XIML, a representation for interaction data.

There have been several papers on task models, and migratability. Barboni and co. [BARB10] have introduced a new notation for bridging the gap between tasks and systems models. Dittmar[DIT11] has worked the support of task migratability. Penichet and co.[PENI07] have worked on task modeling for Collaborative Systems. Wurdel and co.[WUR09] have worked on task modeling for smart environments.

Some papers have concentrated on the context model. Brdiczka and co.[BRD07] have worked on models for context-aware services. Shackel [SHAC09] has worked on the definition of usability. Terosiero, Vanderdon-

ckt and co.[TES10, TES10B] have worked on extending UsiXML. Zaidenberg and co.[ZAI06] have worked on extracting context models from scenarios.

The other papers have focused on other models. Biswas and Robinson [BISW10], McTear [MTE93] have made surveys on existing modeling techniques. Salay and co.[SAL09] have worked on macromodels to manage collections of models. Wagelaar[WAG08] has written a thesis about platform ontologies. Still and Masciocchi[STI10] have worked on a model for predictions of web interfaces.

Approaches

Although we have been through lots of papers we have only found one approach to help developers to create applications. Martinie and co.[MART10] have worked on a model-based development approach to embed requirements at design time. The papers about software supports for DUIs describe their solution without providing the methodology they used.

User Interface

In order to address the problem of designing and developing a DDGUI we have reviewed the literature in HCI. This review has proven that there exist lots of references about UI. We first list them and will describe them later in the section.

In model-driven UI development, Breiner and co.[BRE10] have realized an evaluation of the UI adaptation. Caffiau and Girard[CAF10] have introduced a process for using model-driven approach in UI design. Gonzáles-Calleros, Guerrero-García, Vanderdonckt, and co.[GON09, GUE06, GUE09, VDD01] have worked on conceptual modeling of UIs. Griffiths and co. [GRI01] have worked on Teallach, a model-based UI development environment for object databases. Ladry and co. [LAD10] have worked on usability evaluation of interactive techniques. Pastor and Molina [PAS07] have worked on a software production environment based on conceptual modeling. Rich [RIC09] has worked on building task-based UIs with the ANSI/CEA-2018 standard. Sousa [SOU09] has worked on the model-driven approach for UI in business process modeling. Wolff and Forbrig [WOL10] have worked on the development with the Eclipse Modeling Project. Zhang [ZHA10] has worked on an aspect-oriented UI Modeling.

Howell and co.[HOW03] have evaluated the runtime performance of GUI creation frameworks. Miah and Alty [MIA99] have realized an empirical study of an adaptive window management. Rashid and co. [RASH12] has compared the cost of display switching with mobile, large display and hybrid UI configurations. Vrazalic [VRA03] has realized an evaluation of distributed usability in an activity systems. Xiaojun and Balakrishnan [XIA09]

have presented a study comparing usage of a large display to single/dual-monitor configurations.

Moscovitch [MOS09] have worked on the use of the contact area instead of the contact point with touch UIs. Vanacken and co. [VAN08] have worked on additional UI for Multi-touch Interaction. Aslan and co.[ASL10], Avrahami and co.[AVR89], Beaudouin-Lafon and co.[BEAU00, BEAU01], and Barralon and co.[BARR04, BARR07] have also written about additional UI to control the UI itself, commonly called meta-UI or extra-UI.

Other researchers have focused their work on more specific UI. Ali and co.[ALI01, ALI02], Bishop[BISH06], Ding and Litz[DIN06] have written about multi-platform UI. Kortuem and Kray [KOR05] have studied the HCI issues in multi-display environments (MDEs). Bickmore and co.[BIC99] have worked on a web page filtering for mobile devices. Pierce and Mahaney [PIE04] have also worked on mobile devices. Kavaldjan and co. [KAVA10] have worked on an automated optimization of UIs for screens with limited resolution. Hutchings and Pierce[HUT06] have worked on divisible UIs. Hill[HIL92] has studied the abstraction-link-view paradigm to connect UIs to applications. Lorenz [LOR10] has studied the application of MVC in ambient computing environments. Vernier and Nigay [VERN99, VERN00] have worked on multi-modal UIs. Schlegel [SCHL10] and Schwartze and co. [SCHW10] have worked on the adaptation for UI at runtime.

There are groups of research that have already spent time on the next generation and the future of UIs. Shaer and co .[SHAE08] have worked on the use of UIDL with them. Myers and co. [MYE00, MYE01] have discussed about the future of software tools.

The work on UI for multi-device systems has led to the domain of Distributed User Interfaces (DUIs).

Distributed User Interfaces

There are plenty of papers about DUIs which aim reaching Mark Weiser's dream of Ubiquity[WEIS99, WEIS03]. Aksenov and co. have worked on reasoning over spatial relations for DUIs[AKS08, AKS09]. Balme and co. have worked on a reference model for DUIsband[BAL04].

There are teams of researchers that spent several years on this topic.

Bang, Berglund, Fröberg, Sjölund and co.[BANG05, BER02, FRO11, SJO04] have been among the first to work on DUIs. They have realized a prototype with a smartphone used as a remote for a computer.

Bandelloni, Ghiani, Manca, Mori, Paternò, Santoro and co. have written several papers on DUIs [GHI10, PAT02, PAT07]. Their research has led to the development of TERESA (Transformation Environment for inteRactivE Systems representAtions)[BAND04, MOR03, MOR04, PAT01, PAT08]. An authoring tool which provides designers and developers automatic support for transformations of UIs. It has later led to the development of MARIA

(Modelbased lAnguage foR Interactive Applications)[MANC11, MANC11B, MANC11C, PAT09, PAT09B, PAT10, PAT10B]. A novel model-based language for UIs.

Bailey, Biehl and co. have worked on multi-display, multi-device and DUI systems [BAI04, BIE04]. Their work has first led to the development of ARIS [BIE04, BIE05, BIE05B, BIE06C], SEAPort (Scalable, Enhanced Awareness, Portal-based) [BIE06, BIE06B, BIE06C] and later to IMPROMPTU [BIE08].

Blumendorf, Feuerstack, Roscher, Weingarten and co. have been working on the runtime aspects of DUIs [BLU11, ROS10]. They have defined a Smart Home Energy Assistant (SHEA)[FEU07, WEIN10].
Their research has led to the development of MASP (Multi-Access Service Platform) [BLU10, FEU07, ROS09, ROS09B, WEIN10] which is a model-based run-time system for the creation of DUIs.

de la Guia, Gallud, Garrido, Lozano, Marco, Peñalver, Penichet, Sebastían, Tesoriero, Villanueva, and co.[DLG10, GAR11, MARC11, SEB11, VIL11] is currently active on formally defined DUIs with models. Their work has led to the definition of an AUI model[PENA11, PENA11B, PENA12].

Coninx, Luyten, Meskens, Vanderhulst, Vandervelpen and co. [LUY02, LUY04, VDH07, VDV04] have also been working a lot on DUIs. Vanderhulst has written a thesis about Dynamic Distributed User Interfaces (DDUI) [VDH05]. They have also worked on models[VDH08C, VDH09]. Their work has led to the development of Light-Weight Distributed Web Interfaces[LUY05, LUY06, VDV05, VDV05B]. And it has also led to the development of GUMMY[MESK08, MESK09], a multi-platform GUI builder. They have also worked on ReWiRe and pervasive environments[VDH08, VDH08B, VDH08C, VDH10, VDH10C], and on software support[VDH09B, VDH10B].

Finally Grolaux, Lepreux, Vanderdonckt and Van Roy [GRO04, GRO05, LEP06, LEP06B, LEP11, VDD10] have worked on migratory UIs. Their work has led to the development of the AttachMe, DetachMe demonstration which is based on EBL/Tk (Enhanced Binding Layer/Toolkit), a middleware that interfaces with one or more graphical toolkits. This tool has led to this thesis.

Lots of small teams have also worked on DUIs:

Bardram and co. [BARD11], Barth and co. [BART11], Bharat and co. [BHA95], Cagle [CAG05], Chang and Li [CHA11], Chen and co. [CHE11], Dadlani and co. [DAD11, DAD11B], Ens and co. [ENS11], Fardoun and co. [FAR11], Lambropoulos and Danis [LAM11], Larsson, Ingmarsson and co. [LAR06, LAR07], Linten and Price [LIN93], Löchtefeld, and co. [LOC11], Marquardt and Greenberg [MARQ07], Molina and co. [MOL06, MOL06B], Qiu and Graham [QIU09], Rodden and co. [ROD04], Seifried and co. [SEI11], Sendin and López [SEN11, SEN11B], Shen and co .[SHE04], , Yanagida and Nonaka [YAN08], and Zöllner, and co. [ZOL11]. Bell [BEL05]

has written a doctoral thesis on DUIs. Recently, Elmqvist [ELM11] has listed the state of the art of DUIs.

There are also lots of papers talking about multi-display and multi-device systems.

Hutchings, Stasko and co. have worked a lot on multiple monitors [HUT02, HUT02B, HUT03, HUT04, HUT04B, HUT04C, HUT04D, HUT05, HUT05B, HUT05C, HUT05D, HUT07, HUT07B] and on QuickSpace which provides new operations for computers[HUT02].

Ashdown and Sato [ASH04], Grudin [GRU01], Inkpen and Mandryk [INK05], Kaviani and co. [KAVI11], Lee and co. [LEE08], Mansoux and Nigay [MANS05] have also worked on multiple monitors.

Beale and Edmondson [BEAL07], Cardinaels and co. [CAR06], Dearman and Pierce [DEA08] have worked on multi-device and multi-screen systems.

Applications

Along with all these more theoretical papers, there are some applications that have grown up from these topics. Air Display [Air Display] and AirPlay [AirPlay] are applications to turn a device into a monitor. Ayatsuka and co. have worked on HyperPalette[AYA00]. Benoît and co. have worked on a multimodal driving simulator[BEN07]. Englebert and Heymans have worked on MetaCASE tools[ENG07]. Black, Edwards and co. have worked on the Speakeasy approach[BLA02, EDW02]. Eychaner has developed a UI framework for controlling DS[EYC03]. Etherpad is a website where a user can create a text document and share it with other users[Etherpad]. Han and co. have worked on WebSplitter[HAN00]. Drag&Share is a shared workspace for distributed synchronous collaboration [MARC11]. Rekimoto and co. have worked on Pick-and-Drop and Proximal interactions software applications [REK97, REK03]. Rey and Coutaz have worked on the Contextor, an application for dynamic distribution of contextual information[REY04, REY06]. Multi is a multi-user laser table interface[STU04]. Wincuts allows the manipulation of window regions[TAN03, TAN04].

Distributed computing

When writing about DUIs it is hard to avoid writing about Distributed computing. There are already many papers on distributed computing to list them all but here are the main litterature we have been through to understand and benefits from this topic. Andrews [AND00], Attiya and Welch [ATT98], Collet [COL07], Coulouris and co. [COUL05], Elmqvist [ELM15], Emmerich [EMM98], Ghosh [1], Peleg [PEL00], Tel [TEL95] have written about distributed computing.

A subtopic of distributed computing is the Peer-to-Peer architecture which has been covered by the following papers. Loeser and co.[LOE03],

Mesarov and co.[MESA04] have worked on peer-to-peer networks. Chung and Dewan have worked on dynamic collaboration architectures[CHU04]. Recently, Fisher and co.[FIS14] have worked on P2P DUIs.

Pen-based UI

While a bit off topic we have also look at how a pen could be used with UI. There is an ISO standard for pen-based interfaces[ISO99]. Long and co. have also worked on pen-based interfaces[LON00, LON01].

Others

Some teams of researchers have worked on many of these topics together and cannot be put in one of these.

Calvary, Coutaz, Demeure, Frey, Graham, Lachenal, Roudaut, Sottet, Thevenin, Vanderdonckt and co. have worked on models [CAL97, COU05, COU05B, DEM05, SOT07], on plastic UI [CAL01, CAL04, COU10, SOT07, THE99, THE02, VDD08B], on ambient space [COU06, COU07, COU07B], on multi-device [COU03, COU03B, GRA00, LAC03], on DUI [DEM05B, DEM08, FRE09, ROU06, ROU06B] and on gestures [ROU09].

Dewan and co. have worked on multi-user systems[DEW98, DEW98B]. Dey and Abowd have worked on context-aware systems[DEY00, DEY01]. Jourde and co. have worked on multimodal systems[JOU10]. Robertson and co. have worked on a flexible task management[ROB04]. Hyperspace is a high resolution display[SCREEN]. Seifried and co. have worked on CRISTAL, a collaborative home controller[SEI09]. Tullis and Albert have worked on usability metrics[TUL08]. Vanderdonckt has worked on a knowledge-based system for interaction styles[VDD97].

2.0.1 Meta-UI for Ambient Spaces[COU06]

In their paper on Meta-UI for Ambient Spaces[COU06]. They compare the tools that exist in 2006 such as ARIS [BIE04], Jigsaw [ROD04], AttachMe [GRO05] and SpeakEasy [BLA02, EDW02]. The dimensions used for the comparison were the discoverability, the coupling, the ability to redistribute and to remold the UI. The main conclusion is that researchers should be careful about the development of automatic systems for scientific challenge instead of involving and considering the level of control left to end-users. You can find their classification of related work in Figure 2.1. This table shows the different capabilities offered by the toolkits according to several axis: discovery, coupling, re-distribution, re-molding, parametrization and extensibility.

		Objets manipulés						Qualités		Puissance																						
		Nature			Manipulation			Mixage		Services et contrôle des services																				Extensibilité		
		Num. indep	Num. dép	Num	Directe	R.Num	R.Phys	Tissée	Ext	Découvrir				Assembler				Redistribuer				Remodeler				Parametrer				Voca	Synt	Sem
										Existance	O	T	C	Existance	O	T	C	Existance	O	T	C	Existance	O	T	C	Existance	O	T	C			
7	ARIS		X	X		X			X	X	X			X				X	X	X	X					X	X				X	X
13	a CAPpella	X	X	X		X			X	X	X															X	X		X	X		X
25	DongleRelate		X			X			X	X	X		X	X	X		X													X	X	
28	MigreXML		X	X		X			X	X	X			X				X	X	X	X	X									X	X
3	AmbientDesktop		X	X	X	X			X	X	X			X	X		X	X	X		X	X									X	X
30	SpeakEasy		X	X		X			X	X	X		X	X				X			X	X								X	X	X
36	JIGSAW	X	X	X		X	X		X	X	X			X	X		X									X	X		X	X	X	X
26	E-Gadget	X	X			X			X	X	X			X	X		X									X	X		X	X	X	X
37	ICAP	X	X	X		X			X	X	X		X													X			X	X	X	X
27	LEGOLOGO	X	X				X		X					X	X		X									X	X				X	
32	Elope		X	X		X	X		X					X			X	X			X											X
9	MightyMouse		X	X		X			X	X	X			X				X			X										X	
24	Dynamo		X	X		X			X	X	X															X	X		X		X	
29	Peebles		X	X		X			X					X				X	X		X										X	
19	PutThatThere			X	X	X			X									X			X											X
2	IStuff	X	X		X				X																	X			X		?	?
33	Icrafter		X	X		X			X	X	X		X													X			X		X	?
4	Collapse			X	X			X														X			X						X	X
18	AttachMe			X	X			X						X			X	X			X	X									X	X
21	Stitching			X	X	X			X					X			X	X	X	X	X											X
8	CubeTV			X			X		X																	X						
35	DataTiles			X		X	X		X					X	X		X	X			X											
16	Triangle		X	X		X	X		X					X	X		X									X			X			
38	Media Bloc			X			X		X																	X			X	X	?	X
20	Hinckley		X	X	X				X					X	X		X	X	X		X	X										
34	SyncTap		X	X	X	X			X					X			X														X	X
1	TranSticks		X	X			X		X					X			X															X

Figure 2.1: Comparison between toolkits supporting DUIs[ROU06]

2.0.2 Mobile and Intelligent Environments[DEE10]

Walter Dees has worked on mobile and intelligent interaction environments [DEE10]. He studied the importance of the usability and compared the difference of perception between partial and total distribution and with consistency or inconsistency. The figure 2.2 shows the results of his study. It shows that ergonomic is also an important aspect in Human-Computer Interaction.

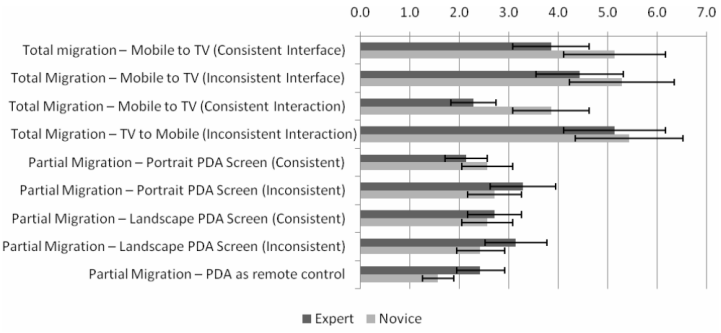


Figure 2.2: The ergonomic aspect of the User Interface[DEE10]

2.0.3 The Fiaa Platform Model

Another paper on modeling is the Fiaa Platform Model[QIU09]. The Fiia Platform Model described is based on a publish and subscribe architecture. Each devices may subscribe to information it is able to use or display. Less-powerful devices may only get partial and more specific information from parts of the application while more-powerful ones will user the complete application with the information related to it. Each device which is represented as a node of the Fiaa Platform Model which stores its own platform information. The resource model of this Fiaa Platform Model is depicted in Figure 2.3.

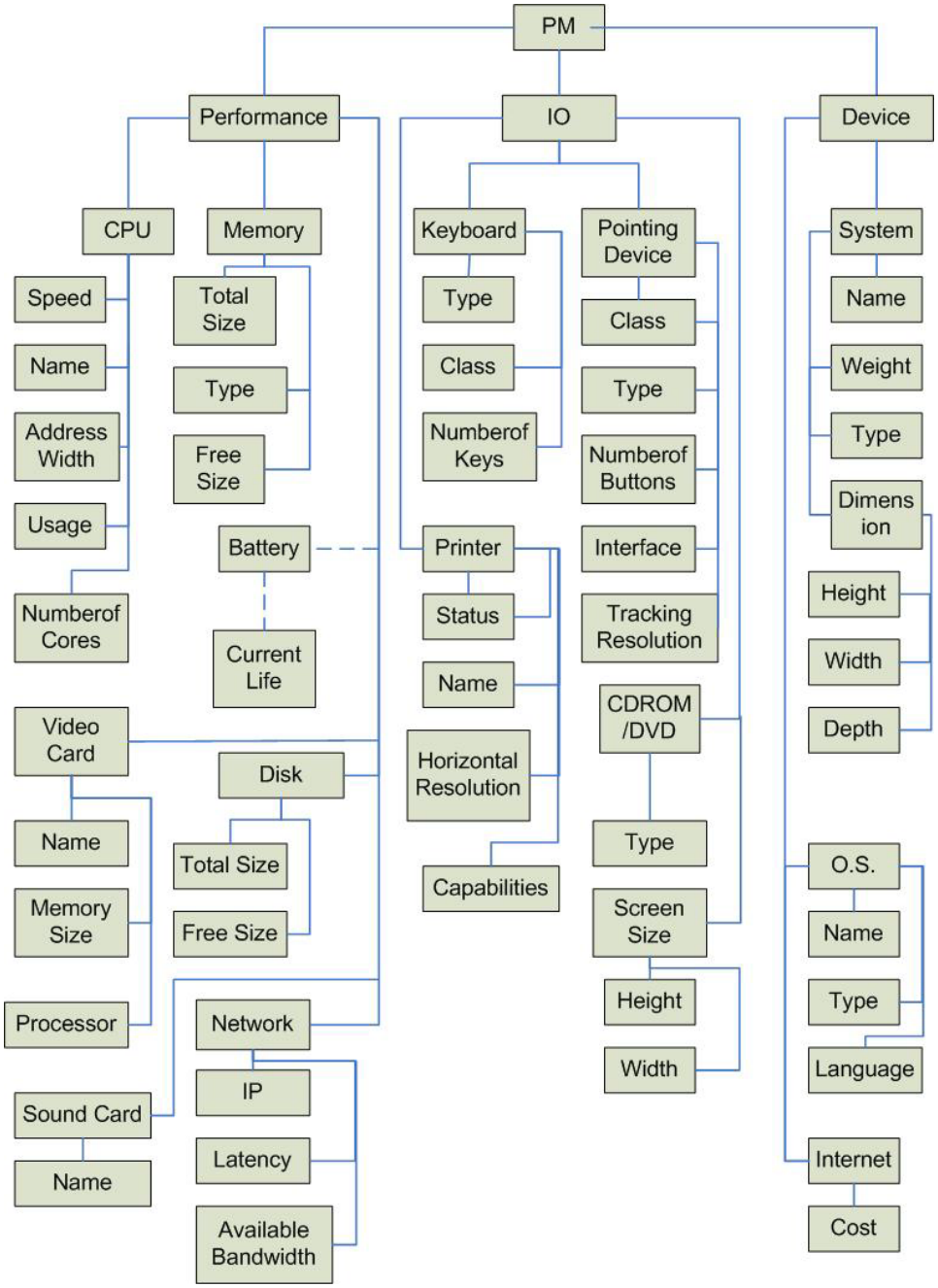


Figure 2.3: The *Fiaa Platform Model* resource model from [QIU09]

2.0.4 An AUI model to support DUIs

An new Abstract User Interface (AUI) model has been developed in order to support DUIs[PENA11B]. They define the *Status* of a UI as the temporal point in which the UI lies after using part of the UI's elements. They also define the *State* of a DUI as being the combination of all the status of UIs that composed the DUI. Their AUI model provided in Figure 2.4 allows the separation of *interactionElement* along with their *subTarget*.

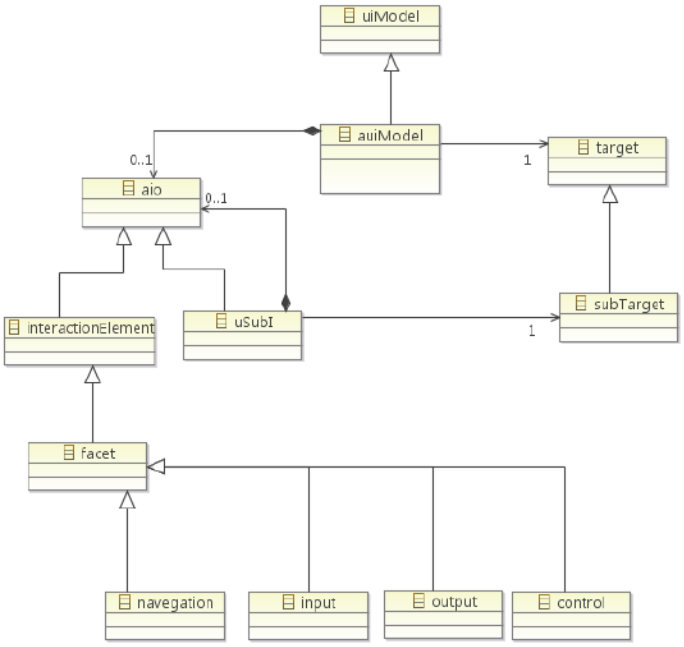


Figure 2.4: Their AUI model with the DUIs perspective from [PENA11B]

Although the research is at an early stage, this model supporting DUIs is the base of an approach that they are currently building.

2.0.5 FRESCO

FRESCO[LIN93] is a set of programming interfaces that expand the X window system in Linux. It models UI components as objects in order to allow users to run an application on a remote machine. It allows the partial management a DS. While *X* provides network-transparent access to some user's display remotely; Fresco allows the distribution of components from UIs across several devices.

2.0.6 CESAM

CESAM[ROU06B] is a prototype to show how a UI could be extended to support some operations to distribute the UI.

In CESAM prototype, it is possible to provide the following information about a device: a name and, the width and the height of the screen resolution as in Figure 2.5. There is also other information such as a place, and the interconnection between devices. The two devices that are in the example have a different form factor: there is a laptop and a PDA.

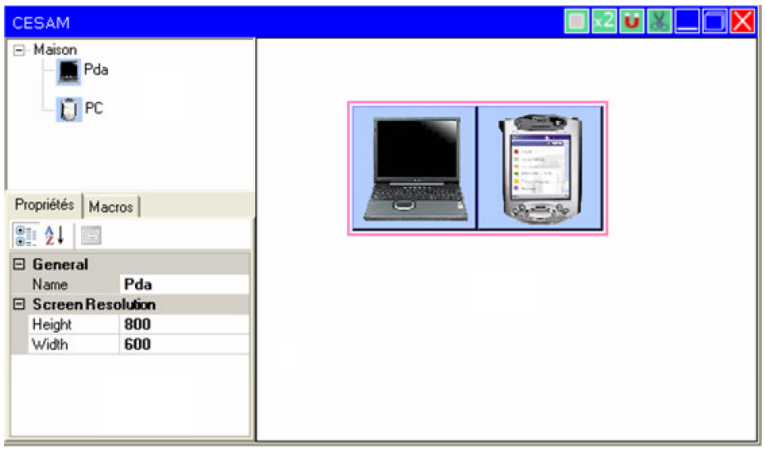


Figure 2.5: A print screen of the CESAM prototype with two devices connected[ROU06B].

Via a drag&drop gesture of an item from the upper-left part to the right part of the window you can assemble devices together. In the example the two devices are linked together which is visible thanks to the red border that encapsulates them. For devices to be assembled they have to be in the same logical space. In this case the PDA and the laptop are in the space called *Maison*. CESAM also allows users to cut the UI into several parts. An example is provided in Figure 2.6.

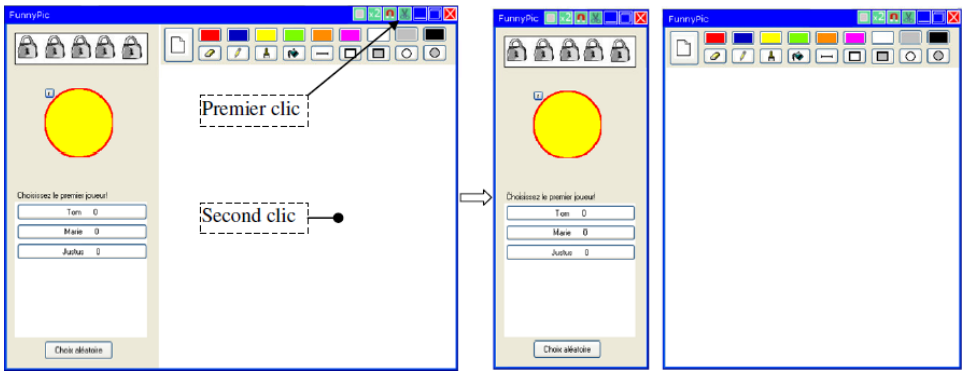


Figure 2.6: A print screen of an application that has been cut into parts[ROU06B].

2.0.7 Windows snipping in MME[HUT07]

Hutchings and co. have worked on gaining simplicity in multi-monitor environments (MME) [HUT07]. In Figure 2.7, there are two different representations of four windows. The left part shows a MME without any improvement. In the other image there is free space in about half the screen size. The free space allows users to improve readability to find the right information. Windows are snipped to avoid displaying uninteresting part of the windows.

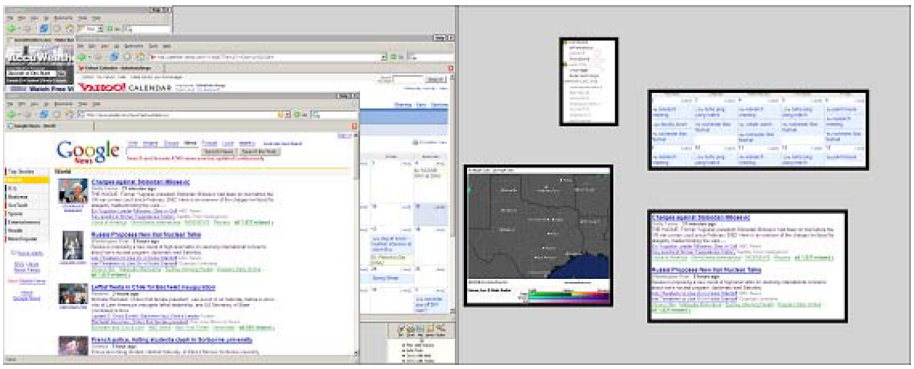


Figure 2.7: Example of simplicity gains from research on DUIs[HUT07]

2.0.8 ARIS

The ARIS[BIE04] interface allows users to relocate applications across a fixed disposition of devices.

In their setup, they have 5 screens hold by walls, with a PDA and two tablets. The tool provides an iconic map as a visualization of the static DS as in Figure 2.8.

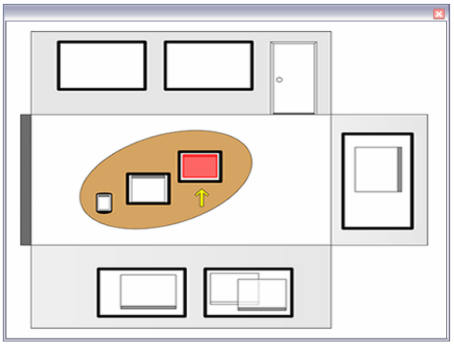


Figure 2.8: The iconic map in ARIS[BIE04]

2.0.9 IMPROMPTU

ARIS has been extended to an interaction framework in 2008. They renamed it to IMPROMPTU[BIE08]. It is a simplification of the interactive space as in Figure 2.9.

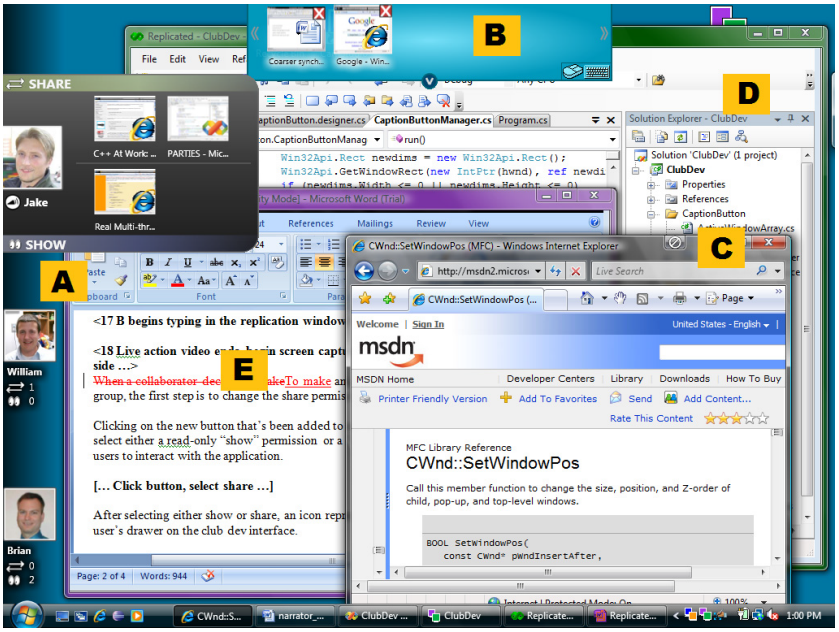


Figure 2.9: Screen shot of the IMPROMPTU's UI[BIE08]

Users have the ability to share a view (read-only or editable) of their windows. Applications can be used remotely and each device is a server for its own applications. IMPROMPTU brings a specific UI (Figure 2.10) to support collaboration between several users.

There is a view on the windows shared by any user (Figure 2.10a), and there is a view on the windows shared by a user from the device where it is displayed (Figure 2.10b). The windows sharing is possible by capturing application window's pixel data and reproducing it on other devices.

2.0.10 GUMMY

Gummy[MESK08] is a tool for building generic multi-platform GUIs. It starts with an initial GUI on a device. Then it adapts and combines its features into a new GUI for another device. It allows people to target new devices and keep UI consistent without requiring designers to start from scratch. A screen shot of Gummy is depicted in Figure 2.11.

In order to generate a Final User Interface, Gummy starts with a UI description which is converted thanks to a UIML vocabulary into the Final User Interface (Figure 2.12).

Figure 2.10: IMPROMPTU's extra-UI for windows sharing (a) and displaying (b) [BIE08].

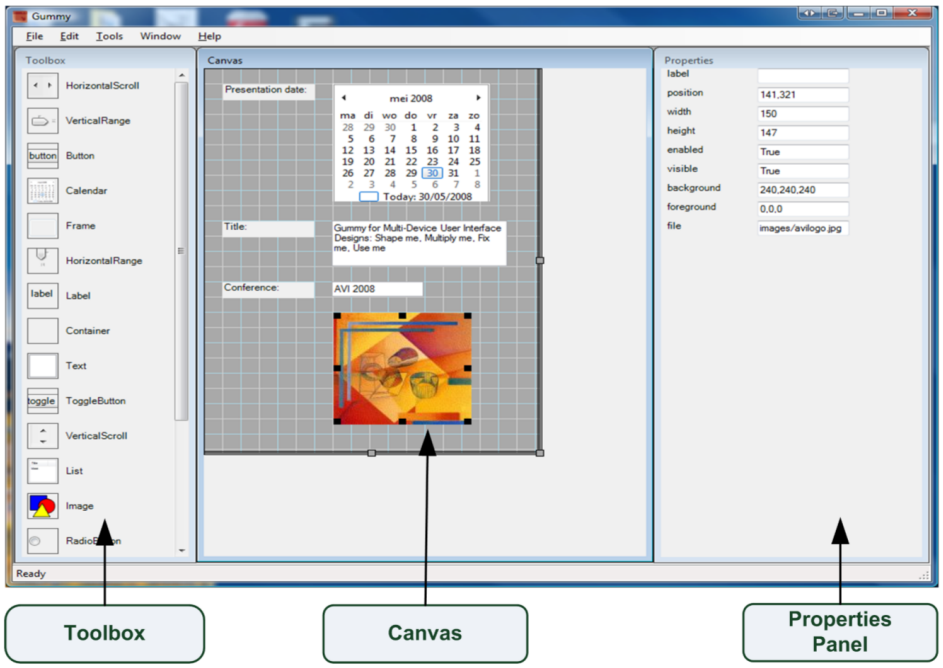


Figure 2.11: The three main dialogues of the Gummy tool[MESK08].

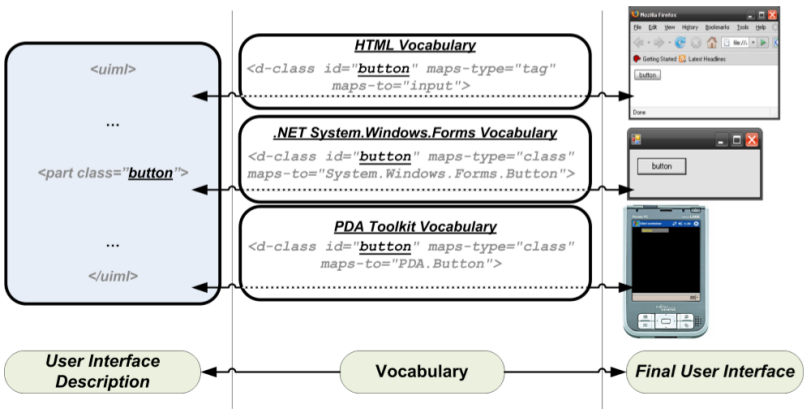


Figure 2.12: A UIML vocabulary relates generic terms to concrete representations [MESK08].

2.0.11 Light-weight Services

Light-weight Services [VDV05] supports multi-user collaboration thanks to an HTTP-based daemon allowing the distribution of web applications. It offers distribution of web applications through services. In Figure 2.13, there is an example of a website that is distributed across three devices. The zoom services is currently distributed on the device on the left.

This toolkit allows the full control of the distribution through user-driven distribution and support automatic distribution through system-driven distribution. They allows automatic redistribution in case of changes in the interaction space [VDH08C, VDH09]. The data and the UI of a disconnected device will not be lost, thanks to the redistribution of this part. The figure 2.14 shows an additional UI to control the application's UI, one of the UI and the list of tasks provided by the example.

2.0.12 MASP

The Multi-Access Service Platform (MASP)[ROS09, ROS09B, BLU10] is an additional UI to control Smart Environments. It allows the control of the UIs from their SHEA assistant (Smart Home Energy Assistant) through an additional UI (Figure 2.15) based on a graphical representation of the different tasks that a user can display on a screen. Supported distribution primitives are involved in four services: migration for transferring a service from one interaction resource to another, adaptation to an interaction resource, distribution of UI elements across interaction resources, and multimodality. It allows the distribution of a task on a screen. They also support a dynamic environment and support multimodal interactions with devices.

It is based on services to allow a lot of flexibility. The main goal of the

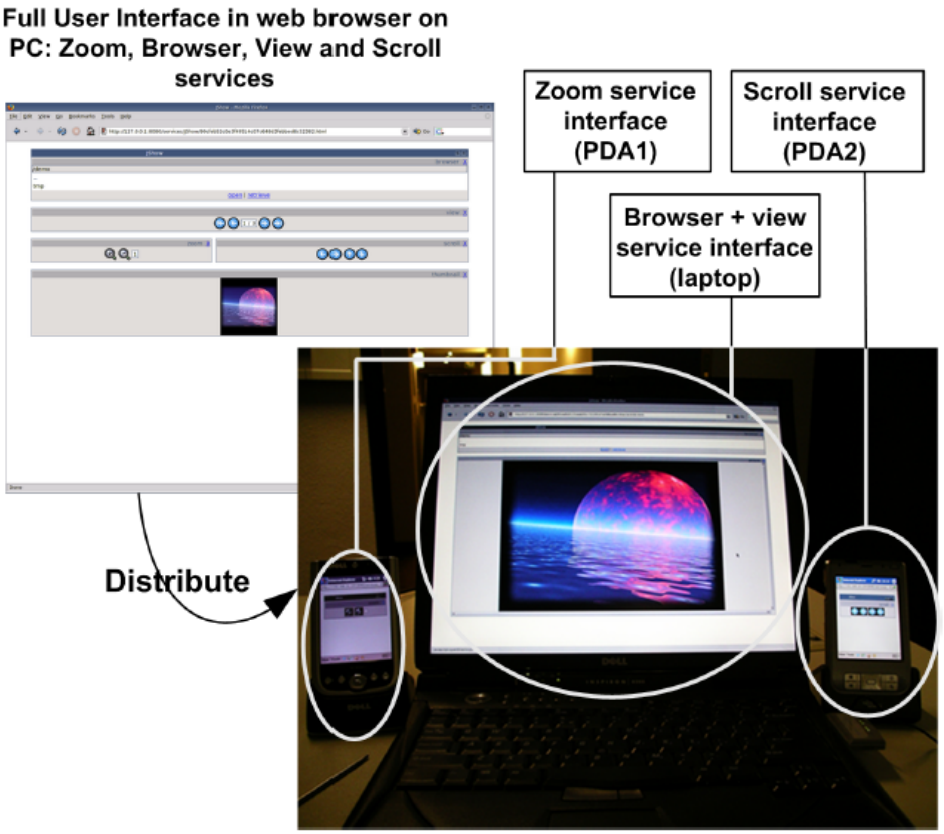


Figure 2.13: An example of website distributed across 3 devices thanks to the Light-weight services[VDV05].

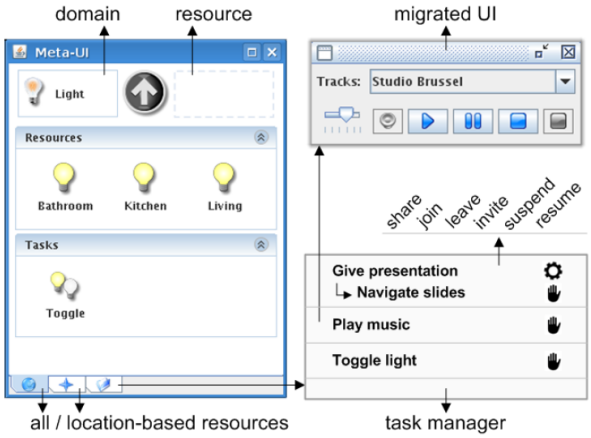


Figure 2.14: The additional UI and the UI associated with the task *play music*[VDH08C].

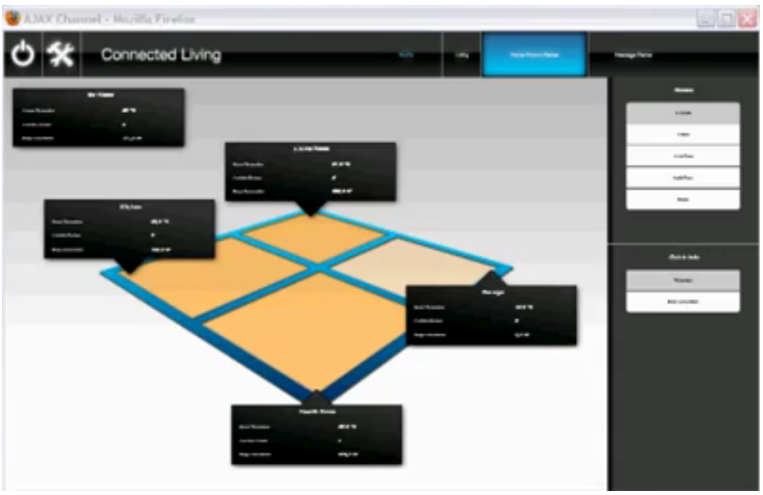


Figure 2.15: A screen shot of MASP's UI[MASP].

application is the distribution of User Interface to allow the move of some services UI from a device to another.

The description of the context with the room, and each users, is depicted in Figure 2.16.

2.0.13 Web sites and applications

Web sites and applications have been particularly investigated through the angle of distribution primitives with the Migration project [MOS09], Cameleon-RT [BAL04]. Most of the time, the extra-UI is separated [ROU06] from the UI subject to remolding or distribution.

Figure 2.16: A screen shot of the description of the context[MASP].

2.1 The related work along the three dimensions

2.1.1 Modeling for Distributed Systems

The first dimension we have explored is the model dimension. There are several concepts that can be modeled (e.g., users, tasks, context, environment, devices, connections).

Distributed Systems

The concept of a DS has been widely used in the related work however there are several terms that refer to it.

It was first introduced as *ubiquitous computing* in this famous statement: "Specialized elements of hardware and software, connected by wires, radio waves and infrared, will be so ubiquitous that no one will notice their presence" [WEIS99].

Many papers instead used the term *Interactive Space*. An Interactive Space is a DS where users can interact with all the devices. But today there are DS where users cannot directly interact with all the devices (e.g., internet, cloud computing). They can only remotely access them. An interactive space is the subset of devices in a DS that users can interact with.

Later, the concept of Interactive Space has been replaced by *Multiple Display Environments* (MDEs) or *Multiple Monitor Environments* (MMEs) as in IMPROMPTU[BIE08]. The concept of user disappears and the only remaining concepts are the displaysand applications' GUIs. An MDE is also a subset of a DS where the users are not represented and devices are only represented as displays. A device that has two displays will be considered as two different entities in this concept.

