T E C H N I C A L
R E P O R T

TR-UH-EDM-0603

USING DEVICE FEDERATIONS TO ENHANCE THE CREATIVE PROCESS IN A DISTRIBUTED INTERACTION ENVIRONMENT

Tom Van Laerhoven, Geert Vanderhulst, Kris Luyten,
Frank Van Reeth and Karin Coninx
{tom.vanlaerhoven, geert.vanderhulst, kris.luyten, frank.vanreeth, karin.coninx}@uhasselt.be

Hasselt University - Expertise Centre for Digital Media
and
transnationale Universiteit Limburg
Wetenschapspark 2
BE-3590 Diepenbeek, Belgium
Tel. +32 (0) 11.26.84.11 Fax. +32 (0) 11.26.84.00
http://www.edm.uhasselt.be/

Using device federations to enhance the creative process in a distributed interaction environment

Tom Van Laerhoven, Geert Vanderhulst, Kris Luyten,
Frank Van Reeth and Karin Coninx
{tom.vanlaerhoven, geert.vanderhulst, kris.luyten, frank.vanreeth, karin.coninx}@uhasselt.be

Abstract

In this work, we examine how the advantages of a digital painting application can be supplemented by distributing its elements among several devices that participate in a device federation.

The paint system we describe is a proof-of-concept for an architecture that facilitates the creation of a distributed interaction space. The architecture handles end-user device discovery, distribution of user interfaces, coordination of actions and propagation of events within the interaction environment. Central to this approach is a web-based middleware layer built on the *Web To Peer* (W2P) framework, a custom framework that supports user interface distribution for heterogeneous environments.

The motivation for developing a distributed painting application is to offer the artist a more natural environment so that digital painting resembles a traditional painting process. We accomplish this goal by creating a computer-enhanced paint environment that closely matches the real world painting process, incorporating devices such as an interactive whiteboard, a graphics tablet interface, a tablet PC and PDAs.

keywords I.3.4 [Graphics Utilities]: Paint systems, H.5.2 [User Interfaces]: Input devices and strategies, C.2.4 [Distributed Systems]: Distributed applications, C.5.3 [Microcomputers]: Portable devices,

1 Introduction

The domain of human computer interaction (HCI) explores how users interact with computers. Consider for instance the case of the "digital painter". In this case, a painter is equipped with a range of computer devices to create her/his artwork. In order to guarantee fluent interaction, the digital painter's actions should match those of the "real world painter". The latter typically uses the following attributes: a set of paint brushes, a palette to select and mix pigments and a painting canvas to draw on. However, a desktop computer with monitor, mouse and keyboard can hardly provide these attributes in a usable manner. A combination of pen based hardware, portable devices and touch screen interaction is better suited to support the digital painter's actions.

With new emerging communication technologies (such as Web services) becoming available on a wide diversity of devices, there is an opportunity to exploit the potential of interconnected devices in the user's surroundings to create a *distributed interaction space* [16]. The federation of heterogeneous devices and resources involved in the interaction space operates as one logical whole when allocated to a certain task, e.g. create a painting. An interactive environment focused on a painting task can accommodate a single user (*personal interaction space*) as well as a collection of users working cooperatively on a painting (*collaborative interaction space*).

1.1 Contributions

In our work, we investigate how various paint components (palette, canvas, ...) can be distributed over heterogeneous computing devices to support the traditional actions of a painter in a 3D environment. We present an interactive paint environment together with specialized software for discovering devices, distributing user interfaces, coordinating actions and propagating events among the interaction space. This computer-augmented environment allows for personal and collaborative paint tasks, regardless whether users (and their device federations) are collocated or geographically dispersed.

1.2 Overview

The remainder of this document is organized as follows. Section 2 discusses some previous and ongoing work in related domains. In section 3 we present an architecture for user interface distribution. We evaluated our framework by means of a case study focused on a ubiquitous painting environment which is outlined in section 4. Section 5 summarizes the results and experiences learnt from this case study and section 6 finally draws a conclusion.

2 Related work

This section gives an overview of the relevant related work that has been done in the different domains encompassed in this text. The combination of distributed user interfaces (DUI) that work on top of a service-oriented architecture (SOA) makes up the basis for the solution we developed. The applicability of our approach for software that supports a creative process is shown by developing a painting application using this basis. Sections 2.1 and 2.2 respectively discuss DUIs and SOAs. Section 2.3 discusses relevant background on painting applications. Although we will discuss only this one example application in this text, we will emphasize the generic approach during the remainder of the text.

2.1 Distributed User Interfaces

Distributable user interfaces are user interfaces that can be split up in parts that are migrated to different interconnected (and embedded) devices in the environment of the end-user that cooperate to offer functionality to ease the tasks of the user [23, 6, 16]. The distribution process presented in [27] relies on an XML schema language to guide the migration of user interface description documents to a federation of devices. This

work focuses on the schema-driven generation and distribution of Web-based interfaces, but the methods discussed can be generalized to generic XML-based interface markup languages. Distributable user interfaces can be split up in parts that are migrated to different interconnected (and embedded) devices in the environment of the end-user. These cooperate to offer functionality to ease the tasks of the user [24, 7].

Bandelloni and Paternò have shown that a web interface can be partially or completely migrated [2]. Here, partial migration implies the web interface is split up in two or more parts that each run on a separate device. This is accomplished by exploiting information about the interactive system that is available, and by using a flexible language to describe the interface presentation. The former relies on the models that describe an interactive system [20], while the latter on a specific user interface description language [4]. Recent approaches [2, 20, 4, 19] have shown that a combination of XML-based user interface description languages (UIDLs) [15] and model-based interface design are best suited to adapt the user interface for new contexts of use.