There is also a concept called *Multi-Surface Interaction*. It has been introduced in [COU03B]. Public walls, blackboards, desks and tables, the back of an envelope are some of the surfaces that can be used. In com-

puting science, these words define the same areas. However they can be augmented with computational capabilities. Each surface has its own interaction properties. To define the concept, they first introduce the concept of an information surface which represents a physical surface able to display information such as digital information. They call it a multi-surface interaction, when the information surfaces can be manipulated through a UI.

These concepts are also very close to our definitions of a device and a DS. Information surfaces as well as multi-surface interactions are some kind of devices. A device is an information surface if it has the ability to show information (i.e., the device has a display). The multi-surface interaction is the set or a subset of all the devices that support interactions in the DS.

In Figure 2.17 the distributed system set is represented as a super-set of all the other terms.

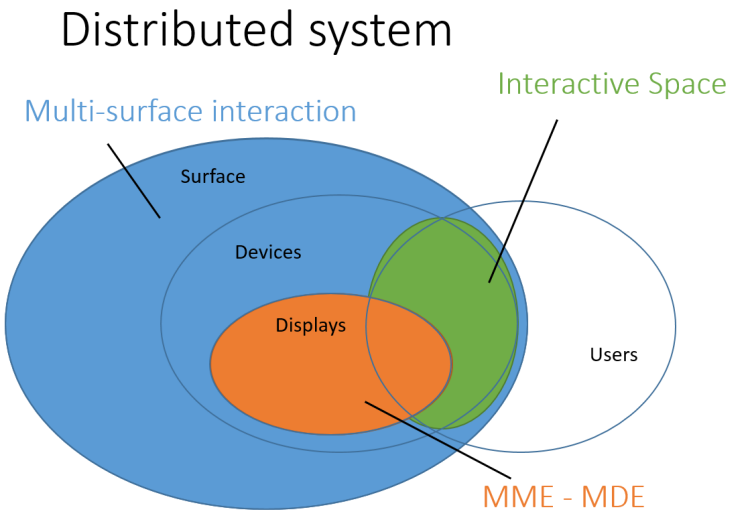


Figure 2.17: The distributed system set and the other sets

Lack of support for the dynamic aspect

Most of the papers consider the DS as something static and already known. In ARIS[BIE04], they have a iconic map with one PDA and two tablets. What will happen with their iconic map if we want to add another device in the DS? They do not support the dynamic aspect of a natural environment: users and devices can appear and disappear at run-time.

Also in all the papers we have been through, the connection between devices is considered as perfect. This means that the connection is assumed as permanent but in a real world there are failures and delays. None of the existing studies provide a tool to deal with these failures and delays.

Representation of a Distributed System

There have been several attempts to represent the configuration of a DS. Since the early years of computing science, the basic configuration was one user on one device in one environment. A basic scenario for this configuration is that a user wants to accomplish a task with the help of a tool. This is the model of the four principal components: the user, the task, the tool and the environment[SHAC09] (Figure 2.18).

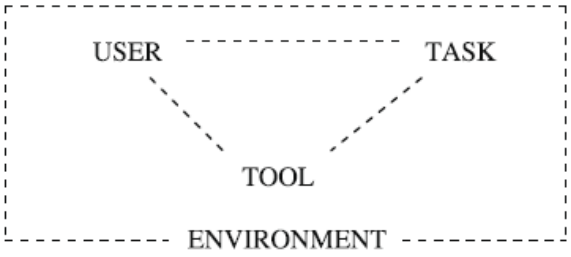


Figure 2.18: The four principal components in a human-machine system [SHAC09]

Later the term *tool* has been replaced by the term *device* which encapsulates the applications that run on top of it. The task has been dropped since it is a goal to achieve and not an agent in the system. This has led to the definition of the *context of use*[CAL04, COU05B, DEY00, DEY01]. The context of use has been defined as a triplet $C = (U, P, E)$ where there is one user, one platform and one environment. In this definition a platform can be a cluster of platforms (e.g., a desktop computer with a tower, a monitor, a keyboard and a mouse). A context of use is always attached to one and only one user.

To deal with today's reality there is a need to extend this highly-used concept to take care of all the components of a DS. In a DS, there can be several users, several devices and they can dynamically appear, disappear, join and leave at any time. The term platforms and devices are equivalent.

In the Painter's Palette example there is only one user and one device at the beginning. The current definition of the context of use does not allow us to represent a second device, or another user. Multi-device and multi-user scenarios appear more often than in the past. The support of both kinds of scenarios could be added to the representation.

At first we wanted to extend the definition of the context of use to a distributed context of use. A distributed context of use would have been $C_d = (U_d, P_d, E_d)$ where U_d is the set of Users, P_d a set of Platforms and E_d is the environment that contains all the users and platforms. Each user would have got his/her own context of use and environment.

Configurations like in Figure 2.19 would have then be possible.

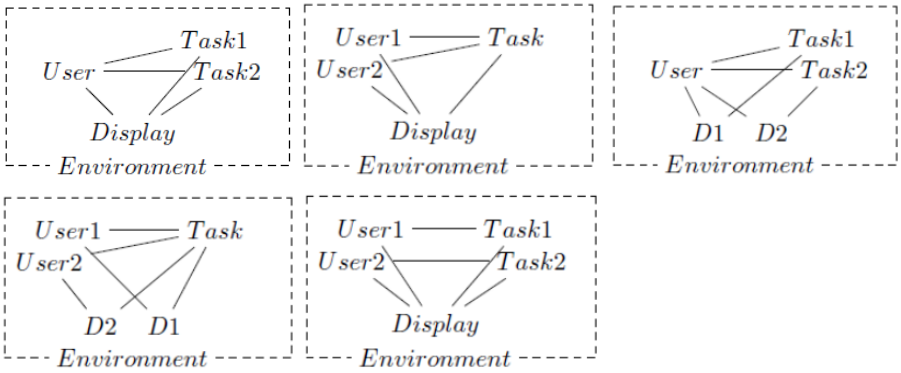


Figure 2.19: Possible distribution with two users, two displays and two tasks

This extension would allow the representation of the user with more than one device in the environment. However this definition does not support any dynamic behavior and there is no way to model the interactions between users and devices.

Thus we wanted to add the concepts of events and actions that happen between all the entities that are in the environment. For this reason we have decided to introduce the concept of a DS which contains all the contexts of use and allows the representation of events and actions. Each user has still his/her own context of use where the device can be a cluster of devices (e.g., a desktop computer with a laptop and a smartphone). This allows each user to have a set of devices and choose how he/she wants to interact with them.

Task life cycle

Russel and co.[RUS05] have introduced a workflow based on a classical task life cycle. A task corresponds to a single unit of work. The classical task life cycle is a task having the following states: *created*, *offered*, *allocated*, *started*, *suspended*, *failed*, *completed*. As depicted in Figure 2.20 a task first need to be created. Then the task can be offered to one or several resources. The task can then be allocated to one and only one resource. After being allocated, the task is then started and will go to completion if not canceled or failed. While started a task can be suspended and resumed at any time.

A *distributed* task is a task whose resources are not centralized in a single device. We have extended this task life cycle to support distributed tasks. The corresponding workflow is depicted in Figure 2.21.

In this workflow, we have introduced three types of nodes:

- *Initial state* as a white oval, that represents any state where the life cycle could be initiated.
- *Active state* as a blue oval, that represents any state where the task is

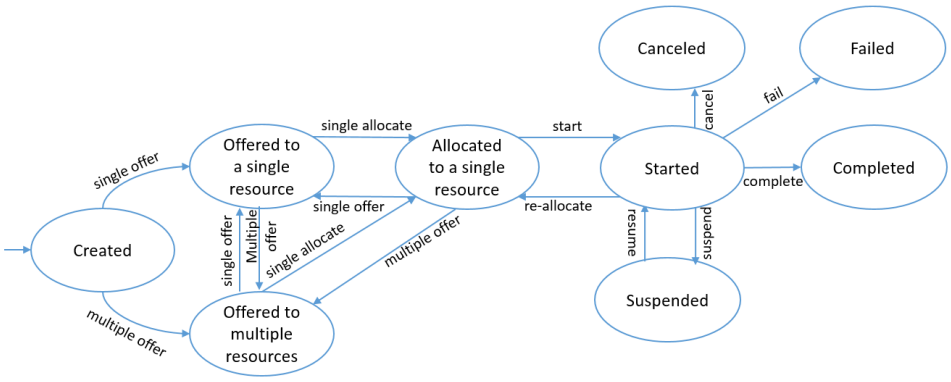


Figure 2.20: A workflow representing the life cycle of a task[RUS05].

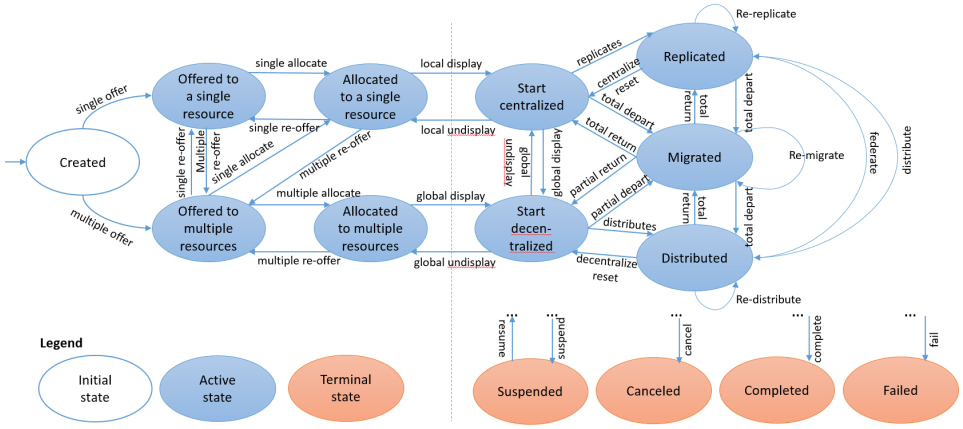


Figure 2.21: A workflow representing the life cycle of a distributed task.

active or in progress which could lead to any terminal state.

- *Terminal state* as a red oval, that represents any state where the life cycle could be terminated.

A distributed task can also be offered to a single or several resources. All these resources can allocate a distributed task. This leads to a task being allocated to multiple resources. If a task has been allocated to only one resource, the task is centralized. However if a task has been allocated to several resources then the task is decentralized. As soon as the task is started the task will reach the corresponding status of distribution. A centralized task can become decentralized if it becomes allocated to other resources. The opposite is also possible if other resources gives the allocation back to a single recourse.

The distribution of the task becomes more complex when the task is either:

- *replicated*, several resources get a copy of the task;
- *migrated*, the task is moved from the allocated resources to other resources;
- *distributed*, the task is spread across all the resources but is not strictly replicated or migrated.

2.1.2 Tools for creating DDGUIs

In this section we come back to the tools that partially support DDGUIs and that were previously described. We will explain what are their limitations and what we are going to support from their work.

FRESCO

FRESCO[LIN93] is an old tool. In FRESCO it was already possible to distribute the UI. Unfortunately it has had a limited availability. Indeed it was only compatible with X which is only present in some Unix operating systems.

CESAM

CESAM[ROU06B] is a prototype which means that there is no available version of CESAM. This prevented us from testing how the tool would work.

In CESAM it is possible to use different form of devices running different operating systems. However no information about the operating system is be provided.

The connection between devices is a manual process. It is not possible to know the status of the real connection between the devices.

IMPROMPTU

IMPROMPTU[BIE08] also has a limited availability. It only supports one operating system (Windows).

The way IMPROMPTU works does not cover distribution. It only enables sharing capability of windows with several users.

There is also no support for failures and delays. However this would not be useful as each device always keeps its applications.

This tool is interesting for collaboration but has a few drawbacks that prevent it from being the solution for our Painter's Palette example. Indeed it is not possible to independently manipulate the view within a shared application. It is also not possible to separate a window into several windows or merge two windows into one single window. In the case of multi-touch drawing area, this would remove the multi-touch capability of the device using the shared view of the drawing area.

GUMMY

GUMMY[MESK08] is a multi-platform user interfaces designer. The goal of this tool is not to distribute UIs.

We think that the choice of using UIDL in GUMMY is really important. Adding the ability to distribute their UI description would easily extend the tool and allow the creation of DDGUIs.

MASP

MASP[ROS09] is the most advanced tool for migrating UIs across devices. However this tool does not give any choice of the UI that will be displayed. It is not possible to control the distribution in finer granularity. It only focuses on one single task and has a limited multi-user interaction.

There is also no support for collaboration. MASP is an application. It is not a toolkit that developers can integrate into their own applications.There is no way to separate, copy, create UI at run-time.

Light-weight services

In Light-weight services[VDV05], each element can be distributed on only one device at a time. It is not possible to display on another device without moving it from the current device.

2.2 Summary

After a thorough reading of the related work, we did not find any toolkit managing the dynamic aspect of the environment while offering a vast way of controlling the distribution of the applications. There is almost no support for devices and users entering and leaving the system at run-time and for dynamic evolution of the environment. Either it is a static environment or with pre-programmed and known evolution.

In this section we will summarize the lacks we found in the related work in order to use it as a base for specifying requirements that our tool should take care of.

2.2.1 List of shortcomings

Here is the list of shortcomings for the model dimension (MS_i):

- MS_1: *Lack of ontological understanding of DDGUIs*: Fundamental concepts that are underlying to DDGUIs are rarely defined or are not solid enough for defining an ontology. When such concepts exist, they often appear as heterogeneous from one implementation to another. For example, the concept of user may sometimes refer to a human

person in front of a computer, sometimes to a computing service. The concept of *device* is confused: is it the hardware part, the software part or both parts together?

- MS_2:*Lack of DDGUI modeling*: Few DS have been modeled. The way it is often modeled is not sufficient and too dependent on the case study researchers put their focus on. Many papers have been using the definition of context of use[COU05B]. However this definition limits the system to at most one device and one user which is no more sufficient in a DS.

And then the shortcomings for the approach dimension (AS_i):

- AS_1: *Predefined or limited distribution*: The scenarios of distribution are often created at design-time preventing distribution from happening at run-time. It is specific and limited to the domain of the scenarios [AYA00, GRO05, HAN00, NEW02, SJO04, TAN04].

- AS_2: *Limited systematical approach in DDGUI development*: DDGUIs are rarely developed in a principle-based way, they are developed in an opportunistic way that results into a complex DDGUI like in [BIE08].

- AS_3: *Limited re-usability of existing tools and approaches*: As a consequence of the shortcoming MS_2, DDGUIs are implemented for very specific circumstances (e.g. specifically for an application, a task, a domain, or on a specific device) that gives rise to little or no re-usability of these development efforts for another application. This means that neither the models, the concepts and fragments of code are available for other projects.

- AS_4: *Lack of basic distribution operations*: DDGUIs are not necessarily implemented on a RISD (Reduced Instruction Set for Distribution), thus allowing little or no reuse of these basic distribution operations and limited possibility of incrementation for their development.

- AS_5: *Difficult extensibility*: Since DDGUIs are implemented with limited re-usability in mind, it is also hard to expand an existing DDGUI in order to support a new user, a new device, a new task, etc.

- AS_6: *Little or no separation of concerns*: When different aspects of a DDGUI are subject to distribution, e.g., task, user, and device, these concerns are too often intertwined.

- AS_7: *Lack of integration*: DDGUIs are rarely integrated with themselves, thus limiting composition or decomposition.

Shortcomings with respect to the software support dimension (SS_i):

- SS_1: *Limited availability of DDGUI*: UI elements stay in initial context. It is not possible to merge two applications created with the same toolkit. They communicate with each other but without any possibility to be rearranged[VDV05, BIE04, BIE08].

- SS_2: *Coarse grain distribution*: When they are effectively developed, DDGUIs provide a distribution granularity that is often limited in scope and coarse grain[BIE04, BIE08].

- SS_3: *Lack of repeatability*: Some tools do not allow anyone to change the way the UI is distributed or to redistribute it. E.g., the distribution of an visual element is possible but can only happen once or one element at a time, or the element cannot be distributed on two devices at the same time[VDV05].

2.2.2 Comparison of software support

Based on the shortcomings we have just listed we have made a comparison of the main software support for building GUI, DUI, DGUI, DDUI and DDGUI (see Figure 2.2.2).

Softawre support	Working	Availability	Models	Approach	DDGUI
FRESCO	●	○	○	○	DGUI
CESAM	○	○	○	○	DGUI
Windows snipping	◑	○	○	○	DGUI
ARIS	●	○	○	○	DGUI
IMPROMPTU	●	○	○	○	DDGUI
GUMMY	●	○	○	○	GUI
Light-weight services	●	○	○	○	DGUI
ReWiRe	●	○	◑	◑	DGUI
MASP	●	○	○	○	DGUI
JayTk	●	◕	●	●	DDGUI

Figure 2.22: Comparison of software support

A software support is considered as fully working (●) if there are demonstrations that have been accomplished with the software support. It is only working half (◑) if the concept is pretty clear but there is no proof of a working demonstration behind the concept. Prototypes are not considered as working (○) as they are not really software.

A software support is either available (●), or close to release (◕) or not available (○).

A software support may have some models or an approach that is recommended and use along with it (●). If not this means that there is no models and approach support (○).

Along with the state of each software support we also wanted to reuse the table from Meta-UI for Ambient Spaces[COU06]. We propose an update of this table which expresses the possibility to display, undisplay, replicate, move, move back, migrate and distribute the GUIs (see Figure 2.23).

In this table we can see the state of each possibility where the numbers correspond to these states:

1 Unsupported
2 Observable
3 Traceable
4 Controllable
5 Flexible
? Undetermined
- No information

		Display		Undisplay							Depart		Return				
		Local	Global	Local	Global	Replicate	Re-replicate	Centralize	Decentralize	Federate	Partial	Total	Partial	Total	Re-migrate	Distribute	Re-distribute
FormsVBT	1989	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1
CPN2000	2000	3	3	3	3	1	1	1	3	1	3	3	3	3	1	3	1
iCrafter	2001	3	3	3	3	3	1	1	3	1	1	1	3	3	1	3	1
Proximal UI	2003	4	4	3	3	4	1	1	1	1	3	3	3	3	1	3	1
Aris	2004	4	4	3	3	1	1	1	3	1	3	3	3	3	1	3	1
JigSaw	2004	3	3	3	3	3	?	1	3	1	3	3	3	3	?	3	?
AttachMe/ DetachMe	2005	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4
Lightweight Services	2005	4	4	4	4	1	1	1	1	1	1	4	1	4	3	3	3
Diamond	2005	4	4	4	4	3	1	1	3	1	4	4	4	4	1	3	1
CESAM	2006	3	3	3	3	3	3	3	3	3	4	4	4	4	3	4	3
MigriXML	2006	3	3	3	3	1	1	2	4	2	4	4	2	2	1	4	1
Impromptu	2008	3	3	3	3	3	?	2	3	2	2	4	2	4	?	3	?
MASP	2009	4	4	4	4	3	1	2	3	3	3	3	3	3	1	3	1
DistriXML	2009	4	4	4	4	4	2	4	4	4	4	4	4	4	2	4	2
Hugin	2010	4	4	4	4	4	?	?	4	4	4	4	4	4	?	4	?
IUDSM	2014	4	4	4	4	4	1	2	4	3	4	4	4	4	1	3	1
JayTK	2015	5	5	5	5	4	3	4	4	4	4	4	4	4	3	4	3

Figure 2.23: Update of the comparison of Meta-UI[COU06].

2.3 Survey on User Preferences for additional UI

The dissemination of the related work lead us to conclude that there is a common ground that would benefit from both a theoretical and empirical analysis. Dearman & Pierce [DEA08] report in their US study how 27 people from academic and industrial research were using their devices: they revealed that on average they employ more than five computing devices in four different configurations. Participants reported managing information across their devices as the most challenging aspect of multiple devices[DEA08], thus encouraging improvements for an additional UI that effectively and efficiently supports these capabilities.

These results suggest that functions should be provided for the end user on how to share UI portions. They could share them across several devices and contexts. It is also important to show them how to divide them[HUT06].

In order to address the aforementioned shortcomings and the problem independently of the application domain as possible, we have described a comparative evaluation of user preferences for additional UIs in a journal paper[MEL12-IJHCI].

Roudaut & Coutaz [ROU06] has established a state of the art for the domain of DUIs. In this article the term *Meta-UI* is used for additional UI. They compared 25 additional UIs in ambient intelligence against criteria representing: (i) the objects manipulated by the additional UI (i.e., their nature and manipulation), (ii) the additional UI external presentation (i.e., embedded or not in the final UI, offering observability and/or predictability), (iii) capability to ensure services (i.e., resource discovery, assembly, distribution, remolding, and parameterizing), and (iv) their expandability. This survey revealed that many additional UIs exist that provide similar or dissimilar capabilities, but with very different metaphors and interaction styles, thus suggesting a further study. Their survey compared high-level services, which suggests that also low-level services could be used as new comparison criteria to complement the survey. Vanderhulst et al. [VDH09] pioneered the field by conceptualizing the first reference framework for additional UIs for pervasive systems. In this framework, an additional UI runs on devices that offer services depending on tasks executed by users. The environment, the interaction resources, and the domains resources are then characterized and mapped. Based on this framework, we would like to further examine the services offered and the way they are offered to determine whether there are some aspects affected by the users carrying out these corresponding tasks.

2.3.1 Experimental study

We have defined and applied a procedure for conducting a comparative analysis of the state of the art in order to identify which basic services are supported by each tool/toolkit, which interaction styles are used and for these services in order to discuss the rationale behind these usages.

This procedure consists of the following steps:

1. Literature review: All references identified in [ROU06] were initially selected and complemented with related work (only references taken from DataBase systems and logic programming (DBLP)).

2. Classification of references: We listed all basic services found and classified them into four categories:

   (a) Simple primitives [Set, Display, Undisplay, Expose]

   (b) Basic primitives [Copy, Move, Switch, Permute]

   (c) Advanced primitives [Merge, Split, Replace, Distribute, Reset, Append]

(d) Management primitives [Save, Restore, Import, Export]

3. Reduction of scope: The criteria used for selection is any article recognizing the need for distribution of any kind, describing explicitly an implementation, and working at the user interface level (other levels are relevant to distributed computing). A final comparative analysis was obtained in Figure 2.24. In this figure, *P* stands for *partial support* and *T* stands for *total support*.

Name, reference, and year	Interaction style	Set	Display	Undisplay	Copy	Move	Replace	Transform	Merge	Switch	Separate	Distribute	Reset	Import	Export	Expose	Append	Permute
FormsVBT (1989)	DD,DM,MW,CL	T	T	T		T			P	P							P	
CPN2000 (2000)	PL									P	P	P					P	
Proximal UI (2003)	DD,DM,MW		T	T	T	T	P										P	
Aris (2004)	MW,II					T						P		P	P			
JigSaw (2004)	DD,II		P	P	P	P					P	P					P	P
AttachMe (2005)	II,MS		T	T	T	T				T			T			T		
Lightweight services (2005)	MS		P	P		P			P		P	P					T	
CESAM (2006)	DD,II				P	P	P	P	P		P	P						
MigriXML (2006)	VR,DD,DM		P	P		T						P			T			
Impromptu (2008)	MS		P	P	P	P					P	P			T			
MASP (2009)			P	P		P					P	P						
Ext. Tcl/Tk (2009)	PL	P	T	T	T	T								P	P			
JayTk (2012)	DD,DM,MW,CL	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T

Note. DD = Drag & Drop; DM = Direct Manipulation; MW = Multi-Windowing; II = Iconic Interaction; MS = Menu Selection; VR = Virtual Reality; CPN2000 from Beaudouin-Lafon & Lassen (2000).MASP = Multi-Access Service Platform; CL = Command Lin; Forms VBT from Avrahami et al. (1989);

Figure 2.24: Comparative Analysis of additional UIs: Their Interaction Styles

From this comparative analysis of related work, we could draw this conclusion: distribution primitives are effectively implemented in some studies, in many different ways, with different interaction styles and techniques.

We do not know today which one is prevalent depending on the distribution primitive for an additional UI to become widely acceptable.

Work on additional additional UIs has proven to be very rare (according to Chapter 2) since nothing was found for supporting the manipulation of the abstract UI, the context model, the task model or the domain model with direct implication on the lower levels. One notable exception concerns COMETS [SOT07, CAL04], a toolkit that supports manipulating the models required for running the final UI, either at design-time or at run-time.

Due to the heterogeneity of distribution primitives and their implementation demonstrated in the previous section, this section aims to conduct an experimental study in order to determine the user preferences for particular interaction styles for each major distribution operation. The goal is to provide us information about how users perceive each distribution primitive

and which are the interaction styles that are well understood and which are the ones that lead users to confusion.

Method and protocol

- **Participants and apparatus.** We conducted a user trial of 14 participants (8 female, 6 male) who were recruited from a database of volunteers coming from different disciplines (e.g., marketing, finance, medicine, management) and having different ages (22% between 19 and 25-year old participants, 57% between 26 and 40-year old participants, and 21% of more than 40-year old participants). Participants do not have any prior knowledge of an additional UI. The physical setup was similar for all participants. The computer used for the experiment was equipped with an Intel Pentium M 1.6GHz CPU, 2Gb DDR of RAM and a 15 inches LCD screen with a resolution of 1400 x 1050 pixels. This apparatus was selected as it was considered representative.

- **Task and procedure.** Each participant received a detailed explanation of the experimental study that was uniformly conveyed through an interactive presentation (implemented with the animation language of Microsoft PowerPoint 2010): to express their preference for interaction styles for distribution primitives. Following the short introduction to the test procedure and test purpose, they performed some training with the tool. Following the training session, the 14 participants were presented 6 distribution primitives (i.e. Set, Copy, Move, Replace, Merge, and Split), each with four different interaction styles for each operator (i.e., A=form filling with iconic interaction, B=direct manipulation with drag & drop, C=command language, and D=menu selection). Each interaction style was implemented as animated examples thanks to Microsoft PowerPoint 2010 macro-command language. Each participant received a different sequence of interaction styles in order to avoid the order influencing the results. Each participant therefore provided an answer to the following questions:

  - Which devices have you already used or owned? (desktop, mobile phone, tablet PC, interactive kiosk)
  - What is your experience level with a computer/laptop, mobile phone/smartphone, tablet PC? Interactive kiosk? (preference on a 5-point Likert scale)
  - What is your favorite style for each distribution primitive (A, B, C, or D)?
  - What are the aspects that you found the most interesting, if any? (open question)

– What are the aspects that you found the less interesting, if any? (open question)

– What do you think of using pen-based gestures for some distribution primitives? (open question)

Two examples of metaphors that were presented to the participants for the MOVE and MERGE operations are shown in Figure 2.25.

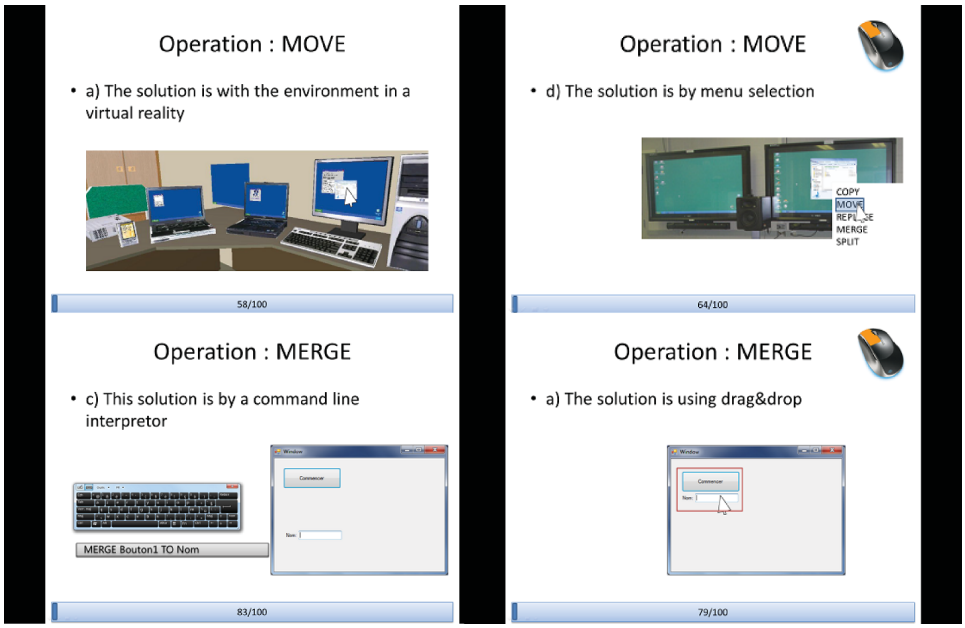


Figure 2.25: Example of metaphors for MOVE and MERGE operations

The dependent variables used to assess the participant task performances were twofold: the preferences for an interaction style for each distribution primitive (i.e., A, B, C, or D) and the comments gathered for each interaction style (i.e., number of positive and negative comments for each interaction style).

All the relevant data captured from the presentation were captured in a log file to be imported in a statistical software package. We now justify these main choices of this setup.

- **Justifications.** When it comes to *evaluate alternative UI designs*, Tullis & Albert [TUL08] report that a couple of self-reported metrics are particularly relevant. One is asking each participant to choose which alternative interaction style they would most like to use in the future for a distribution primitive as a forced choice comparison. Another one is asking each participant to provide comments on each alternative interaction style dived into two classes: what were the most

> positive aspects that you appreciated (if any), what were the most negative aspects that you regretted (if any). This is what we did. In order to consistently analyze all verbal comments, Tullis & Albert's protocol for verbal analysis was used that classifies any comment into three classes: positive (when a clearly positive tone is expressed), negative (when a clearly negative tone is expressed), or neutral (when no clear tone is expressed or in any other case).

The six distribution primitives selected were the most frequently found ones in the literature that were considered fundamental for an additional UI and because of their associated spectrum of interaction styles. Indeed, we did not find all interaction styles possible for each primitive. SET was the first primitive selected for its simplicity and its largest scope of possible interaction styles. COPY and MOVE were also selected for the same reasons in the set of basic primitives. SWITCH and PERMUTE are less interesting because they could be obtained as a composition of other primitives such as MOVE. The three last operations were REPLACE, MERGE, and SPLIT because of their large coverage and representativeness of the aims and goals of an additional UI. The DISTRIBUTE and RESET primitives require too much implementation effort, APPEND primitive is a sub-primitive of MERGE, they were not selected. The four interaction styles were selected based on their intrinsic properties and because some interaction styles are typically combined with others. For instance, the command language is assumed to be appropriate when the task prerequisites are moderate, the task productivity should be high, and so forth.

2.3.2 Results and discussion

The survey was based on a 14x6x4 factorial design: 14 participants were involved, 6 distribution operators were selected and 4 different interaction styles for each operator. All the fourteen participants completed the 24 trials, thus giving a total sampling of 336 trials. No outlier was removed since all tasks have been completed without any problem and interruption. Figure 2.26 summarizes the participants' experiences regarding their usage of various devices, ranging from no experience to expert.

Figure 2.27 graphically reproduces the results obtained for the user preferences:

- Form filling with iconic interaction (A) was considered globally as the most preferred interaction style for advanced primitives taken together (i.e., Replace, Merge, and Split) in terms of occurrences
- a percentile analysis revealed the 50% percentile (x50%) is also in favor of iconic interaction, followed by menu selection, and direct manipulation.

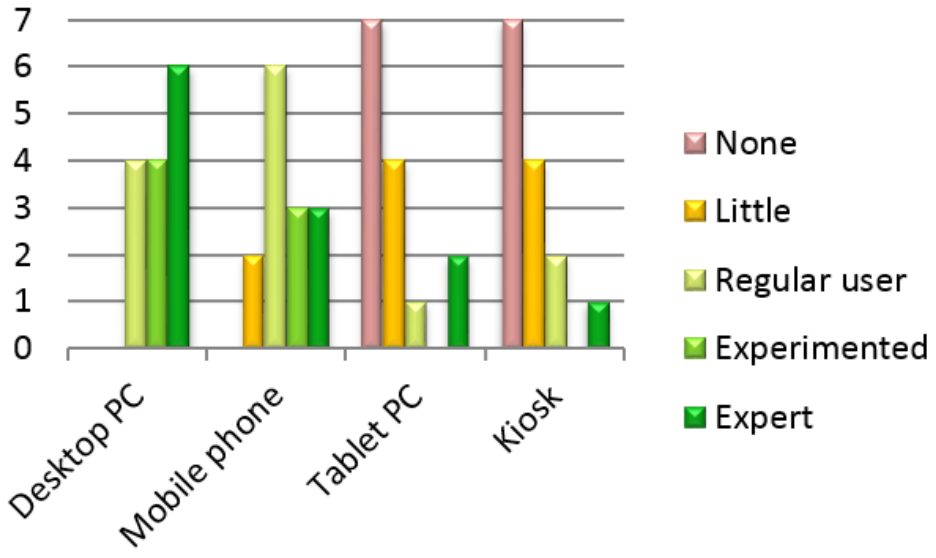


Figure 2.26: Platform experience of participants.

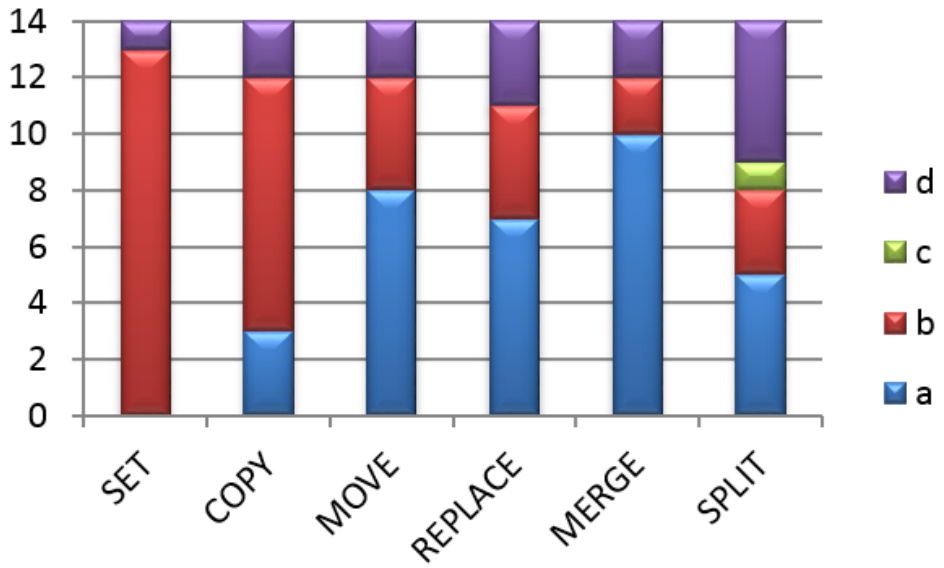


Figure 2.27: Distribution of preferred interaction styles for each distribution primitive (color figure available online).

- Direct manipulation (e.g., with drag and drop - B) is the most preferred interaction style (B) for the Set primitive belonging to the set of simple primitives, probably because of the visual counterpart of the UI element property. This was assessed in terms of occurrences and of

the 90%-percentile (x50%) which is in favor of this interaction style.

- Direct manipulation is also the preferred interaction style for Copy and Move, belonging to the set of advanced primitives. This was assessed in terms of occurrences and of the 55%-percentile (x55%) in favor of this alternative
- Command language (C) remains the least preferred interaction style. However, a one-way ANOVA procedure (F=1.6933, p=0.1588) suggests that only highly experienced participants appreciated this interaction style in general. No other statistically significant correlation was found. Command language was only appreciated for the Split primitive since it involves several complex parameters as opposed to more obvious parameters for other primitives.

Figure 2.28 and 2.29 graphically reproduces the distribution of comments gathered from the participants regarding the overall usage of the 6 distribution primitives (Figure 2.28) and regarding the overall usage of interaction styles for all primitives (Figure 2.29).

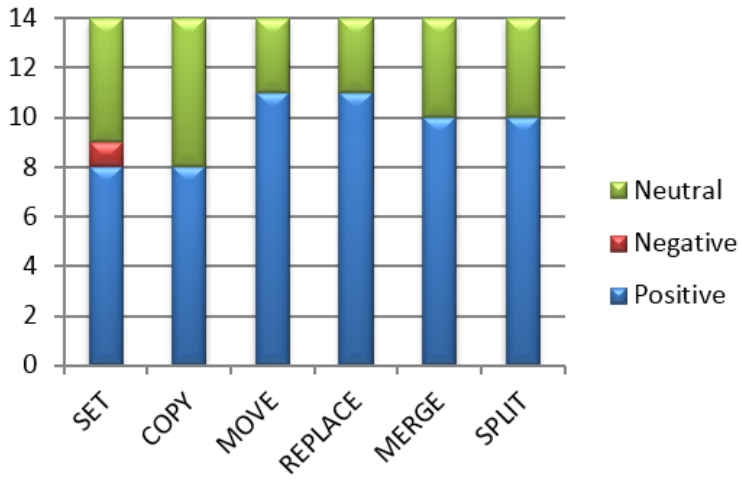


Figure 2.28: Distribution of participants' comments for all distribution primitives.

All the tested primitives are considered vital by participants (in terms of occurrences and a 50%-percentile (x50%) which is in favor globally speaking). Participants' feedbacks are mostly positive for the primitives. Each primitive has always at least one interaction style that makes it easy to use and natural to understand, particularly Drag & Drop.

The styles can vary depending on the context of use, such as the SET operation. There is a question about how precise a primitive like SET should be if we use direct manipulation or Drag & Drop.

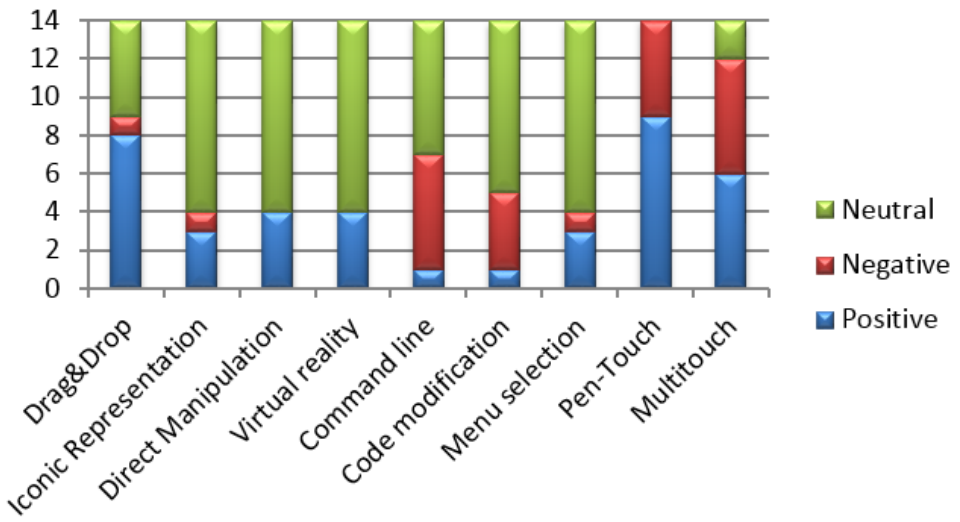


Figure 2.29: Distribution of participants' comments for all interaction styles, explicit or implicit.

The diversity of styles proposed for each operation is well appreciated. If the SET has to be precise, they will use direct values by command language, modifying code or changing values of a property. The results for each style taken individually are less mitigated.

While Drag & Drop is widely the most interesting and natural one for the participants, command language and code modification are again the less interesting and less natural. Participants have approved the other styles but seem less convinced by them. The menu selection has been criticized because it was less intuitive than direct manipulation and Drag & Drop. The ability to simulate virtual reality is very interesting for the MOVE operation.

Participants said that it is nice to have Drag & Drop features in virtual reality. It allows the user to see what she is going to do and to specify exactly where she moves the UI elements. The reactions for pen-touch-multitouch interactions are separated. About a half of the participants likes these interactions because it is more natural and because we have fingers that can be used for that.

There are more positive feedbacks for single-touch than multitouch because participants found multitouch a little less precise and easy to use. The negative feedbacks got for both interactions are the complexity and the feeling of using fingers as interaction mechanisms.

Thanks to this study we have realized that at the time of this study (in 2010) there were very few people who own or use a touch screen. Most people had a computer (desktop PC or laptop). It was not very clear what was the best metaphor for each distribution primitive. The clearest thing

for us was that there is no perfect metaphor for all the primitives.

Another important point raised by the survey is that users do prefer to directly interact with the visual representation of the object rather than using an icon or a menu. Drag&drop and Pen-Touch interactions got the best results while iconic command line interface and code modification got the worst preferences.

We have also tried to see what would be the impact if these primitives were realized thanks to touch or multitouch gestures such as the ones depicted in figure 2.30.

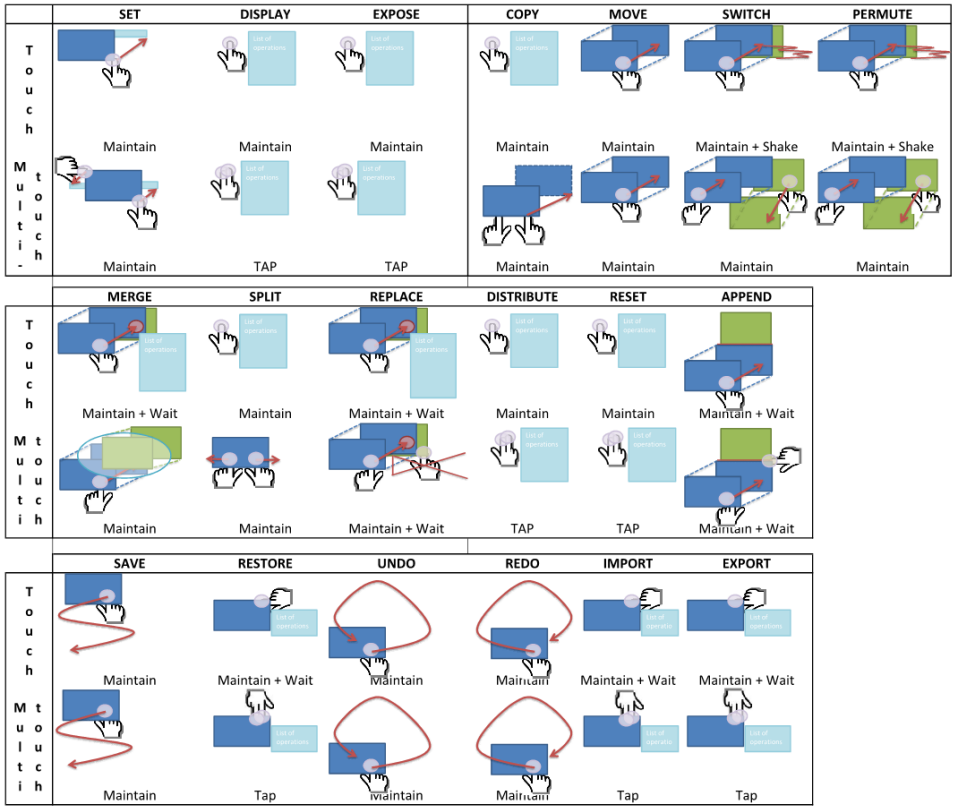


Figure 2.30: Examples of some possible touch and multi-touch gestures.

For each primitive we have provided them with a proposition of possible touch and multitouch interactions. When we found no obvious way to trigger a primitive (e.g., for Display, Expose, Distributed and Reset), we offered the ability to pop up a menu with the list of other available primitives.

While half of the users did like touch and multitouch gestures, some didn't like it at all. This is very important to consider for the future where almost all the device will be touch or multitouch enabled. Applications should not only work with a touch screen but also with a keyboard and a mouse.

As we only focus on GUIs we have not tested other modalities such as using voice to interact with the computer.

2.4 List of requirements

Based on the shortcomings we have identified in section 2.2.1 and with the results of the survey we have conducted in section 2.3. We have created a list of the requirements that we want a toolkit to fulfill. Several shortcomings may lead to only one requirement.

Requirements for the model dimension (MR_i):

- MR_1: *Description of the concepts of a DS*: The fundamental concepts of a DS should be clearly defined. They should be independent of any implementation.
- MR_2:*Establishment a model-based approach for DDGUI*: Describe the models needed for the representation of a DS.

Requirements for the approach dimension (AR_i):

- AR_1: *Support of dynamic distribution*: DDGUI can be dynamically created and distributed at runtime without predefined scenarios.
- AR_2: *Based the approach on models*: The approach must rely on models.
- AR_3: *Basic distribution operations*: The distribution operations must be defined and independent of the implementation.
- AR_4: *Transparent distribution*: The distribution mechanisms should be transparent to the applications.
- AR_5: *Simple distribution mechanisms*: DDGUI should be easily created and manipulated.

Requirement with respect to the software support dimension (SR):

- SR_1: *Availability of the toolkit*: The toolkit should be easy-to-use and available widely.
- SR_2: *Support of different levels of granularity*: The distribution should be possible at several levels of granularity.

The requirements are directly derived from the shortcomings (Figure 2.31).

Shortcomings		**Requirements**
MS_1	$\rightarrow$	MR_1
MS_2	$\rightarrow$	MR_2
AS_1	$\rightarrow$	AR_1
AS_2	$\rightarrow$	AR_2
AS_3	$\rightarrow$	AR_2
AS_4	$\rightarrow$	AR_3
AS_5	$\rightarrow$	AR_2
AS_6	$\rightarrow$	AR_2, AR_4
AS_7	$\rightarrow$	AR_2, AR_4, AR_5
SS_1	$\rightarrow$	SR_1
SS_2	$\rightarrow$	SR_2
SS_3	$\rightarrow$	SR_1

Figure 2.31: Links between shortcomings and requirements.

Chapter 3

Conceptual modeling of a Distributed System

In this chapter we introduce a formal model of a distributed system, which we call a distribution graph. This model lets us design Dynamic Distributed Graphical User Interfaces (DDGUI) and reason about their behaviors. We first discuss about the way applications are usually created and how distribution can be applied to them. We first give a formal definition of the Distribution Graph and then we give it an example of graphical representation. Then we give a list of events and actions that we are interested into. We end this chapter with a discussion about the possible behaviors that may happen with distribution.

3.1 Architecture of a distributed system

According to the presentation-abstraction-control (PAC)[COU87], there are three parts in a software support: one for the core (control), one for the data (abstraction) and one for the UI (presentation). This pattern allows us to separate the development of the UI from the rest. We could consider that there are two different kinds of parts: the application logic and the user interface. The abstraction and the control components represent the application logic while the presentation is for the UI. The control is the mean to ensure that the abstraction and the presentation components can communicate.

In the domain of distributed computing the application logic can be distributed onto several devices. A distributed application is started on one computer but is processed on several computers. This is pretty interesting for time consuming computations. Work can be processed on several computers which is faster than one computer and then saving time.

Some applications also support distribution using the Peer-to-Peer technology. The most famous examples are Skype and torrents. These ap-

plications have information about the distribution which means they are distribution-aware. However the user interface is not part of this distribution. In Skype, each computer has its own application.

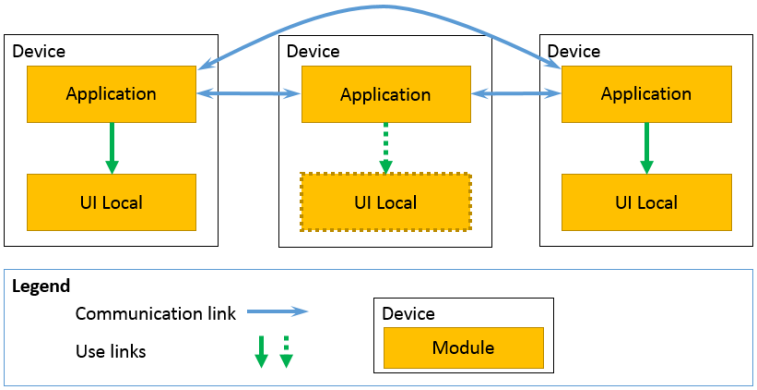


Figure 3.1: The structure of common distributed applications

The structure of these applications is depicted in Figure 3.1. While the application needs to run on all the devices it does not always need to have a user interface. In the figure, you can see that application use the *UI Local* module to create the user interface. The communication between all the devices happens at the *Applications* layer. The user interface modules are local and never communicate together. At time of writing the thesis cloud computing appears as an evolution where there are a lot of computers available and shared. They offer a lot of computing power. The logic can now be fully processed outside the computer where the application was started. The only limitation to a fully distributed application is the user interface that is stick to the device where it has been created.

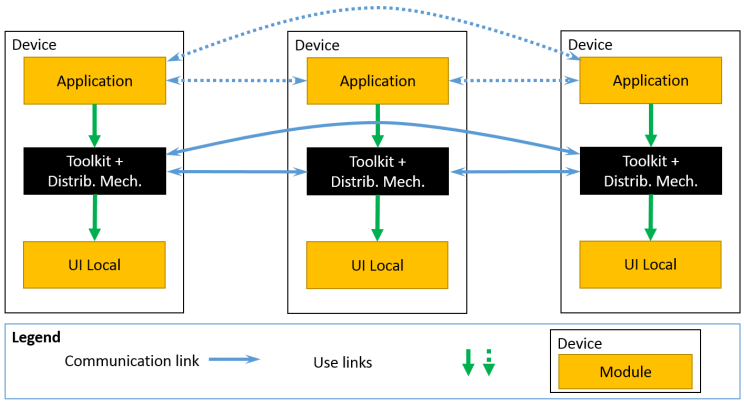


Figure 3.2: The structure of a distributed application

To create a toolkit that allows the distribution of Graphical User Inter-

faces (GUI) we first need to model a distributed system and an application. Based on this we want to create a toolkit in order to define, implement and test our new concepts. We think that an application should use existing distribution mechanisms to support the distribution of the GUI. A module, represented as a black box, is added between the application and the *UI Local* module as in Figure 3.2. This black box represents a group of interconnected modules which we will consider as a single module to keep our structure simple. The communication mostly happens between these black boxes. They allows applications to react to delays, failures and other special behaviors that happen in a network of computers. Thanks to the toolkit and the distribution mechanisms a device will be able to distribute the user interface to all the devices connected to it.

3.1.1 Non-distributed and distribution-aware applications

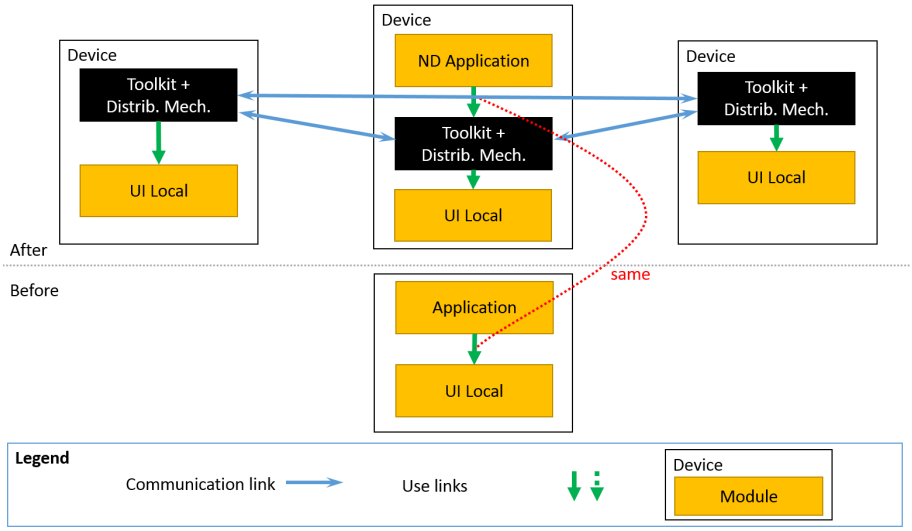


Figure 3.3: An example of a non-distributed application with and without distribution support.

Most of current applications are not distributed. They are simply running on one device and use the *UI Local* module to create the UI. These applications can simply use the added black box in the same way as they were using the *UI Local* module to support distribution without any change in the code and in its behavior. The distribution is managed by the black box which can communicate and distribute the user interface to other devices without the need of the application to run on these devices. The structure of such application with and without the black box is depicted in Figure 3.3.

If these applications want to be aware of distribution they can use distribution operations to interact with the distribution mechanisms. They

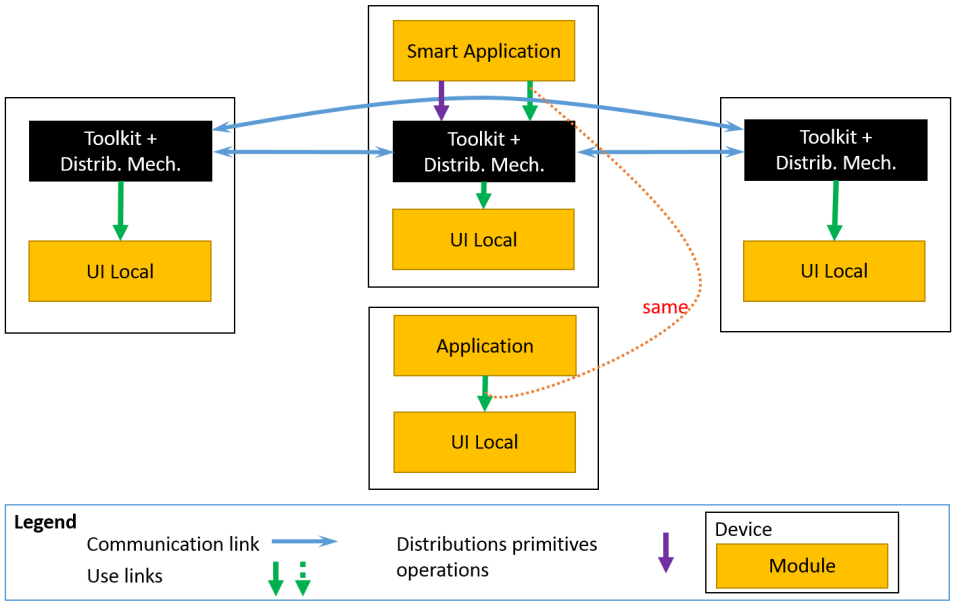


Figure 3.4: An example of a distribution-aware application

will not only use the same interface as for the *UI Local* module but also use another interface to interact with the distribution mechanisms. They can register for events (e.g., device joining, device leaving), they can ask for information regarding the distribution (e.g., how many devices are connected?, are there devices supporting touch interaction?). These apps are called *Distribution-Aware Applications* because they are aware of the distribution and can react to events in the distributed system. Their structure is depicted in Figure 3.4.

3.1.2 JayTk's architecture

The final organization of all the modules is represented in Figure 3.5. The black box has now been exploded to show the three main modules that were encapsulated. We have added information regarding the layer for each module. This is the structure of an application created in Mozart and using our toolkit and Beernet. We propose this solution for supporting the creation of applications with a DDGUI.

There are two kinds of applications: distribution-aware and unaware applications. Both will use the UI operations to interact with our toolkit JayTk as they would with QTk or Tk. Distribution-aware applications will also use distribution operations to handle distribution events and manage the distributed system.

The *distribution mechanisms* module represents all the mechanisms that currently exist for creating and managing the distribution of a system. This

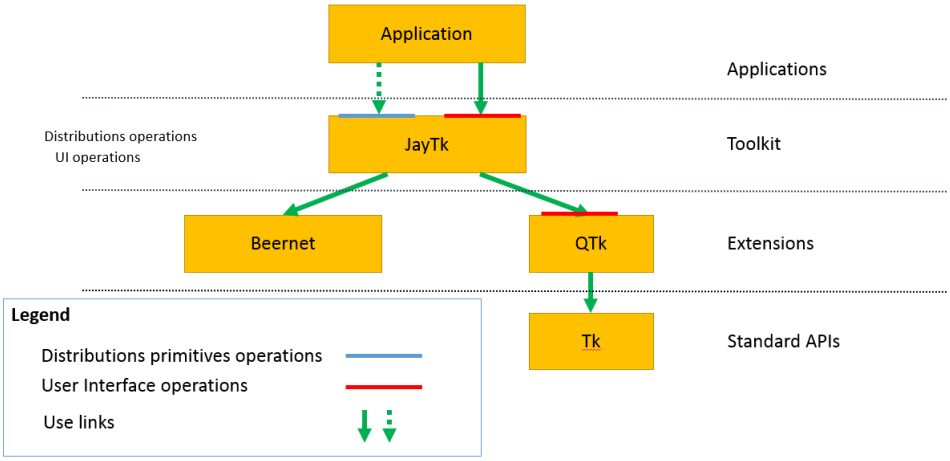


Figure 3.5: The structure of an application using the JayTk

module brings the ability to create, join and leave a peer-to-peer network, to send messages to other peers through communication protocols such as the reliable broadcast (referred later as RB). For this module we have chosen to use Beernet. Beernet is a peer-to-peer network for building self-managing scalable systems with transactional and robust storage. Thanks to Beernet, it is easy to use transactions, be notified when a failure is detected in the network and react to it. It also supports basic functionalities such as creating, joining and leaving the network.

The module *distribution mechanisms* is not mandatory but we recommend to use distribution mechanisms to fully support distribution. Without this module it is possible to distribute the UI but there is no guarantee and no possible management on the distributed system. This may be sufficient for a very short distribution (i.e., if all the devices are in the same room).

3.1.3 Example of a non-distributed application: Painter's palette

The Painter's palette is an example to explain how a non-distributed application may be distributed while not being aware and not supporting it natively. What would happen if a smartphone or a tablet becomes available to the user? As the application is not aware of the distribution, it is not notified that a new device is available and nothing happens. However the toolkit is aware that the distributed system is now composed of two devices and that the new device has a touch screen which could be interesting for both the drawing area and for the palette. A default behavior could be to move the palette to the new device while the drawing area stays on the desktop computer. The application is not aware that the palette will not be displayed on the desktop computer anymore. The application will react as

if the palette was still there.

There are several ways to enable the distribution. Another user interface provided by the toolkit could let the user choose where each part of the UI will be displayed. The toolkit could react to the appearance of a new device and automatically distribute the user interface.

This shows how a non-distributed application can be easily transformed into a distributed application without being aware of the distribution.

3.2 Model-based approach for DDGUIs

We still need to model the state of the whole system. The reason we want to use a model-based approach is to find a solution to the shortcomings listed in Chapter 2. Especially the shortcomings MS_1 (lack of ontological understanding of DDGUIs), MS_2 (lack of DDGUI modeling) and AS_3 (limited re-usability of existing tools and approaches).

3.2.1 Modeling in software engineering

Before trying to model the system, let us take inspiration from software engineering which allow people to model things.

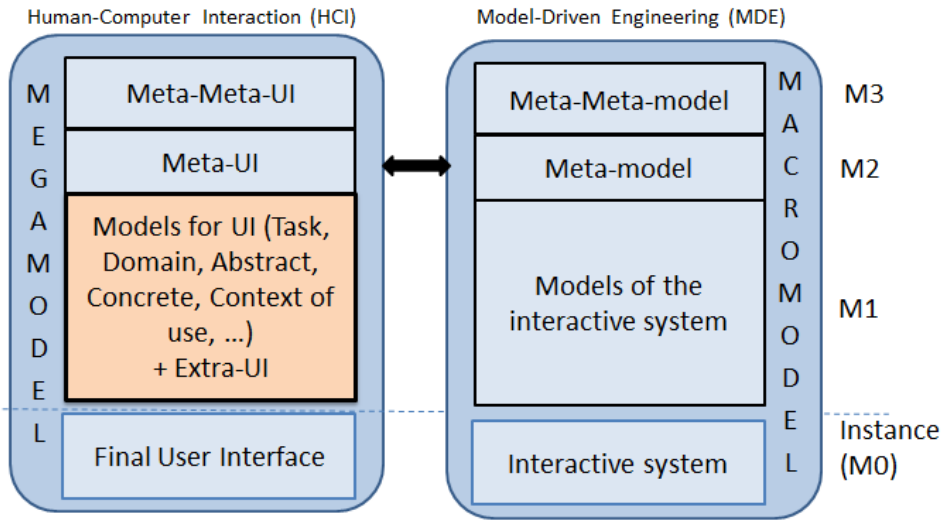


Figure 3.6: Modeling in software engineering and in HCI

In Figure 3.6, there is a comparison between the macromodel from Model-Driven Engineering (MDE) and what we called the megamodel for Human-Computer Interaction (HCI). The level 0 (M0) is the instantiation of the whole system. In MDE, the result is an interactive system based on models from level 1 (M1). For HCI, the result is the Final User Interface,

which is the UI the application provides to the user. In MDE, when we work on top of models, we are in the meta-modeling level (M2). The results from meta-modeling is a model. In HCI, a UI that controls the UI is an extra-UI, and a model that can generate a UI is a model for the UI (task model, domain model, ...). The level 2 (M2) in HCI is the meta-UI, this means that there is a UI that will control the models or the extra-UI. The level M3 in both domain is pretty rare. This means that there are models that control the meta-models, or a UI that controls the meta-UI. The whole structure is called the megamodel for HCI and the macromodel for MDE.

3.2.2 Core models

We now need to model precisely the entities of a DS. We thus need a model for the users and for the devices. There already exist several user and device models however we are not reusing one of these. The reason is that we want to create a model that is generic and independent of the other models. It needs to be expressive enough for our approach and to allow us to be compliant with the models that have already been defined in other research.

3.2.3 User Model

The first concept that we have defined in this chapter is the user. In order to adapt the system according to the user needs and characteristics, it is the first and important piece in the system. It is also the only human part in a computing system. The whole system should be able to adapt itself to the user.

Each user is a set of pairs of properties and values. Here are some examples of properties the user can have:

- Id of the user (cannot change)
- Name of the user (can change)
- Title of the user (can change)
- Location (with or without GPS, so dynamic)
- Tracks (music, video, can change)
- Dropbox (some links to personal files, can change)
- Comments (additional infos, can change)
- Connections (between users, can change)
- Experience (can change)

A User Model is divided into four stereotypes:

- a teenager: between 8 and 14 years old
- a student: from 18 to 24 years old
- an adult: from 25 to 60 years old
- a senior: more than 61 years old

3.2.4 Device Model

The second concept we modeled is the device. It is the tool for the user to interact with the system. The more the system knows about the device, the better it can adapt to the device constraints.

A device is also a set of pairs of properties and values. The properties available for the device are part of the device model of Figure 3.7. Each property is a class in the diagram. The possible values for each property are the attributes of each class.

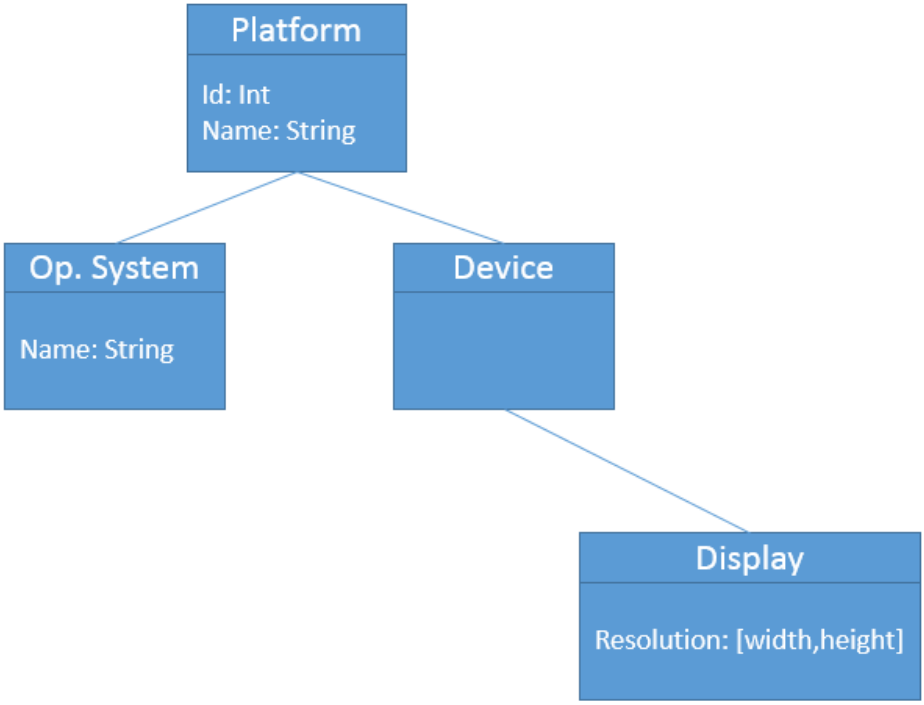


Figure 3.7: Device model

3.2.5 The selection mechanism

For our selection mechanism we will introduce the concept of *selector*. A selector consists of a selection of UI element types of a particular CUI model

that satisfy a first-order predicate logical formula. In this way, it will be possible to apply a template for a selector instead of a (potentially long) sequence of widgets. Four types of selectors are defined that will be later on used for addressing UI elements and for specifying source and targets elements of distribution primitives:

- universalSelector: applies the template to all UI elements belonging to a particular CUI, whatever they are.

- elementTypeSelector: applies the template to all UI elements belonging to a particular CUI which correspond to the selector's type (e.g., all containers, all list boxes).

- classSelector: applies the template to all UI elements belonging to a particular CUI which correspond to the selector's type whose definition makes them part of the class (e.g., all containers having an id greater or equal to 10, all list boxes having more than 10 items).

- idSelector: applies the template to only one UI element belonging to a particular CUI: the one whose id property matches the string contained in the parameter.

3.2.6 EBNF grammar

In order to formally define the language expressing distribution primitives, an Extended Backus Naur Form (EBNF) grammar has been defined. EBNF has been selected because it is widely used to formally define programming languages and markup languages (e.g., XML and SGML), the syntax of the language is precisely defined, thus leaving no ambiguity on its interpretation, and it is easier to develop a parser for such a language, because the parser can be generated automatically with a compiler (e.g., YACC). EBNF only differs from BNF in the usage of the following symbols: ? means that the symbol (or group of symbols in parenthesis) to the left of the operator is optional, * means that something can be repeated any number of times, and + means that something can appear one or more times. In this notation, brackets indicate an optional section, while parentheses denote a simple choice in a set of possible values. We use our selection mechanism in order to specify the widgets that will be affected.

Instances of distribution primitives are called by statements. The definitions of an operation, a source, a target, a selector and some other ones are defined in Figure 3.8. The definitions can be extended later to support more distribution primitives.

- statement = operation , source , "TO" , target ;
- operation = "SET" | "DISPLAY" | "UNDISPLAY" | "EXPOSE" | "COPY" | "MOVE" | "SWITCH" | "PERMUTE" | "MERGE" | "SEPARATE" | "REPLACE" | "DISTRIBUTE" | "RESET" | "APPEND" | "TRANSFORM" | "SAVE" | "RESTORE" | "IMPORT" | "EXPORT";
- source = selector ;
- target = displays | selector , "ON" , displays ;
- displays = display_platform , { "," , display_platform};
- display_platform = display , [, "OF" , platform] ;
- selector = identifier , { "," , identifier } | universal ;
- display = identifier ;
- platform = identifier ;

Figure 3.8: EBNF grammar for the main terms

3.3 Distribution Graph

As previously defined, a distributed system is dynamic. There are devices that join and leave it, users that appear and disappear, and links between devices that can fail.

To support the representation of a DS we have introduced the concept of Distribution Graph. An example of metaphor used for a Distribution Graph is an iconic map but there are other ways to represent it. All the devices that are in the DS are also in the DG. When another device appears in the DS a new distribution graph is produced. The iconic map should then be updated to display the device that has appeared.

Definition 7. *A **distribution graph [DG]** is the representation of a distributed system in a directed graph where vertices are either devices or users and arcs are links between them.*

In the Painter's Palette example there are only one device and one user. The distribution graph that represents it has two vertices (i.e., one for the user and one for the device) and an arc linking the vertices together. An example of this DG can be found in Figure 3.9.

The distribution graph is static and thus cannot represent this dynamic behavior. However vertices and arcs can be added or removed. This is exactly what happens in our example when another device is now available. A

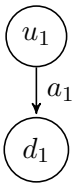


Figure 3.9: Example of distribution graph for the example of the Painter's Palette

third vertex appears which leads to a new distribution graph. The appearance of this new device is depicted in Figure 3.10. As soon as the user will be logged into it, an arc will link the user to this second device in a new distribution graph.

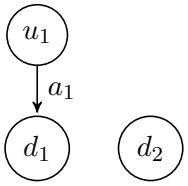


Figure 3.10: Evolution of the distribution graph when a second device has appeared

Thus in order to represent the distributed system and its dynamic behavior we need a sequence of distribution graphs called an execution.

We now formally defined it based on graph theory. In this section, we are going into detail for all these concepts. For this let us first start by the mathematical definition of a graph and then define the concept of a distribution graph as a special kind of graph.

3.3.1 Graph

A *graph* is a mathematical concept that describes links between pairs of objects. A graph $G = (V,E)$ is defined by a finite set of Vertices V and a finite set of pairs $E \subseteq V \times V$ called edges. Typically $V = \{v_1, ..., v_{n_v}\}$ and $|E| = m_e$.

A *directed graph* D = (V, A) is a graph where A is a set of ordered pairs of vertices that are called arcs. In the remainder of the thesis we will only consider directed graph and refer to links as arcs.

3.3.2 Attribute-Value Pair pattern

The pattern *Attribute-value pair* is a data representation that is commonly used in computing science. This allows us to create a link between an

attribute and its value. The attribute is often represented as a name. An example of this pattern can be found in machine learning[AN02].

The main advantages of this pattern is that it is a fundamental data representation. It offers some flexibility thanks to its open data structure which allows extensions. It is straightforward to model a physical entity as a record using this pattern. A record is a value that contains other values, here values are attribute-value pairs as in Figure 3.3.1

Definition 3.3.1. ***(Record)****:* $R = \{(a_i,v_i)\}_{i=1,..,n_a}$

In this definition a_i represents an attribute and v_i the value associated with it. For the thesis we will use this pattern to model our data as a record. It is very easy to compute and sufficient to represent our concepts.

3.3.3 Vertices

The first concept we need to model is the user. Each user has his/her own characteristics. A child will not use the computer in the same way than an adult. There also exist certain persons with disabilities that need a system adapted to them. The more the system knows about the user, the more it can be adapted to it. Every physical entity can be considered as a user depending on the importance of his/her interactions. An observer does not need to be considered as a user. A user is generally defined by an account.

Each user is represented by a unique constant, an identification, that is part of the set of constants U_{Total}. The set of users U is a subset of U_{Total} and is defined in Definition 3.3.2.

Definition 3.3.2. ***(Set of Users)****:*
$U = \{u_i\}_{i=1,...,n_u}$

A record is associated with this constant/identification that characterized it. The function $\mathcal{U}$ associates a constant/identification with a record as defined in Definition 3.3.3. For example, a user record may have the pair (Experience, High). It means that the experience of the user is high.

Definition 3.3.3. ***(User function)****:*
$\mathcal{U} : U_{Total} \rightarrow R_u : u_i \in U_{Total} \mapsto r_i \in R_u$

We have defined the concept of devices as the combination of the hardware and the software (the Operating System). It means that it is the combination of the possible restrictions brought by the device and the interaction capabilities brought by the OS. Both are needed for most of the common questions we could have about a device such as : *"Is is touch-sensitive?"*. It depends if both the hardware and the software support it. Or *"What are the possible resolutions and what is the current?"*. This latter question depends on the native resolution provided by the hardware (the

screen, the display) but also by the drivers running on the operating system that can allow more than the native resolution. As for the user, a device is represented by a unique constant, an identification, that is part of the set of constants D_{Total}. The set of devices D is a subset of D_{Total} as defined in 3.3.4.

Definition 3.3.4. ***(Set of Devices)****:*
$D = \{d_i\}_{i=1,...,n_d}$

The function $\mathcal{D}$ associates a constant/identification with a record. This function is defined in Definition 3.3.5. For example, a device record may have the pair (Resolution, 1366x768). It means that the device has a resolution of 1366 by 768 pixels.

Definition 3.3.5. ***(Device function)****:*
$\mathcal{D} : D_{Total} \rightarrow R_d : d_i \in D_{Total} \mapsto r_i \in R_d$

Applied to the domain of the thesis, there are two kinds of vertices: a user vertex and a device vertex. Let us define U as the set of user constants and D as the set of device constants. Consequently the set of vertices applied to the DDGUI domain is $V = U \cup D$ as in Definition 3.3.6.

Definition 3.3.6. ***(Set of DDGUI Vertices)****:*
$V = \{v_i \mid v_i \in (U \cup D)\}_{i=1,...,n_v}$

This definition allows us to model several users and devices of a distributed system. This multiplicity is needed to extend the definition of the context of use which allows only one device and one user.

3.3.4 Arcs

The set of arcs A is defined as a 2-element subset of DDGUI vertices so that

Definition 3.3.7. ***(Set of Distribution Arcs)****:*
$A = \{\ a_i \mid a_i = (x, y)\ \&\ x, y \in V\ \}$

A distribution arc represents a connection between two DDGUI vertices. Here are the possible distribution arcs : $u_i \rightarrow d_j$ and $d_i \rightarrow d_j$ There are two kinds of DDGUI vertices and two kinds of arcs. An arc between a user vertex and a device vertex is labeled as *logged*. One between two device vertices is labeled as *connection*. A user vertex can only have arcs towards device vertices. These arcs are neither reflexive, nor symmetric, nor transitive. A user can log into several devices at the same time. This relation between a user vertex and a device vertex is defined as a the logging relationship $\mathcal{L}$:

Definition 3.3.8. ***(Logging relationship)****:*
$\mathcal{L} : U \rightarrow D :$
$\forall i \in 1..n_u, \exists j \in 1..n_d | \forall u_i \in U, d_j \in D : u_i \in U \mapsto d_j \in D$
denoted by $\mathcal{L}(u_i, d_j) \mathrel{\hat{=}} u_i \xrightarrow{l} d_j$

A device vertex can only have arcs going to other device vertices or coming from other DDGUI vertices, both user and device vertices. An arc from a device vertex d_i towards a device vertex d_j means that d_i can communicate to d_j through at least one communication protocol (e.g., Bluetooth, WiFi, NFC). Some protocols may allow more than one channel between two devices. However we only represent the ability to communicate from one device to another no matter what is the communication protocol used. These arcs can be transitive: if a device vertex d_x can communicate with vertex d_y and d_y can communicate to d_z, d_x can communicate to d_z without a link. However a link between d_x and d_z is correct if they are directly connected through at least one communication protocol too. It is nonetheless not guaranteed that the transition is always possible (i.e., it may depends on the way communication is implemented). This relation between two device vertices is defined as a the connection relationship $\mathcal{C}$:

Definition 3.3.9. ***(Connection relationship)****:*
$\mathcal{C} : D \to D :$
$\forall i \in 1..n_d, \exists j \in 1..n_d | \forall d_i \in D, d_j \in D : d_i \in D \mapsto d_j \in D \ \& \ d_i \neq d_j$
denoted by $\mathcal{C}(d_i, d_j) \hat{=} d_i \xrightarrow{c} d_j \Rightarrow \exists a_k \in A$ *so that* $a_k = (d_i, d_j)$

3.3.5 Distribution Graph

A *Distribution Graph* (DG) is defined as a pair of the set of vertices V and the set of arcs A. It allows people to represent a snapshot of a Distributed System at a certain time. Formally:

Definition 3.3.10. ***(Distribution Graph)****:* $DG = (V, A)$ *where* V *is the set of vertices and* A *is the set of arcs.*

An example of a distribution graph is depicted in Figure 3.11. There are two users and three devices. We will now refer to user vertex as user and device vertex as device. The user u_1 is logged into the three devices d_1, d_2 and d_3 while user u_2 is not logged into any device. There is a connection between devices d_1 and d_2, and between d_2 and d_1. The device d_3 is not connected to any other device.

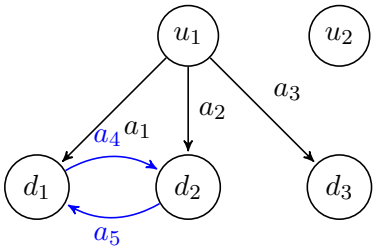


Figure 3.11: An example of distribution graph with 2 users and 3 devices

Even if u_2 is not linked to any vertex it is still in the DG. It means that u_2 was previously logged into a device but this is not the case anymore. The device the user u_2 was logged into may be d_1, d_2, d_3 or another device. A device that has no arc anymore cannot interact with the distributed system.

Each vertex has its own view of the distribution graph. It is only a partial view as it does not know all the views of each vertex. Formally :

Definition 3.3.11. ***(Partial Distribution Graph)****: $DG_{v_i} = (V_{v_i}, A_{v_i})$ where $V_{v_i} \subseteq V$ and $A_{v_i} \subseteq A$*

To process the complete DG it is necessary to use a distribution algorithm that will create a snapshot of the DS. The source vertex needs to ask other vertices their own view in order to collect all the information . Each vertex will then share it back to the source vertex. Thanks to the information collected the source vertex can create a global DG. There is no guarantee that this global DG will be the real distribution at a certain time because while collecting information, some views may change. E.g., some devices may have joined or left after the information was shared to the source vertex.

3.3.6 Environment

In Human-Computer Interaction, the environment is often defined to include aspects of the physical world, such as placement of objects and persons, and sensory characteristics such as visual imagery, temperature, sound and so forth. In our formal model we consider the *environment* only as a source of events. Any event that is not triggered by a user is considered to have been triggered by the environment. This formalization is done without loss of generality because any physical embodiement of an environment must interact with the system in terms of events. It also has the advantage of not making any premature choices about the representation of the physical world.

3.4 Behavior of a Distributed System

The Distribution Graph allows people to model the elements and relationships that constitute a distributed system. In order to model the behavior of each of these elements we need to model its dynamic aspects.

3.4.1 Events

The communication between all the devices, and between a user and a device are represented by events. An *event* is defined as a triple of two vertices and the content of the event which is a record. This is defined in Definition 3.4.1.

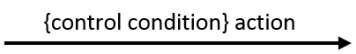


Figure 3.12: The graphical representation of the Event-Condition-Action pattern[PAS07]

Definition 3.4.1. ***(Event)****:* $e_k = (v, w, r)$ *where* $v, w \in V$ & $r \in R_e$

An event is a communication of the content r from v to w. We introduce the special value *environment* for v when the event comes from the environment. The appearance of a device or a user in the distributed system, as well as the login of a user into a device or the connection between two devices are examples of an event that could trigger a modification in the distribution graph.

3.4.2 Execution trace of the distributed system

A Distribution Graph is a snapshot of a distributed system at a certain moment of time. To model the dynamic aspect of a distributed system we need to allow the distribution graph to change with time. An execution trace is an alternate sequence of distribution graphs and events that starts with the initial distribution graph dg_0. Formally:

Definition 3.4.2. ***(Execution trace)****:* $E = (\{dg_0, e_0, dg_1, e_1, \ldots\})$

3.4.3 Event-Condition-Action pattern

The *Event-Condition-Action pattern* (ECA) allows people to describe a dynamic concept. It is commonly used in Software Engineering, Database Management and in HCI. The structure is defined as: ON event IF condition THEN action. It means that when something happen, the event, under a certain condition, a set of actions is triggered. Graphically it is an optional conditional and the set of actions as in Figure 3.12.

We will now introduce a similar pattern for distributed systems. There are three kinds of events: graph events, distribution events and interaction events. The appearance of a device or a user in the distributed system, as well as the login of a user into a device or the connection between two devices are example of events that could trigger a modification in the distribution graph. We will call these events: **graph events**. As we already introduced in the introduction a distributed system may have crashes, failures or other complex behavior. The information about the quality of the communication between devices will be through **distribution events**. Users while being logged into devices can interact with them: click on button, fill text boxes, press a key on the keyboard, ... These interactions are **interaction**

events. The two first categories are events triggered by the environment. The last category is for events triggered by users.

The condition is a predicate as in mathematical logic: a boolean-valued function P: $X \rightarrow$ {true, false}. An example of condition is to check if an attribute of the device or the user is set to a certain value (e.g., device.isMultiTouch == true). This will allow us to automatically adapt the Graphical User Interface according some rules.

The last category is the set of actions that are triggered for a certain event. There are two possible actions in this set: distribution operations and graph modifications. The graph modifications are transitions from one distribution graph to another (i.e., appearance of a vertex, disappearance of a vertex, connection between two devices, user logging into a device or disconnection between two devices).

3.5 Graphical representation

Figure 3.14 is an example of DG with two users, u_1 and u_2, and two devices d_1 and d_2. The device d_1 is currently connected to user u_1. The other device and user, u_2 and d_2, are not connected together or to anything else.

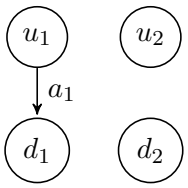


Figure 3.13: Representation of a distribution graph with two users and two devices

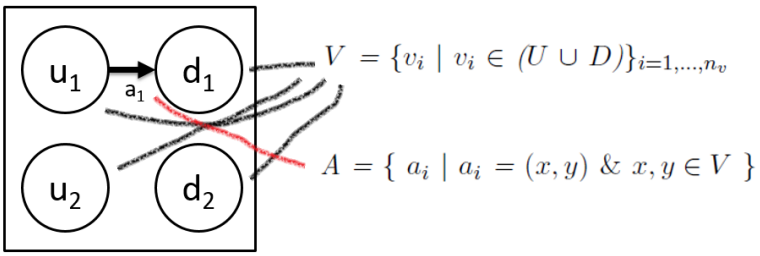


Figure 3.14: Representation of a distribution graph with two users and two devices

Several changes can happen in the DG. The first example is a vertex that disappears. Figure 3.15 shows the disappearance of the user u_2 and the device d_2. The visual feedback that allows us to show the disappearance of

a vertex is to remove it from the distribution graph. In the figure, we only see the user u_1 and the device d_1.

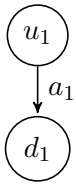


Figure 3.15: Representation of the same DG after disappearance of u_2 and d_2

Another example of change in the distribution graph is the disconnection between two vertices. For example, in Figure 3.16 the user u_1 has left the session on the device d_1. They are then left not connected to any other vertex. The disconnection between two vertices is only possible if they were connected together, graphically it means that there was an arc between the vertices. The disconnection will remove this arc.

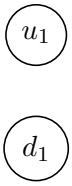


Figure 3.16: Representation of the same DG after disconnection of user vertex u_1

And finally, the disappearance of the user u_1 left the device d_1 alone in the DG. As depicted in Figure 3.17 the device d_1 is not anymore in the DG. A disappearance could happen with one vertex connected to another one. If one of theses vertices disappear the arcs between them will also be removed.

Figure 3.17: Representation of the same DG after disappearance of u_1

3.6 Application Graph

Another concept that we are introducing in the thesis is the *Application Graph* (AG). Instead of a complete definition of the concept as made for DG in Section 3.3 this section only gives an introduction to the concept.

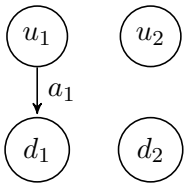


Figure 3.18: Simple example of Distribution Graph.

The AG is a more fine-grained view of the DG. It focus on the distribution of applications inside devices.

An example of DG can be found in Figure 3.18. In this example, there is one user u_1 currently connected to two devices d_1 and d_2. The two devices are not connected together.

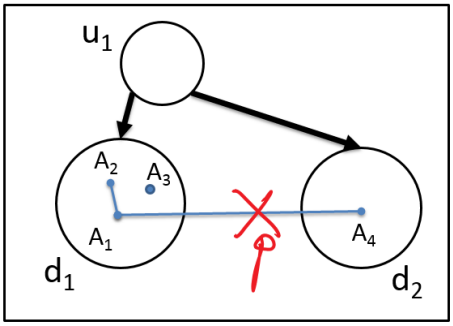


Figure 3.19: Simple Application Graph from the Distribution Graph of Figure 3.18

From this DG, Figure 3.19 shows a possible Application Graph. This AG shows that the are currently three applications running on device d_1 and another running on the device d_2. The applications A_1 and A_2 are conntected together and can communicate and exchange UI elements. The application A_3 is currently not connected and then working alone. The figure also shows that it is not possible for applications A_1 and A_4 to connect or communicate to each other. In fact, the connection between two applications is possible if and only if:

- The two applications are on the same devices
- The devices where are the two applications located are connected together in the DG.

As seen in the previous figure, the DG has no connection between d_1 and d_2. Thus, A_1 and other applications from device d_1 cannot communicate with application A_4.

Distribution State

The distribution graph allows a graphical representation of the relationships between users and devices. We also wanted to provide a view on the actual state of the elements that composed the distributed system. We thus propose to model this through a new concept: the distribution state. A *Distribution State* is a snapshot of what constitutes the DG at a certain moment of time in which the distributed system is stable.

Distribution Initial State

There is a distribution initial state. We will write DS_i for the ith Distribution State. In this initial state the set of vertices (users and devices) and set of arcs are all empty.

Formally:

$$DIS = DS_0 = \begin{cases} V = \emptyset \\ A = \emptyset \\ U = \emptyset \\ D = \emptyset \end{cases} \tag{3.1}$$

Other Distribution States

From this distribution initial state there is only one possible event that can happen: APPEAR. In this case, only a vertex can appear at this stage. Even if vertices can either be users or devices we assume that a user always connects to the system through a device. This means that a device should first appear and then a user. A sequence of distribution states will help us clarify the possible transitions. The sequence is neither total nor complete. There is only one possible state for DS_1 which is a device appears in the system.