Larsson and Berglund identify a set of new requirements for designing distributed interfaces [14]. A model-based design approach seems appropriate to realize this [26]. This, however, requires specialized knowledge about the different models that are used to accomplish this, and considers distribution mostly on a single level of abstraction.

The distribution of an interface among several locations can also be observed on traditional desktop systems, where multiple displays can be used to spread the user interface of a single application. Several researchers presented work that goes beyond simple multi-display systems [11, 21]. A virtual cohesive interaction space is created where different surfaces are connected to each other to create one logical display surface.

In previous work we presented a framework to migrate user interface from one device to another device while switching modality [17, 18]. Although we aimed at a generic framework, this work also shows the feasibility to use another device with another modality to access the same type of functionality. Depending on the capabilities of the interaction device

2.2 Service-oriented Architectures

Several technologies have been proposed for creating device-level service-oriented architectures, most notably UPnP [1] and Jini [29]. In these systems (mobile) devices and resources are discovered in the local subnet together with the services they offer. Using UPnP or Jini communication middleware, heterogeneous software entities can exchange action and event messages.

Web-based service-oriented environments are gaining in popularity due to the openness of Web standards and their emerging support on all kinds of devices. [22] and [28] study Web-based service discovery and invocation on mobile devices. When dealing with low-bandwidth or intermitted connections, it may also be beneficial to compact Web service messages and transfer them in compressed format rather then in plain XML [5]. A generalized middleware framework for the Web can serve as a foundation to integrate such developments.

The Web Service Invocation framework (WSIF) [9] is a Java-based initiative towards uniform middleware. Instead of focusing on a specific protocol (e.g. SOAP) it focuses on a high-level description language for Web services. This enables services

to be invoked in a consistent way regardless of how they are implemented or accessed.

2.3 Painting Applications

There has been considerable work in interactive computer painting, but only a few efforts specifically focus on a natural interaction with the software.

Baxter *et al.* adopt in their system for creating images viscous paint a minimalistic interface that entirely depends on a PHANToM haptic feedback device [3]. The force feedback enhances the sense of realism and enables the user to manipulate a virtual brush model in an intuitive way, creating a setting that is conceptually equivalent to a real-world painting environment.

The "CavePainting" system [12] allows 3D painting in an immersive CAVE environment. The environment makes it possible to paint while standing up and walking around, and is very similar to how a painter works on a large canvas. 3D brush strokes are created by tracking the position and orientation of physical props, such as a paint brush or a bucket. In addition, a pinch glove worn on the non-dominant hand modifies brush attributes like size and color.

A related approach for drawing in 3D space, the "HoloSketch" application, adopts head-tracked stereo shutter glasses and a CRT display in combination with a 6DOF hand input device [8].

"CoolPaint" also relies on props-based interaction [13]. In this case, real paint brushes instrumented with 6DOF trackers allow direct interaction on a tabletop display surface.

Although the approaches discussed in this section have some similarities to the system presented in section 4, we focus on interaction using a combination of different devices and setups, and additionally, the collaboration of several users working on the same artwork.

2.4 Relation with Our Work

Our work builds on the aforementioned schema-driven interface distribution approach and leverages Web service technology to establish communication between distributed service parts. These technologies are applied to create a distributed interaction space in which a ubiquitous paint application is deployed.

3 An Architecture for Dynamic User Interface Distribution

This section gives an overview of the architectural elements in our paint environment. Our implementation is based on the following software components:

- The services that are distributed to heterogeneous devices in the user's surroundings;
- A distribution handler responsible for the actual distribution of service interfaces;
- Middleware for transparent two-way communication between software entities.

These parts are shown in figure 1 and discussed in the next sections.

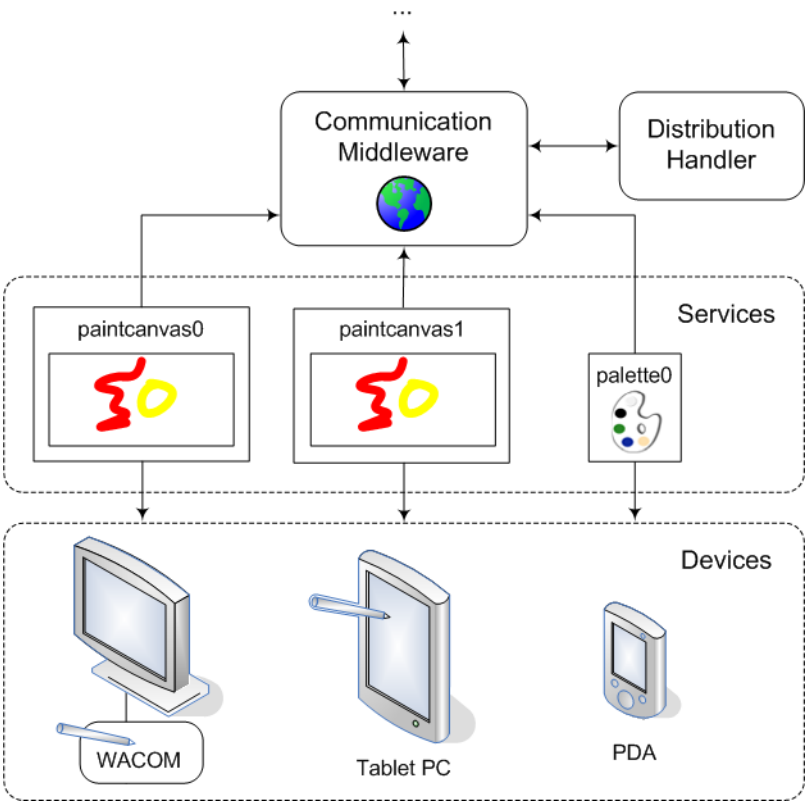


Figure 1: Architecture

3.1 Service User Interfaces

A service encapsulates application logic that can be accessed remotely by a service user interface. We refer to the service application logic as the *service core* and the service user interface as the *service interface* in the remainder of this text.