Formally the next state can be reached with event : $e_0 = (env,\ d_1,$ APPEAR). It means that a device has shown up.

$$DS_1 = \begin{cases} V = \{d_1\} \\ A = \emptyset \\ U = \emptyset \\ D = \{d_1\} \end{cases} \tag{3.2}$$

From DS_1 it is now possible for the vertex to disappear and the situation comes back to DIS. If another device appears in the environment we stay in a state equivalent to DS_1. What is new here is the ability for a user to log into a device leading to DS_2.

Formally the next state can be reached with: $e_1 = (env,\ u_1,$ APPEAR). The previous state can be reached with the event : $e_1 = (env,\ d_1,$ DISAPPEAR).

$$DS_2 = \begin{cases} V &= \{d_1, u_1\} \\ A &= \emptyset \\ U &= \{u_1\} \\ D &= \{d_1\} \end{cases} \tag{3.3}$$

The state DS_2 should not be visible as the user appears by logging into a device, the next event is : $e_2 = (u_1, d_1, \text{LOGIN})$. It leads to DS_3 :

$$DS_3 = \begin{cases} V &= \{d_1, u_1\} \\ A &= \{a_1(u_1, d_1)\} \\ U &= \{u_1\} \\ D &= \{d_1\} \end{cases} \tag{3.4}$$

From DS_3 it is still possible for a vertex to disappear. We only consider that devices can leave the distribution graph. A user that logs out of a device is kept in the system. A user needs to be connected to a device to enter in the environment but can stay in the environment without any connection to a vertex. The appearance of another vertex will be equivalent to DS_2 and DS_3. The last possibility is to connect two devices together which leads to DS_4.

Formally the next state can be reached with: $e_3 = (d_1, d_2, \text{CONNECT})$. It leads to DS_4 :

$$DS_4 = \begin{cases} V &= \{d_1, d_2\} \\ A &= \{a_1(d_1, d_2)\} \\ U &= \emptyset \\ D &= \{d_1, d_2\} \end{cases} \tag{3.5}$$

And finally, the last possibility is that two vertices disconnect from each other. Formally the disconnection can be reached with: $e_4 = (d_1, d_2, \text{DISCONNECT})$.

Application State

In the previous section we have defined and exemplified the distribution state and all the actions that can happen in the distributed system around users, devices and environment. However this concept does not cover any application state. For this reason we introduce the concept of *Application State*(AS). As for the distribution state an application state is a snapshot of what constitutes the application graph without repeating what is already specified in a distribution state.

A basic application in computing science is a process executed on one device for a certain user. It can be separated in two parts: the application logic and the user interface. With distributed applications, the application logic is distributed across several devices with the user interface kept in the device where the application has been created. In any case, we have a

very basic configuration of an application that is started on a device. An application starts on only one device and depends on this device. This leads to the first application state. The application initial state has no application at all.

Application Initial State

The application state is a function that for each application gives the set of devices in which the application is distributed and the set of applications that are started in the distributed system. Let us concentrate our efforts in a simple distribution graph with two devices and one user as in figure 3.18.

Formally, the application initial state (AIS) is:

$$AIS = AS_0 = App = \emptyset \tag{3.6}$$

At this time, no application is running and the distributed system is stable. But with the start of an application, the dynamic behavior of a distributed system appears again.

Other Application State

From the IAS the only way to leave the state is that an interaction happens in the distributed system. This interaction is a user or a device starting an application. No matter if the application starts on device d_1 or d_2 it is equivalent.

Formally, the application state (AS_1) is:

$$AS_1 = \left\{ \begin{array}{rcl} App & = & \{A_1\} \\ f(A_1) & = & \{d_1\} \end{array} \right. \tag{3.7}$$

We have decided to use an uppercase notation for the application to differentiate the AG elements from the DG elements. With this state, there is one application, A_1, which has been started. The function allows us to know on which device is the application running. It returns the set of devices. From this application state, it is possible to start another application or to stop the one that was running. The first possibility will lead to the application state AS_2 while the other will come back to IAS.

This may raise a question: is it possible to have the same application running on the same device or on two devices? The answer is complicated. If the application A_1 is running on d_1 it is possible to also run the application on d_1 and connect them together. However for A_1 to start on d_2 it needs d_1 and d_2 to be connected together. If there exist two versions of the same application running on two devices that are not connected, they are two different applications.

Formally:

$$AS_2 = \begin{cases} App & = & \{A_1, A_2\} \\ f(A_1) & = & \{d_1\} \\ f(A_2) & = & \{d_2\} \end{cases} \tag{3.8}$$

$$or \begin{cases} App & = & \{A_1, A_2\} \\ f(A_1) & = & \{d_1\} \\ f(A_2) & = & \{d_1\} \end{cases} \tag{3.9}$$

$$or \begin{cases} App & = & \{A_1\} \\ f(A_1) & = & \{d_1, d_1\} \end{cases} \tag{3.10}$$

With the concept of application states we allow the application to be distributed across several devices. Before going into details of the distribution we need to introduce the possible actions.

3.7 Distribution Primitives

Let us talk about the actions with a finer granularity. In this section, we will formally define some possible actions. As described in section 3.4.3, actions are part of the Event-Condition-Action pattern. An *action* is thus either a reaction to an event or it is triggered when a condition is fulfilled.

It is not possible and useful to support and list all the possible actions. The thesis defines a set of actions relative to a DS. Moreover, the thesis only covers a subset that was necessary for the case studies or already provided in the related work. We also think that this subset is sufficiently representative to support the DS properties.

3.7.1 Local User Interface actions

Several actions can happen between vertices in a distribution graph. It means between several users, several devices and both kind of vertices. However local actions can only happen in one and only one device. They do not support distribution in any way. Even if they can be triggered by a user we assume that these actions are directly triggered by the device itself and does not even need a user.

There are a lot of possible actions allowed by applications and operating systems. We do not cover all these actions because they are already supported by the graphical toolkits and the operating systems that we used in the thesis. For example, the creation of a window with buttons to minimize, maximimize and close it are often provided by the environment and operating system APIs. For mobile and modern UI there exists other ways to support these actions such as hardware buttons.

We do not currently cover mouse, keyboard and gesture actions. They are already provided by APIs.

Creating a button and reacting to a click on this button, and recognizing some gestures are also available in the environment itself. Thus we will reuse these technologies to only focus on more complex aspects of a distributed system. The main actions that are needed by any application is to display and hide elements in a UI. While it is provided by all the environments to create a UI at design stage; it is not provided to display and hide elements at run-time. For this reason we will now specify these interactions.

These action are called *local User Interface actions*. Some examples of local User Interface actions are: a user moving, minimizing or maximizing a window.

3.7.2 Global UI actions

The Global UI actions is the set of actions that interact with the user interface without the need of any information from the distribution graph. The actions will not change the distribution graph however they can use some distribution mechanisms. They are based on the Local User Interface actions and can be seen as a unification of all the set of local User Interface actions provided by each operating system API.

Display - Undisplay

As the devices and users have an appear and disappear event, the actions *display* and *undisplay* exist for the user interface elements. Let us first take an example of what it can be and separate this action through the whole concept of displaying elements.

A user is currently at home with his smartphone, a computer and a television. He would like thanks to his smartphone to remotely play a game from his computer while displaying it on his television. This first step to acquire this is to already have the user and the three devices in the distributed system. Then to connect the user to all these devices and all the devices together. If we assume that we already have this DS we then need to start the game on the computer and display the interface on the television.

Let us then assume that the game is started on the computer without any user interface. In order to display some elements the computer needs an action that will ask the television to display the elements and then the television will need an action to display the elements. This is the latter action that we care about because it is the local action of displaying an element.

These actions are very important because they allow the application to show and hide elements on all the devices. It is based on the local actions that allow the creation of UI elements. Thanks to these actions the application can dynamically show and hide things on each screen of the DS. Not only it can display things on one device but it can also display things

to other devices without any knowledge of distribution. These actions are already provided in some toolkits [AVR89, REK03, GRO05, MEL09].

The statement for this DISPLAY action is:
DISPLAY list_element [IN element [AS position]]

There are two possible actions with DISPLAY: either we create a new element independently from other elements or we insert this element into an existing element.

A small and simple example is to an empty window which would be:
DISPLAY window

The concept of *window* depends on the device it will be displayed. It is the root of any user interface displayed on a device. There can be several windows on a device if the device allows it which is not always the case.

Here is a more complex example which needs to display several complex elements in another element:
DISPLAY label(text:"Here is a pop up windows"), lr(button(text:"Cancel"), button(text:"Ok")) IN win AS top

This example show how to display a label and two buttons inside the windows we have already created. It also specifies that the label is upon the buttons and that the buttons will be beside each other. The new elements will be put at the top of the existing ones if there are already elements in it.

The statement for the UNDISPLAY action is:
UNDISPLAY list_element [IN element]

It can either hide all the elements from *list_element* found on the device or inside *element*.

Some feedback can be provided for these actions. The display action may have an increasing opacity or size. The undisplay action may have a decreasing opacity or size.

Get - Set

When some elements have already been displayed, it is possible to get or change information from them. The local action *get* reads the value of a property without modification. This action allows us to directly access or change properties of elements in DS. Without these actions each modification would require to undisplay the elements and display a new one with the modification. These actions are only provided in one toolkit[AVR89].

The statement for GET action is:
GET p FROM id OF element

The action will get the value from the property *p*. The action *set* allows us to change the value of a property.

The statement for SET action is:
SET p TO value FROM id OF element

The action will set a new value to the property *p* of *element*. This value can either be dependent or independent from the previous one. Let us the

old value be v_{old} and the new value be v_{new}.

- $v_{new} = p$, if p is a value independent from the old one (e.g. 10, 100, "tablet").
- $v_{new} = p * v_{old}$, if p is a percentage (e.g. 10%, 20%, 50%,...).
  i.e. if $p = 10\%$, $v_{new} = 0.1 * v_{old}$.
- $v_{new} = v_{old} + sign(p) * value(p)$, if p is either $-X$ or $+X$.
  i.e. if $p = -5$, $v_{new} = v_{old}$ - 5.

For each property there is a possible range of values call the *type*. For example, boolean property like *isVisible* can only have value $FALSE$ and $TRUE$. If there is a type violation, the result of the action gets a $FAIL$ result. If the action succeeds the result of the action gets the new value as result.

Move

Thanks to the ability to display and undisplay graphical elements on each device we are able to perform some actions that we use to do locally. For this we will create a scenario of actions that will be executed sequentially. This action is the one that is mostly supported by all the existing toolkits. It is totally supported in [AVR89, REK03, BIE04, GRO05, MEL09] and partially in Jigsaw and Cesam[COU06] and in [VDV05, BIE08, ROS09, ROS09B, BLU10].

An example is a *move* operation. Any movement of graphical elements is a combination of hiding the element (undisplay) and showing it somewhere else (display).

The statement for the MOVE action is:
MOVE list_element [IN element1 [AS position1]] FROM device1 TO device2 [IN element2 [AS position2]]

The scenario asspciated with this action is then:
Send(UNDISPLAY list_element [IN element1 [AS position1]] , device1)
Send(DISPLAY list_element [IN element2 [AS position2]] , device2)

Some feedback is needed to understand what is happening. What we recommend for this is to set the opacity as decreasing for the undisplay action and increasing for display action.

Copy

The action *copy* is also possible thanks to the display action. While the display action creates the user interface in a predefined state, the copy action needs to get the current state of the user interface and display the user interface according to this state.

This action is already provided in few toolkits[REK03, GRO05, MEL09]. It is also interesting to note that this action could allow us to create a copy of a UI and then modify it. It would allow us to compare them quickly and choose which suits best the users.

The statement for the COPY action is:
COPY list_element [IN element1 [AS position1]] FROM device1 TO device2 [IN element2 [AS position2]]

The scenario associated with this action is then:
Send(DISPLAY list_element [IN element2 [AS position2]] , device2)

A feedback for this interaction is also recommended. It is not easy how to represent this operation. What we suggest is to create the copy upon the original and then move the copy thanks to an animation. It will allow the user to understand that the element has been copied.

Here is the algorithm to copy an element to a device:

Algorithm 1: Copy to a device

PRE : The id *ID* refers to an element that already exists in the system.
POST: The element with id *ID* is now copied to the device *Target*.

```
function Copy(id:ID device:Target)
  Element el = new Element()
  display(el,Target)
  for each property in ID->properties
  do
    el->set(property, ID->get(property)
  end
end
```

Here is the algorithm for the same ditribution primitive but in a container:

Algorithm 2: Copy in a container of a device

PRE : The id *ID* refers to an element that already exists in the system.
POST: The element with id *ID* is now copied to the device *Target* in the element *Container*.

```
function Copy(id:ID id:Container device:Target)
  Element el = new Element()
  display(el,Container,Target)
  for each property in ID->properties
  do
    el->set(property, ID->get(property)
  end
end
```

Finally, here is the algorithm to copy several elements to a device:

Algorithm 3: Copy several elements to a device

PRE : The list of ids *List_id* refers to a list of elements that already exist in the system. POST: All the elements of the list are copied to the device *Target*.

```
function Copy(list:List_id<ID> device:Target)
  for each element in List_id
  do
    Element el = new Element()
    display(el,Target)
    for each property in ID->properties
    do
      el->set(property, ID->get(property)
    end
  end
end
```

Merge - Separate

These actions allow developers to merge or separate some elements. I.e., the creation of bars with buttons or the split of these buttons. These actions consist of a spatial grouping or degrouping of UI elements.

The *separate* action has been first introduce in CESAM's prototype in the form of a scissor [ROU06]. The *merge* action has already been provided in JigSaw and CESAM[ROU06] and in [REK03].

This action is the one that would be useful for our Painter's palette example. It will allows us to separate the palette and the drawing area in two parts. The GUI of the application is composed of a lot of graphical elements. A nice way to create this GUI is to create two containers, one for the palette and one for the drawing area. Typically the drawing area is a canvas and the palette is a set of buttons arranged in a sort of layout (e.g., a grid, a bar).

Each graphical element is a record which is represented by an id: canvas(id:"drawingarea"), td(id:"palette"). The generic statement for the SEPARATE action is:
SEPARATE list_element1 [from element] [to list_element2]

The number of elements in list_element2 should either be one or equal to the number of elements in list_element1. If list_element2 has only one element, all the elements in list_element1 will be placed into it. If there are several elements in list_element2, each element of list_element1 will be placed into the element of list_element2 that is at the same position.

For our example, the statement will be :
SEPARATE drawingarea, palette from win to win,win2

In this case, we start with an element with id *win* which currently contains *drawingarea* and *palette*. They are undisplayed from *win* and then *drawingarea* is placed back into *win* while *palette* is placed into *win2*. If *win2* already exists, *palette* will be added to the end of the children of *win2*. If it does not exist then a new element will be created and its id will be *win2*.

There are three possible scenarios regarding where *win2* exists or will be created. If it does not exist it will be created in the device where it was displayed before. If it does exist on the same device it will be displayed on this local element. The last possible scenario is that *win2* exists on another device. This means that not only the two elements will be separated but also that *palette* will be distributed to the device(s) where *win2* exists.

The SEPARATE action is a composition of the DISPLAY and UNDISPLAY actions. The same result is possible by composing this sequence of actions : UNDISPLAY drawingarea, palette IN win
DISPLAY drawingarea IN win
DISPLAY palette IN win2

A shortcut is also possible because it is not mandatory to undisplay *drawingarea* from *win*. This leads to the following sequence of actions : UNDISPLAY palette IN win
DISPLAY palette IN win2

Some feedback can also be provided for this action. Either we reuse the feedback from both UNDISPLAY and DISPLAY actions or we create another feedback. An idea is to show the two elements going out of *win* and then moving to their destination. The shortcut sequence would only show *palette* going out of *win* and moving to *win2* or disappearing.

3.7.3 Distributed actions

Appear - Disappear

We have already defined the distribution initial state (DIS) and its transition to DS_1 through an action. When a device or a user appears in the system, the change *appear* happens. It is the creation of a vertex corresponding to the device or the user in the DG.

The opposite change may happen when a vertex (either a device or a user) is not anymore in the system. This change is *disappear*. It is the suppression of the vertex in the DG.

The statement for the APPEAR change is:
APPEAR id [AS new_name]

We recommend to use the optional name in order to simplify how a vertex can be identified in the system:

- APPEAR device_id AS new_name

- APPEAR user_id AS new_name

In both case it corresponds to a vertex entering the distributed system. The device may notify its presence with some kind of *HELLO* or *PING* messages. A user notifies its presence by logging with an existing or creating some user profile. The device used by the user to connect have to already been appeared in the DG.

Algorithm 4: Appearance of a device

PRE : The id *ID* refers to a device that exists in the environment.
POST: A vertex corresponding to the device with id *ID* is created in the DG.

```
function Appear(id:ID)
  if (ID->type == Device.type)
  then Device.add(ID)
  end
end
```

Algorithm 5: Appearance of a user

PRE : The id *ID* refers to a user that exists in the environment.
POST: A vertex corresponding to the user with id *ID* is created in the DG.
The User is connected to the device he/she just logged in.

```
function Appear(id:ID)
  if (ID->type == User.type)
  then
    User.add(ID)
    p_ID<-IdentifyDevice()
    Connect(User.ID, Device.p_ID)
  end
end
```

Connect - Disconnect

When a device appears in the distributed system, it is not connected to any other vertex. In order to be able to perform communications between two vertices they need to connect together. Two vertices need a *connect* change. This change will create a directed edge with a source vertex and a target vertex. If both direct edges exist, they are merged into a non-directed edge. If several connections exist there is still only one edge representing it. An edge represents the ability of a vertex to communicate with another one.

When a user appears in the DS it directly connects to the device used for the login.

The statement for the CONNECT change is:
CONNECT vertex1 TO vertex2

The opposite change *disconnect* can happen if a vertex close a connection with another vertex. All the possible communication between both vertices are then stopped and the edge in the distribution graph is removed.

The statement for the DISCONNECT change is:
DISCONNECT vertex1 from vertex2

Process

The distributed action *process* brings the ability to perform a set of actions on a set of devices. These action may fail or succeed. There is no guarantee that they all succeed or failed.

The statement for the PROCESS action is:
PROCESS list_actions ON list_devices

The algorithms for this action are:

Algorithm 6: Process a list of actions on a list of devices

```
function Process(List_action<Action> actions,
                 List_device<Device> devices)
  for each device in devices
  do
    for each action in actions
    do Process(action,device)
    end
  end
end
```

Algorithm 7: Process a list of actions on a device

```
function Process(Action action, Device device)
  if (this->device equals device)
    Execute(action)
  else
    Send(action, device)
  end
end
```

Transaction

The action *transaction* allows developers to force one or more actions to be performed atomically. The transaction can either succeed or fail.

The statement for the TRANSACTION interaction is:
TRANSACTION sequence_of_actions

Algorithm 8: Transaction of several interactions without freezing resources.

PRE: -
POST: All the interactions succeed or fail

```
function Transaction(actions:List_action<Action>)
  List<Action> succeed = empty
  for each action in List_action
  do
    perform(action)
    if succeed(action)
    then succeed union action
    else break
    end
  end
  if size(List_action) > size(succeed)
  then for each action in succeed
  do
    undo(action)
  end
  else
    return succeed
  end
  return fail
end
```

Crash - Suspect - Resume

When a device is not reachable anymore, the vertex is suspected to be crashed. This is possible thanks to a failure detector available in the set of distribution mechanisms. A *suspect* action happens to give suspission.

The action does not induce a change in the DG because it is not a disappearance of a vertex. In a disappear change, the device has left the environment and is not available anymore while in a suspect action, the device is still in the enviroment but is not reachable.

When the device is available again, the action *resume* tells the application that the device is not suspected anymore.

Sometimes a device is recognized as permanently crashed. In this case, the action *crash* happens.

Delay

An action also happens when something takes longer than expected. The *delay* action notifies the application that there is a delay before the action ends. The application can give feedbacks to the users to ensure the application keeps being responsive and does not seem to be crashed.

Algorithm 9: Transaction of several interactions by freezing resources.

PRE: -
POST: All the interactions succeed or fail

```
function Transaction(actions:List_action<Action>)
  try
    block(resources(List_action))
  catch fail
  then
    free(resources(List_action))
    return fail
  end
  List<Action> succeed = empty
  for each action in List_action
  do
    perform(action)
    if succeed(action)
    then succeed union action
    else break
    end
  end
  if size(List_action) > size(succeed)
  then
    for each action in succeed
    do
      undo(action)
    end
        else
                return succeed
        end
        return fail
end
```

List of vertices

This action will list all the users and/or devices that are currently in the distributed system.

Snapshot

This action will take a snapshot of the distribution graph at a certain moment of time in the past.

Update all widgets

This action will try to update all the widget. If the action fails no widget should have been updated. This action is possible thanks to the use of the transaction action.

3.7.4 Behavior behind the User Interface

When we introduced the actions we only focused on what happened with the UI. This is good for local actions but it is not the case for distributed actions. The problem is that there is a behavior associated with an action. When the action os local the behavior is also local.

When executing a distributed action we need to know where the behavior of the action should be: locally, remotely or hybrid (mainly copied).

This was a huge question that was raised when doing the thesis even with very small examples. In fact there is no real study on this topic. Taking small examples shows that this is not possible to know what the user expect. Here are a few examples which cover all the different behaviors:

- Example 1: a presentation with a computer attached to a projector and with a smartphone used as a remote for changing slides.
- Example 2: a slide-show of pictures from a remote computer with a smartphone or another display
- Example 3: the number of the current slide in a presentation
- Example 4: a presentation with a computer displaying the current slide, the number of the slide, and the smartphone remotely changing the slides (with navigation buttons), while the next slide is changed locally and the number of the slides is synchronized both locally and remotely

We just need to look at the example 4 to notice that the behavior behind the UI running on the smartphone is quite common but pretty difficult to guess from the UI itself.

This example has already been the focus of a research[VDV05]. They tried to solve the problem by creating web services with a predefined scenario and a fixed behavior. However in real life it is not the same behavior that is expected by all the users.

3.7.5 Distribution mechanisms

In order to support actions between vertices we need some distribution mechanisms. The complexity of such mechanisms have already long been study in the domain of distributed applications. In this thesis we wanted to reuse such algorithms developed for distributed applications to adapt them for distributed systems.

As we have already said about in the document there are several problems that happen once we try to distribute the system. While connecting and sending messages from device to device may seem quite straight forward it has some unpredictable behaviors. Indeed a message that has been sent may not be received or received later or even received more than once. There already exist algorithms that ensure that a message sent is received once and only once by a device.

The devices are often connected with unstable communication mechanisms and the device itself can be unstable or crash. The dynamic distribution that we face in distributed system also allows devices to join and leave at run-time. This leads to the need of other mechanisms to detect and maintain such behaviors. There also exist mechanisms for this. They are called failure detectors. They detect when failures or important delays happen between two devices.

Another important point is how and where the data are stored. In an usual non distributed system or in a client-server architecture the data are stored in one and only one device. If this device crashes or leaves these data are then unavailable and sometimes lost. Using technologies such as peer-to-peer have solved this issue by creating replication and by running independently of a server. This is for sure something that we are really interested to see in distributed system.

This solution is also interesting to support such a simple situation as a user starting an application or a game and then leaving. Without distribution mechanisms the system will be stopped when the user leaves while distribution mechanisms can allow the system to go on living normally without even noticing the creator left it.

Reliable Broadcast

Sending a message through a network is not that difficult. If we know the target ip, we can create a connection and then send our message. If we want to be sure that the target get the message, we can send back a message to

acknowledge the reception of the message.

When you want to send a message to all the peers that are in your network and you want to ensure that each peer receive once and only once the message, you need to use a more complex algorithm: the Reliable Broadcast.

The definition of the reliable broadcast is: Events

- Request: $\langle rbBroadcast|m\rangle$
- Indication: $\langle rbDeliver|src, m\rangle$

Properties

- RB1. Validity: if correct d_i broadcasts m, d_i itself eventually delivers m
- RB2. No duplication: no message delivered more than once
- RB3. No creation: no message delivered unless broadcast
- RB4. Agreement: if a correct vertex delivers m, then every correct vertex delivers m

Concepts of the thesis

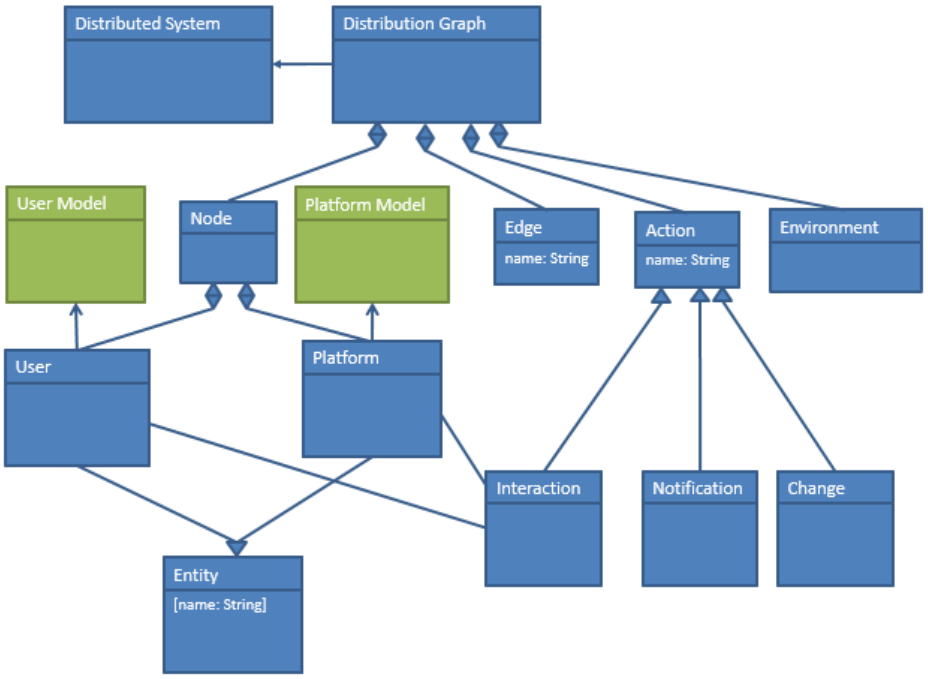


Figure 3.20: Concepts of the thesis

Chapter 4

Specifications of the toolkit

In the previous chapter we have introduced a formal model of a distributed system and have formally defined the concept of distribution primitives. Based on these formal definitions we want to create a toolkit that implements them.

In this chapter we first define the concept of Dynamic Distributed User Interfaces. Then we give the specification of the toolkit: JayTk. We explain what JayTk offers, how it instantiates the concept of Dynamic Distributed User Interfaces and what are the limitations with respect to this definition. After the specification of JayTk, we give the specification of Beernet which is an implementation of distribution mechanisms that we use for JayTk.

Finally we explain how applications can integrate the toolkit based on these specifications.

4.1 Definition of a Dynamic Distributed User Interfaces

A *Dynamic Distributed User Interface* (DDUI) is a user interface that can be partitioned and distributed at run-time over multiple devices. There is a protocol that defines what UI event reaches the application when there is an event on any of the devices.

DDUI is an extension of the concept of UI by adding the ability to distribute it across multiple devices and allowing this distribution to happen at run-time without requiring a predefined distribution scenario.

The User Interface (UI) of an application is the interface offered to the user in order to interact with the application. This UI is usually limited to the device where the application is executed and is strongly integrated into it. Applications do not offer a way to display the whole or parts of the UI to any other devices.

A Distributed User Interface (DUI) as defined in *Definition 2* allows end users to distribute any user interface element across different users and

devices. By allowing distribution, another device can control the application remotely without requiring that the core of this application is running on the remote device. When an event occurs on a DUI, this event is forwarded to the application which processes the action linked to this event.

A User Interface is dynamic if it is possible to remold it during the execution of the application. By opposition, a UI is static if it offers the same features in the same form during any execution of the application. A dynamic UI allows the users or the application itself to split parts of the UI into new UIs and to recombine UIs into a single UI.

A DDUI is a UI that can be dynamically remolded at run-time and any UI, and parts of UI, can be displayed, migrated, copied, on any other devices. When a device connects to the application, the UI of the application can partially or totally extends its interactions to it. If a device disconnects, the application can dynamically redistribute the UIs across the remaining connected devices in order to react to this disconnection. A dynamic UI also allows the users to change at run-time on which devices the UIs and their parts are displayed.

4.2 Specification of JayTk

JayTk allows applications to support, create and manage DDUIs. In this section we explain how we offer dynamics and distribution to UIs, what parts can be distributable and how dynamic, consistent and fault tolerant it can be.

In the section 3.1 we described two kinds of applications: distribution-aware and unaware applications. Almost all existing applications are non-distributed applications. We would like to offer developers a toolkit to extend current applications or create new applications that support DDUIs. By using the toolkit, an application is able to distribute parts or whole of any UI to other devices.

An application UI can offer several modalities: *graphical, vocal, and their combination* [PAT07]. A Graphical User Interface (GUI) is a UI where all the interaction elements are graphical. This is the standard UI to most of computing systems, as well as distributed systems. Our toolkit focuses on Graphical User Interfaces (GUIs). Thus JayTk offers the support for *Dynamic Distributed Graphical User Interface* (DDGUI), the subset of DDUIs only constituted of GUIs.

Most of applications have a graphical user interface (GUI) that is not distributable. It means that the actions that are triggered by events are always affecting the device where the graphical component is displayed. All these graphical components are displayed on the same device that runs the application. A DDGUI is a GUI that is decentralized from the application itself. It allows a device to trigger an event on another device.

GUIs elements are usually of the WIMP type (Window, Icon, Menu, Pointing device). Here is an non-exhaustive list of common graphical elements: button, label, check box, text box, radio button and canvas.

Each graphical element has several properties that are attached to it. E.g. for a button, although it may differ with the operating system and the toolkit, they usually have:

- content: the text that will be put inside the button
- action: the reaction to an event that has been triggered (e.g., when the button is pressed).

There can be other events supported by a graphical component. They do not all trigger an action.

JayTk offers a coherent and consistent storage where the GUI data will be stored. Each device connected to the application have a partial view of this storage. A device may only have a partial view of this storage.

In order to offer consistent storage, JayTk uses transactions to ensure that:

- either the whole transaction succeed, all the changes have been applied
- or the whole transaction is aborted, all the changes are canceled and the storage is left unmodified.

This allows JayTk to handle concurrent access to the storage in order to ensure consistency. If two events are concurrent, at least one of them will be aborted.

JayTk offers a mechanism that allows an event happening on another device through a DDGUI to reach the application and triggers the actions resulting from it. When an event occurs on a remote device:

- either the device attempt to modify the storage
- or the device forwards the event to the application

It depends from the nature of the event.

The text contained in a text box is a value in the storage. Changing this value through the DDGUI will lead to a modification of the value in the storage through a transaction.

However the action linked to a button is not a value in the storage. The event must be forwarded to the application which will trigger the action. We use message passing to forward the action from a remote device to the application.

Finally JayTk offers the ability to distribute any GUI across all the devices through distribution primitives. This distribution can be automatically triggered by the application or manually triggered by a user.

4.2.1 Distribution primitives

Based on the definition in the previous chapter we will now explain how JayTk instantiate the distribution primitives. To demonstrate this we will use the *Display* primitive has an example.

The statement for the display primitive is:
DISPLAY list_element [IN element [AS position]]

This means that there are 3 parameters for this procedure: a list of components to display, a container component where to display them, and the position.

Without parameters the *Display* primitive only occurs on the device where the application is running. A GUI will be created and will contain the list of elements contained in *list_element*.

If there is a parameter for *element* and *element* exists in the storage then all the elements in *list_element* are added into the container. This will trigger a transaction to modify the value of *element*. Any device that currently displays *element* will be notified that *element* has been updated and will change its GUI to fit with this change.

If there is a parameter for *element* but *element* does not exist in the storage then JayTk creates a GUI that contains all the elements in *list_element*, create a value *element* associated to this GUI and add *element* to the storage.

4.2.2 Events

JayTk allows applications to register for events. Here are the events related to devices joining and leaving the distributed system.

- *join*: every time a device joins the application, JayTk notifies the application
- *left*: every time a device leaves the application, JayTk notifies the application

To support the *left* event JayTk uses a failure detector mechanism. If a device is suspected of being crashed the failure detector notifies the application that the device may be crashed. A crash is considered as leaving the network while being back alive is considered as joining the network. When a suspicion is detected the device is removed from the application and the *left* event is triggered.

In order to support the *joined* event each device registers itself in the list of available devices. When a device is added to the list of available devices a *joined* event is triggered. This happens if a new device connects to the application or if a device that was suspected of being crashed is now back alive.

The specification for both events are:

Event:

---

upon event ⟨ *Crash*, d_i ⟩ **do**
  trigger ⟨ *Left*, *deviceId* ⟩
end

This event notifies the application that the device *deviceId* corresponding to d_i has left the network.

Events:

---

upon event ⟨ *Alive*, d_i ⟩ **do**
  trigger ⟨ *Join*, *deviceId* ⟩
end

This event notifies the application that the device *deviceId* corresponding to d_i has joined the distributed system.

4.3 Specification of Beernet[MEJ10]

The choice for supporting distribution mechanisms brought us to Beernet. Beernet is a peer-to-peer network implementation that comes with distribution mechanisms that we want to use in our toolkit. Using it allows us to separate the complexity of the distribution from the toolkit. It fulfills all the requirements regarding we set in the section 3.7.5. It deals with all the connections, disconnections, communication delays and communication failures. It is implemented on top Mozart; a multi-platform environment supporting Windows, Mac OS X and Linux.

4.3.1 P2P network

Beernet offers a peer-to-peer network communication.

Each peer of Beernet is called a node. Several nodes can connect together to form a network, the ring. Each node can connect and send messages to other nodes.

Its architecture is depicted in Figure 4.1.

The architecture is organized in layers. Let us read them from bottom to top. The first layer handles the network communication and uses TCP-IP protocol. It also ensures that the exchange of messages is reliable and check if failures happen.

The layer above is responsible for the ring. This is used for nodes to create and maintain the network and react if a node joins, leaves or fails.

The third layer stores data in a distributed hash table (DHT) with the primary operations: PUT and GET.

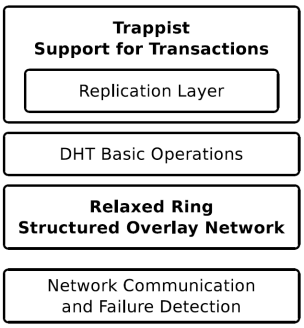


Figure 4.1: Global view of Beernet's architecture[MEJ10]

The upper layer is the transaction manager which handles more complex algorithms to ensure that all nodes always see the DHT as a coherent database. It also supports replications to ensure that no data is lost if less than 4 nodes crash at the same time.

4.3.2 Coherent storage

As already mentioned, Beernet provides a Distributed Hast Table (DHT) as a coherent storage. Every node is identified with a key that is produced through a hashing mechanism.

The DHT offers a robust and coherent storage which can resist to at least one failure without any problem. As we already discussed each data is stored in four replicas. To create replicas, the network is separated in groups of four nodes. Each object in the DHT is stored in one and only one group. The four nodes of this group are the replicas for the object.

Here is the failure model which guarantees the following results depending on the number of failures:

- If 1 replica fails, the DHT is not interrupted. The network will fix the failure by creating a new replica.
- If 2-3 replicas fail, transactions will be temporary interrupted until the network fixes the failures. This might not be noticed as this happens really fast.
- If the 4 replicas fail, all the data stored by these replicas might be lost.

In case of partitions in the network, it is still possible to read value from the DHT but not to change the DHT. It will stay in a read-only mode until there is no partition anymore.

Beernet also provides a registration mechanism which allow us to register as a reader of a key in the storage. If this key is modified, all the readers for this key will be notified that the value has been updated.

Here is the specification for the DHT:

Algorithm 10: Interface of the Beernet's DHT

Module:
Name: BeernetDHT (BDHT)
Events:
Operation: ⟨ Get | *key* ⟩
Returns the value stored with key *key*. It returns the atom 'NOT_FOUND' in case that no value is associated with such key.
Operation: ⟨ Put | *key*, *value* ⟩
Stores the value *value* stored with the key *key*, only in the node responsible for the key resulting from applying the hash function to key *key*.
Operation: ⟨ Delete | *key* ⟩
Delete the item associated with key *key*.
Properties:
D1: Validity
Any value stored is a value proposed.
D2: Coherence
All the nodes always have the same value.
D3: Termination
Every transaction eventually ends with a *commit* or *abort* decision.
D4: Atomicity
Either all operations take place or none of them

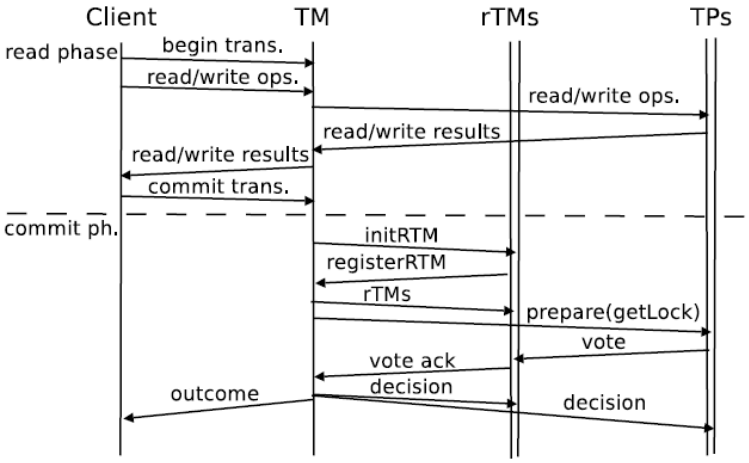


Figure 4.2: Consensus atomic commit on a DHT.

4.3.3 Atomicity

An important property of a distributed system is the atomicity. If an operation is atomic it means that either the whole operation succeed or that it fails. It is a guarantee to prevent some modification from happening partially and let the application in an incoherent state.

For example, an operation that can modify the size of all the buttons. If the operation is atomic it means that all the buttons either keep their current size (the operation fails) or they all get the new size (the operation succeed). The atomicity is not difficult to ensure on a local device but it is much more complicated on several devices. The delay and the reliability of the communication between all the devices lead to some uncertainty.

A way to ensure that all the devices get the modification is to use a reliable broadcast protocol which ensures that each device that is still available will be notified about the change. But there is a problem with devices that may have crashed before or during the broadcast. Even if the broadcast is reliable some devices may have crashed or some communication may fall down and some devices may never be notified about this modification. This leads the whole system in an incoherent state. The use of distribution mechanisms allows us to correctly handle this kind of situation.

To ensure atomicity Beernet provides a transactional replicated storage. Each data is replicated in different nodes and in order to change a value the majority of the replicas needs to be updated instead of all the nodes.

Beernet implements a consensus protocol to achieve the majority. Figure 4.2 describes how the consensus protocol works.

The client is the node which starts a read or a write transaction. This node becomes the transaction manager (TM) for the current transaction.

In order to write a value, it first starts by a *read phase* and then a *commit phase*. During the *read phase*, the TM contacts all transaction participants (TPs) involved in the transaction (i.e. holding a replica of an item in the transaction). The modification to the data is done without asking for a lock. When the client decides to commit the transaction, the *commit phase* starts. This means that it needs to get the lock of the majority of TPs for all the items involved in the transactions. But first a set of replicated transaction managers (rTMs) are registered. They will backup the TM in case it crashes. Then a vote is made by all the TPs and when the decision is achieved, it is sent to the client. The locks are released when all the TPs have received the decision.

Using transactions allows us to reach atomicity if there is no crash during the transaction. If the TM crashes after the vote but before taking the decision, one of the rTM can take over the transaction.

Here are the specifications for the transaction mechanism used in Beernet:

4.3.4 Failure Detector

A distributed system contains several computing devices that communicates together. These devices can appear, disappear and sometimes crash randomly.

When a device is trying to communicate with another one it is important to know if the communication is reliable and efficient. A message sent from a computing device to another one can be lost, received once or received several times, and the order of the messages is not always guaranteed. For an application to stay coherent in such a complex system it needs to get guarantees and acknowledgment that messages are correctly received only once and in the right order. It also needs to know what will happen in case of a device disappear and suddenly reappear in the system.

There are also issues regarding the sharing of data across several devices. If two computing devices want to modify the status of an object at the same time, there is a conflict that can lead to incoherence, mistakes and even failures. There is a need for a mechanism that will prevent the computing devices from accessing and modifying the same thing at the same time. In distributed computing they use transactions to guarantee that the modification either succeeds or fails.

It means that applications need distribution mechanisms to efficiently support the properties of a DS. Instead of creating new mechanisms we wanted to reuse existing mechanisms. Thus we recommend as part of a new methodology to use algorithms from distributed system to handle the inter-devices communications.

Here is the specification for Beernet's failure detector:

Events:

Algorithm 11: Paxos Consensus's API

Module:
Name: Paxos Consensus (TM)
Events:
Operation: ⟨ RunTransaction | *trans*, *client* ⟩
Run the transaction *trans* using the *paxos* protocol. The answer, *commit* or *abort* is sent to the port *client*.
Operation: ⟨ Read | *key* ⟩
Returns the latest value associated with key *key*. Strong consistency is guaranteed by reading from the majority of the replicas.
Operation: ⟨ Write | *key*, *value* ⟩
Write value *value* using key *key*. The new value is stored at least in the majority of the replicas. Updating the value gives a new version number to the item.
Operation: ⟨ Delete | *key* ⟩
Remove the item associated to the key *key* from the majority of the replicas.
Properties:
D1: Validity
Any value stored is a value proposed.
D2: Strong Consistency
Strong consistency is guaranteed by reading from the majority of the replicas.
D3: Termination
Every transaction eventually ends with a *commit* or *abort* decision.
D4: Atomicity
Either all operations take place or none of them

Algorithm 12: Interface of Beernet's failure detector

Module:
Name: BeernetFailureDetector (BDF)
Events:
Indication: ⟨ Crash | p_i ⟩
Notifies that node p_i is suspected to have crashed
Indication: ⟨ Alive | p_i ⟩
Notifies that node p_i is not suspected anymore
Properties:
PFD1: Strong completeness
Eventually every node that crashes is permanently detected by every correct node.
PFD2: Eventual strong accuracy
Eventually, no correct node is suspected by any correct node.

4.4 Applications

We consider two different kind of applications which can use our toolkit: mozart applications, and JayTk applications. Mozart applications are unaware of the toolkit running behind them. They are extended without any change to the functional core, which we will refer to as the *logic* in the remaining of this chapter. JayTk applications are distribution-aware applications that take advantage of being aware of the distribution to efficiently react to failures and connections.

4.4.1 Mozart applications

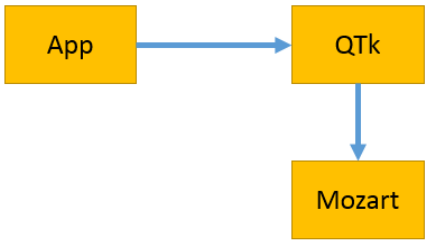


Figure 4.3: How mozart applications usually create their GUI.

Applications that have already been developed with Mozart can use QTk to create graphical user interfaces as in Figure 4.3.

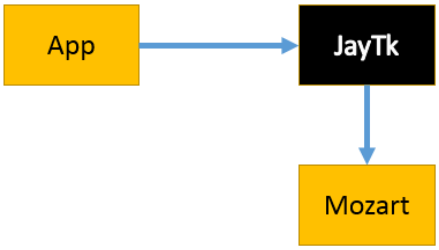


Figure 4.4: How mozart applications create their GUI with JayTk.

JayTk extends QTk in order to provide distribution capabilities to widgets. QTk is encapsulated inside JayTk. JayTk is the module that allows developers to create GUI as in Figure 4.4.

The applications still run on the device where it has been started but the GUI will be separated from the logic. All the devices are connected together as a large virtual space for GUI without overlapping. Figure 4.5 is a simple example of an application created without and with JayTk.

On the left, you can see that you have the application with its own logic that uses QTk to create its GUI. This GUI is displayed on the same device.

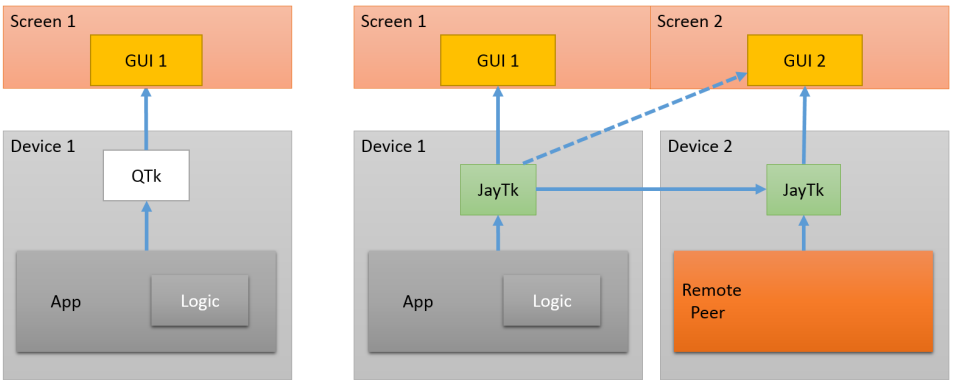


Figure 4.5: Comparison of an application created without and with JayTk

On the right, the same application, logic unmodified, now uses JayTk to create the same GUI.

This configuration allows any device to be aware of the existence of the application's GUI. Any other node can now get the whole GUI in order to create its own GUI which can be the same or a different GUI than the one displayed on the other device. In the example the numbers correspond to an id of the device. There are two devices in the example but there can be an infinite number of devices connected to *Device*1.

4.4.2 JayTk applications

Unless Mozart application in section 4.4.1 applications that are aware of JayTk can use more advanced features: notifications if a device joins or leaves the application, save current state of distribution, load a previous state of distribution, create distribution scenarios, and other features.

First let us talk about what can change with the current Build procedure. As previously discussed with Mozart applications it was not possible to add information inside widgets. With JayTk we want to allow applications to add a name into widgets in order to be able to distribute them.

This has an important consequences: widgets that are sent to JayTk are no more compliant with QTk. It is the reason why we process QTk_GUI from Jay_GUI. This is a way to purify Jay_GUI from information added inside widgets.

4.5 Conclusion

In this chapter we have defined what is a DDUI. Then we have instantiate this concept by giving the specification of JayTk, a software support to build DDGUI. We have then described Beernet that we use to create a Peer-to-Peer network that allows us to build a self-managing and scalable system

with transactional and robust storage. Beernet brought us the solution for several properties of a DS: concurrency, partial failure, lack of global knowledge and dynamic. It also allows JayTk to support Linux, Windows and Mac OS X which are the operating system for desktop computer that we have targeted. Finally we have explained how application can use JayTk to support DDGUI.

Chapter 5

Implementation of JayTk

This chapter describes the different phases and aspects covered by the implementation of JayTk, a toolkit to support Dynamic Distributed Graphical User Interfaces (DDGUIs). The most difficult aspect with JayTk is to support all the operating systems. We will briefly describethe results we achieved for each operating system.

As described in Chapter 3.1 there are two kinds of applications: distribution-aware and non-distributed applications. Both are supported by JayTk. Non-distributed applications are not aware that their UIs can be accessed from other devices. On the contrary distribution-aware applications can take advantage of the distribution capabilities. JayTk provides an API that allows these applications to react to connection, disconnection and failures of devices.

It was not possible to provide the toolkit on all the existing operating systems. We have selected a subset of operating systems to cover a maximum range of devices that is sufficient to represent most of the market. The selected operating systems are the following:

- Android: 3 phones + 1 tablet
- iOS: 1 iPod, 1 iPhone and 1 iPad
- Windows Phone 7 and newer: 4 phones
- Windows 7 and newer: 1 computers + 4 tablets
- Linux: 1 computer
- Mac OS X: 2 computers

To simplify the realization of the toolkit and maximize the compatibility with all these operating systems, we have decided to use an environment that already supports most of them. Our choice is the Mozart environment because it already supports: Windows, Linux and Mac OS X.

5.1 Structure of the toolkit

The thesis has been realized with a goal of separation of concerns. There are several layers that communicate together in a complex structure as in figure 5.1.

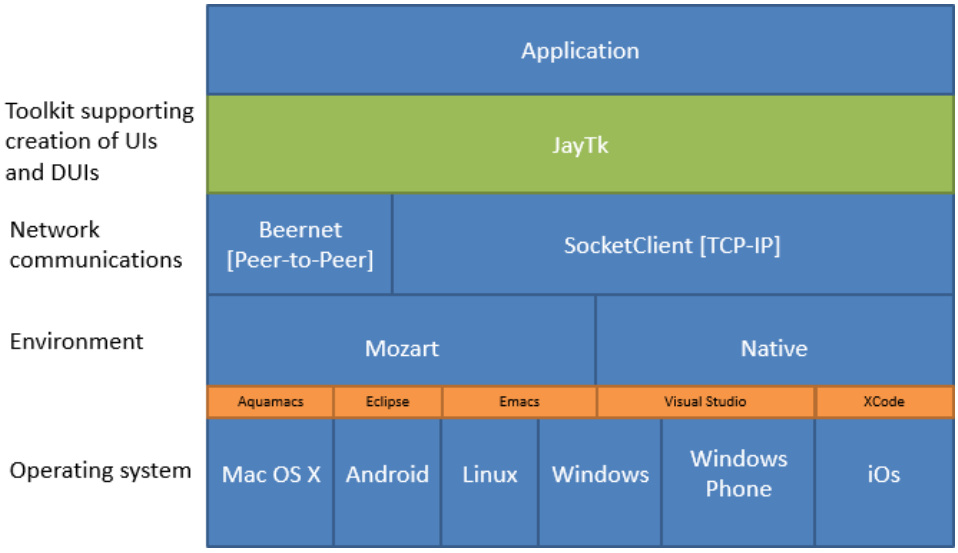


Figure 5.1: Structure of the toolkit

In the structure used for the thesis, the application runs on top of JayTk which encapsulates the creation of UIs and extends it with the ability to manage the GUI in real-time as described in Chapter 3 and specified in Chapter 4.

JayTk has been tested with two network communication mechanisms: Peer-to-peer and sockets. JayTk uses TCP-IP sockets for communications between devices that are not supported by Mozart. Each operating system supports TCP-IP sockets to use with native applications. TCP-IP is the standard used for communication between devices.

For peer-to-peer (P2P) communications, we use Beernet[MEJ10] which works on top of the Mozart environment. Beernet is also based on TCP-IP sockets. It allows us to connect devices together while hiding the complexity of TCP-IP sockets and also supports distribution algorithms to manage and maintain the network.

To connect devices together, one of the devices offers a ticket that allows other devices to communicate with it. Other devices use this ticket to establish a TCP connection no matter which of the network mechanism is used. With Beernet this TCP connection allows devices to join the network. Other communication applications using TCP-IP could also be used.

5.2 Running as a daemon

An application will be installed and run on only one device. The other devices will connect to this device through a daemon. The application can make part or whole GUIs available for other devices which will turn these GUIs into a DDGUIs. Thanks to this daemon a device can either retrieve part or whole of the DDGUI.

Each daemon will support:

- communication to other devices
- a rendering engine to create and manage GUIs

5.3 Mozart Environment

The Mozart Programming System is an advanced development environment for intelligent and distributed applications[Mozart]. It is available from the Mozart's website [Mozart] via the download link. The current version of Mozart is 2.0 and is still under development. There is also a version of Mozart 1.4 that has been ported to Android.

Mozart uses a multi-paradigm Oz language which supports declarative programming, object-oriented programming, constraint programming, and concurrency as part of a coherent whole. The GUI is created using the open-source widget toolkit called Tk. This toolkit is supported on Linux, Windows and Mac OS X. Mozart provides a module called QTk which helps people use Tk with a high-level windows programming interface.

Each widget in QTk is a record with a label being the type of widget and the parameters being the inner widgets and the widget's parameters. An example of a window created in QTk thanks to Mozart is provided in Figure 5.2.

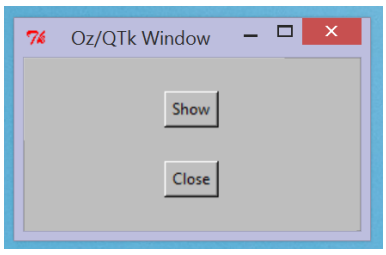


Figure 5.2: An example of GUI created with QTk

The code to create this GUI is provided in Listing 5.1.

```
declare
QTk={Module.link ['x-oz://system/wp/QTk.ozf']}.1
D=td(button(text:"Show"
```

```
            action:proc{$$} {Show 'Hello World'} end)
      button(text:"Close"
            action:toplevel#close))
Win={QTk.build D}
{Win show}
```

Listing 5.1: The code for creating Figure5.2.

5.3.1 Communication between devices

In order to support communications between all the devices it was important to use a standard communication protocol. There are two possible protocols: UDP or TCP. We have chosen to use TCP. In Mozart, there are also two possibilities to connect and communicate from one application to another: either by port or by socket.

A port is an asynchronous many-to-one channel that respects FIFO for messages sent from within the same thread. The messages are appended to a stream on the port's site. Messages from the same thread appear in the stream in the same order in which they were sent in the thread.

The sockets are standards TCP sockets. This allows us to use TCP sockets with other devices. Now we will describe how we could use both solutions in Mozart.

Ports

The creation of a port is very easy as depicted in Listing 5.2.

```
declare
% Stream of the port:
% Msg1|Msg2|...|MsgN|_
Str
% Creation of the PORT bound to the stream Str
ThisPort = {NewPort Str}
```

Listing 5.2: Creation of a port in Mozart.

To create a port, we first need to create an empty stream which is passed to the function *NewPort* as a parameter. Note that *NewPort* function is a variable that references the *Port.new* function. This new port, stored in *ThisPort* variable, can then be used as a destination.

We now have created the port. By default, we are not notified if the port receives any message. We need to use the stream to see if new messages arrive. The code to listen to new messages is depicted in Listing 5.3.

```
% A pointer to the tail of the stream
Tail = {NewCell Str}
proc{Listen}
   case @Tail
   % The port is now closed.
```

```
   % We stop listening
   of nil then skip
   [] Msg|Future then
      % Set the pointer to the new tail of the stream
      Tail := Future
      % Process the message
      % (through an external method)
      {Process Msg}
      {Listen}
      % Loop to listen to next messages
   end
end
```

Listing 5.3: Creation of a loop which listen to new messages received in the port in Mozart.

In order to listen to new messages we first create a pointer, *Tail*, to the tail of the stream. This pointer will be updated each time a message is received to directly point to new messages and forget older ones. Its format is a list where the first element is the next message to be received and the second element is a list to the unknown future of the stream. The pointer can either be the list *nil* which means that we have closed the port or a new message and the list to future messages. In case the port is closed there is nothing to do. The procedure is then terminated. If there is a message, we process the message *Msg* through another procedure called *Process* and we then loop in order to listen to upcoming messages.

An example of a *Process* procedure is depicted in Listing 5.4.

```
proc{Process Msg}
   {System.show 'Message rcvd : '#Msg}
end
```

Listing 5.4: An example of procedure to process a message received from the port in Mozart.

It is possible to send a message through the port (see Listing 5.5).

```
declare
SenderId = node1000
Msg='hello world'
{Port.send ThisPort packet(id:SenderId msg:Msg)}
```

Listing 5.5: Sending a message through the port in Mozart.

The *Post.send* procedure allows developers to send a message to the first parameter, *ThisPort*, which is the destination. In the example we build a message in the form of a record $packet(id : SenderIdmsg : Message)$ but it can simply be $'HelloWorld'$. The first feature of the record allows us to quickly find which node has sent the message. In this example the sender's id is the value $node1000$. The second feature is the message itself which in this case is 'hello world'.

Sockets

```
ServSocket={New Open.socket init()}
TCPPort={ServSocket bind(port:$$)}
{ServSocket listen()}
```

Listing 5.6: Creation of a TCP Socket and starting to listen to the port.

The creation of a socket is realized in 3 steps (see Listing 5.6): The first step is to create the socket with the operation New and using the method *init* on the newly created socket. The socket must then be bound to a TCP port which in this case is an integer (e.g. 50000). Then all that is needed is to listen to this port waiting for a connection request.

The next steps are the acceptance of a connection and then the establishment of a bi-directional connection. The socket can then be used to send or receive messages.

5.3.2 Mozart for all

Using Beernet allowed us to develop for all the operating systems supported by the Mozart Environment while offering a wide set of features thanks to distribution mechanisms. Our first goal was thus to port Mozart on other operating systems.

One of the thesis contribution is the porting of Mozart to Android. As Android is also based on a unix kernel, it was interesting to run Mozart on smaller devices. For the porting, we first needed to cross-compile Mozart for linux on an ARM-processor. Although Android is based on Linux kernel the user space where applications are executed are in Dalvik, a custom JVM (Java Virtual machine) designed by Google. Mozart cannot be run in Dalvik. It needs to be executed below the user space, directly in the Linux part of the android operating system.

Mozart for Android is a pre-compiled archive which is brought to the device by a custom Android application, that we have developed in Java. This archive is then opened and run by the same application to start the Mozart emulator. The main issue with this Mozart archive was that no graphical output was possible in the Linux part of Android. The output is in Java only through Dalvik. We have solved this problem by binding the Android application and the Mozart emulator. We have successfully run some prototypes on Mozart for Android but the whole system was slow and hard to use. Thus we have decided to stop working on porting Mozart to Android.Mozart for Android is not compatible anymore with the recent versions of Android and the prototypes we made are not fully functional anymore.

Due to these issues and time consumed in porting Mozart to Android, we have decided not to port Mozart to other operating systems. Instead we have

decided that the toolkit will support other operating systems through native applications.These applications are able to connect to and communicate with other devices. However they do not support yet all the aspects implemented in JayTk. This is left for future work.

5.4 Implementation of the actions

The first part of the toolkit was to implement the actions. In order to support main features of the toolkit, one important action to implement is DISPLAY.

With this action it is possible to display some UI element to a device wherever the device is located. Almost all the other actions can be deduced from this one. E.g. a copy action is just a display of an existing element, a move action is a display action of an existing element followed by a undisplay action of the element from the source device.

5.4.1 Using actions through different ways

The actions may be either triggered by the application itself, by the user or by the developer itself. As we never know who will manage the distribution of the application we wanted to provide several ways of using the actions.

Distribution scenarios

The concept of **distribution scenario** has been added in order to automatically trigger a predefined sequence of actions. This becomes interesting when there are a lot of actions to execute in a predefined sequence or when a sequence of actions may happen several times.

A distribution scenario is a script that will execute all the actions sequentially. These scenarios can be triggered in the same way as the actions.

An application that needs to modify the distribution regarding to some change in the distribution graph can trigger a distribution scenario. A simple example is an application running on several computers and then a computer becomes unavailable. To react to this loss the UI elements that was displayed on this computer can now be displayed on the remaining devices.

An example of scenario coded in Mozart is depicted in Listing 5.7.

```
% Called by JayTk when an Event occurs
proc {ReactTo Event}
   case Event
   of connect(Device)
   then
      {Display Device mainUI}
      {Copy Device manageUI}
   [] disconnect(Device)
   then
```

```
      {RestoreFrom Device}
   [] ...
   ...
   end
end
```

Listing 5.7: Example of scenario.

This example of scenario allows the application to receive parameters and react as precise as possible to the event that occurs. Scenarios can also be written into files but this will not allow us to use parameters.

Automatic distribution

Along with the distribution scenario we have the automatic distribution.

This is the execution of one or several scenarios. The scenarios are created by the developer and loaded when the application starts. It is a sequence of distribution primitives that will executed in the given order. The developer can either defines the scenarios inside the applications or load a file containing these scenarios at run-time.

The scenarios and actions will be triggered by the application at a certain moment. There is no way to cancel or undo the actions that have been triggered by this mechanism. However other distribution mechanisms makes it possible to repair or redistribute the application through manual distribution.

Manual distribution

When an application is running it is still possible to manage the distribution. This allows users to distribute the UI according to their own needs. There is no need for a device to join or leave the network to trigger distribution manually. Someone responsible for the application can fix, undo, reset the distribution if automatic distribution went wrong.

An example of a manual distribution is a drawing tool in which you can manually choose which device will be used to draw. This can be set by automatic distribution but if the result does not fit with the user needs, or if the user decides to go on drawing on another device, then the user can manually change where the drawing tool is displayed.

In order to allow this we need to give some ways for developers and end-users to perform the possible actions. The choice of enabling the user to have such power is let to the designer and the developer.

The manual distribution can be triggered thanks to an additional UI or through a command-line interface as in Figure 5.3.

A command-line interface is a good way to allow the developer of an application to keep full control on its distribution.

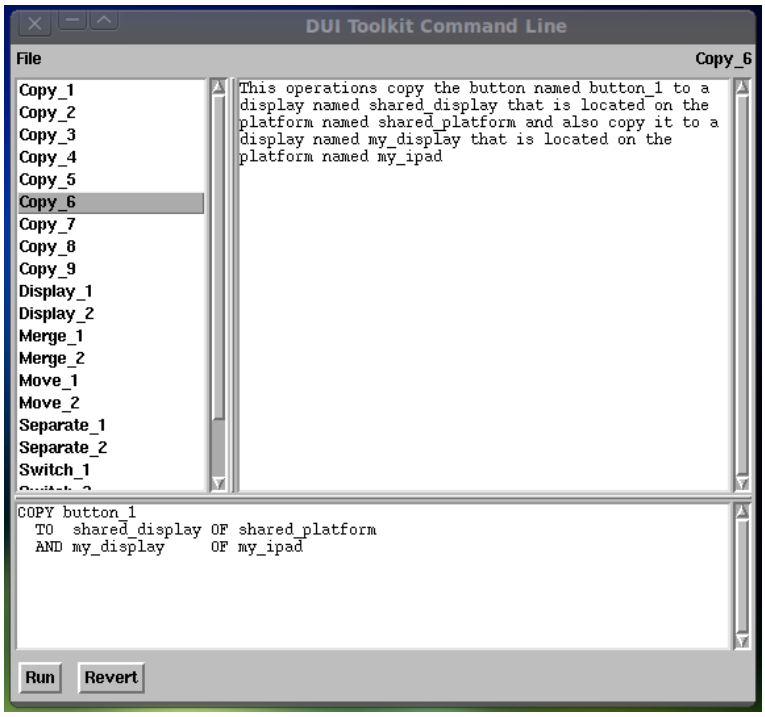


Figure 5.3: Example of a command-line user interface.

5.5 Porting Mozart to Android

As Android was based on a unix kernel, it seemed to be a good idea to port Mozart on Android in order to support the toolkit without efforts. We successfully ported Mozart to Android but with some important drawbacks.

The first issue is the fact that the Mozart emulator runs as a unix process which is in the low-level part of Android. Thus, the environment has been deployed, unzipped and installed thanks to a specific application that we have made. After the application is installed, it still needs to run and stay running. It is not possible to maintain, pause or stop the environment. This drawback leads Android to be running slower because there is an environment switch between Android and Mozart which leads to a lot of environment save and load which is time consuming.

Another drawback is the lack of QTk implementation in Android. Indeed, QTk has been dropped from Unix kernel and Mozart uses QTk. The solution we have provided in the port is an interface in Java that communicates to the Mozart environment. Instead of creating UI elements in QTk, the elements are created in Java using native elements.

However using native elements from Android has a big drawback. The way Android displays elements is not the same way that we have been used to in Mozart and in some other environments. Instead of asking to draw

something in the display, Android needs to redraw everything a few times by second. This leads to a lot of communication between Mozart engine and the Android run-time. Unfortunately, it also leads to a huge slowdown of the system and the application when too much communication happens between both engines. A solution for this problem would be to create a cache of all the graphical elements and communicate less often with the Mozart engine while creating a maximal time delay that is known and could be adapted.

5.6 Implementation of JayTK on other operating systems

We have decided to support distribution on other operating systems through a limited set of features and quality of service. The reason why we think that this solution is acceptable is that Beernet framework has proven that our method is working.

Native development allows us to quickly develop and use as much as possible native features (such as memory management, user interface API, and other interesting features provided by the device. It would not have been to support them directly in the Mozart environment. Native developments also allow other researchers to reuse and extend them to later support distribution mechanisms.

5.6.1 Mozart

The creation of the rendering engine is the hardest part of implementing JayTk. Each operating system has its own way to create user interface elements. In Mozart, the graphical toolkit used is QTk which is an extension of Tk. The graphical elements are records and a simple button can be written as simply as *button(text:"Hello")*.

Thanks to pattern matching, programming with records in Mozart is very simple and it is easy to handle all the events of the user interface outside the description of the user interface. There is a special widget, the placeholder, which can change what it contains.

The power of records is that the label gives the information which kind of widget it is, the attributes can be tested and are always available under the same name and attributes in type of integer are the children of the widget. The structure of a user interface is a tree where each node can have several children and some attributes.

The code of the main function for the rendering engine on Mozart is depicted in Listing 5.8.

```
fun{JTHelper Wid_init} % td, lr or button
   L = {Label Wid_init}
   ToAdjoin = {NewCell nil}
   fun{CommonForTDandLR W}
```

```
      for I in {Arity W}
      do
         if {Int.is I} then
            Res = {JTHelper W.I}
         in
            if Res == invalid then
               ToAdjoin := invalid
            else
               ToAdjoin := {Append @ToAdjoin [I#Res]}
            end
         else
            ToAdjoin := {Append @ToAdjoin [I#W.I]}
         end
      end
      if @ToAdjoin == invalid then
         invalid
      else
         case {Label W}
         of td then {AdjoinList td @ToAdjoin}
         [] lr then {AdjoinList lr @ToAdjoin}
         end
      end
   end
in
   case L
   of td then {CommonForTDandLR Wid_init}
   [] lr then {CommonForTDandLR Wid_init}
   [] button then
      for I in {Arity Wid_init}
      do
         ToAdjoin := {Append @ToAdjoin [I#Wid_init.I]}
      end
      {AdjoinList button @ToAdjoin}
   else
      invalid
   end
end
```

Listing 5.8: Rendering engine on the Mozart environment.

The function navigates through the tree and process all the children until the leaves are reached. Then go back to the parents and create the whole record which represents the root of the user interface.

The root is a special widget in Mozart. It is either a top-down (td) or a left-right (lr) widget. It has a special property *title* which allows developers to give a name to the windows. Other widgets cannot be used as the root. However, the rendering engine can implicitly create the root widget if it is not provided.

We have implemented a simple function for this which will be called before the creation of the user interface as depicted in Listing 5.9.

```
fun{NotTopLevel Wid}
```

```
   Res = {JTHelper Wid}
in
   if Res == invalid then invalid
   else td(Res)
   end
end
```

Listing 5.9: Small trick to create UI without a root widget.

To support existing applications without additional coding we wanted JayTk to be fully compatible with QTk. The UI can then be created with the exact same code as without JayTk. The widget provided by our rendering engine is directly sent to QTk in only two lines of codes as depicted in Listing 5.10.

```
Win = {QTk.build JT}
{Win show}
```

Listing 5.10: User interface is created with QTk and extension are hidden.

5.6.2 Android

The porting of Mozart to Android allows us to use the logic of applications developed in Mozart without any modification. However there was a big problem with the UI: no Tk support.

This was a problem that we have solved by creating a bridge between the Mozart environment on Android and the java library for creating user interface elements. The reflection provided by Java allowed the Mozart environment to create each object of the user interface in the Mozart code itself. For this, we had to use each class of widget. The classes currently used in our implementation are depicted in Listing 5.11.

```
ButtonC  = {J.c 'android.widget.Button'}
FrameC   = {J.c 'android.widget.FrameLayout'}
LLayoutC = {J.c 'android.widget.LinearLayout'}
RLayoutC = {J.c 'android.widget.RelativeLayout'}
ViewC    = {J.c 'android.view.View'}
WindowC  = {J.c 'android.view.Window'}
```

Listing 5.11: Java widget classes stored as Mozart variables.

We can then find the Button, the Frame (which is equivalent to the top-level widget in mozart), the LinearLayout (which either is a top-down or left-right widget depending on the direction of the layout), the RelativeLayout for other type of layout, the View (which works for several widgets) and the Window is the application view itself.

In order to create a left-right widget all we need to do is to create the linear layout and set the orientation as horizontal as depicted in Listing 5.12.

```
LLLR = {LLayoutC.new i({ContextC.cast This})}
{LLLR.p setOrientation({LLayoutC.get 'HORIZONTAL'})}
```

Listing 5.12: Creating a LR widget in Mozart for Android.

This is equivalent as the code in JAVA depicted in Listing 5.13.

```
LinearLayout LLLR = new LinearLayout();
LLLR.setOrientation(LinearLayout.HORIZONTAL);
```

Listing 5.13: Creating a LR widget in native Android java.

5.6.3 Windows Phone

Windows Phone is not based on Unix and is using a very different programming language and user interface protocol. Unlike other frameworks, in Windows Phone there is only one package from which each widget comes: *System.Windows.Controls*. All the classes in this package inherits from the class *System.Windows.UIElement*.

The classes used for creating the rendering engine are depicted in Listing 5.14.

```
Button
Grid
ScrollViewer
StackPanel
TextBlock
TextBox
```

Listing 5.14: Classes used for creating widgets in Windows Phone.

In order to create a left-right widget all we need to do is to create a StackPanel and set the orientation as horizontal. This is depicted in Listing 5.15.

```
StackPanel sp = new StackPanel();
sp.Orientation = Orientation.Horizontal;
```

Listing 5.15: Creating a LR widget in Windows Phone.

In Windows Phone the use of the native framework allow us to directly modify the instances. The variable *sp* is an instance of a StackPanel and we set the value of the attribute *Orientation* to the value of the enumeration *Orientation.Horizontal*.

5.6.4 Windows

Windows 7 and Windows 8 are very similar to Windows Phone. Thanks to the homogeneity of the operating systems from Microsoft, it is very simple to create a Windows Phone, Windows 7 or Windows 8 applications with

most of the code being common. Even if it is not the same class than for Windows Phone, there is also one package from which each widget come from: System.Windows.Forms. All the classes in this package inherits from the class System.Windows.Forms.Control.

The classes used for creating the rendering engine are depicted in Listing 5.16.

```
Button
DateTimePicker
GroupBox
Label
Panel
PictureBox
RadioButton
RichTextBox
TabControl
TabPage
TextBox
TrackBar
```

Listing 5.16: Classes used for creating widgets in Windows 7.

The creation of a left-right widget is more complex when using Windows Forms. There is no special widget that will add the next widget next or below the other. We need to create a panel and then to add the correction x and y position of the widget to be displayed beside or below the previous one. The code is depicted in Listing 5.17.

```
// init the left position
int left = 0;
/* ... code removed ... */
Panel lr = new Panel();
// set a default size
lr.Width = 300;
// set a default height
lr.Height = 70;
// set the new left position of the widget
widget.Left = left;
// add the widget to the panel
lr.Controls.Add(result);
// process the left position for the next widget
left += result.Width + 2;
```

Listing 5.17: Creating a LR widget in Windows 7.

5.7 Conclusion

In summary we have implemented JayTk in the Mozart environment. This allows us to reuse complex distribution algorithms with no effort thanks to Beernet which manages the whole network configuration and provides

transactions upon a distributed hash table and is resistant to failures. A limited subset of JayTk's features have been implemented on Android, iOS and Windows apps in order to support more operating systems and prepare for the mobile revolution.

Chapter 6

Case Studies

During the thesis, we wanted to test the method and the toolkit on some case studies. Instead of complex applications depending on a huge amount of data, we wanted simple applications with several users and small amount of data.

This chapter describes the case studies that have been studied during the thesis. The first case study is a small chat application which has been extended with a migration capability. The second case study is an efficient collaborative distributed drawing application. An example of Pictionary has then been implemented to demonstrate the multi-user and DDGUI support in JayTk. Another small example has demonstrate the possibility of distributing parts of a UI across several kind of devices.

6.1 DistribuChat

DistribuChat is a small chat application that supports the migration of part of the UI to another device. This demonstration has been presented at EICS 2009.

Complexity	**Nb. Users**	**Nb. Devices**
Low	2	2

Table 6.1: Information about the complexity of case study DistribuChat

6.1.1 Specification

DistribuChat is peer-to-peer application that allows users to send a message from one user to another.

6.1.2 Implementation

DistribuChat has been implemented on Mozart for Linux (see Figure 6.2).

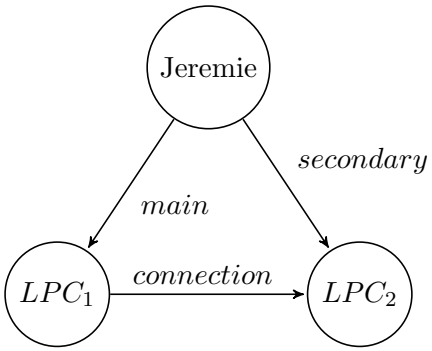


Figure 6.1: Distribution Graph of the DistribuChat demonstration.

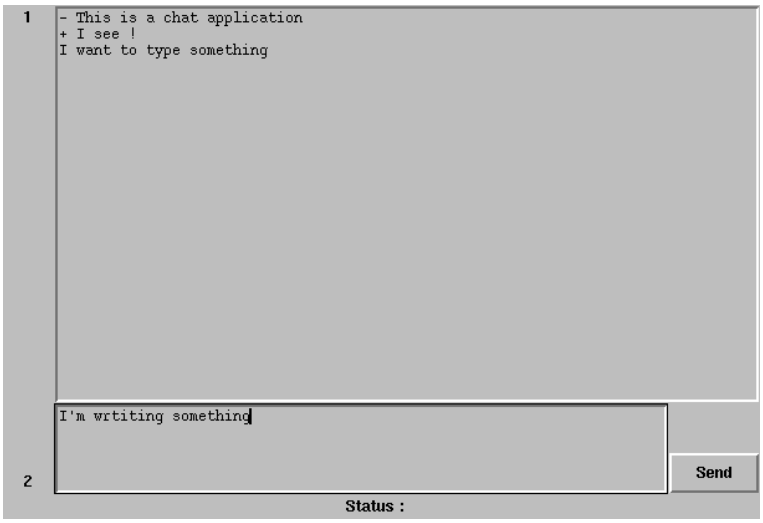


Figure 6.2: The GUI used for the DistribuChat's example

The GUI can be decomposed into pieces (see Figure 6.3).

6.2 DeTransDraw - DeTransDrawid

DeTransDraw (DTD) is a drawing tool that allows several users to collaborate on a drawing area. The goal is to handle the concurrency in an efficient way. This demonstration has been presented at the final review of the SELFMAN project.

Complexity	**Nb. Users**	**Nb. Devices**
Low	1-n	1-n

Table 6.2: Information about the complexity of case study DTD

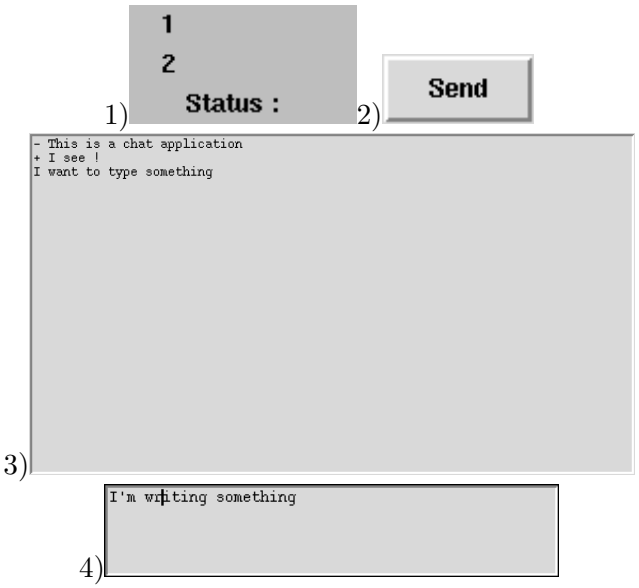


Figure 6.3: The GUI of DistribuChat has been split into pieces: 1) the status of the chat, 2) the send button, 3) the chat window, 4) the typing window

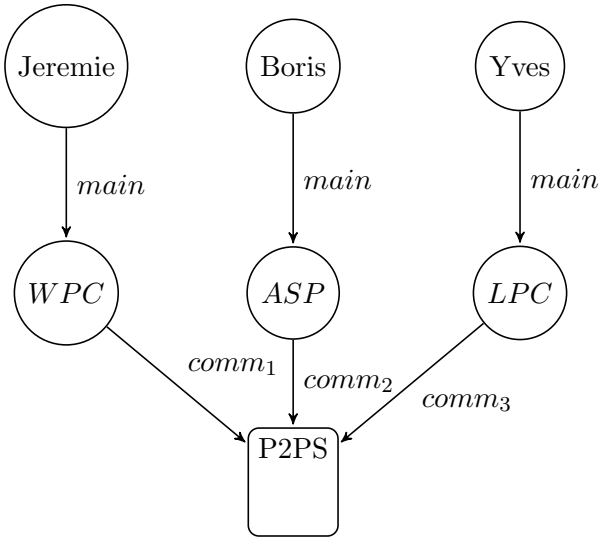


Figure 6.4: Distribution Graph of DTD and DTDid demonstration.

6.2.1 Specification

DTD is an application that allows users to draw and edit drawing while keeping coherence between all the views of the drawing. If a user creates a blue circle, all users can edit the circle by moving it somewhere else in the area or delete it. To support coherence it is important that if someone is

currently editing the circle, others should not be able to edit it.

The main property of DTD is to allow collaborative edition of a drawing such that two properties are satisfied:

1. each user can perform instantaneous edits without delays caused by the network

2. the editor maintains a single coherent drawing

To simultaneously satisfy these apparently conflicting properties, the editor allows local edits to be done speculatively. These edits will eventually be accepted by the editor if they are not in conflict. Speculative edits that are in conflict with other edits may be canceled by the editor. The user interface distinguishes not-yet-accepted edits from accepted edits.

6.2.2 Implementation

There are two implementations of DTD. On Linux and Windows, DTD uses QTk. The Android version uses a native GUI and is called *DeTrans-Drawid*(DTDid).

It starts with the creation of the UI. This UI is the same on all the devices. It is a very basic UI which allows users to create and select rectangles and ovals. The color of the inner bound and outer bound can also be customized. The screen shot of the prototype of DTD with the basic UI can be found in Figure 6.5 and the one of DTDid in Figure 6.6.

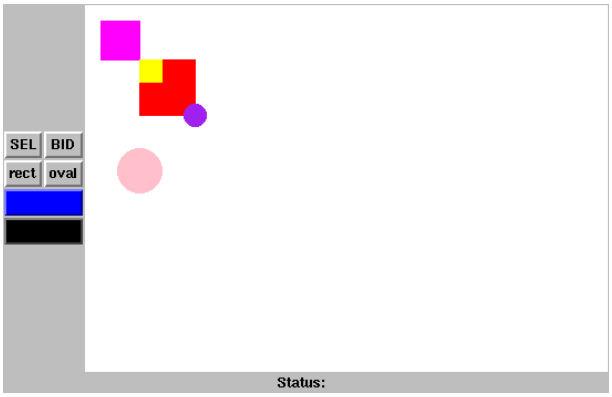


Figure 6.5: The basic UI of the DTD application

To support collaboration when editing elements DTD uses transactions. The user first select the object to lock it and prevent other users from being able to edit it. Then he can edit the object until he clear the selection of the object to commit the changes and release the lock. The selection of the object will initiate a transaction for this object and will keep it locked. If two users try to select the same object, at most one can get the selection.

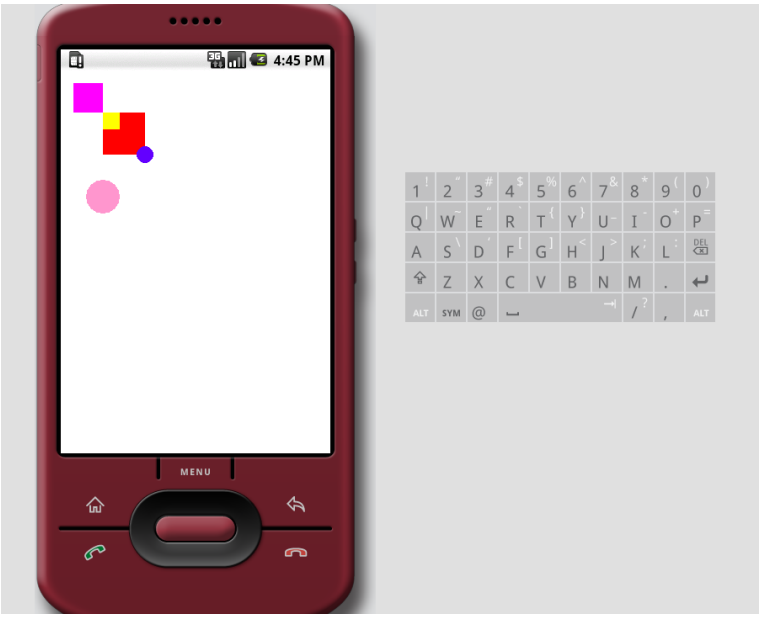


Figure 6.6: The basic UI of the DTDid application

The locking mechanism is more efficient than waiting for the selection before starting to edit.This means that users do not need to wait for the result of the transaction before starting to edit the object. There is no major delay if communications become slow or fail. Users can start editing the object as soon as the object is selected. A visual feedback is given when a user asks for a lock. While waiting for the lock, the object selected has red handles. When the lock is achieved the handles are colored in black as in Figure 6.7. The modifications are not guaranteed until the lock is granted. When the lock is granted the modifications are guaranteed to succeed.

The state diagram of a user is depicted in Figure 6.8.

The structure of the application created with the toolkit is depicted in Figure 6.9. Each DTD application is a node in a peer-to-peer network while DTDid applications are connected to one of the node in the network. A DTDid application is composed in two parts: the logic and the GUI. While the logic has been implemented in Oz based on the same code of DTD, the GUI is created in JAVA thanks to a bridge implemented for the toolkit.

The whole network is based on Beernet and uses the Distributed Hash Table (DHT) to store the data across all the nodes. While DTD has a direct access to the DHT, DTDid asks a node to get access to it.

6.2.3 Evaluation

Thanks to JayTk we have build a demonstration to show that it was possible to create a collaborative drawing tool between several devices running

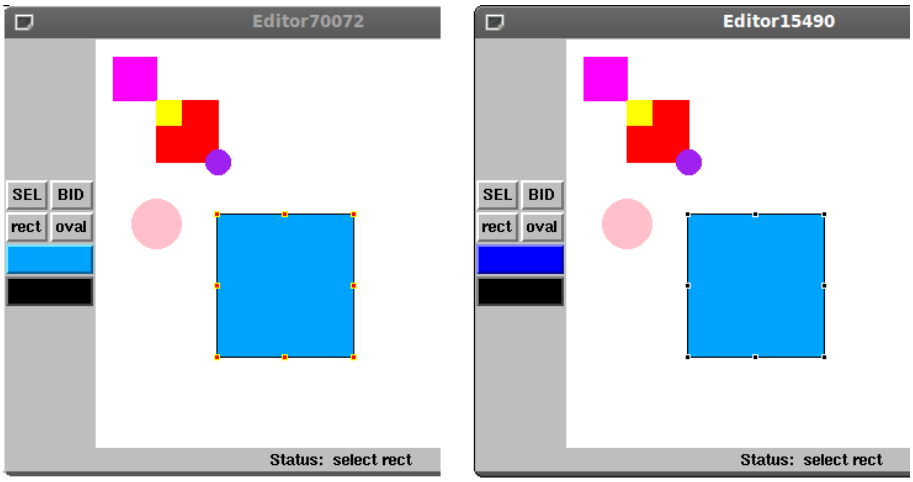


Figure 6.7: On the left, the user is in *Asking for locks* state and the handles of the big square are red. On the right, the user is in *Got locks* state with the handles in black.

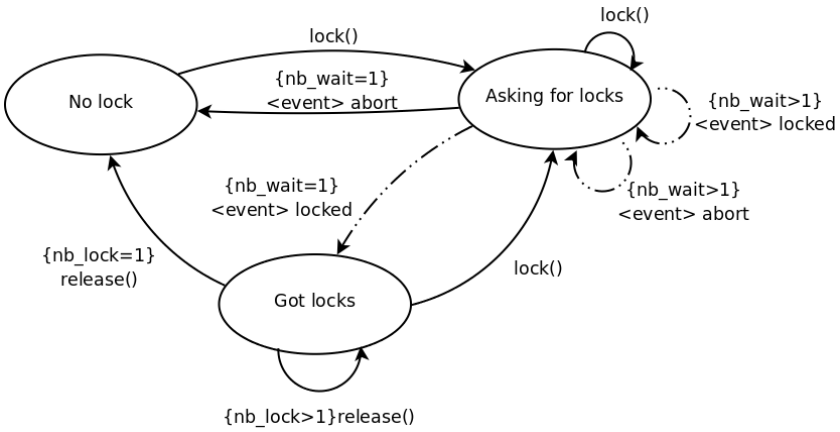


Figure 6.8: The state diagram of a user

different operating systems and which does not suffer from network latency. A description of the demonstration is available in Appendix 1.

The coherence between all the devices is ensured by using transactions for each modification. The eager-locking mechanism allows us to lock an object in the drawing which prevent two devices from modifying the same object at the same time. The transactions and the eager-locking mechanisms are directly provided by Beernet which means that JayTk does not need to know which algorithm is used for the locking and transactions systems.

The porting of Mozart to Android has allowed us to use the same code on both operating systems. This proves how easy it is to port an application on all the operating systems supported by the Mozart Environment.