Service interfaces can be serialized to target devices in high-level XML description documents. These documents can be transformed into XHTML (suitable for rendering in a Web browser) or into another UIDL such as UIML. Since a user interface can be described in a platform-independent way, descriptions are useful to overcome the heterogeneity of devices. They avoid the need to manually fabricate an interface for individual platforms. The quality and usability of the interfaces depends on the used renderer and/or the adaptation of the interface description to the target platform. The approach we present does not depend on the usage of a UIDL however. In many cases, the user interface is available beforehand and there is no need for migrating or dynamically rendering it on a new device. Both "precompiled" and rendered user interfaces are supported.

3.2 Distribution Handler

The task of the distribution handler is to migrate service interfaces to devices available in the environment of the user. Three different types of possible user interface distribu-

tions are identified: user-driven distribution, system-driven distribution and continuous distribution.

User-driven distribution relies on the initiative of the user: the user manually indicates the service user interfaces that are to be visualized on each device in her/his personal interaction space. In many cases user-driven distribution is preferred since the end-user has full control over the distribution of the interface.

System-driven distribution enables the system to decide what services are migrated to what client devices. The user only selects the application she/he wants to interact with and possibly the preferred interface description format. The system-driven distribution algorithm tries to perform an optimal distribution of services among the available clients. Therefore the algorithm needs additional information about the devices, the services, and the user's preferences. This information is stored and exchanged in *profiles*.

Continuous-driven distribution implies that changes in her/his interaction space may trigger dynamic changes in the distribution of the user interface. Two main causes of environment changes can be distinguished in this context: client devices entering or leaving the interaction space. It should be noted that continuous distribution in an interaction space implies some usability issues. This has to do with the fact that the user interface changes while the user is interacting with it. This may be very frustrating for the user when client devices enter and leave the environment all the time.

User-driven and system-driven distribution are relatively easy to accomplish once we have a description of the user's environment including the various devices that are available. In the work presented in this text we do not consider continuous-driven distribution: given the goals we try to accomplish this lies outside the scope of the text.

How to distribute a user interface is also dependent on whether the user knows beforehand the type of application that will be used. In the case we present in section 4, part of the application logic will be distributed beforehand and can not be migrated while using the system: this implies a constraint in the distributed system configurations that can be supported. A service interface can be tied to a specific device because it relies on a device-specific capability. In the type of distributed application we implemented, a paint application, the distribution of the application will be mainly user-driven. It depends on the preferences of the artist using the application, but it also depends on the goals of the user, e.g. whether collaboration with others is envisaged or whether detailed input needs to be provided.

3.3 Messaging Middleware

In order to build distributed applications, a communication protocol is required to let distinct components talk with each other. This communication protocol is often implemented in a middleware layer along with one or more transfer protocols. The middleware takes care of data serialization and transfer over the network to another component where the data is deserialized. As is common in several distributed system

architectures, the middleware entirely takes care of this [10]. Distributed application components can use the middleware API to communicate with other remote components and do not need to know how the data is being transferred.

In our environment, communication between the service interfaces and service cores, both defined in section 3.1, is envisaged. Since both the service interfaces and service cores can be distributed over different computing entities, fairly complex message schema's can be necessary. A service interface can be distributed among different interaction devices, the different parts of the service interface that make up one logical whole together are referred to as user interface parts (UIPs) in the remainder of this text. A service core is distributed as a set of application logic components (ALCs); together they define the behavior of the system and provide the functionality used by the service interfaces. We use the term "entity" if we want to refer to both ALCs and UIPs.

Actions triggered in a UIP should reach the appropriate ALCs. For this purpose an event subscription mechanism is used that makes sure events fired in a particular UIP are propagated to the appropriate listeners. We developed a messaging framework that can be embedded in a middleware layer and provides an open interface by using web standards: *Web To Peer* (W2P).

W2P is an open framework in which communication and transport protocols can be plugged in as independent modules. In our setup, an HTTP transport module is used to exchange messages between distributed components. These components are interconnected via a *W2P service* (W2PS). The W2PS is hosted by a HTTP based servlet container like Tomcat[1] or Jetty[2]. It acts as a message gateway with two-way communication support. The distributed ALCs can use the W2PS to send messages to each other and behave as one functional whole. An ALC does not need to run a local server process, but use a connection to the W2PS that serves as a message bus for the complete system.

3.4 Ubiquitous Message Exchange

A user interacts in its own *service space*: a service space embodies all the services that are available for a user by using the interaction devices available in the environment. Notice a service space can evolve over time: services can appear and disappear. We need a way to identify the available services so they can be used by the software entities participating in a distributed application. When a software entity logs in on the W2P service, a naming service provides it with a unique name w.r.t. the service space. This is either a reserved name or an enumerated name (e.g. `paintcanvas0`, `paintcanvas1`, ...) by which entities can address each other. A reserved name is statically appointed to one particular software entity, often used for software entities that are known to participate in the distributed application. An enumerated name allows new software entities to join a service space without interfering with existing software entities. In practice this has shown to be a simple but effective approach: most distributed applications have a predefined set of software entities that make up the core of the application and live during the complete application's lifetime.

[1] `http://tomcat.apache.org/`
[2] `http://jetty.mortbay.org/`

The W2P service allows to address a group of entities at once. Entities can join or leave groups, which are created on the fly. For instance, a number of similar entities with assigned names `paintcanvas0`, `paintcanvas1`, ... can subscribe themselves to a paintcanvas group. ALCs can then address a message to `@paintcanvas` and use it as a multicast address that forwards the message to each member of the paintcanvas group. The naming service also supports queries to retrieve the composition of a specific group.

Entities communicate with each other by exchanging messages. All messages pass via the W2P service, which routes them to their destination(s). The W2PS uses a message queue that can be compared with traditional message-oriented middleware (MOM) systems. Since messages are buffered in the queue, the MOM can regulate the message flow and protect entities from being flooded. In a highly interactive system messages should be ordered in such a way the latency between input from the user and the related update in the user interface is minimized. We will show our message distributed mechanism is usable for a highly interactive environment in section 4.

W2P messages are designed to carry any kind of information: plain text, XML data, binary files (e.g. images), etc. This information is stored in the message's payload or in parameters and attachments connected with the message. Messages are either delivered asynchronously (one-way, non-blocking) or synchronously (two-way, blocking). Furthermore, a synchronous request/reply mechanism is implemented: when an entity receives a message, it replies with a message to the sender.

The messages exchanged by the distributed entities in our paint environment are XML-based. Figure 2 shows some example messages where a palette and brush movement event are captured in XML. Notice there is no generic schema, but the schema of XML messages is specific for a service. The message schema from a software entity that sends a message can be used for validation by the receiving software entity.

One of the most important benefits of our approach is the ubiquitous nature of the proposed solution. Vandervelpen *et al.* showed that the combination of XML-messaging with a web-based middleware can be used in most environments [27]. It provides an open interface and is independent of any software platform or device.

4 A Case Study: A Natural Interaction Environment for Painting

In this section, we investigate how a painter can benefit from a distributed interaction space. The heterogeneous nature of several devices is exploited to facilitate the artistic process, ensuring that the artist is by no means restrained in his/her expressiveness. This is an important consideration, because the procedure of real-world painting differs somewhat from how painters accomplish the same results using digital tools. The mismatch comes from limitations in the software that is used, as well as the supporting hardware. Mainstream painting systems restrict the user to work individually at a desktop system, equipped with commodity input hardware like a tablet interface. Painting software that is adapted for use in a distributed interaction space has the ability to overcome these limitations. Moreover, it can lever the painting experience to a whole new level, with numerous users collaborating in the creation of an artwork. This is hardly practicable in real life.

```
<?xml version="1.0"?>
<paletteEvent>
  <pigmentMix>
    <pigment index="1" frac="0.5"/>
    <pigment index="3" frac="0.5"/>
  </pigmentMix>
</paletteEvent>
```

```
<?xml version="1.0"?>
<brushMovementEvent t="begin_stroke">
  <pigmentMix>
    <pigment index="1" frac="0.5"/>
    <pigment index="3" frac="0.5"/>
  </pigmentMix>
  <x>20</x>
  <y>30</y>
  <tiltx>5</tiltx>
  <tilty>10</tilty>
  <pressure>0.8</pressure>
</brushMovementEvent>
```

Figure 2: XML messages that are exchanged between distributed parts.

The software that is used throughout this section adopts programmable graphics hardware to simulate watery paint like watercolor and Oriental ink. It relies on physically-based algorithms to recreate the intrinsic complex behavior of such media [25].

In the next section we first explore the approach of a painter that creates an artwork in the traditional way. The subsequent section maps the resulting observations to the digital world.

4.1 Observing a Real-World Painter

Although there are no strict guidelines that tell us how a painting should be made, it is possible to make several observations on a painter's modus operandi.

The traditional painting equipment includes a set of brushes, the canvas and a palette with different pigments. The palette is a wooden or plastic board on which the artist mixes the paints that are used in the artwork. Probably the most recognizable setup used in the process of painting places and secures the canvas onto an easel. The painter holds the palette in one hand, and the brush – or whatever tool is used to manipulate the paint – in the other. This practice has the advantage that the painter can periodically step back and evaluate the work in progress from a distance or from a different perspective.

Another commonly used technique requires the painter to sit down, holding a wooden board on which the canvas is firmly attached in one hand, while the brush

is held in the other. In this case, the palette rests within reach on a table.

4.2 A Taxonomy of Input and Output Devices

In trying to recreate the experience of real-world painting, we want to provide a painter with an intuitive and familiar working environment, while exploiting the flexibility of the digital world. Several devices can be used to support an artist in creating paintings:

- SMART board interactive whiteboard,
- Wacom tablet interface,
- Tablet PC,
- PDA;

User interaction involving the SMART board closely matches the first strategy we described in the previous section. The whiteboard corresponds to the easel on which a large canvas is positioned. It is mounted to the wall and provides touch-sensitive input from a SMART pen that acts as a brush metaphor. The pen has just two degrees of freedom (DOF), so it can only track the brush's 2D position. Front projection is used to map the current working environment on the whiteboard.

The tablet interface and the tablet PC are very similar in usage. Both require a special pen or stylus that registers 2D position and optionally pressure and orientation information, which is used to control the virtual brush. The tablet PC contains an embedded screen showing the canvas, while the tablet interface requires an external canvas representation in the form of a computer screen or whiteboard projection. This way of working maps well to the second painting technique outlined in the previous section.