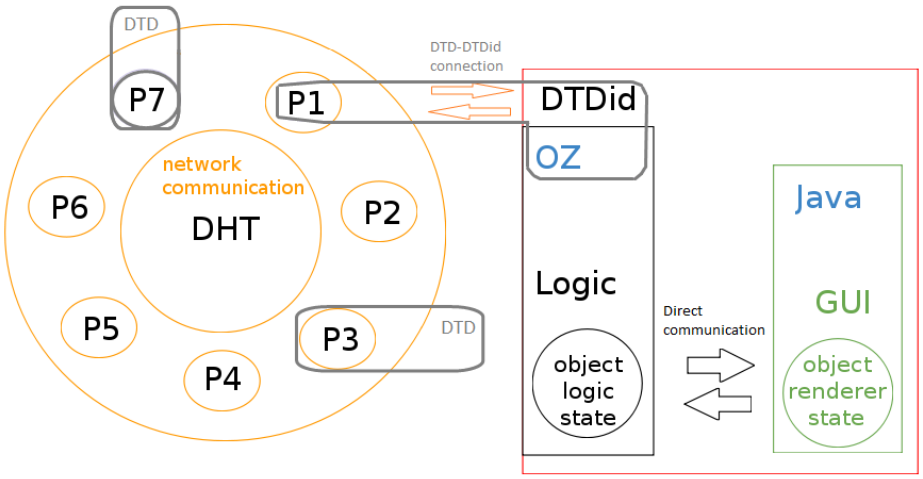


Figure 6.9: The structure of DeTransDraw and DeTransDrawid applications

6.3 Mobictionary

Mobictionary is a distributed Pictionary as described in 1.1. The adaptation of this game to a digital game is not simple. While it is clear that there is only one drawer, it is not clear to know who are the guessers, who chooses the word and who decides who and how to say that a player guess the word.

This demonstration has been presented at several conferences and events.

Complexity	Nb. Users	Nb. Devices
Medium	2-n	2-n

Table 6.3: Information about the complexity of the case study

6.3.1 Specification

Before going into the implementation of the Mobictionary, we need to clearly specify the way it will work. For this, we are going to define what is the initial state of the Mobictionary and the evolution of the game according to the connection of users, devices and with the game advance.

Initial state

Complexity	Nb. Users	Nb. Devices
Low	1-2	1-2

Table 6.4: Information about the initial state of the Mobictionary

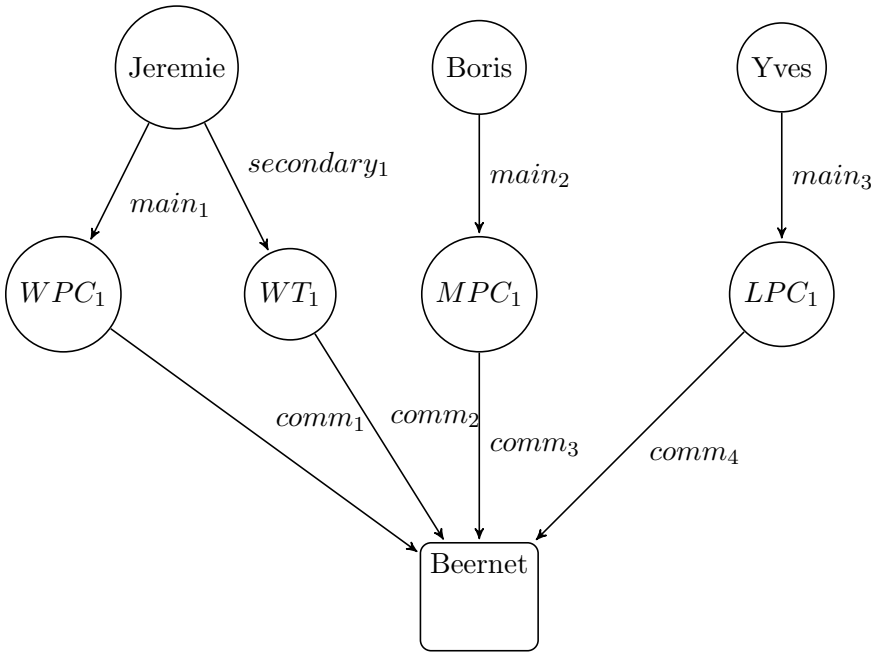


Figure 6.10: Distribution Graph of the Mobictionary demonstration.

As most applications, the Mobictionary is not created when there is no player. In order to take part of the game, a user has to create a new game or connect to an existing game.

When there are at least two players, it is then possible to start playing the Mobictionary. In this scenario, there is a role for each user: the drawer and the guesser. As neither the drawer nor the guesser should choose the word that they will have to guess, it is the application that will provide random words to the drawer.

Two main roles: drawer and guesser

Complexity	**Nb. Users**	**Nb. Devices**
Low	2	2-4

Table 6.5: Information about the complexity with 2 players

The drawer will have a specific UI that will allow him to draw something, to choose the color in which he is going to draw and to see the word that he needs to make the other guess.

The guesser has another UI that will allow him to see what the drawer is currently doing and to say what he thinks that is the current word to be guessed.

The game should alone switch the roles every time the guesser finds the right word. When the word has not been found, the drawer keeps his role. When there is only two players, it is not possible to choose who is the winner. Every time the guesser finds a word he earns one point. If the word has not been guessed, the drawer loses one point.

Mobictionary in team

Complexity	Nb. Users	Nb. Devices
Medium	4-n	2-n

Table 6.6: Information about the complexity of the whole case study

The game becomes more interesting when other players join the game. There is only one drawer and other players become guessers. The guesser who finds the word still earns a point but also becomes the drawer. The other guessers keep their role and the drawer becomes a guesser too. After 10 rounds, the player with highest score wins the game.

During each 10 rounds, it is not possible to join a game. It will only be possible to join the game before or after the 10 rounds to start a fresh new game.

When there are more than 3 players, it is possible to create teams. Each team should at least have two players and the smallest should only be smaller than the biggest one by one player (4-4-5 is ok, 3-5-5 is not). At each round, one team will be selected to draw. There will only be one drawer by round. If another team finds the word, this team will now get the drawing board and will choose one player that will become the drawer. If the active team finds the words, another player of the team will become the drawer and other teams stays as guessers. Any player that guesses a word makes the team earning one point. If no guesser finds the word, the team that is currently drawing loses one point. The first team earning 10 points will become the winner of the game.

6.3.2 Implementation

The Mobictionary example supports two way of distribution: automatic and manual. When roles changes the distribution is automatically triggered. Users have also the ability to manually trigger some distribution. They can split their UI across all the devices they own.

Here are the key roles and their Graphical User Interfaces (GUI):

- **Observer :** a person that is not currently trying to guess the word. This role is for opponents to the playing team. When there are only two players, the drawer will also have the observer UI. For observers,

the UI is mainly the ability to start, end the current round and to enter the word to guess. The round is ended when a guesser has found the right word. This UI is depicted in Figure 6.11.

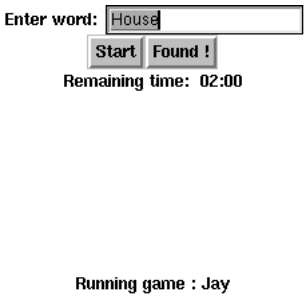


Figure 6.11: User Interface of the observers.

- **Drawer :** a person that helps other players find the word to guess by drawings. The minimum UI needed for this role as in Figure 6.12 is the drawing tools and information about the game. The user sees the word he has to draw about and has the tool to draw on the shared area.

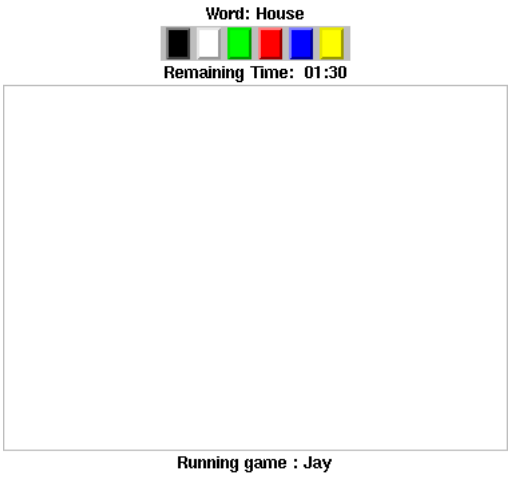


Figure 6.12: User Interface of the drawer.

- **Guesser :** a person that has to guess the word. If there are teams, guessers are in the same team as the drawer. The drawer may not be a guesser and his associated UI is in Figure 6.13.

Depending on the role associated to a player, he will get the appropriate GUI for his role. When roles change, the distribution of the UIs is reprocessed to keep a coherent state.

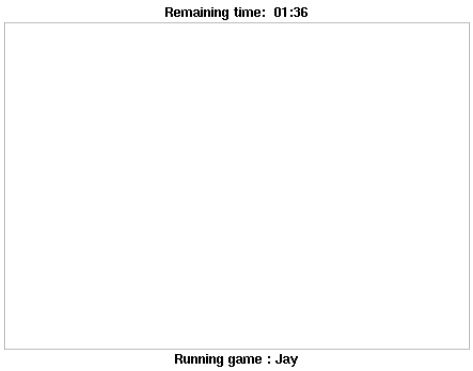


Figure 6.13: User Interface of the guessers.

Simplified Mobictionary

Complexity	Nb. Users	Nb. Devices
Low	2-n	2

Table 6.7: Information about the complexity of the simplified Mobictionary

The first case study is a simplified variant of the Mobictionary, where there is no team. Observers are players waiting for next game to begin. Each person is a single player as the drawer or a guesser. There is only one drawer at the same time but there may be several guessers.

The game starts with an initial state where the application is not started, the current state is empty. When the first user starts the application, he needs to create a room for the game. Other players will then join this room to play.

The UI for this first player allows him to create a new game as in Figure 6.14.

Figure 6.14: Creation of a game when no game is already started.

In Figure 6.15, we can see the pseudo-code to create this UI.

The first steps create and display a button and an entry in a new window. These two widgets will then be associated in a new widget arranging them from left to right. The last step is the creation of the window with all widgets arranged in the desired order. The *td* widget created by the main *Display* contains the widgets *create_game*, *status* and zero to several *b***X**. The name

```
{Display
   td(name:p1
      button(name:b_c glue:e text:"Create:")
   )
}
{Display
   entry(name:e_c glue:w bg:white
         init:"Own game"
         handle:HEntry return:R)
#p1}
{Display lr(name:create_game b_c e_c)#p1}
{Display
   td(name:p1
      create_game
      td(name:join)
      label(name:status glue:swe
            text:"Waiting for a game"
            bg:white)
   )
}
for I in {DiscoverGames} do
   {Display
      button(name:b#I glue:nswe
          text:"Join "#I}
#join}
end
```

Figure 6.15: Pseudo-code for creating initial UI.

used when we create widgets is the key to use the widget later. As we can see, the button named b_c appears in the creation of the lr widget.

As there is currently only one player, the game cannot be started. The application still needs a player before being able to start. The next state is the connection of a second player.

Two players are now connected as Observers waiting for the game to begin. Figure 6.16 shows the current state of the application for the different roles;

To create the UI of the Player 2, it only needs to copy Player 1 UI with code 6.17

As the game is created, both players got an update with their UI looking like in Figure 6.18.

The code to update the UI is presented in Figure 6.19.

Now, both players are waiting for the game to begin. Their UI is the

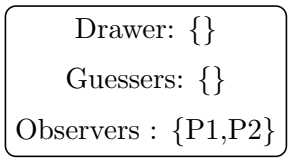


Figure 6.16: Second state. Player 1 and 2 are connected.

```
{Copy p1 td(name:p2)}
```

Figure 6.17: Pseudo-code for updating UI after game creation.

Figure 6.18: Creation of a game when no game is already started.

same until one of the players chooses to start the game. As the minimum number of players is reached, the game can start and each player will now be assigned to a role. In Figure 6.20 Player 1 becomes the drawer and Player 2 a guesser.

When Player 1 becomes the drawer, the UI has to be redistributed to this new role assignment. The result of this redistribution appears in Figure 6.21. This UI slightly differs from 6.12 because Player has to stay an observer. As Player 2 is the guesser, he is not allowed to have the *Start* and *Found*! buttons. This role should be assigned to observers, if there were any. The only solution is to assign this role to the drawer itself.

Thanks to the UI he is already playing with, the adaptation of the UI is only a small piece of code. In three statements, the UI is adapted. The code can be found in Figure 6.22.

Thanks to the UI he is already playing with, the adaptation of the UI is only a small piece of code. In three statements, the UI is adapted. The code can be found in Figure 6.22.

The UI of the second player also has to be adapted to the new role assignment. He now has the guesser UI as in 6.23.

The code for this adaptation can be found in Figure 6.24.

```
{Undisplay create_game#p1}
{Update status "Running game: "#Name}
{Display
   td(name:observer
      lr(name:enter_word glue:nwe bg:white
         label(bg:white text:"Enter word: ")
         entry(glue:w bg:white
               init:"House"
               handle:HEW)
      )
      lr(name:start_found bg:white glue:nwe
         button(glue:e text:"Start")
         button(glue:w text:"Found !")
      )
      lr(name:remaining_time bg:white glue:swe
         label(glue:e bg:white
               text:"Remaining time: ")
         label(glue:w bg:white text:"02:00")
      )
   )
#p1}
{Display td(name:p1 create_game observer status)}
```

Figure 6.19: Pseudo-code for updating UI after game creation.

Here, the transition triggered by the connection of a new player can be a loop from the current state. The only modification happening when a new player connects is an update in the observer list. It is exactly the same if Player 1 leaves the game.

Every time the word to guess is found, the winner becomes the drawer while the drawer becomes a guesser. If the word is not found in the time let for the game, the current drawer wins and stays the drawer. The game needs at least two players.

In the state where Player 1 is the drawer, two transitions can be triggered. If P1 wins, the system stays in the same state. Otherwise, the system has to redistribute the UIs.

In Figure 6.25, the winner is Player 2. As he won, he becomes the new drawer and Player 1 becomes a guesser. The merging of the two first states is represented by a dashed red loop.

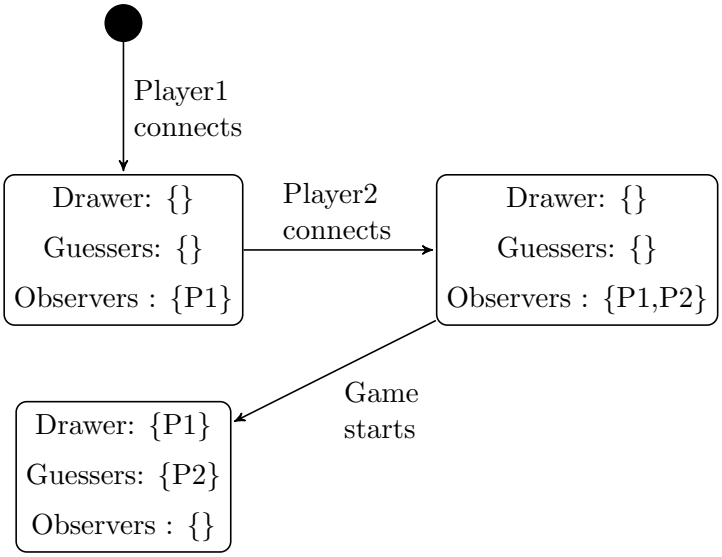


Figure 6.20: State diagram of the current system.

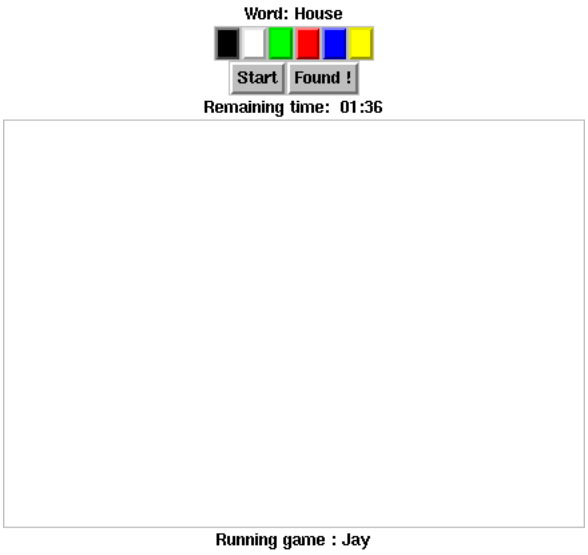


Figure 6.21: Player 1 becomes the drawer and stays observer.

```
{Undisplay enter_word#observer}
{Display
   td(name:drawing_tool
      label(name:word bg:white
            text:"Word: House")
   {Record.adjoin CD lr(name:colors
                        glue:n
                        relief:sunken
                        bg:white)}
   )
#p1}
{Display
   td(name:p1
      drawing_tool
      observer
      canvas(name:drawing area
             bg:white glue:nswe)
      status
   )
}
```

Figure 6.22: Pseudo-code for updating UI from Observer to Observer-Drawer.

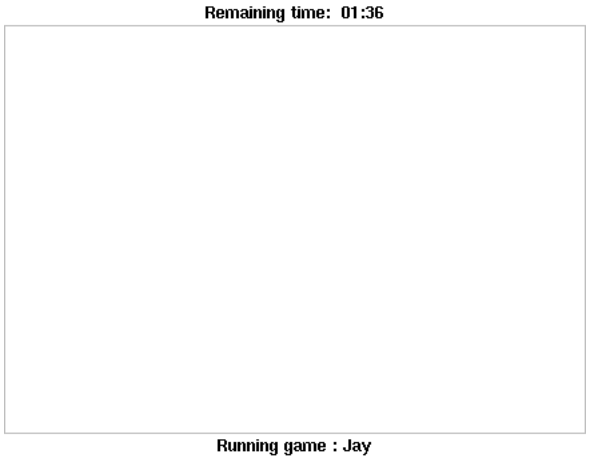


Figure 6.23: Player 2 becomes the guesser.

```
{Undisplay enter_word#p2}
{Undisplay start_found#p2}
{Display canvas(name:drawing_area
                bg:white glue:nswe)#p2}
```

Figure 6.24: Pseudo-code for updating UI for Player 2 becoming a guesser.

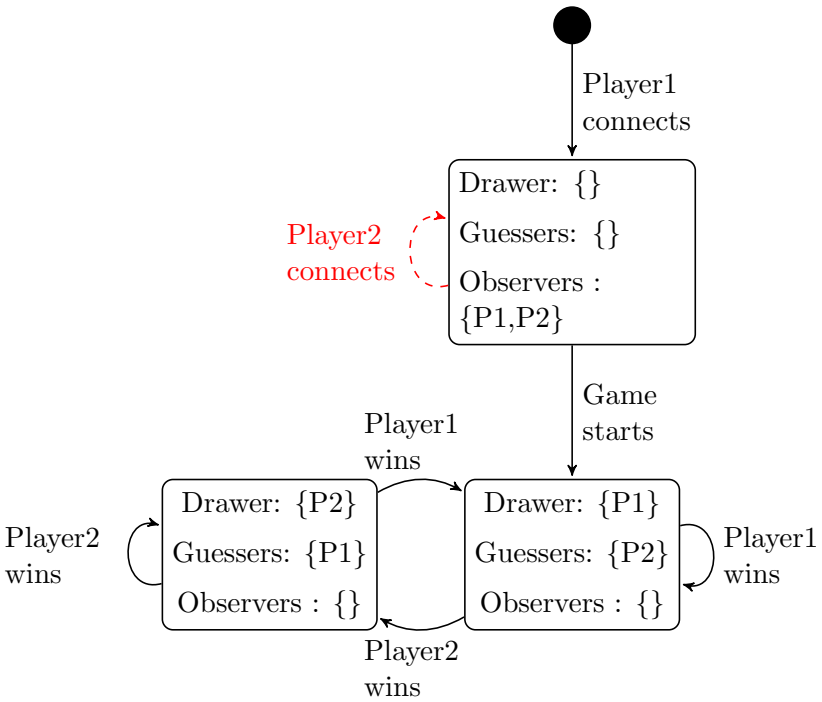


Figure 6.25: Simplified diagram of the whole system.

The last step remaining is the redistribution of the UI when role are exchanged. This means that we want to switch the upper part of Player 1's UI to Player 2's UI.

This can be done with the code in 6.26. Here we introduce the ability to choose the position of the widget. The *drawing_tool* widget will be placed first in widget $p2$.

```
{Move drawing_tool p2 pos:first)}
```

Figure 6.26: Pseudo-code for switching players' role.

Extended Mobictionary

Complexity	Nb. Users	Nb. Devices
Medium	4-n	2-n

Table 6.8: Information about the complexity of the extended Mobictionary

In this section, we introduce a dynamically extended version of the Mobictionary as a real case study. As the number of player increases, the simplified version may be extended to support teams. The minimum required for this variant is four players separated in two teams with two players.

A team is composed by at least two players. The team that is currently drawing needs a drawer and at least one guesser. The members of the other teams are observers.

The distribution graph for two teams is presented in Figure 6.27 based on the one in Figure 6.25.

If a guesser finds the word within the guessing time, he becomes the drawer and the team stays playing. If the guessing time is passed and no guesser found the word, another team is given the ability to play the same word. If the team finds the word, this team becomes the new team playing. If not, another team takes the turn until every team has played with the word. Every time a word is found, it increases the points of the team currently playing.

6.3.3 Evaluation

Thanks to the toolkit it has been possible to create the whole game on one computer and distribute it on several computers. The real power of the toolkit is the ability to assign each role a UI. This allows us to distribute the UI according to the role. The UI assigned to a role can also be merged into the application UI to offer only one UI to each user. A description of the demonstration is available in Appendix 2.

The UI of the drawer is more adapted for a device supporting touch than for a computer with a mouse and a keyboard. The distribution allows us to distribute the UI to smartphones and tablets. This will allow the drawer to use a pen or his fingers.

If we wanted to build this Mobictionary without the toolkit we would need to create one application for each operating system. Then we would need to connect this applications together and exchange information about the current state of the game. If one of the device crashes this would lead the game to an incoherent state or the application of the device that has crashed should be smart enough to save the state and recover it.

The toolkit allows the UI to be redistributed when something wrong happen. The UI that was distributed on the device that has crashed can easily be recovered and be displayed back too one of the other devices.

Another important difference is the change that would need to be made if we wanted to have two drawers instead of one. With the toolkit, it is very simple. We only need to copy the UI of the drawer and display it to the second drawer. This copy can either be with a synchronized behavior (if someone draws, the other drawer sees the current drawing) or with disjoint behavior (they can both draw but they do not see what the other is currently drawing). This would have been very hard without the tool. It would first mean to change the whole game rules, create an entire new UI for the second drawer and then create the code for the synchronization between them in case we want a synchronized behavior.

These small examples show that the toolkit allows us to easily create and manage the game. It is also possible to easily modify the rules of the game without creating a whole news application or without needing heavy changes.

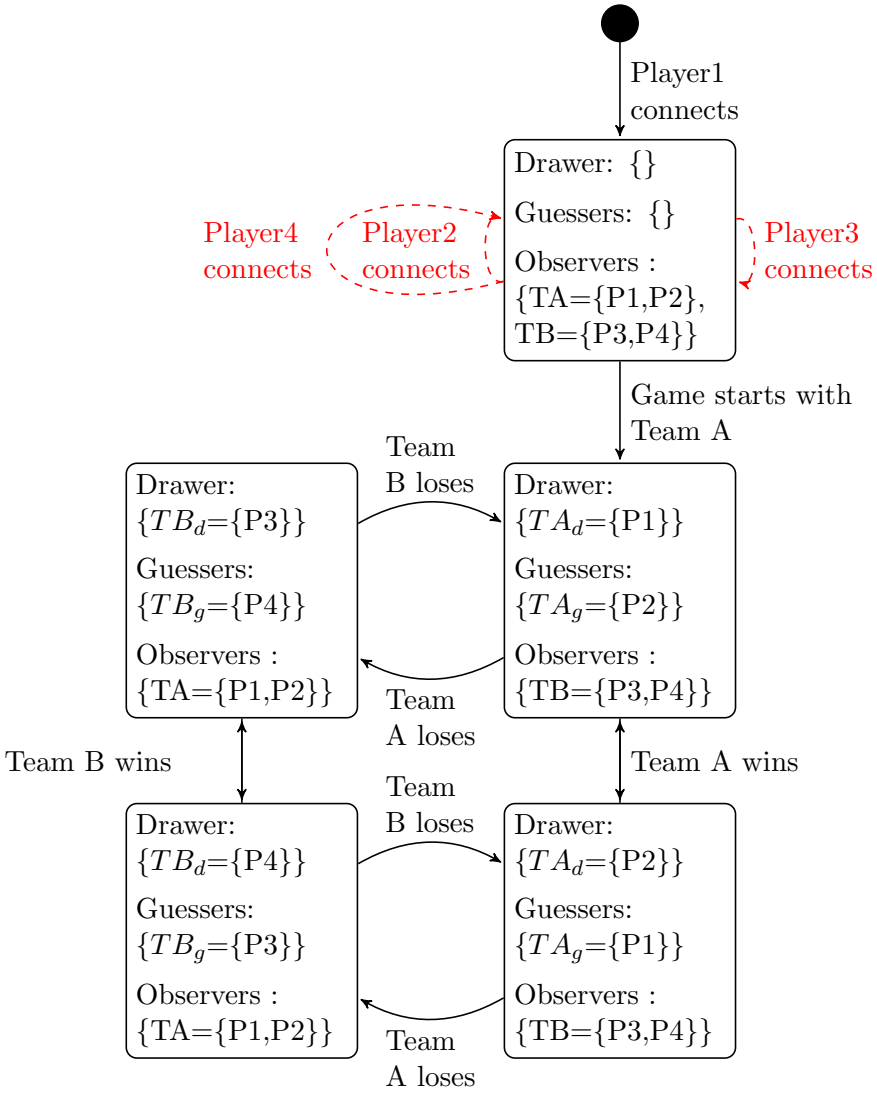


Figure 6.27: Complete diagram of the whole system.

6.4 CarReservation

CarReservation is a small demonstrator realized during the UsiXML project. This demonstration has been presented at the final review of the UsiXML project.

Complexity	Nb. Users	Nb. Devices
Low	1-n	1-n

Table 6.9: Information about the complexity of case study CarReservation

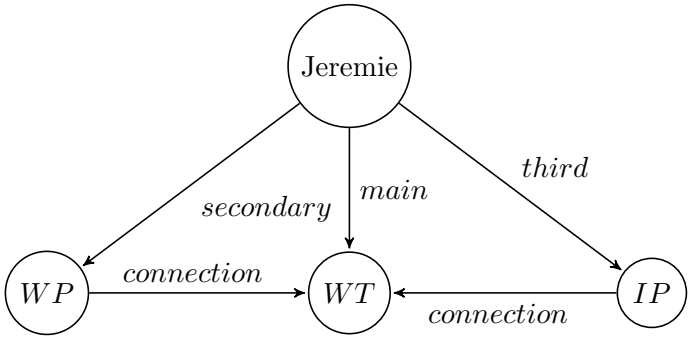


Figure 6.28: Distribution Graph of the CarReservation demonstration.

6.4.1 Specification

CarReservation is a small application that allows a user to fill a form. The GUI of the form is created from a model describing the content of the GUI. No data is contained in this model. Once the GUI has been created, other devices can connect to it to retrieve parts or whole of the GUI.

6.4.2 Implementation

This demonstration has been realized using TCP-IP sockets. Three devices are first connected together with a Windows tablet used as the master and two small devices used as its slaves. The master acts like a server to which the other devices connect to as clients.

Any component of the GUI that has been created on the tablet can be displayed on the two other devices. The resulting widget is sent to each device and then recomposed with native graphical elements to the Windows Phone device and the iOS device.

A screenshot of the form on the Windows tablet is depicted in Figure 6.29.

While the same on the Windows phone is depicted in Figure 6.30.

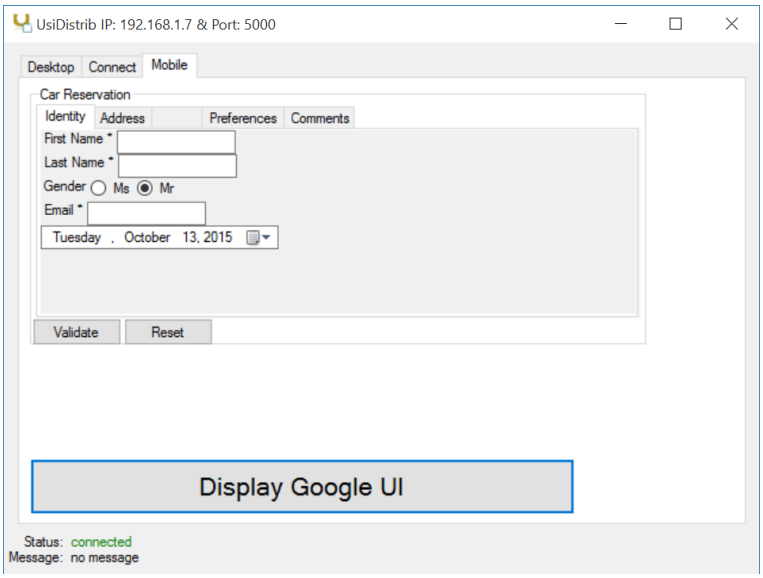


Figure 6.29: The GUI of the CarReservation form on the Windows tablet

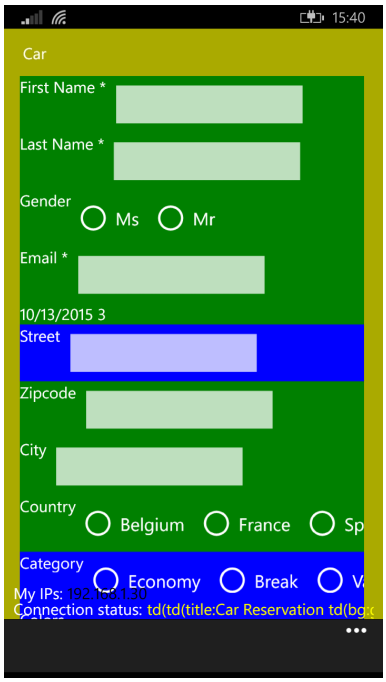


Figure 6.30: The GUI of the CarReservation form on the Windows phone

6.4.3 Evaluation

Thanks to the method we have been able to demonstrate that it was possible to use the same concepts with a different implementation.

In this demonstration there are three different operating systems used: Windows Phone, Windows and iOS. This adds to the demonstrations with Linux, Mac OS X and Android.

We have also been able to demonstrate that only a pre-defined GUI could be distributed but also that any GUI created at run-time, based on a file storing an instantation of a model.

With this DDGUI, we have also demonstrate the granularity of the distribution. It can happen at the window level but also at any graphical widget level.

6.5 Conclusion

With these case studies we have demonstrated how it was possible to use JayTk and Beernet to implement several scenarios. All the specifications have been reached using the toolkit and the distribution mechanisms offered by Beernet.

Thanks to these demonstration we have demonstrated that it was not mandatory to install an application on several devices in order to use it on them. An application started on a PC can easily distributed itself across several devices with the help of the toolkit.

This success has led to a FIRST SpinOff project funding which is currently preparing the creation of the SpinOff. The core business of the SpinOff is the ability to remotely applications that are started on a device with other devices no matter the size and the operating system of the devices.

Chapter 7

Validation of JayTk

While we were creating the new concepts and the toolkit associated with the method we also wanted to evaluate the work done during the thesis.

All the distribution primitives of the toolkit have not been developed. We mainly focused on the *display* and *undisplay* primitives which can be combined to produce some of the other distribution primitives. These primitives are then distributed to the right device and then executed locally.

As we made a survey as introduction of the thesis we wanted another method to evaluate and validate the thesis. Our evaluation was mainly based on the demonstrations that we have realized during the thesis. The next step is to validate our results with another survey. For the last evaluation process we will pass through all the requirements listed in 2.4 and see how the toolkit respond to them.

7.1 Potential benefits

The method and toolkit developed during the thesis allow developers to create complete applications that support distribution and adapt dynamically to the environment. One of the shortcomings of the solution proposed in the related work is that none of them is publicly available. We want our toolkit to be publicly available in the form of an API provided along with Mozart. Everyone can use it and give feedback to us. With JayTk we will soon offer the first toolkit that allows developers to create and manage DUIs across recent devices.

The model-based approach used in this thesis allows us to precisely define all the concepts that are important to distributed systems. Everyone can reuse these concepts with their own toolkit. The toolkit has also been created in a modular way so that it is easily possible to change the graphical user interface language, the distribution mechanisms used or the core between the network and the UI.

7.2 SpinOff: Usidistrib

Thanks to these demonstrations building DDGUI is now possible and can be integrated in commercial applications. This led us to the idea of creating a company that could offer this ability to extend current applications to support distribution and mobility across several devices and different operating systems.

We have started working on a FIRST Spin-Off project since May 2013 and we are close to create a company which use the results of the thesis. This has validated that our technology was indeed working but also interesting for other companies.

7.3 Validation of the Toolkit

In order to validate the method we allow users to test the system.

We have prepared these devices for the testers:

- a smartphone running Windows Phone 8
- a tablet running Windows 8
- a Macbook running Mac OS X
- a tablet running Linux
- a projector that was attached to the tablet or to the Macbook

These following scenarios are offered to the users:

- A small tutorial (5min max) shows how the system works
- Play with the system during 5 minutes to test and get used of it
- Distribute the UI for their own configuration
- Accomplish the same task with 3 different devices
- Accomplish the same task running 2 devices at the same time

7.4 Fulfillment of the requirements

One way to validate the contributions offered by the thesis consists of assessing its characteristics regarding some criteria. We have already listed the requirements that we wanted our work to take into account. We will use these requirements for the criteria. Thus we will refer to the list that have been defined earlier in the section 2.4. For all these requirements,

we have proposed some contributions. We will now see in how much each contribution contributes to the requirements.

First, let's remember what are the contributions of the thesis. We have already listed them in the section 1.3.3. These contributions can be classified in two categories: extension of concepts, and new concepts.

Here is the list of all these contributions according to the three pillars:

- Models and languages
  - MC_1: A user model allows us to describe some properties of who will interact with the system. i.e. the system should act in a different way with blind people and experts.
  - MC_2: A device model allows us to define important characteristics about the device and the platform. e.g. the resolution and the interaction mechanisms provided by the device are important data needed for good adaptation of the UI.
  - MC_3: A distribution graph helps us to see how the devices and users are distributed in the space. It is a quick view of the users, the devices and the environment they evolve in.
- Approach
  - AC_1: An EBNF grammar has been defined in order to use a common language for the distribution mechanisms.
  - AC_2: A catalog of distribution primitives/operations described all the primitives/operations that can be performed with the DUIs. This catalog is based on the knowledge from research that have been done in this topic. The goal of this catalog is to support any kind of distribution that have been proven useful in other studies and some other primitives/operations that we think they are interesting to support.
  - AC_3: Distribution scenarios allow us to manually or automatically trigger several distribution primitives in order to distribute, adapt or manage the UIs in the system.
  - AC_4: Some feedback have been defined in order to provide users with some visual notifications on what the system is currently doing or not able to do. i.e. the states : idle, in a distribution mechanism, successful and failed are differently notified to users.
- Software support
  - SC_1: Allow the creation of User Interfaces
  - SC_2: Allow the distribution of User Interfaces
  - SC_3: Multi-platform (from desktop to mobile operating systems)

- SC_4: Platform and device discovery for automatic visualisation of the system
- SC_5: Feedback mechanisms for multi-user and multi-platform distributions
- SC_6: Transparent distribution and communication mechanisms
- SC_7: Reliable distribution mechanisms with support for failures

The requirement MR_1 is the description of concepts of a Distributed System. The contribution MC_1, MC_2 and MC_3 allow us to satisfy this requirement. Indeed, the user and the device are described in a model while the distribution graph is the representation of the distributed system itself. The same contributions are part of the model-based approach created during the thesis. This satisfies the requirement MR_2. For the requirement AR_1, we provide distribution primitives and distribution scenarios which allows developers and users to modify the current state of distribution. Changes in the DS can also be modeled in the DG and the AG. The requirement AR_4 is the need for the distribution to be transparent. Thanks to choice of using Beernet as the distribution mechanisms this requirement is satisfied without the need of a contribution. We have added transparency in the use of the toolkit with the contributions AC_3 and SC_6. It also fulfill the requirement AR_5 which was to provide simple distribution mechanisms. The requirement SR_1 is the availability of the toolkit. It will be available for developers in 2016. Usidistrib is going to provide services for companies to make these features more broadly available. For the easy of use, we have described how easy it was for a non-distributed application to support distrubtion without being aware of it. This satisfies this requirement. JayTk provides different levels of granularity: action/service, widget, windows, application. This satisfies the SR_2 requirement.

Id.	Name	Fulfillment (/5)
MR_1	Description of the concepts of a DS	4
MR_2	Model-based approach for DDGUI	3
AR_1	Support of dynamic distribution	3
AR_2	Based the approach on models	3
AR_3	Basic distribution operations	4
AR_4	Transparent distribution	4
AR_5	Simple distribution mechanisms	5
SR_1	Availability of the toolkit	4
SR_2	Support of different levels of granularity	4

Table 7.1: Fulfilment of the requirements

The summary of how these requirements have been met can be found in the table 7.1. The score is a subjective evaluation according to our own

expectation. The scale goes from 0 to 5 where:

- 0 means that work has not started
- 1 means we have not got any good result
- 2 means that we fail fulfilling the requirement but we are close to it
- 3 means that we have a good base but we still have work to do
- 4 means that we fulfilled the requirement but there is still room for improvements or some validation is still needed
- 5 means that we fulfilled the requirement and finished working on it

7.5 Conclusion

In this chapter we have briefly discussed how we handle the validation and evaluation process. We mainly focused on demonstrations to validate and evaluate rather than surveying lots of people.

With the project that could lead to a company we think that this is an important step to evaluate and validate, our ideas, our method and our software support.

Through the translation of shortcomings into requirements which we attempted to fulfill with our contributions, we have been able to demonstrate the feasibility and the potential of using DDGUIs in applications.

Chapter 8

Conclusion

Throughout the thesis, the concepts have been introduced, exemplified and finally evaluated and validated. This leads us to summarize the work done during the thesis.

We have introduced some concepts in a model-based approach for managing DUIs and a toolkit that supports it. The goal is to provide a common base for researchers on DUIs. They now have the same set of primitives and can follow a common approach. It allows them to share the same possibilities regardless of the UI implementation.

Thanks to the use of distribution mechanisms from the domain of Distributed Programming, the existing applications and the new applications can both support distributed user interfaces. They both get a strong basis allowing them to react to delays, failures and other behaviors of a distributed system.

The conclusion is divided into two parts. First, we have all the contributions and then we consider the possibilities for future work.

8.1 Summary of the contributions

Like in the whole document, the contributions are divided into the three pillars.

Model

In order to deploy a complete approach for creating and managing DUIs, we needed to define the ontology and the language of the topic. We have created some models to represent the various agents that are used with the system. These models are the *user model* described in section 3.2.3 and the *device model* from section 3.2.4.

Thanks to these definitions, we have introduced the concept of a distributed system in section 1.2.1, a distribution graph in section 3.3 and an

application graph in section 3.6. The distributed system includes the users and the devices related to it, it is also constituted of an environment which contains all the users and devices. The distribution graph is a model of the interactions between the users and the devices through *actions*. The application graph is a more specific model of an application running on several devices. It allows users to see how the user interface is distributed across these devices.

Basing ourselves on these contributions, we have established a language to define the way each action can be defined and triggered. This is a very important contribution to simplify the reusability and the availability of the method.

Approach

This dissertation introduced the notion of distribution graph in section 3.3 as a way of modeling and developing Distributed User Interfaces. The graph is a state diagram where states represent the current distribution of the system. It also describes the implementation of two case studies, a simple and an extended Pictionary. This new methodology needs to be validated, i.e. with these case studies applied as real game. A more complex kind of game could be developed with the idea of the *snakes and ladders* game. Each square would be a different game and an additional UI would enable users to change the game inside it at running time.

Basing ourselves on the language that we have defined we have listed and detailed with algorithms the main actions that we wanted our method to support in section 3.7. These actions can also be grouped into distribution scenarios. The distribution scenario is a concept we have introduced to allow the execution of several actions in a predefined order. This is very interesting to support dynamic distribution or to repeat a sequence of actions.

Thanks to the use of complex distribution mechanisms, our method is able to support the complex aspect of a distributed system as detailed in section 3.7.5. It adds the support for crashes, failures, delays, and users and devices leaving at run-time.

Software support

A toolkit based on this approach has also been created. It allows developers to see the power of this catalog. This toolkit is multi-platform and will be released publicly. You can read more details about it in section 5.

The toolkit makes it possible to create applications that support the distribution of their user interfaces across several devices ranging from a mobile phone to a desktop computer, a projector or a kiosk, and it allows the creation of a real distributed system.

Thanks to the distribution mechanisms used and detailed in section 4.3 the toolkit provides device discovery and supports feedback, multi-user collaboration, failure detection and other important aspects of distributed systems.

8.2 Progress and Shortcomings

The toolkit created during the thesis does not fully implement all the concepts and all the primitives we have defined. As discussed before, the goal was to test the toolkit on sufficient case studies to prove the concepts introduced in the thesis. We are aware that conducting the survey without being able to use the toolkit is not sufficient to validate the toolkit itself. In the near future we want to create several applications with the help of the toolkit and carry out a larger experiment.

8.3 Future Work

One of the first steps after the thesis will be to test the toolkit with a lot of users. For this reason, the toolkit will be released publicly.

A larger case study has already been decided for the future of the toolkit. The toolkit will be used by the CyborgOS. It is an entire ecosystem that runs as an executable on Windows, Linux and Mac OS X. The application allows the creation of the user interface from an editor which triggers the actions defined in the toolkit. All the devices get the same UI and thanks to the toolkit the UI can be created and modified at real-time. If a device becomes unavailable the CyborgOS will still work and nothing is lost.

The thesis provides some models, an approach and a software support for designers and developers to manage distributed systems and allow the distribution of the user interfaces at run-time. There is no added support for end-user programming. Developers can allow the end-users to manage the distribution but it is up to them to offer this.

In the thesis, we have selected the version of each operating system that we wanted to support. These operating systems have or will have newer versions in the future. We will update the toolkit to stay compliant and use these systems as efficiently as possible. This will be part of the SpinOff UsiDistrib that will be created soon after the thesis, whose goal is to commercialize and update the toolkit created in the thesis.

The concept of Distribution Graph has been defined precisely in the thesis. We think that it would be interesting to see the Distribution Graph in real-time. This means that we could provide an implementation of a graphical representation of the DG. The actions that have been defined would then be translated into some representation in the DG. When a device would appear in the DS, a node representing thise device will be created in

the DG. When two devices are connected together, we would see the edge created between both nodes.