Finally, a PDA equipped with wireless connection embodies very portable processing capabilities, but has limited screen size and likewise computing power. It is therefore unsuitable as a means of canvas representation. However, just as a real-world painter can reserve one hand for holding the palette and mixing colors, it would be useful to assign this task to the PDA. This approach is also advantageous because it makes more viewing space available on the device that is presenting the canvas.

4.3 Description of Usage

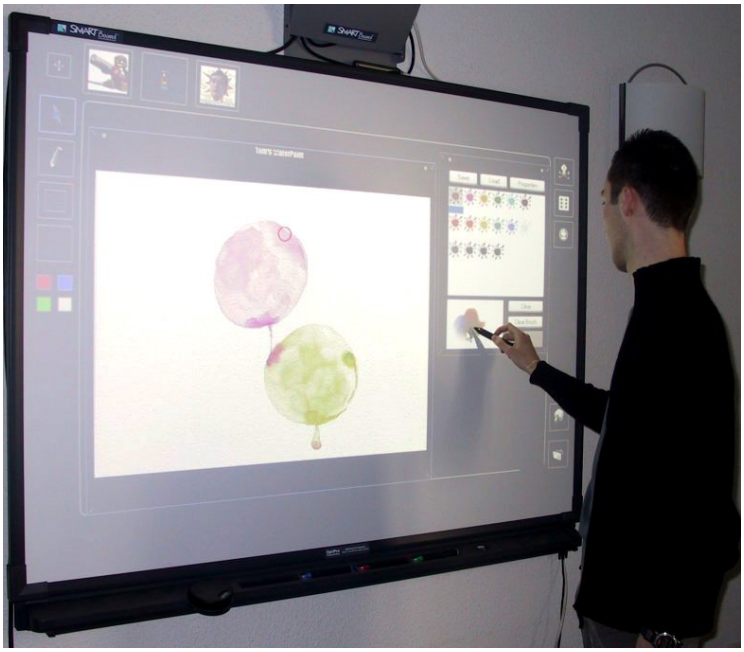
The scene depicted in figure 3 shows a user working on a painting using the SMART board. Both palette and canvas components are projected onto the whiteboard. With the SMART pen, the user can prepare a paint mix (figure 3(a)) and use it to apply strokes on the canvas (figure 3(b)). As the pen provides just positional information, the strokes are formed as a trail of circular markings.

Alternatively, the palette service can be migrated to the PDA device in the user's left hand. The user is then able to mix paint on the PDA (figure 4(a)) and apply paint on the canvas in the same way as before. However, as more space is now available on the whiteboard, the canvas can be enlarged so that the user is able to concentrate on the finer details of the artwork. Also, the palette is always within reach, making the painting process more efficient (figure 4(b)).

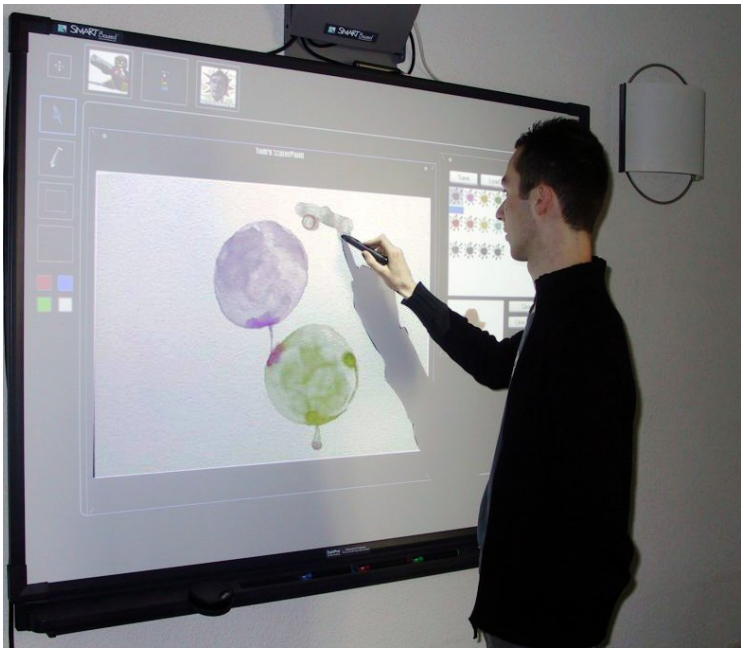
A different setup is shown in figure 5(a), where the painter sits at a desktop PC with a Wacom tablet interface, which also participates in the device federation. The extra capabilities of the 5DOF tablet interface – orientation and pressure level – are adopted to manipulate a 3D virtual deformable brush that is able to create complex, more realistically shaped strokes.

Finally, the setting in figure 5(b) demonstrates how two users are cooperatively creating an image by sharing the same canvas, but shown on different devices. One user is currently operating the tablet interface. In the meantime, the user at the whiteboard is preparing a paint mix with the mobile palette.

The system sequence diagrams in figure 6 and 7 show the message flow between the different system components when an artist interacts with the palette and canvas interfaces respectively. Notice the service interfaces use the W2PS as proxy for the actual service cores that need to be used. The W2PS acts as a central bus mechanism embedded in a middleware layer.

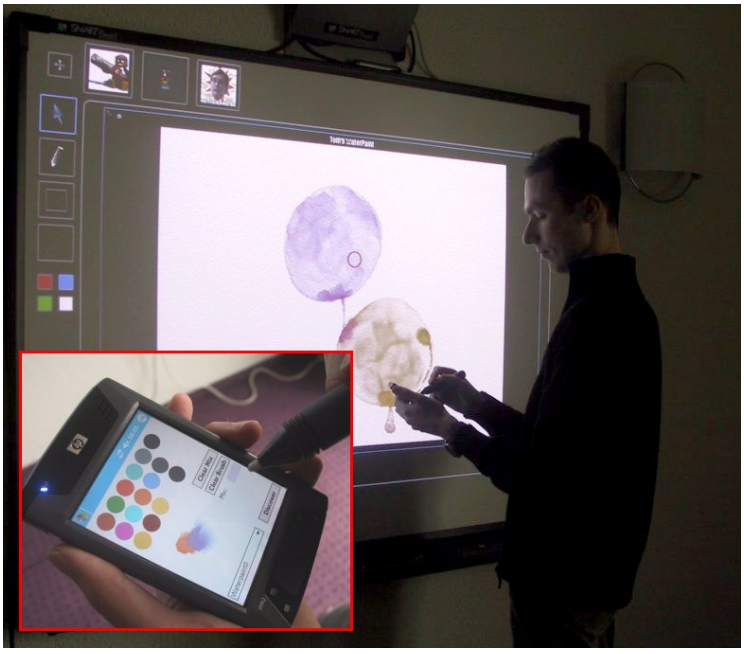


(a)

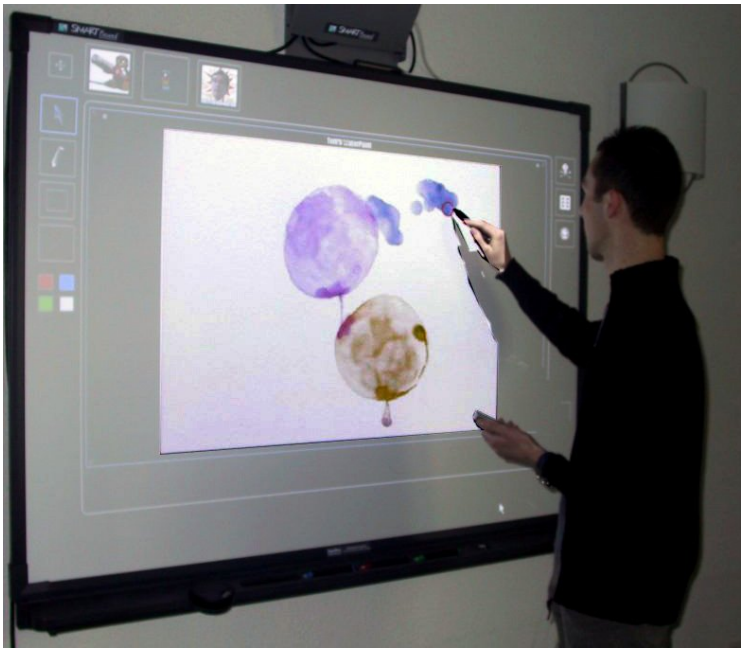


(b)

Figure 3: A user painting on the SMART board. The virtual palette is adopted to prepare a paint mix (*a*), which is then used to draw strokes on the canvas (*b*).

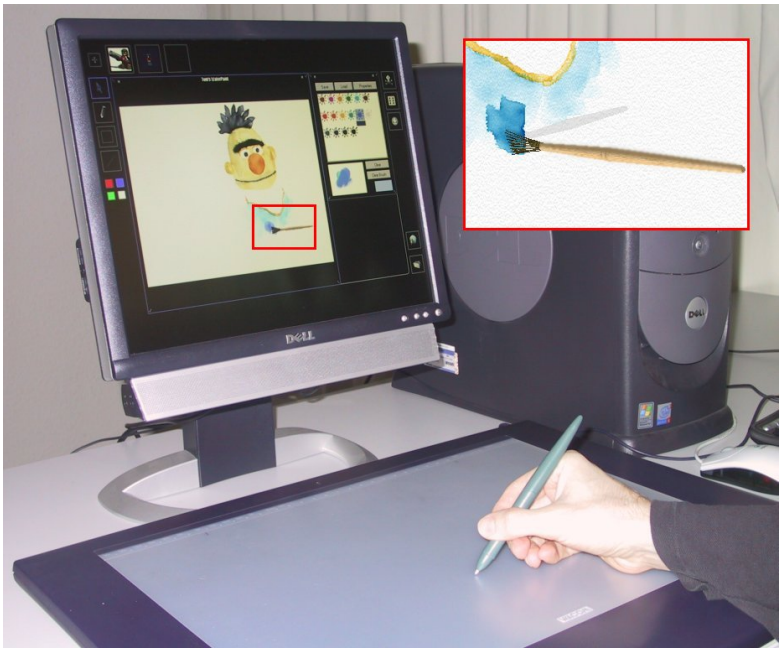


(a)