In Chapter 3, the concept of Application Graph is just introduced without details. It would be interesting to develop this concept more deeply and test it with some other applications.

8.4 Final Word

The work done during the thesis has opened the doors to a completely new kind of applications which can work seamlessly inside a distributed system constituted of several computers and mobile devices. This embraces the mobile vision which is the future already knocking at our doors.

With the power of mobility increasing every year, and the number of devices launched every year and multiplying, there is a high need of supporting such technology so that people are not obliged to duplicate their work each time they need to use another operating system.

Chapter 9

Appendices

Appendix: Demonstration 1

I. Name:

DeTransDraw – DeTransDrawid

II. Features:

- Distributed application
- Collaborative application
- Redundancy (CARE properties)
- Eager-Locking mechanism
- Transactions

III. Description:

DeTransDraw and **DeTransDrawid** together is a distributed transaction drawing tool that can runs on PCs (**DeTransDraw**, ***DTD***) and android smartphones (**DeTransDrawid**, ***DTDid***). This tool allows users to collaborate on simple drawings with 2 shapes (rectangles and ovals) and several colors. Each shape is created as an object that users can edit (move and resize) or delete later. This tool allows to draw and edit shapes without delays.
There is no possible conflict when drawing a shape. However there are conflicts when some users are trying to edit or delete the same object. To handle this, the tool supports transactions and a smart selection process. In order to edit an object, a user must first select the object which will ask to lock it. During this temporary state, the user can edit the object without any delay but the action is not guaranteed to be accepted. As soon as the lock is granted, the action is guaranteed to succeed and as soon as the action is over, the result will be synced across all the devices.

IV. Devices:

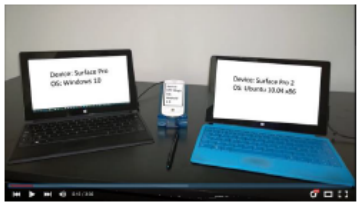
	Desktop PC	Tablet	Smartphone
Operating System	Linux	Windows	Android
OS Version	Ubuntu 10.04 x86	Windows 10	Android 1.5
Environment	Mozart 1.4	Mozart 1.4	Mozart 1.4
Communication	P2PS	P2PS	P2PS
Sources	DTD-Linux.zip	DTD-Windows.zip	DTDid.zip

V. Presented:

At the Internet of Service Collaboration Meeting in the Crowne Plaza Brussels City Centre on June 10-11, 2009.

At the final review of SELFMAN European FP6 Project under convention n°034080 which took place at Université catholique de Louvain (UCL, Belgium) on November 29, 2009.

VI. Videos:

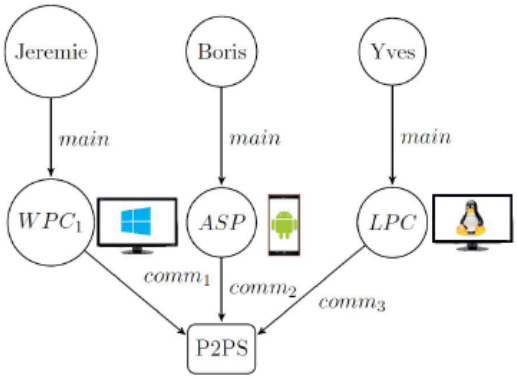


https://youtu.be/cts5ReiffUc

https://youtu.be/kAXmCFzshMs

VII. Distribution Graph:

There are 3 users: Jeremie, Boris and Yves. While Jeremie uses a Windows PC, Boris uses an Android smartphone, and Yves uses a Linux PC. All the devices are connected to P2PS, a peer-to-peer network. Here is the Distribution Graph corresponding to the demonstration.



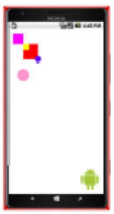
VIII. Screenshots

Here are screenshots of DeTransDraw running on a Desktop PC and DeTransDrawid running on a Smartphone.

1. DeTransDraw

DeTransDraw runs in a window no matter if it runs on a Linux or Windows PC.

2. DeTransDrawid



DeTransDrawid runs as a full screen application in the Android application.

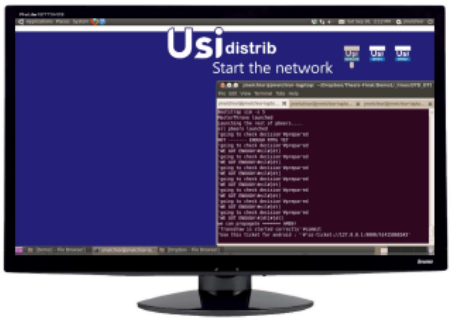
IX. Scenarios

The application contains buttons to create rectangles and ovals on a drawing area. The drawing area is shared with all the other users. The UI is the same for each device. Everyone sees the same drawings and everyone can create and modify the drawings. Any device can join or leave the network without affecting the application. When a device joins the network, it automatically gets the up-to-date drawings.

To demonstrate the possible scenarios we have decided to separate the multi-platform aspect from the locking mechanism.

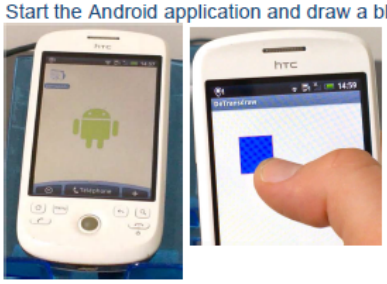
1. Multi-platform scenario

Start the network



Yves starts the P2P network to allow other devices to connect to.

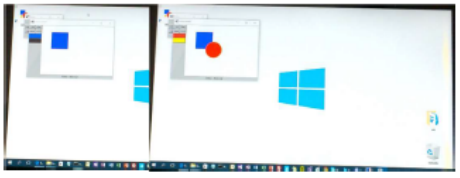
Start the Android application and draw a blue rectangle



Boriss starts the application on its Android smartphone. Connect it to the network.

Then he draws a blue rectangle with his finger.

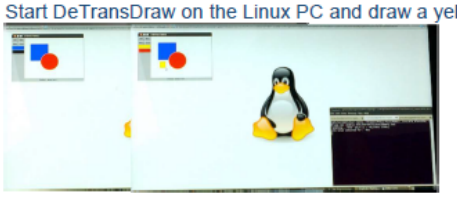
Start DeTransDraw on the Windows tablet and draw a red oval



Jérémie starts the application on its Windows tablet. Connect it to the network.

Then he draws a red oval with his stylus.

Start DeTransDraw on the Linux PC and draw a yellow rectangle



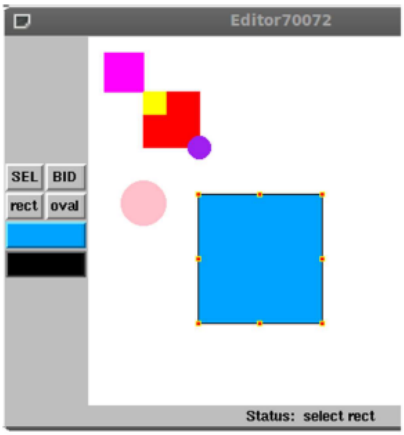
Yves finally starts the application on its Linux PC. Connect it to the network.

Then he draws a yellow rectangle with his mouse.

2. Locking mechanism scenario

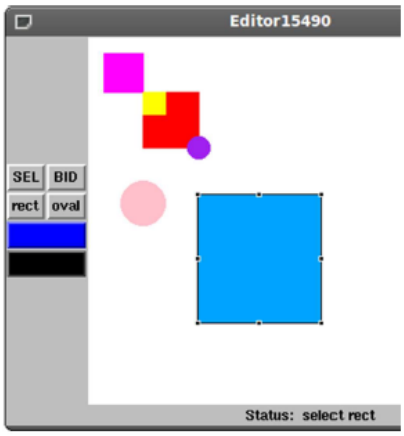
In case a user wants to edit an object, he can select the object to lock it. This lock will prevent 2 users to edit the same object at the same time. A graphical feedback is given to the user to know if the application is waiting for the lock, or if the lock is granted. Here are 2 screenshots with the 2 states (waiting for the lock, lock granted).

If the lock is refused then the object will not be selected anymore and the object will go back to its initial position.



The blue rectangle is selected. Handles around the rectangle show the selection. It is red because the application is currently trying to lock the object.

If the lock is refused, then the object will go back to its initial position and the edition is canceled. Then none of the devices will see any change.



As soon as the lock is acquired the handles become black. This means that the edition will succeed.

All the devices will be updated with the new position of the rectangle.

Appendix: Demo 2

I. Name:

Mobictionary

II. Features:

- Distributed application
- Interlaced User Interfaces (UI)
- Dynamic Distributed Graphical User Interface (DDGUI)
- Multi-player game
- Selective Distribution
- Complementarity, Redundancy (CARE properties)

III. Description:

Mobictionary is a multi-player game with screens that can be displayed on several devices. It is an implementation of the Pictionary. There are several players drawing or guessing words depending on their roles: draw player, guess player, and observer. Each player uses its own computer and gets the interface that comes with its role. All players are located in the same room.

IV. Devices:

Desktop PC

Operating System	Windows	Linux	Mac OS
OS Version	Windows 10	Ubuntu 14.04 x64	Mac OS X 10.8
Environment	Mozart 2.0	Mozart 2.0	Mozart 2.0
Communication	Beernet	Beernet	Beernet
Sources	Win-Mobictionary.zip	Lin-Mobictionary.zip	Mac-Mobictionary.zip

Tablet

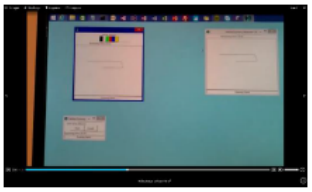
Operating System	Windows
OS Version	Windows 10
Environment	Mozart 2.0
Communication	Beernet
Sources	Win-Mobictionary.zip

V. Presented:

This demo has been presented at ICT Forum 2010 (Brussels, Belgium) on October 27-29, 2010.

VI. Videos:

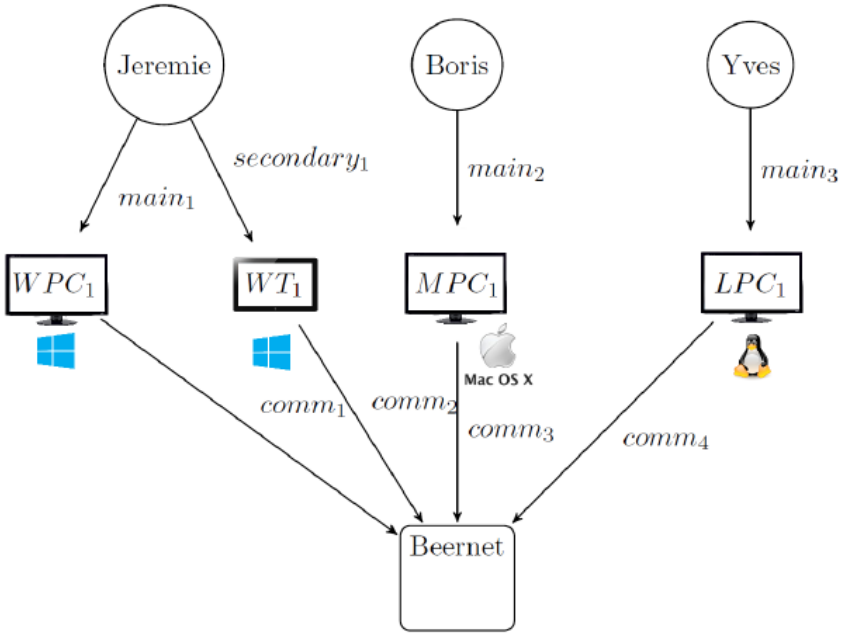
Local-Mobictionary.mp4



Distributed-Mobictionary.mp4

VII. Distribution Graph:

There are 3 users: Jeremie, Boris and Yves. Jérémie uses a Windows PC and a Windows tablet, Boris uses a Mac OS X PC and Yves uses a Linux PC. All the devices are connected to Beernet, a peer-to-peer network.
Here is the Distribution Graph corresponding to the demonstration.



VIII. Screenshots:

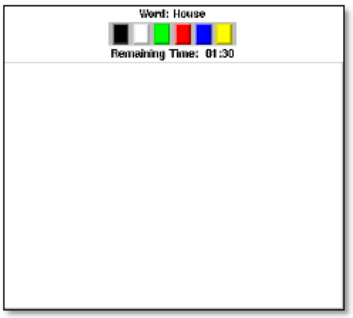
Here are the screenshots of all the possible UI. Each UI is attached to a player role.

Observer UI

The observer UI displays:

- The word to find
- The buttons to start and stop the game
- The timer that shows the remaining time

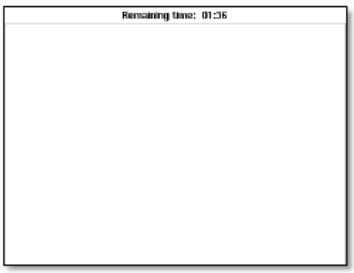
Draw player UI



The draw player UI displays:

- The word to find
- 5 colors and an eraser (white).
- The timer
- The drawing area

Guess player UI



The guess player UI shows:

- The timer
- The drawing

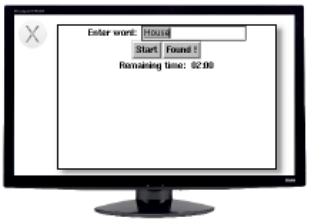
There is no possibility to draw on the screen.

IX. Scenario

The game is created.

Any screen can be displayed on any device. In this setup we decided that the Mac PC will be used to control the game, the draw player will use a tablet to draw and a Linux PC at the same time to select the color of the pen.
Each guess player will use either a PC or a tablet.

Observer screen



The Mac PC displays the observer UI. It allows to choose the word. This player starts the game.

This screen can only be displayed on one device. Either a Mac, Linux or Windows PC, or a Windows tablet

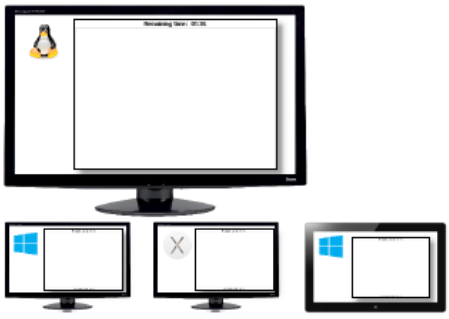
Draw player screens

The draw player UI is displayed on the Windows tablet.

The draw player cannot start drawing unless the game is started by the observer.

The draw player can choose the color of its pen either from his tablet or from the Windows PC.

Guess player screens



The PCs and Windows tablets display the guess player UI so that everyone can see the drawing.

Any PC can be connected to an external monitor or to a video projector to let more people see the drawing.

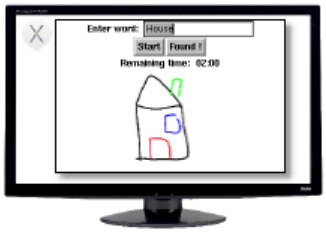
The draw player is drawing.

Draw player screens

The draw player draws something on the tablet.

The Windows PC displays the different colors.

Observer screen

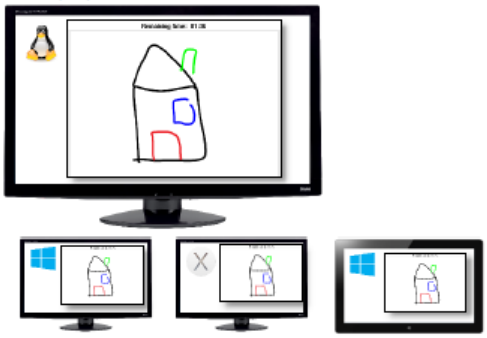


The Mac PC still displays the observer UI.

It also instantly displays the drawing.

The observer can stop the game as soon as a guess player finds the word, or if the time is over

Guess player screens



The drawing is instantly displayed on all the guess player screens.

The guess players tell their answers to the observer.

Glossary

computing device

A device that can be programmed to accomplish automatic computation tasks. It includes a hardware architecture and a software framework (including application frameworks), where the combination allows software applications to run. Typical computing devices include a computer architecture, at least one operating system, programming languages and a related user interface (run-time system libraries or graphical user interface).e.g., desktop computers, laptops, tablets, mobile phones (features phones and smartphones), video game consoles, televisions.

distributed user interface [DUI]

Any application UI whose components can be distributed across different computing devices that can be used by different users.

distributed user interface [DDUI]

Any DUI whose components can be dynamically distributed.

distributed graphical user interface [DGUI]

Any application UI whose only graphical components can be distributed across different computing devices that can be used by different users.

dynamic distributed graphical user interface [DDGUI]

Any application UI whose only graphical components can be dynamically distributed across different computing devices that can be used by different users.

environment

It covers objects, persons, and events that are peripheral to the current task but that may have an impact on the distributed system and/or the user's behavior[THE99].

interaction

An action that occurs between a user and a computing device. e.g., clicking on a button, writing down your user name or password, touching an icon to start an application on your tablet or smartphone.

user

It can either be a human, an animal, a robot or any mechanical entity that is able to use or interact with the computing devices.

user interface [UI]

An interface provided by an application to allow interactions with the users. It can either let the users control or observe the application. The most common UI is the Graphical User Interface [GUI] which is often represented in the windows, icons, menus, pointer (WIMP) style of interaction.

Bibliography

[AKS08] Aksenov, P., Luyten, K., and Coninx, K. Reasoning Over Spatial Relations for Context-Aware Distributed User Interfaces, In Fifth International Workshop Modeling and Reasoning in Context (MRC 2008), Delft, The Netherlands, 9-12 June 2008.

[AKS09] Aksenov, P., Vanderhulst, G., Luyten, K., and Coninx, K. Ambient Compass: One Approach to Model Spatial Relations, 2009.

[Air Display] Avatron Software, Air Display turn your iPad or Android tablet into a computer monitor https://avatron.com/applications/air-display/,
Last visited on July 11, 2015.

[AirPlay] AirPlay: Play content from your iPhone, iPad, iPod touch, or Mac on your HDTV, https://www.apple.com/airplay/, Last visited on January 11, 2015.

[ALI01] Ali, M.F., Pérez-Quinones, M.A., and Abrams, M. A Multi-Step Process for Generating Multi-Platform User Interfaces using UIML, CoRRcs.HC/0111025, 2001.

[ALI02] Ali, M.F., Pérez-Quinones, M.A., Abrams, M. and Shell, E. Building Multi-Platform User Interfaces with UIML, In Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces (CADUI 2002), Kluwer Academics Publisher, Dordrecht, 2002, pp. 255-266, Valencienne, France, 15-17 May 2002.

[AN02] An, A. Learning Classification Rules from Data, Computers and Mathematics with Applications 45 (2003), pp. 737-748, 2003.

[AND00] Foundations of Multithreaded, Parallel, and Distributed Programming, Andrews, G. R., AddisonWesley, 2000, ISBN 0-201-35752-6.

[ASH04] Ashdown, M., and Sato, Y. Attentive Interfaces for Multiple Monitors, 2004.

[ASL10] Aslan, I., Menon, D., Brauer, R., Albert, K., and Maugg, C. E-Composer: Enabling the Composition of Mobile Assistants, In Proceedings of MDDAUI2010 & CHI2010, pp. 37-40, 2010.

[ATT98] Attiya, H., Welch, J. Distributed Computing: Fundamentals, Simulations and Advanced Topics, McGraw-Hill Publishing Company, UK, ISBN 0-07-709352 6

[AVR89] Avrahami, G., Brooks, K.P., and Brown, M.H. A Two-View Approach to Constructing User Interfaces, In Proceedings of SIGGRAPH'89 (Boston, USA), ACM Computer Graphics, vol. 23, no. 3, pp. 137-146, July 31 - August 4, 1989.

[AYA00] Ayatsuka, Y., Matsushita, N., and Rekimoto, J. HyperPalette: a Hybrid Computing Environment for Small Computing devices. In Proceedings of CHI'00 (The Hague), ACM Press, New York, pp. 133-134, April 1-6, 2000.

[BAI04] Bailey, B.P. A Distributed Display System for Interactive Sketching, 2004.

[BAL04] Balme, L., Demeure, A., Barralon, N., Coutaz, J., and Calvary, G. Cameleon-RT: A software architecture reference model for distributed, migratable, and plastic user interfaces, In Proceedings of 2nd European Symposium on Ambient Intelligence EUSAI'2004, Springer, Heidelberg, LNCS, vol. 3295, pp. 291-302, 2004.

[BAND04] Bandelloni, R., and Paternò, F. Migratory user interfaces able to adapt to various interaction platforms, International Journal of Human-Computer Studies 60, 5-6, pp. 621-639, 2004.

[BANG05] Bang, M., Larsson, A., Berglund, E., and Eriksson, H. Distributed user interfaces for clinical ubiquitous computing applications, International Journal of Medical Informatics, Elsevier, 2005.

[BARB10] Barboni, E., Ladry, J.-F., Navarre, D., Palaque, P., and Winckler, M. Beyond Modelling: An Integrated Environment Supporting Co-Execution of Tasks and Systems Models, In Proceedings of EICS'10, Berlin, Germany, June 19-23, 2010.

[BARD11] Bardram, J., Doryab, A., Gueddana, S. Activity-Based Computing Metaphors and Technologies for Distributed User Interfaces, In ... ACM Press, New York, pp. 67-74, 2011. DOI 10.1007/978-1-4471-2271-5_8

[BARR04] Barralon, N. Meta UI: vers un Desktop++,
In Proceedings of Rencontres Jeunes Chercheurs en Interaction Homme Machine RJHCI'2004, 2004.

[BARR07] Barralon, N., and Coutaz, J. Coupling Interaction Resources in Ambient Spaces: There Is More Than Meets the Eye!. In Engineering interactive Systems: EIS 2007 Joint Working Conferences, EHCI 2007, DSV-IS 2007, HCSE 2007, Salamanca, Spain, Selected Papers, J. Gulliksen, M.B. Harning, P. Palanque, G.C. Veer, and J. Wesson, Eds. Lecture Notes in Computer Science, vol.4940. Springer-Verlag, Berlin, Heidelberg, pp. 537-554, March 22-24, 2007.

[BART11] Barth, T., Fielenbach, T., Bourimi, M., Kesdogan, D., Villanueva, P.G. Supporting Distributed Decision Making Using Secure Distributed User Interfaces, In ... ACM Press, New York, pp. 177-184, 2011. DOI 10.1007/978-1-4471-2271-5_20

[BEAL07] Beale, R., Edmondson, W. Multiple Carets, Multiple Screens and Multi-Tasking: New Behaviours with Multiple Computers, In Proceedings of HCI'2007 (Lancaster, England), British Computer Society, Swinson, pp. 55-64, September 3-7, 2007.

[BEAU00] Beaudouin-Lafon, M., and Lassen, H.M. The architecture and implementation of CPN2000, a post-WIMP graphical application, In Proceedings of the 13th ACM Symposium on User interface Software and Technology UIST'2000, ACM Press, New York, pp. 181-190, 2000.

[BEAU01] Beaudouin-Lafon M., Novel Interface Software and Technology for Overlapping Windows, In Proceedings of 14th ACM Symposium on User Interface Software and Technology UIST'2001, ACM Press, New York, pp. 153-154, 2001.

[Beernet] Programming Languages and Distributed Computing Research Group, UCLouvain Beernet: pbeer-to-pbeer network, http://beernet.info.ucl.ac.be

[BEL05] Bell, B.A. View Management for Distributed User Interfaces, Doctoral Thesis. UMI Order Number: AAI3174746, Columbia University, 2005.

[BEN07] Benoît, A., Bonnaud, L., Caplier, A., Jourde, F., Nigay, L., Serrano, M., Damousis, I., Tzovaras, D., and Lawson, L. Multimodal Signal Processing and Interaction for a Driving Simulator: Component-based Architecture., Journal of Multi. UIs, vol. 1, no. 1, pp. 49-58, 2007.

[BER02] Berglund, E., Bang, M. Requirements for Distributed User Interfaces in Ubiquitous Computing Networks, In Proceedings of ACM Conferences on Mobile and Ubiquitous MultiMedia. MUM'02 (Oulo, Finland), ACM Press, New York, NY, December 11-13, 2002.

[BHA95] Bharat, K.A., and Cardelli, L. Migratory applications, In Proceedings of the 8th Annual ACM Symposium on User interface and Software Technology (Pittsburgh, ennsylvania, USA). UIST'95, ACM Press, New York, NY, pp. 132-142, November 15-17, 1995.

[BIC99] Bickmore, T., Girgensohn, A., and Sullivan, J.W. Web Page Filtering and Re-Authoring for Mobile Users, Computer Journal, Oxford University Press for British Computing ociety, vol. 42, No. 6, pp. 534-546, 1999.

[BIE04] Biehl, J.T., and Bailey, B.P. ARIS: An Inferface for Application Relocation in an Interactive Space, In Proceedings of Graphics interface 2004 (London, Ontario, Canada), GI'2004, ACM International Conference Proceeding Series, vol. 62. Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, pp. 107-116, May 17-19,2004.

[BIE05] Biehl, J.T., and Bailey, B.P. Interfaces for Managing Information in Distributed Display Environments, 2005.

[BIE05B] Biehl, J.T., and Bailey, B.P. A Toolset for Creating Iconic Interfaces for Interactive Workspaces, 2005.

[BIE06] Biehl, J.T., and Bailey, B.P. Improving Scalability and Awareness in Iconic Interfaces for Multiple-Device Environments, In Proceedings of AVI'06, Venezia, Italy, May 23-26, 2006.

[BIE06B] Biehl, J.T., and Bailey, B.P. Improving Interfaces for Managing Applications in Multiple-Device Environments, In Proceedings of AVI'06, Venezia, Italy, May 23-26, 2006.

[BIE06C] Biehl, J.T., and Bailey, B.P. Interfaces for Managing Applications and Input in Multi-Device Environments, In ..., 2006.

[BIE08] Biehl, J.T., Baker, W.T., Bailey, B.P., Tan, D.S., Inkpen, K.M., and Czerwinski, M. IMPROMPTU: A New Interaction Framework for Supporting Collaboration in Multiple Display Environments and its Field Evaluation for Co-located Software Development, In Proceedings of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems (Florence, Italy). CHI'08, ACM Press, New York, NY, pp. 939-948, April 05-10, 2008.

[BISH06] Bishop, J. Multi-platform User Interface Construction - a Challenge for Software Engineering-in-the-Small, In Proceedings of the 28th international Conference on Software Engineering (Shangai, China).ICSE'06, ACM, New York, NY, pp. 751-760, May 20-28, 2006.

[BISW10] Biswas, P., and Robinson, P., A Brief Survey on User Modelling in HCI, 2010.

[BLA02] Black, J.A., Hong, J.I., Newman, M.W., Edwards, W.K., Izadi, S., Sedivy, J., Smith, T.F. Speakeasy: A Platform for Interactive Public Displays, In Community and Situated Displays (Workshop at CSCW 2002), New Orleans, L.A., USA. November 16-20, 2002.

[BLU10] Blumendorf, M., Roscher, D., and Albayrak, S. Dynamic User Interface Distribution for Flexible Multimodal Interaction, In Proceedings of the International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction (ICMI-MLMI'10), ACM Press, New York, Article 20, 8 pages. November 8-10, 2010.

[BLU11] Blumendorf, M., Roscher, D., and Albayrak, S. Distributed User Interfaces for Smart Environments: Characteristics and Challenges, In Proceedings of CHI2011 workshop on DUI, ACM Press, New York, pp. 25-28, May 7-12, 2011.

[BRD07] Brdiczka, O., Crowley, J.L., and Reignier, P. Learning Situation Models for Providing Context-Aware Services, Lecture Notes in Computer Science, pp. 23-32, 2007.

[BRE10] Breiner, K., Gauckler, V., Seissler, M., and Meixner, G. Evaluation of User Interface Adaptation Strategies in the Process of Model-driven User Interface Development, In Proceedings of MDDAUI2010 & CHI2010, pp. 17-20, 2010.

[CAF10] Caffiau, S., and Girard, P. A Global Process for Using Model-driven Approaches in User Interface Design, In Proceedings of MDDAUI2010 & CHI2010, pp. 33-36, 2010.

[CAG05] Cagle, K. Distributed User Interfaces: Toward SVG 1.2, In the book Visualizing Information Using SVG and X3D, pp. 119-152 (Chapter 6), 2005.

[CAL97] Calvary, G., Coutaz, J., and Nigay, L. From Single-User Architecture Design to PAC*: a Generic Software Architecture Model for CSCW, In Proceedings of CHI'97, 1997.

[CAL01] Calvary, G., Coutaz, J., and Thevenin, D. A unifying Reference Framework for the Development of Plastic User Interfaces, In Proceedings of EHCI'01, pp. 137-192, 2001.

[CAL04] Calvary, G., Coutaz, J., Dâassi, O., Balme, L., and Demeure, A. Towards a New Generation of Widgets for Supporting Software Plasticity: The "Comet", Engineering Human Computer Interaction and In-

teractive Systems, Joint Working Conferences EHCI-DSVIS 2004, Hamburg, Germany, Revised Selected Papers, Springer, pp. 306-324, July 11-13, 2004.

[CAR06] Cardinaels, M., Vanderhulst, G., Wijnants, M., Raymaekers, C., Luyten, K., Coninx, K. Seamless Interaction Between Multiple Devices and Meeting Rooms, In Proceedings of CHI'06, 2006.

[CHA11] Chang, T.-H., and Li, Y. Deep shot: a framework for migrating tasks across devices using mobile phone cameras. In Proceedings of the 2011 annual conference on Human factors in computing systems (CHI '11). ACM, New York, NY, USA, pp. 2163-2172.
DOI=10.1145/1978942.1979257
http://doi.acm.org/10.1145/1978942.1979257

[CHE11] Chen, N., Guimbretiere, F., and Sellen, A. Distributed User Interface for a Multi-Tablet Active Reading System, In Proceedings of CHI2011 workshop on DUI, ACM Press, New York, pp. 73-76, May 7-12, 2011.

[CHU04] Chung, G., and Dewan, P. Towards Dynamic Collaboration Architectures, In Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work (Chicago, Illinois, USA). CSCW'04, ACM, New York, NY, pp. 1-10, November 06-10, 2004.

[cloud computing] http://aws.amazon.com/what-is-cloud-computing/

[COL07] Collet, R. The Limits of Network Transparency in a Distributed Programming Language, Thesis, 2007.

[Compiz] Compiz: an OpenGL compositing manager to texture object,
http://www.compiz.org,
Last visited on July 10, 2015.

[Continuity] iPhone, iPad, and Mac. Connected like never before,
https://www.apple.com/ios/whats-new/continuity/,
Last visited on January 11, 2015.

[COUL05] Coulouris, G. F., Dollimore, J., Kindberg, T. Distributed Systems: Concepts and Design, Addison Wesley, 2005.

[COU87] Coutaz, J. PAC: an Implementation Model for Dialog Design, In Proceedings of Interact'87 (Stuttgart, Germany), H-J. Bullinger, B. Shackel ed., North Holland, pp. 431-436, September 1987.

[COU03] Coutaz, J., Barralon, N., Lachenal, C., Rey, G. Final Reference Framework for Interaction Surfaces, GLOSS Project. 2003.

[COU03B] Coutaz, J., Lachenal, C., and Dupuy-Chessa, S. Ontology for Multi-surface Interaction, In Proceedings of IFIP INTERACT'03: Human-Computer Interaction 2003, pp. 447, 2003.

[COU05] Coutaz, J., Borkowski, S., and Barralon, N. Coupling interaction resources: an analytical model, In Proceedings of the 2005 Joint Conference on Smart Objects and Ambient intelligence: innovative Context-Aware Services: Usages and Technologies (Grenoble, France). sOc-EUSAI'05, vol. 121, ACM, New York, NY, pp. 183-188, October 12-14, 2005.

[COU05B] Coutaz, J., Crowley, J.L., Dobson, S., and Garlan, D. Context is key, Commun, ACM 48, 3, pp. 49-53, Mars, 2005.

[COU06] Coutaz, J. Meta-User Interfaces for Ambient Spaces, In Proc. of the 5th International Workshop on Task Models and Diagrams for Users Interface Design, TAMODIA'2006 (Hasselt, Belgium), LNCS, Vol. 4385. Springer, Berlin, pp. 1-15, October 23-24, 2006.

[COU07] Coutaz, Balme, L., Alvaro, X., Calvary, G., Demeure, A., and Sottet, J-S. An MDE-SOA Approach to Support Plastic User Interfaces in Ambient Spaces, In Proceedings of the Universal Access in Human-Computer Interaction. Ambient Interaction, 4th International Conference on Universal Access in Human-Computer Interaction, UAHCI 2007 Held as Part of HCI International 2007 Beijing, China, Springer, pp. 63-72, July 22-27, 2007.

[COU07B] Coutaz, J., Meta-User Interfaces for Ambient Spaces: Can Model-Driven-Engineering Help?, 2007.

[COU10] Coutaz, J. User Interface Plasticity: Model Driven Engineering to the Limit! In Proceedings of 2nd International Conference on Engineering Interactive Computing Systems, EICS'2010, pp. 1-8, 2010.

[DAD11] Dadlani, P., Emparanza J.P., and Markopoulos, P., Exploring Distributed User Interfaces in Ambient Intelligent Environments, In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-Computer Interaction Series, Springer-Verlag, London, pp. 161-168, 2011. DOI 10.1007/978-1-4471-2271-5_18

[DAD11B] Dadlani, P., Emparanza, J.P., and Markopoulos, P. Distributed User Interfaces in Ambient Intelligent Environments: A Tale of Three Studies, In Proceedings of CHI2011 workshop on DUI, ACM Press, New York, May 7-12, 2011.

[DAM05] Damas, C., Lambeau, B., Dupont, P., and van Lamsweerde, A. Generating Annotated Behavior Models from End-User Scenarios, IEEE

Transactions on Software Engineering, Special Issue on Interaction and State-based Modeling, Vol. 31, No. 12, pp. 1056-1073, 2005.

[DAM06] Damas, C., Lambeau, B., and van Lamsweerde, A. Scenarios, Goals, and State Machines: a Win-Win Partnership for Model Synthesis, In Proceedings of SIGSOFT'06/FSE-14, Portland, Oregon, USA. November 5-11, 2006.

[DEA08] Dearman, D. and Pierce, J.S. It's on my other computer!: computing with multiple devices, In Proceedings of the Conference on Human Factors in Computing Systems, CHI'08, pp. 767-776, 2008.

[DEE10] Dees, W. Lecture Notes in Computer Science, Volume 6763, Human-Computer Interaction. Towards Mobile and Intelligent Interaction Environments, pp. 195-204, 2011.

[DEM05] Demeure, A., Sottet, J. and Calvary, G. A Model-Driven Home Heating Control System. Presented at Plastic Services for Mobile Devices (PSMD), Workshop hel in conjunction with Interact'05. Rome, September 12, 2005.

[DEM05B] Demeure, A., Calvary, G., Sottet, J., and Vanderdonckt, J. A reference model for distributed user interfaces, In Proceedings of the 4th international Workshop on Task Models and Diagrams (Gdansk, Poland). TAMODIA'05, vol. 127, ACM, New York, NY, pp. 79-86, September 26-27, 2005.

[DEM08] Demeure, A., Sottet, J., Calvary, G., Coutaz, J., Ganneau, V., and Vanderdonckt, J. The 4C Reference Model for Distributed User Interfaces, In Proceedings of the Fourth international Conference on Autonomic and Autonomous Systems. ICAS'08 (Gosier), IEEE Computer Society, Washington, DC, pp. 61-69, March 16-21, 2008.

[DEW98] Dewan, P., and Choudhary, R. Coupling the User Interfaces of a Multiuser Program, ACM Transacations on Computer-Human Interaction, pp. 34-62, 1998.

[DEW98B] Dewan, P., and Shen, H. Controlling access in multiuser interfaces, ACM Transactions on Computer-Human Interaction, 5, 1, pp. 34-62, 1998.

[DEY00] Dey, A.K., and Abowd, G.D. CybreMinder: A Context-Aware System for Supporting Reminders, Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K), Bristol, UK, pp. 172-186, September 25-27, 2000.

[DEY01] Dey, A.K. Understanding and Using Context, Personal and Ubiquitous Computing Journal, Volume 5 (1), pp. 4-7, 2001.

[DIN06] Ding, Y., and Litz, H. Creating Multiplatform User Interfaces by Annotation and Adaptation, In Proceedings of the 11th international Conference on intelligent User Interfaces (Sydney, Australia). IUI'06, ACM, New York, NY, pp. 270-272, January 29 - February 01, 2006.

[DIT11] Dittmar, A., and Forbrig, P. Selective Modeling to Support Task Migratability of Interactive Artifacts, In Proceedings of Interact 2011, pp. 571-588. 2011.

[DLG10] de la Guía, E., Lozano, M.D., and Penichet, V.M.R. Co-Interactive Table: a New Facility Based on Distributed User Interfaces to Improve Collaborative Meetings, In Proceedings of the 12th Conference on Human-Computer Interaction with Mobile Devices and Services, Mobile HCI 2010, Lisbon, Portugal, September 7-10, 2010. DOI: 10.1145/1851600.1851702

[EDW02] Edwards, W.K., Newman, M.W., Sedivy, J., Smith, T., and Izadi, S. Challenge: Recombinant Computing and the Speakeasy Approach, In Proceedings of MOBICOM'02, Atlanta, Georgia, USA, September 23-26, 2002.

[EIS01] Eisenstein, J., Vanderdonckt, J., and Puerta, A. Applying Model-Based Techniques to the Developments of UIs for Mobile Computers, In Proceedings of the 6th international Conference on intelligent User interfaces (Santa Fe, New Mexico, USA). IUI'01, ACM Press, New York, pp. 69-76, January 14-17, 2001.

[ELM11] Elmqvist, N., Distributed User Interfaces: State of the Art, In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-Computer Interaction Series, Springer-Verlag, London, pp. 1-12, 2011. DOI 10.1007/978-1-4471-2271-5_1

[ELM15] Badam, S.K., Fisher, E., Elmqvist, N. Munin: A Peer-to-Peer Middleware 10.1007/978-1-4471-2271-5_1for Ubiquitous Analytics and Visualization Spaces, In IEEE Transactions on Visualization & Computer Graphics, vol.21, no.2 pp. 215-228, Febryary, 2015. DOI 10.1109/TVCG.2014.2337337

[EMM98] Emmerich, W. Distributed System Principles, http://www0.cs.ucl.ac.uk/staff/ucacwxe/lectures/ds98-99/dsee3.pdf, 1997.

[ENG07] Englebert, V., and Heymans, P. Towars More Extensible Meta-CASE Tools, In Proceedings of CAiSE2007, Springer-Verlag, Berlin Heidelberg, pp. 454-468, 2007.

[ENS11] Ens, B., Eskicioglu, R., Irani, P. Visually Augmented Interfaces for Co-located Mobile Collaboration In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-Computer Interaction Series, Springer-Verlag, London, pp. 169-176, 2011. DOI 10.1007/978-1-4471-2271-5_19

[EYC03] Eychaner, G. An Extensible Java User Interface Framework for Controlling Distributed Systems. July 08, 2003.

[Eyefinity] AMD Eyefinity,
http://www.amd.com/en-us/innovations/
software-technologies/technologies-gaming/eyefinity#,
Last visited on July 10, 2015.

[Etherpad] EtherPad: Realtime collaborative text editing,
http://www.etherpad.com

[FAR11] Fardoun, H.M., López, S.R., Villanueva, P. G. , Improving E-Learning Using Distributed User Interfaces, In ... ACM Press, New York, pp. 75-85, 2011. DOI 10.1007/978-1-4471-2271-5_9

[FEU07] Feuerstack, S., Blumendorf, M., and Albayrak, S. Prototyping of Multimodal Interactions for Smart Environments based on Task Models, In European Conference on Ambient Intelligence: Workshop on Model Driven Software Engineering for Ambient Intelligence Applications, Darmstadt, Germany, 2007.

[FIS14] Fisher, E.R., Badam, S.K., Elmqvist, N. Designing peer-to-peer distributed user interfaces: Case studies on building distributed applications, In the International Journal of Human-Computer Studies (IJHCS), 72(1), pp. 100110, January 2014. DOI: 10.1016/j.ijhcs.2013.08.011

[FRE09] Frey, A.G., Calvary, G., and Dupuy-Chessa, S. Self-Explanatory User Interfaces by Model-Driven Engineering, In Proceedings of CHI 2009, ACM Press, New York, April 4-9, 2009, Boston, Massachussets, USA.

[FRO11] Fröberg, A., Eriksson, H., Berglund, E., Developing a DUI Based Operator Control Station, In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-Computer Interaction Series, Springer-Verlag, London, pp. 41-49, 2011. DOI 10.1007/978-1-4471-2271-5_5

[GotG] Game of the Goose - Wikipedia,
https://en.wikipedia.org/wiki/Game_of_the_Goose
last visited on October 5th, 2015.

[GAR11] Garrido, J.E., Penichet, V. M. R., Lozano, M.D., Improving Ubiquitous Environments Through Collaborative Features, In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-Computer Interaction Series, Springer-Verlag, London, pp. 59-66, 2011. DOI 10.1007/978-1-4471-2271-5_7

[GHI10] Ghiani, G., and Paternò, F. Supporting Mobile Users in Selecting Target Devices, In Journal of Universal Computer Science, vol. 16, no. 15, 2010.

[1] GHO07 Distributed Systems: An Algorithmic Approach, Ghosh, S., Chapman & Hall/CRC, 2007, ISBN 978-1-58488-564-1.

[GON09] Gonzáles-Calleros, J.M., Muños-Arteaga, J. Towards Canonical Task Types for User Interfaces Design, In Latin American Web Congress, 2009.

[GRA00] Graham, T.C., Watts, L.A., Calvary, G., Coutaz, J., Dubois, E., and Nigay, L. A dimension space for the design of interactive systems within their physical environments, In Proceedings of the 3rd Conference on Designing interactive Systems: Processes, Practices, Methods, and Techniques (New York City, New York, United States). D. Boyarski, and W.A. Kellogg, Eds. DIS'00, ACM Press, New York, pp. 406-416, August 17-19, 2000.

[GRI01] Griffiths, T., Barclay, P.J., Paton, N.W., McKirdy, K., Kennedy, J., Gray, P.D., Cooper, R., Goble, C.A., and Pinheiro, P. Teallach: a Model-based User Interface Development Environment for Object Databases, Interacting with Computers 14, 1, pp. 31-68, December 2001.

[GRO04] Grolaux, D., Van Roy, P., and Vanderdonckt, J. Migratable User Interfaces: Beyond Migratory User Interfaces, In Proceedings of 1st IEEE-ACM Annual Int. Conf. on Mobile and Ubiquitous Systems: Networking and Services MOBIQUITOUS'04 (Cambridge, England), ACM Press, New York, pp. 422-430, August 22-26, 2004.