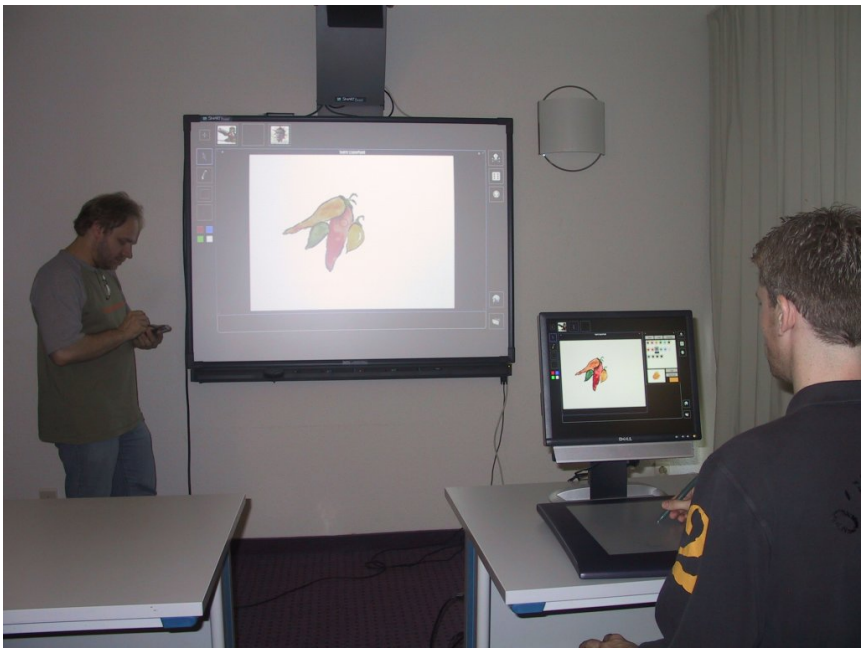


(b)

Figure 4: A user painting on the SMART board, while holding a PDA that contains the virtual palette.



(a)



(b)

Figure 5: A painting in progress, using a desktop PC with a Wacom tablet interface (*a*), and two users collaborating in the creation of a single painting by means of several devices (*b*).

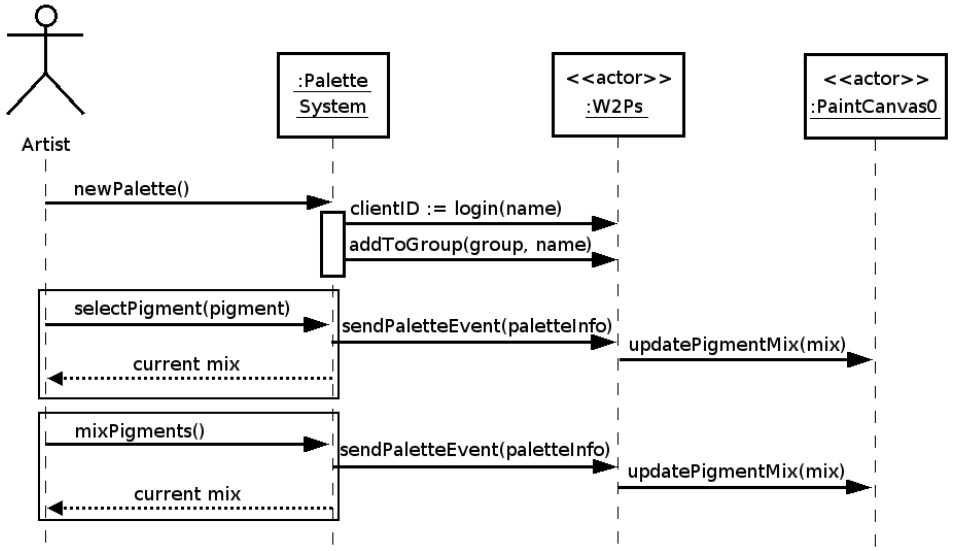


Figure 6: System sequence diagram (SSD) describing the actions of an artist interacting with a palette.

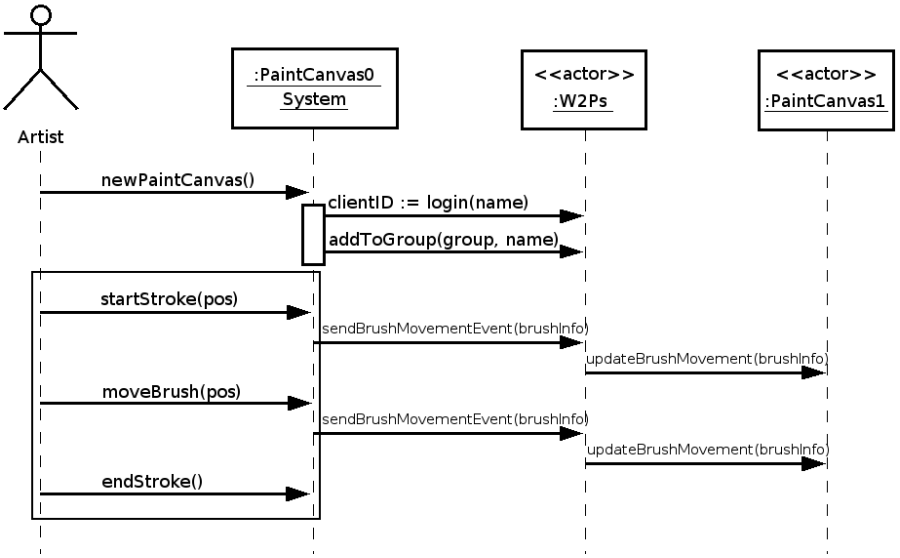


Figure 7: System sequence diagram (SSD) describing the actions of an artist interacting with a canvas.

5 Evaluation

This section reflects on the case study outlined in the previous section and discusses some major foundations. We can subdivide results originating from a personal interaction space and observations arising from a collaborative context. In the former case, a single artist creates a digital artwork and in the collaborative case, a number of artists work together on a shared painting canvas.

When observing an artist's painting habits (both in a traditional and digital environment), we noticed that the adoption of personal preferences is important in creating natural interaction with the system. Therefore we let users create customized palette presets, e.g. with skin pigments for portraits and nature colors for landscape paintings. Furthermore, an artist is able to guide the migration of services to devices of their choice and experiment with various combinations.