[GRO05] Grolaux, D., Vanderdonckt, J., and Van Roy, P. Attach me, Detach me, Assemble me like You Work, In Proceedings of the 10th IFIP TC 13 International Conference on Human-Computer Interaction, INTERACT'05 (Rome, Italy), Springer-Verlag, Berlin, LNCS, Vol. 3585, pp. 198-212, September 12-16, 2005.

[GRO07] Grolaux, D. Transparent Migration and Adaptation in a Graphical User Interface toolkit, Ph.D. dissertation, Department of Computing Science and Engineering, Université catholique de Louvain, 2007.

[GRU01] Grudin, J. Partitioning Digital Worlds: Focal and Peripheral Awareness in Multiple Monitor Use, In Proceedings of the ACM Conference on Human Factors in Computing Systems CHI'01 (Seattle, USA), ACM Press, New York, pp. 458-465, 2001.

[GUE06] Guerrero-García, J. Conceptual Modeling of User Interfaces to Workflow Information Systems, Master thesis, 2006.

[GUE09] Guerrero-García, J., Vanderdonckt, J., and Gonzáles-Calleros, J.M. Towards a Multi-User Interaction Meta-Model, Working Paper 2008 - 2009, 2009.

[HAN00] Han, R., Perret, V., and Naghshineh, M. WebSplitter: A Unified XML Framwork for Multi-Device Collaborative Web Browsing, In Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work (Philadelphia, Pennsylvania, USA). CSCW'00, ACM Press, New York, pp. 221-230, December 2-6, 2000.

[HydraVision] Hydravision, https://en.wikipedia.org/wiki/AMD_Catalyst#HydraVision, Last visited on July 10, 2015.

[HIL92] Hill, R.D. The Abstraction Link-View Paradigm: Using Constraints to Connect User Interfaces to Applications, In Proceedings of CHI'92, 1992.

[HOW03] Howell, C.J., Kapfhammer, G.M., Roos, R.S. An Examination of the Run-time Performance of GUI Creation Frameworks, In PPPJ 2003, Kilkenny City, Ireland, June 16-18, 2003.

[HUT02] Hutchings, D.R., and Stasko, J. QuickSpace: New Operations for the Desktop Metaphor, In CHI'02 Extended Abstracts on Human Factors in Computing Systems (Minneapolis, Minnesota, USA). CHI'02, ACM Press, New York, pp. 802-803, April 20-25, 2002.

[HUT02B] Hutchings, D.R., and Stasko, J. New Operations for Display Space Management and Window Management, Technical Report, August, 2002.

[HUT03] Hutchings, D.R., and Stasko, J. An Interview-Based Study of Display Space Management, Technical Report GIT-GVU-03-17, 2003.

[HUT04] Hutchings, D.R., Smith, G., Meyers, B., Czerwinski, M., and Robertson, G. Display Space Usage and Window Management Operation Comparisons between Single Monitor and Multiple Monitor Users, In Proceedings of the Working Conference on Advanced Visual interfaces (Gallipoli, Italy). AVI'04, ACM Press, New York, pp. 32-39, May 25-28, 2004.

[HUT04B] Hutchings, D.R., and Stasko, J. Revisiting display space management: understanding current practice to inform next-generation design, In Proceedings of Graphics interface 2004 (London, Ontario, Canada). ACM International Conference Proceeding Series, vol. 62. Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, pp. 127-134, May 17-19, 2004.

[HUT04C] Hutchings, D.R., Czerwinski, M., Meyers, B., and Stasko, J. Exploring the Use and Affordances of Multiple Display Environments, 2004.

[HUT04D] Hutchings, D.R., and Stasko, J. Shrinking Window Operations for Expanding Display Space, In Proceedings of the working conference on Advanced Visual Interfaces, AVI'04, 2004.

[HUT05] Hutchings, D.R. M5: Making Multiple Monitors More Manageable, In Proceedings of UIST'05, Seattle, Washington, USA. October 23-27, 2005.

[HUT05B] Hutchings, D.R., and Stasko, J. mudibo: Multiple Dialog Boxes for Multiple Monitors, In Proceedings of CHI 2005, Portland, Oregon, USA. April 2-7, 2005.

[HUT05C] Hutchings, D.R., Czerwinski, M., Robbins, D., and Robertson, G. TaskZones: A Task Manager for Multiple-Monitor Systems, In Proceedings of UIST'05, Seattle, Washington, USA. October 23-27, 2005.

[HUT05D] Hutchings, D.R., Stasko, J., and Czerwinski, M.
http://facstaff.elon.edu/dhutching/dde.
CHI 2005 Workshop on Distributed Display Environments, Sunday, April 3rd, 2005.

[HUT06] Hutchings, H.M., and Pierce, J.S. Understanding the whethers, hows, and whys of divisible interfaces, In Proceedings of the Working Conference on Advanced Visual Interfaces (Venezia, Italy). AVI'06, ACM Press, New York, pp. 274-277, May 23-26, 2006.

[HUT07] Hutchings, D.R., and Stasko, J. Quantifying the Performance Effect of Window Snipping in Multiple-monitor Environments, In Human-Computer Interaction - INTERACT 2007: 11th IFIP TC 13 International conference Rio de Janerio, Brazil Proceedings, Part II., Springer, pp. 461-474, September 10-14, 2007.

[HUT07B] Hutchings, D.R., and Stasko, J. Consistency, Multiple Monitors, and Multiple Windows, 2007.

[INK05] Inkpen, K.M., and Mandryk, R.L. Multi-Display Environments for Co-located Collaboration, 2005.

[ISO99] ISO/IEC 14754: Pen-based interfaces - Common Gestures for text editing with pen-based system, International Standard Organization, Geneva, 1999.

[JOU10] Jourde, F., Laurillau, Y., Nigay, L. COMM Notation for Specifying Collaborative and MultiModal Interactive Systems, In Proceedings of EICS2010, 2010.

[KAVA10] Kavaldjian, S., Raneburger, D., Popp, R., Letiner, M., Falb, J., and Kaindl, H. Automated Optimization of User Interfaces for Screens with Limited Resolution, In Proceedings of MDDAUI2010 & CHI2010, pp. 13-16, 2010.

[KAVI11] Kaviani, N., Finke, M., Lea, R., Fels, S. Investigating the Design Space for Multi-display Environments, In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-Computer Interaction Series, Springer-Verlag, London, pp. 103-112, 2011. DOI 10.1007/978-1-4471-2271-5_12

[KOR05] Kortuem, G., and Kray, C. HCI Issues of Dispersed Public Displays, 2005.

[LAC03] Lachenal, C., and Coutaz, J. A Reference Framework For Multi-Surface Interaction, In Proceedings of HCI International 2003 (Crete, Greece), pp. 424-428, 2003.

[LAD10] Ladry, J., Palanque, P., Barboni, E., and Navarre, D. Model-Based Usability Evaluation and Analysis of Interactive Techniques, In Proceedings of MDDAUI2010 & CHI2010, pp. 21-24, 2010.

[LAM11] Lambropoulos, N., and Danis, S. Humane Machine Interaction Design: DUIs Design Thinking for Social Innovation, In Proceedings of CHI2011 workshop on DUI, ACM Press, New York, May 7-12, 2011.

[LAR06] Larsson, A., and Ingmarsson, M. Ubiquitous Information Access Through Distributed User Interfaces and Ontology-based Service-discovery, In Proceedings of MU3I'06, Sydney, Australia, 2006.

[LAR07] Larsson, A., Ingmarsson, M., and Sun, B. A Development Platform for Distributed User Interfaces, In Proceedings of the Nineteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2007), Boston, Massachusetts, USA, July 9-11, 2007.

[LEE08] Lee, J.C., Hudson, S.E., and Tse, E. Foldable Interactive Displays, In Proceedings of UIST'08, Monterey, California, USA, October 19-22, 2008.

[LEP06] Lepreux, S., Vanderdonckt, J., and Michotte, B. Visual Design of User Interfaces by (De)composition, In Proceedings of DSVIS2006, LNCS 4323, pp. 157-170, 2007.

[LEP06B] Lepreux, S., and Vanderdonckt, J. Toward a Support of The Design of User Interfaces by Using Composition Rules, CADUI 2006, pp. 231-244, 2006.

[LEP11] Lepreux, S., Kubicki, S., Kolski, C., and Caelen, J. Distributed Interactive Surfaces: A Step Towards the Distribution of Tangible and Virtual Objects, In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-Computer Interaction Series, Springer-Verlag, London, pp. 133-143, 2011. DOI 10.1007/978-1-4471-2271-5_15

[LET01] Letier, E. Reasoning about Agents in Goal-Oriented Requirements Engineering, Thesis, 2001.

[LIN93] Linten, M., and Price, C. Building Distributed User Interfaces with Fresco, In Proceedings of the Seventh X Technical Conference, pp. 77-87, 1993.

[LOC11] Löchtefeld, M., Gehring, S., and Krüger, A., Distributed User Interfaces for Projector Phones, In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-Computer Interaction Series, Springer-Verlag, London, pp. 113-123, 2011. DOI 10.1007/978-1-4471-2271-5_13

[LOE03] Loeser, C., Mueller, W., Berger, F., and Eikerling, H.J. Peer-to-peer Networks for Virtual Home Environments, In Proceedings of HICSS-36 (Big Island), IEEE Computer Society Press, Los Alamitos, pp. 282, January 6-9, 2003.

[LON00] Long Jr, A.C., Landay, J.A., Rowe, L.A., and Michiels, J. Visual Similarity of Pen Gestures, In Proceedings of SIGCHI'00, 2000.

[LON01] Long Jr, A.C., Landay, J.A., and Rowe, L.A. quill: a Gesture Design Tool for Pen-based User Interfaces, 2001.

[LOP11] López-Espin, J.J., Gallud, J.A., Lazcorreta, E., Peñalver, A., and Botella, F. Formal Specification of Distributed User Interface, In Proceedings of CHI2011 workshop on DUI, ACM Press, New York, May 7-12, 2011.

[LOR10] Lorenz, A. Research Directions for the Applications of MVC in Ambient Computing Environments, In Proceedings of the 1st International Workshop on Pattern-Driven Engineering of interactive Computing Systems PEICS'10 (Berlin, Germany), ACM Press, New York, pp. 28-31, July 20, 2010.

[LUY02] Luyten, K., Vandervelpen, Ch., and Coninx, K. Migratable User Interface Descriptions in Component-Based Development, In Proceedings of DSV-IS'2002 (Rostock, Germany), Lecture Notes in Computer Science, VOl. 2545. Springer-Verlag, London, pp. 44-58, June 12-14, 2002.

[LUY04] Luyten, K. Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development, Thesis, 2004.

[LUY05] Luyten, K., and Coninx, K. Distributed User Interface Elements to support Smart Interaction Spaces, In Proceedings of the 7th IEEE International Symposium on Multimedia ISM'2005, IEEE Computer Society Press, Washington, DC, pp. 277-286, December 12-14, 2005.

[LUY06] Luyten, K., Van den Bergh, J., Vandervelpen, Ch., and Coninx, K. Designing Distributed User Interfaces for Ambient Intelligent Environments using Models and Simulations, Computers & Graphics 30, 5, pp. 702-713, 2006.

[MTE93] McTear, M.F. User Modelling for Adaptive Computer Systems: a Survey of Recent Developments Artificial Intelligence Review 7, Kluwer Academic Publishers, Netherlands, pp. 157-184, 1993.

[MANC11] Manca, M., and Paternò, F. Distributed User Interfaces with MARIA, In Proceedings of CHI2011 workshop on DUI, ACM Press, New York, pp. -, May 7-12, 2011.

[MANC11B] Manca, M., and Paternò, F. Flexible Support for Distributed User Interfaces Across Multiple Devices, In Proceedings of CHI2011, September 13-16, 2011.

[MANC11C] Manca, M., Paternò, F., Extending MARIA to Support Distributed User Interfaces, In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-Computer Interaction Series, Springer-Verlag, London, pp. 33-40, 2011. DOI 10.1007/978-1-4471-2271-5_4

[MAND94] Mandviwalla, M. What Do Groups Need? A Proposed Set of Generic Groupware Requirements, 1994.

[MANS05] Mansoux, B., and Nigay, L. Distributed Display Environments in Computer-Assisted Surgery Systems, 2005.

[MARC11] Marco F.A., Penichet, V. M. R., Gallud, J.A. , Drag & Share: A Shared Workspace for Distributed Synchronous Collaboration In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosys-

tem, Human-Computer Interaction Series, Springer-Verlag, London, pp. 125-132, 2011. DOI 10.1007/978-1-4471-2271-5_14

[MARQ07] Marquardt, N., and Greenberg, S. Shared Phidgets: A Toolkit for Rapidly Prototyping Distributed Physical User Interfaces, In TEI'07: Proceedings of the 1st international conference on Tangible and embedded interaction (Baton Rouge, Louisiana, USA), ACM Press, New York, pp. 13-20, February 15-17, 2007.

[MART10] Martinie, C., Ladry, J., Navarre, D., Palanque, P., and Winckler, M. Embedding Requirements in Design Rationale to Deal Explicitely with User eXperience and Usability in an "intensive" Model-Based Development Approach, In Proceedings of MDDAUI2010 & CHI2010, pp. 29-32, 2010.

[MASP] MASP.
http://masp.dai-labor.de,
Visited on December 8th, 2015.

[MEJ10] Mejías, B. Beernet: A Relaxed Approach to the Design of Scalable Systems with Self-Managing Behaviour and Transactional Robust Storage, Thesis, 2010.

[MEL09] Melchior, J., Grolaux, D., Vanderdonckt, J., and Van Roy, P. A Toolkit for Peer-to-Peer Distributed User Interfaces: Concepts, Implementation, and Applications, In Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2009), ACM Press, New York, pp. 69-78, Pittsburgh, PA, USA, July 15-17, 2009.

[MEL11] Melchior, J., Vanderdonckt, J., Van Roy, P. A Model-Based Approach for Distributed User Interfaces, In Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2011), Pisa, Italy, June 13-16, 2011.

[MEL11dc] Melchior, J. Distributed User Interfaces in Space and Time, In Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2011), Pisa, Italy, June 13-16, 2011.

[MEL11dui] Melchior, J., Vanderdonckt, J., Van Roy, P. Distribution Primitives for Distributed User Interfaces, In Proceeding of DUI 2011, ACM Press, New York, pp. 29-32, Vancouver, British Columbia, Canada, 2011. DOI 10.1007/978-1-4471-2271-5_3

[MEL11dui2] Melchior, J., Vanderdonckt, J., Van Roy, P. Distribution Primitives for Distributed User Interfaces, In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-

Computer Interaction Series, Springer-Verlag, London, pp. 23-31, 2011. DOI 10.1007/978-1-4471-2271-5_3

[MEL12dui] Melchior, J., Mejías, B., Jaradin, Y., Van Roy, P., and Vanderdonckt, J. Improving DUIs with a decentralized approach with transactions and feedbacks, In Proceeding of DUI 2012, ACM Press, New York, pp. 65-68, Austin, Texas, USA, 2012. DOI 10.1007/978-1-4471-2271-5_3

[MEL12] Melchior, J., Vanderdonckt, J., Van Roy, P. Modelling and Developing Distributed User Interfaces based on Distribution Graph, In Proceedings of the Sixth International Conference on Research Challenges in Information Science (RCIS 2012), Valencia, Spain, May 16-18, 2012.

[MEL12-IJHCI] Melchior, J., Vanderdonckt, J., and Van Roy, P. A Comparative Evaluation of User Preferences for Extra-User Interfaces, In the International Journal of Human-Computer Interaction (IJHCI), 28:11, Taylor & Francis, pp. 760-767, 2012. DOI:10.1080/10447318.2012.715544

[MESA04] Mesarov, V., Carton, B., and Van Roy, P. P2PS: Peer-to-Peer Development Platform for Mozart, In Proceedings of Second International Mozart/Oz Conference MOZ'04, LNCS, Vol. 3389, Springer, Berlin, pp. 125-136, 2004.

[MESK08] Meskens, J., Vermeulen, J., Luyten, K., and Coninx, K. Gummy for Multi-Platform User Interface Designs: Shape me, Multiply me, Fix me, Use me, In Proceedings of the Working Conference on Advanced Visual interfaces (Napoli, Italy). AVI'08, ACM Press, New York, pp. 233-240, May 28-30, 2008.

[MESK09] Meskens, J., Luyten, K., and Coninx, K. Plug-and-Design: Embracing Mobile Devices as Part of the Design Environment, In Proceedings of EICS'2009, 2009.

[MIA99] Miah, T., and Alty, J.L. Vanishing Windows: an Empirical Study of Adaptive Window Management, In Proceedings of CADUI99, 1999.

[Miracast] Wi-Fi CERTIFIED Miracast, http://www.wi-fi.org/discover-wi-fi/wi-fi-certified-miracast,
Last visited on July 11, 2015.

[MOL06] Molina, J.P., Vanderdonckt, J., González, P.,
Fernández-Caballero, A., and Lozano, M.D.
Rapid Prototyping of Distributed User Interfaces, In Proceedings of the CADUI'2006 (Bucharest, Romania), Springer-Verlag, Berlin, pp. 151-166, June 6-8, 2006.

[MOL06B] Molina, J.P., Vanderdonckt, J., and González, P. Direct Manipulation of User Interfaces for Migration, In Proceedings of International Conference on Intelligent User Interfaces, IUI'2006, pp. 140-147, 2006.

[MOR03] Mori, G., Paternò, and Santoro, C. Tool Support for Designing Nomadic Applications, In Proceedings of the 8th international Conference on intelligent User interfaces (Miami, Florida, USA). IUI'03, ACM Press, New York, pp. 141-148, January 12-15, 2003.

[MOR04] Mori, G., Paternò, F., and Santoro, C. Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions, IEEE Transactions on Software Engineering, vol. 30, no. 8, pp. 507-520, August, 2004.

[MOS09] Moscovitch, T. Contact Area Interaction with Sliding Widgets, In Proceedings of the 22nd ACM Symposium on User interface software and technology, UIST'2009, ACM Press, New York, pp. 13-22, 2009.

[Mozart] The Mozart Programming System,
http://mozart.github.io/,
Visited on October 13th, 2015.

[MYE00] Myers, B., Hudson, S.E., and Pausch, R. Past, present, and future of user interface software tools, ACM Trans. Comput.-Hum. Interact. 7, 1., pp. 3-28, Mars 2000.

[MYE01] Myers, B.A. Using Handhelds and PCs Together, Communication of the ACM 44,11, pp. 34-41, November 2001.

[NEW02] Newman, M.W., Izadi, S., Edwards, W.K., Sedivy, J.Z., and Smith, T.F. User Interfaces When and Where They are Needed: An Infrastructure for Recombinant Computing, In Proceedings of the 15th ACM Symposium on User interface Software and Tehcnology UIST'02 (Paris, France), ACM Press, New York, pp. 171-180, October 27-30, 2002.

[NEW02B] Newman, M.W., and Sedivy, J. Designing for Serendipity: Supporting End-User Configuration of Ubiquitous Computing Environments, In Proceedings of DIS2002, London, England, 2002.

[NFC] ISO/IEC 18092:2004, Information technology - Telecommunications and information exchange between systems - Near Field Communication - Interface and Protocol (NFCIP-1),
http://www.iso.org/iso/home/store/catalogue_tc/
catalogue_detail.htm?csnumber=38578,
Last visited on July 10, 2015.

[NIE11] http://www.nielsen.com/us/en/insights/news/2011/in-the-u-s-tablets-are-tv-buddies-while-ereaders-make-great-bedfellows.html, Last visited on September 4, 2015.

[nView] Display and desktop management, nView desktop management software, http://www.nvidia.com/object/nview-display-us.html, Last visited on July 10, 2015.

[Paint.NET] http://www.getpaint.net

[PAS07] Pastor, O., and Molina, J. C., Model-Driven Architecture in Practice : A Software Production Environment Based on Conceptual Modeling, 2007, XVI, 302 p. 48 Illus.

[PAT01] Paternò, F., and Santoro, C. A Unified Method for Designing Interactive Systems Adaptable to Mobile and Stationary Platforms, In IWC, 2001.

[PAT02] Paternò, F., and Santoro, C. One Model, Many Interfaces, In Proceedings of CADUI'2002, the 4th International Conference on Computer-Aided Design of User Interfaces, Valencinnes, France, Kluwer Academics Publisher, Dordrecht, pp. 143-154, May 15-17, 2002.

[PAT07] Paternò, F., Santoro, C., Scorcia, A., Bandelloni, R., and Mori, G. Web user interface migration through difference modalities with dynamic device discovery, In Proceedings of the 2nd International Workshop on Adaptation and Evolution in Web Systems Engineering, AEWSE'2007, Vol. 267, CEUR Workshop Proceedings, 2007. http://sunsite.informatik.rwth-aachen.de/ Publications/CEUR-WS/Vol-267/paper4.pdf,

[PAT08] Paternò, F., Santoro, C., Mäntyjärvi, J., Mori, G., and Sansone, S. Authoring Pervasive Multimodal User Interfaces, In Int. J. Web Engineering and Technology, Vol.4, No. 2, 2008.

[PAT09] Paternò, F., Santoro, C., and Spano, L.D., MARIA: A Universal Declarative, Multiple Abstract-Level Language for Service-Oriented Applications in Ubiquitous Environments, 2009.

[PAT09B] Paternò, F., Santoro, C., and Spano, L.D., Model-Based Design of Multi-device Interactive Applications Based on Web Services, In Proceedings of the 12th IFIP TC 13 international Conference on Human-Computer interaction: PART I (Uppsala, Sweden). T. Gross, J. Gulliksen, P. Kotz, L. Oestreicher, P. Palanque, R. O. Prates, and M. Winckler, Eds. Lecture Notes In Computer Science, vol. 5726, Springer-Verlag, Berlin, Heidelberg, pp. 892-905, Auguste 24-28, 2009.

[PAT10] Paternò, F., and Zichitella, G. End-User Customization of Multi-Device Ubiquitous User Interfaces, In Proceedings of MDDAUI2010 & CHI2010, pp. 41-44, 2010.

[PAT10B] Paternò, F., and Zichitella, G. Desktop-to-Mobile Web Adaptation through Customizable Two-Dimensional Semantic Redesign, In Proceedings of HCSE 2010, pp. 79-94, 2010.

[PEL00] Peleg, D. Distributed Computing: A Locality-Sensitive Approach, SIAM, 2000, ISBN 0-89871-464-8.

[PENA11] Peñalver, A., López-Espín, J. J., Gallud, J. A., Lazcorreta, E., and Botella, F. Distributed User Interfaces: Specification of Essential Properties, In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-Computer Interaction Series, Springer-Verlag, London, pp. 13-21, 2011. DOI 10.1007/978-1-4471-2271-5_2

[PENA11B] Peñalver, A., López-Espín, J. J., Gallud, J. A., Lazcorreta, E., and Botella, F. An AUI Model to Support Distributed User Interfaces, In Proceedings of UIDL'2011, 2011.

[PENA12] Peñalver, A., López-Espín, J. J., Botella, F., Lazcorreta, E., and Gallud, J. A. Schema Driven Distributed User Interface Generation, In Proceeding of the XIII Congreso Internacional de Interacción Persona-Ordenador, Interacción 2012, 2012.

[PENI07] Penichet, V.M.R., Lozano, M., Gallud, J.A., and Tesoriero, R. Task Modelling for Collaborative Systems, In Proceedings of TAMODIA 2007, 2007.

[Pictionary] Pictionary - Wikipedia,
https://en.wikipedia.org/wiki/Pictionary
last visited on October 5th, 2015.

[PIE04] Pierce, J.S., Mahaney, H.E., Opportunistic Annexing for Handheld Devices: Opportunities and Challenges,
In Proceedings of Human-Computer Interface Consortium 2004,
accessible on-line at:
http://wwwstatic.cc.gatech.edu/∼jpierce/papers/OA-HCIC2004.pdf,
2004.

[Project My Screen] Project my phone screen to a TV or PC,
http://www.windowsphone.com/en-us/how-to/wp8/
connectivity/project-my-phone-screen,
Last visited on July 11, 2015.

[PUE02] Puerta, A., and Eisenstein, J. XIML: A Common Representation for Interaction Data, In Proceedings of, IUI'02, San Francisco, California, USA, January 13-16, 2002.

[QIU09] Qiu, X.F., and Graham, T.N. Flexible and Efficient Platform Modeling for Distributed Interactive Systems, In Proc. of the 1st ACM SIGCHI Symposium on Engineering interactive Computing Systems (Pittsburgh, PA, USA). EICS'09, ACM Press, New York, pp. 29-34, July 15-17, 2009.

[RASH12] Rashid, U., Nacenta, M. A., and Quigley, A. The Cost of Display Switching: A Comparison of Mobile, Large Display and Hybrid UI Configurations, In Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI '12), ACM Press, New York, May 21, 2012. pp. 99106 DOI 10.1145/2254556.2254577

[REK97] Rekimoto, J. Pick-and-Drop: a Direct Manipulation Technique for Multiple Computer Environments, In Proceedings of the 10th Annual ACM Symposium on User interface Software and Technology (Banff, Alberta, Canada). UIST'97, ACM Press, New York, pp. 31-39, October 14-17, 1997.

[REK03] Rekimoto, J., Ayatsuka, Y., Kohno, M., and Oba, H. Proximal Interactions: A Direct Manipulation Technique for Wireless Networking, In Proceedings of the 9th IFIP TC 13 International Conference on Human-Computer Interaction, INTERACT'2003, IOS Press, Amsterdam, pp. 511-518, 2003.

[REY04] Rey, G., and Coutaz, J. Contextor: capture and dynamic distribution of contextual information, In Proceedings of the 1st French-Speaking Conference on Mobility and Ubiquity Computing (Nice, France). UbiMob'04, vol. 64, ACM Press, New York, pp. 131-138, June 01-03, 2004.

[REY06] Rey, G. Context en Interaction Homme-Machine: le contexteur, Thesis, 2006.

[RIC09] Rich, C. Building Task-Based User Interfaces with ANSI/CEA-2018, In IEEE Computer Society, 2009.

[ROB04] Robertson, G., Horvitz, E., Czerwinski, M., Baudisch, P., Hutchings, D., Meyers, B., Robbins, D., and Smith, G. Scalable Fabric: Flexible Task Management, In Proceedings of the working conference on Advanced Visual Interfaces, 2004.

[ROD04] Rodden, T., Crabtree, A., Hemmings, T., Koleva, B., Humble, J., Akesson, K.P., and Hansson, P. Configuring the Ubiquitous Home,

In Proceedings of International Workshop on Cooperative Systems Design, Scenario-Based Design of Collaborative Systems, COOP'2004, IOS Press, Amsterdam, pp. 227-242, 2004.

[ROS09] Roscher, D., Blumendorf, M., and Albayrak, S. A Meta User Interface to Control Multimodal Interaction in Smart Environments, In Proceedings of the 13rd International Conference on Intelligent User Interface, IUI'2009, ACM Press, New York, pp. 481-482, 2009.

[ROS09B] Roscher, D., Blumendorf, M., and Albayrak, S. Using Meta User Interfaces to Control Multi-modal Interaction in Smart Environments, In Proceedings of the 4th International Workshop on Model Driven Development of Advanced User Intrefaces MDDAUI'2009, Vol. 439, EUR Workshop Proceedings, 2009. http://ceur-ws.org/Vol-439/paper4.pdf,

[ROS10] Roscher, D., Blumendorf, M., and Albayrak, S. Multimodal User Interface Model for Runtime Distribution, In Proceedings of MDAAUI2010 & CHI2010, pp. 5-8, 2010.

[ROU06] Roudaut, A., and Coutaz, J. Méta-IHM ou comment contrôler l'espace interactif ambiant, In Proceedings of Actes des Troisièmes Journées Francophones: Mobilité et Ubiquité 2006 (Paris, France). UbiMob'2006, ACM Press, New York, 2006.

[ROU06B] Roudaut, A. Méta-IHM: Pour le contrôle d'espace interactif ambiant, Ph.D. dissertation, Ingénierie de l'Interaction Homme-Machine, Institut National Polytechnique de Grenoble ENSIMAG, 2006.

[ROU09] Roudaut, A., Lecolinet, E., and Guiard, Y. MicroRolls: Expanding Touch-Screen Input Vocabulary by Distringuishing Rolls vs. Slides of the Thumb, In Proceedings of ACM Conference on Human Aspects in Computing Systems, CHI'2009, ACM Press, New York, pp. 927-936, 2009.

[RUS05] Russel, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D. Workflow Resource Patterns: Identification, Representation and Tool Support. In: Pastor, O., Falcão e Cunha, J. (eds): 17th Int. Conf. Advanced Information Systems Engineering CAiSE 2005. LNCS, vol. 3520, pp. 216-232, Springer, Heidelberg, 2005.

[SAL09] Salay, R., Mylopoulos, J., and Easterbrook, S.M. Using Macromodels to Manage Collections of Related Models, In Proceedings of CAiSE'2009, Springer, LNCS, vol. 5565, pp. 141-155, 2009.

[SCHL10] Schlegel, T. An Interactive Process Meta Model for Runtime User Interface Generation and Adaptation, In Proceedings of MDDAAUI2010 & CHI 2010, pp. 9-12, 2010.

[SCHW10] Schwartze, V., Blumendorf, M., and Albayrak, S. Adjustable Context Adaptations for User Interfaces at Runtime, In Proceedings of AVI'10, Rome, Italy, May 25-29, 2010.

[SCREEN] Hiperspace Is The World's Highest Resolution Display, http://gizmodo.com/5023526/ hiperspace-is-the-worlds-highest-resolution-display, September 2009.

[SEB11] Sebastián, G., Villanueva, P.G. , Tesoriero, R., and Gallud, J.A. Multi-touch Collaborative DUI to Create Mobile Services, ACM Press, New York, pp. 145-151, 2011. DOI 10.1007/978-1-4471-2271-5_16

[SEI09] Seifried, T., Haller, M., Scott, S.D., Perteneder, F., Rendl, C., Sakamoto, D., and Inami, M. CRISTAL: A Collaborative Home Media and Device Controller Based on a Multi-touch Display, In Proceedings of ITS'09, Banff, Alberta, Canada. November 23-25, 2009.

[SEI11] Seifried, T., Jetter, H.-C., Haller, M., and Reiterer H. Lessons Learned from the Design and Implementation of Distributed Post-WIMP User Interfaces, ACM Press, New York, pp. 95-102, 2011. DOI 10.1007/978-1-4471-2271-5_11

[SEN11] Sendín, M., López, J.-M., Software Infrastructure for Enriching Distributed User Interfaces with Awareness, In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-Computer Interaction Series, Springer-Verlag, London, pp. 51-58, 2011. DOI 10.1007/978-1-4471-2271-5_6

[SEN11B] Sendin, M., López, J.M. Approach for Enriching Distributed User Interfaces with Awareness, In Proceedings of CHI2011 workshop on DUI, ACM Press, New York, pp. 13-16 May 7-12, 2011.

[SHAC09] Shackel, B. Usability - Context, framework, definition, design and evaluation, In Interacting with Computers 21 (5-6), pp. 339-346 December 2009.

[SHAE08] Shaer, O., Jacob, R.J., Green, M., and Luyten, K. User Interface Description Languages for Next Generation User Interfaces, In CHI'08 Extended Abstracts on Human Factors in Computing Systems (Florence, Italy). CHI'08, ACM Press, New York, pp. 3949-3952, April 05-10, 2008.

[SHE04] Shen, C., Ryall, K., and Everitt, K. Facets of Distributed Display Environments, 2004.

[SJO04] Sjölund, M., Larsson, A., and Berglund, E. Smartphone Views: Building Multi-Device Distributed User Interfaces, In Proceedings of MobileHCI'2004 (Glasgow, Scotland), LNCS, Vol. 3160., Springer, Berlin, pp. 507-511, September 13-16, 2004.

[SOT07] Sottet, J.-S., Calvary, G., Favre, J.-M., and Coutaz, J. Megamodeling and Metamodel-driven Engineering for Plastic User Interface: Mega-UI, In Seffah, A., Vanderdonckt, J., Desmarais, M. (eds.), Human-Centered Software Engineering, Vol. II, Springer, pp. 173-200, 2007.

[SOU09] Sousa, K. Model-Driven Approach for User Interface - Business Alignment, In Proceedings of EICS'2009, Pittsburgh, Pennsylvania, USA, July 15-17, 2009.

[STI10] Still, J.D., and Masciocchi, C.M. A Saliency Model Predicts Fixations in Web Interfaces, In Proceedings of MDDAUI2010 & CHI2010, pp. 25-28, 2010.

[STU04] Stuerzlinger, W. MULTI: Multi-User Laser Table Interface, 2004.

[TAN03] Tan, D.S., and Czerwinski, M. Effects of Visual Separation and Physical Discontinuities when Distributing Information across Multiple Displays, In Proceedings of INTERACT'03 (Zurich, Germany), IOS Press, pp. 252-260, September 1-5, 2003.

[TAN04] WinCuts: Manipulating Arbitrary Window Regions for More Effective User of Screen Space, In Proceedings of CHI'2004 (Vienna, Austria), ACM Press, New York, pp. 1525-1528, April 24-29, 2004.

[TEL95] Tel, G. Introduction to Distributed Algorithms, Cambridge University Press, 1995. ISBN 0-521-47069-2

[TES10] Tesoriero, R., Vanderdonckt, J., Gallud, J.A., and Lozano, M. Extending UsiXML to Support Location Awareness, 2010.

[TES10B] Tesoriero, R., and Vanderdonckt, J. Extending UsiXML to support User-aware Interfaces, 2010.

[TeXnicCenter] http://www.texniccenter.org

[THE99] Thevenin, D., and Coutaz, J. Plasticity of User Interfaces: Framework and Research Agenda., In Proceedings of IFIP INTERACT'99: Human-Computer Interaction, 1999.

[THE02] Thevenin, D., and Coutaz, J. Adaptation des IHM: taxonomies et archi. logicielle, In Proceedings of the 14th French-Speaking Conference on Human-Computer interaction (Conférence Francophone Sur l'interaction Homme-Machine) (Poitiers, France). M. Beaudouin-Lafon, ED. IHM'02, vol. 32, ACM Press, New York, pp. 207-210, November 26-29, 2002.

[TUL08] Tullis, T., and Albert, B. Measuring The User Experience, Collecting, Analyzing, and Presenting Usability Metrics, Morgan Kaufmann Publishers, San Francisco, 2008.

[VAN08] Vanacken, D., Demeure, A., Luyten, K., and Coninx, K. Ghosts in the Interface: Meta-user Interface Visualizations as Guides for Multi-touch Interaction, In Proceedings of IEEE International Workshop on Horizontal Interactive Human Computer System, TABLETOP'2008, IEEE Computer Society Press, Los Alamitos, pp. 87-90, 2008.

[VDD97] Vanderdonckt, J. A Small Knowledge-Based System for Selecting Interaction Styles, 1997.

[VDD01] Vanderdonckt, J., Furtado, E., Furtado, V., Limbourg, Q., Silva, W., Rodrigues, D., and Taddeo, L. Multi-model and Multi-level Development of User Interfaces, In Multiple User Interfaces - Cross-Platform Applications and Context-Aware Interfaces, John Wiley & Sons, pp. 193-216, 2001.

[VDD08] Vanderdonckt, J., Calvary, G., Coutaz, J., and Stanciulescu, A. Multimodality for Plastic User Interfaces: Models, Methods, and Principles, In Multimodal User Interfaces: Signals and Communication Technology, Springer, 2008.

[VDD08B] Vanderdonckt, J., Mendonca, H., and Molina, J.-P. Distributed User Interfaces in Ambient Environment, In Proceeding of AmI 2007 Workshop, 2008.

[VDD10] Vanderdonckt, J. Distributed User Interfaces: How to Distribute User Interface Elements across Users, Platforms, and Environments, In Proceedings of the XIth Congreso Internacional de Interaccion Persona-Ordenador Interaccion'2010 (Valencia, Spain), AIPO, Valencia, pp. 3-14, September 7-10, 2010.

[VDH05] Vanderhulst, G., Dynamic Distributed User Interfaces: Supporting Mobile Interaction Spaces, (Thesis), 2005

[VDH07] Vanderhulst, G., Luyten, K., and Coninx, K. Middleware for Ubiquitous Service-oriented Spaces on the Web, 2007.

[VDH08] Vanderhulst, G., Luyten, K., and Coninx, K. ReWiRe: Designing Reactive Systems for Pervasive Environments, 2008.

[VDH08B] Vanderhulst, G., Luyten, K., and Coninx, K. ReWiRe: Creating Interactive Pervasive Systems that cope with Changing Environments by Rewiring, In Proceedings of the 4th IET International Conference on Intelligence Environment (IE'08), pp. 1-8, 2008.

[VDH08C] Vanderhulst, G., Luyten, K., and Coninx, K. Put the User in Control: Ontology-driven Meta-level Interaction for Pervasive Environments, In the First International Workshop on Ontologies in Interactive Systems, 2008.

[VDH09] Vanderhulst, G., Schreiber, D., Luyten, K., Muhlhauser, M., and Coninx, K. Edit, Inspect and Connect your Surroundings: A Reference Framework for Meta-UIs, In Proceedings of 1st Symposium on Engineering Interactive Computing Systems, EICS'2009, ACM Press, New York, pp. 167-175, 2009.

[VDH09B] Vanderhulst, G., Luyten, K., and Coninx, K. Photo-Based User Interfaces: Picture It, Tag It, Use It, In OTM 2009 Workshop, 2009.

[VDH10] Vanderhulst, G., Luyten, K., and Coninx, K. Pervasive Maps: Explore and Interact with Pervasive Environments, 2009.

[VDH10B] Vanderhulst, G., Luyten, K., and Coninx, K. SemSon: Connecting Ontologies and Web Applications, In Proceeding of the 6th International Conference on Web Information Systems and Techonologies, WEBIST 2010, pp. 163-166, 2010.

[VDH10C] Vanderhulst, G., Luyten, K., and Coninx, K. On a Journey from Message to Observable Pervasive Application, In the International Conference on Complex, Intelligent and Software Intensive Systems, 2010.

[VDV04] Vandervelpen, C., and Coninx, K. Towards model-based design support for distributed user interfaces, In Proceedings of the Third Nordic Conference on Human-Computer interaction (Tampere, Finland), NordiCHI'04, vol. 82, ACM Press, New York, pp. 61-70, October 23-27, 2004.

[VDV05] Vandervelpen, C., Vanderhulst, G., Luyten, K., and Coninx, K. Light-Weight Distributed Web Interfaces: Preparing the Web for Heterogeneous Environments, In Proceedings of the 5th International Conference on Web Engineering ICWE'2005 (Sydney, Australia), Springer, Berlin, pp. 197-202, July 27-29, 2005.

[VDV05B] Vandervelpen, C., Vanderhulst, G., Luyten, K., and Coninx, K. Light-Weight Distributed Web Interfaces: Preparing the Web for Heterogeneous Environments, In Proceedings of the 5th International Conference on Web Engineering ICWE'2005 (Sydney, Australia), Springer, Berlin, 2005.

[VERN99] Vernier, F., and Nigay, L. Interfaces Multimodales: Composition et Caractérisation des Modalités de Sortie 1999 ou 2000.

[VERN00] Vernier, F., and Nigay, L. Espace de Conception pour les Interfaces Multimodales, In Colloque sur les modalités, Mai, 2000.

[VIL11] Villanueva, P.G., Tesoriero, R., Sebastin, G., and Gallud, J.A. Using Multi-touch Technologies to Perform Collaborative Map Exploration,

In Proceedings of CHI2011 workshop on DUI, ACM Press, New York, pp. 57-60, May 7-12, 2011.

[VS] http://www.microsoft.com/visualstudio

[VMDE13] Developer Economics Q3 2013, http://www.developereconomics.com/reports/q3-2013/, 2013.

[VRA03] Vrazalic, L. Evaluating Distributed Usability: the role of user interfaces in an activity systems, Australasian Journal of Information Systems, Vol.11, No 1, 2003.

[VRO03] Van Roy, P., and Haridi, S. Concepts, Techniques, and Models of Computer Programming, MIT Press, Cambridge, 2004.

[WAG08] Wagelaar, D. Platform Ontologies for the Model-Driven Architecture, Thesis, 2008.

[WEIN10] Weingarten, F., Blumendorf, M., and Albayrak, S. Towards multimodal interaction in smart home environments: the home operating system, In Proceedings of the 8th ACM Conference on Designing Interactive Systems (DIS '10), ACM, New York, NY, USA, pp. 430-433, 2010. DOI=10.1145/1858171.1858255 http://doi.acm.org/10.1145/1858171.1858255

[WEIS99] Weiser, M. The computer for the 21st century. SIGMOBILE Mob. Comput. Commun. Rev. 3, pp. 3-11, July 3, 1999. DOI=10.1145/329124.329126 http://doi.acm.org/10.1145/329124.329126

[WEIS03] Weiser, M. The Computer for the Twenty-First Century, Scientific American, Vol. 265, No. 3., ACM Press, New York, pp. 94-104, 2003.

[WOL10] Wolff, A., and Forbrig, P. Model-driven User Interface Development with the Eclipse Modeling Project, In Proceedings of MDDAUI2010 & CHI2010, pp. 49-52, 2010.

[WUR09] Wurdel, M., Burghardt, C., and Forbrig, P. Making Task Modeling Suitable for Smart Environments, 2009.

[Xbox SmartGlass] Xbox SmartGlass, http://www.xbox.com/smartglass, Last visited on January 11, 2015.

[XIA09] Xiaojun, B., and Balakrishnan, R. Comparing Usage of a Large High-Resolution Display to Single or Dual Desktop Displays for Daily Work, In Proceedings of the 27th International Conference on Human

factors in Computing Systems CHI'09 (Boston, USA), ACM Press, New York, pp. 1005-1014, April 4-9, 2009.

[YAN08] Yanagida, T., and Nonaka, H. Architecture for Migratory Adaptive User Interfaces, In Proceedings of the 8th IEEE International Conference on Computer and Information Technology CIT'2008 (Sydney, Australia), pp. 450-455, July 8-11, 2008.

[ZAI06] Zaidenberg, S., Brdiczka, O., Reignier, P., and Crowley, J.L. Learning context models for the recognition of scenarios, FIP International Federation for Information Processing of Artificial Intelligence Applications and Innovations, pp. 86-97, 2006.

[ZHA10] Zhang, G. Aspect-Oriented UI Modeling with State Machines, In Proceedings of MDDAUI2010 & CHI2010, pp. 45-48, 2010.

[ZOL11] Zöllner, M., Jetter, H.-C., Reiterer, H., ZOIL: A Design Paradigm and Software Framework for Post-WIMP Distributed User Interfaces, In Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Human-Computer Interaction Series, Springer-Verlag, London, pp. 87-94, 2011. DOI 10.1007/978-1-4471-2271-5_10