The palette service interface can be migrated to a PDA held in the non-dominant hand or even to a larger tablet PC placed on a table next to a Wacom tablet device. The choice of standing in front of the whiteboard or sitting down behind a desktop monitor or tablet PC is something that is common in practice and supported by our system.

The service interfaces in the interaction space of the distributed paint application can also be changed according to the user goals (as mentioned in section 3.2). This is a typical example of a user-driven distribution. An artist can draw some rough strokes or broad washes on the SMART board and then finalize the painting using a tablet interface. The Wacom tablet interface results in the most detailed artwork thanks to its accurate orientation and pressure registration, but interacts with the same service cores as the SMART board interface.

A prototype of the system was positively evaluated by an artist. While working at the whiteboard, the preparation of a paint mix on a mobile palette was reported as being especially favorable because it felt familiar with the real-life practice of working in front of an easel.

6 Conclusions

The benefits of digital paint systems are profound: digital paint is inexhaustible, should not dry and artists can digitally load, save, transfer and manipulate their artwork. Moreover, the digital world can provide users with collaboration features which are hardly practicable in a traditional paint environment.

In this document we have illustrated the use of heterogeneous device federations to support a painter's actions in a computer-augmented environment. The migration of service user interfaces to devices in the user's surroundings can help to optimally integrate these devices in the paint environment. We combined a schema-driven distribution approach and Web-based technology in a framework that serves as middleware layer.

Future Directions

Additional research can go into various aspects of the paint environment and contribute to an even more enriched paint experience.

We already stated in a previous section that during a cooperative painting session the communication between collaborating parties should be emphasized in order to avoid confusion. This can be done using, for example, voice traffic, unambiguous representation of a user's presence and behavior in the environment, or some sort of floor control mechanism.

It may also be worthwhile to increase the intelligence of the distributed paint system. The system could for instance learn from a user's habits and use this information to prepare a paint session and migrate user interfaces to preferred devices in advance.

Finally, additional devices could be incorporated in the device federation. The PHANToM haptic feedback device, for example, can further enhance the realism of the painting experience by providing the user with a brush's reactive forces to deformations.

Acknowledgments

Our gratitude goes to Koen Beets and Bjorn Geuns for their aid in taking pictures of the setup, and Karel Robert for evaluating the setup.

References

[1] *Universal Plug and Play (UPnP)*. `http://www.upnp.org/resources/`, 2006. 2.2

[2] Renata Bandelloni and Fabio Paternò. Flexible Interface Migration. In *Proceedings of Intelligent User Interface 2004 (IUI 04)*, pages 148–155, 2004. 2.1

[3] William V. Baxter, Vincent Scheib, Ming C. Lin, and Dinesh Manocha. dAb: interactive haptic painting with 3D virtual brushes. In Eugene Fiume, editor, *Proceedings of ACM SIGGRAPH 2001*, pages 461–468. ACM Press, NY, USA, August 2001. 2.3

[4] Silvia Berti, Francesco Correanim, Fabio Paternò, and Carmen Santoro. The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction Levels. In Luyten et al. [15], pages 103–110. 2.1

[5] Min Cai, Shahram Ghandeharizadeh, Rolfe R. Schmidt, and Saihong Song. A Comparison of Alternative Encoding Mechanisms for Web Services. In *DEXA '02: Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 93–102, London, UK, 2002. Springer-Verlag. 2.2

[6] Joëlle Coutaz, Lionel Balme, Christophe Lachenal, and Nicolas Barralon. Software Infrastructure for Distributed Migratable User Interfaces. *UbiHCISys 03 Workshop on UbiComp 2003*, 2003. 2.1

[7] Joëlle Coutaz, Lionel Balme, Christophe Lachenal, and Nicolas Barralon. Software Infrastructure for Distributed Migratable User Interfaces. *UbiHCISys Workshop on UbiComp 2003*, 2003. 2.1

[8] Michael F. Deering. HoloSketch: A Virtual Reality Sketching/Animation Tool. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 2:3:220–238, 1995. 2.3

[9] M. Duflter, N. Aleksander, S. Sanjiva, W. IBM, and T. Research. *Web Service Invocation Framework (WSIF)*. 2006. 2.2

[10] George F. Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed systems: concepts and design*. Addison-Wesley, Boston, MA, USA, 2001. 3.3

[11] Ken Hinckley, Gonzalo Ramos, Francois Guimbretiere, Patrick Baudisch, and Marc Smith. Stitching: Pen Gestures that span Multiple Displays. In *Proceedings of Advanced Visual Interfaces 2004 (AVI 04)*, pages 23–31, 2004. 2.1

[12] Daniel F. Keefe, Daniel Acevedo Feliz, Tomer Moscovich, David H. Laidlaw, and Joseph J. LaViola Jr. CavePainting: a fully immersive 3D artistic medium and interactive experience. In *Symposium on Interactive 3D Graphics*, pages 85–93, 2001. 2.3

[13] Dustin Lang, Leih Findlater, and Michael Shaver. CoolPaint: Direct Interaction Painting. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology (UIST)*, Vancouver, Canada, 2003. Poster. 2.3

[14] Anders Larsson and Erik Berglund. Programming ubiquitous software applications: requirments for distributed user interface. In *Proceedings of Software Engineering and Knowledge Engineering 2004 (SEKE 04)*, 2004. 2.1

[15] Kris Luyten, Marc Abrams, Quentin Limbourg, and Jean Vanderdonckt, editors. *Developing User Interfaces with XML: Advances on User Interface Description Languages*, Gallipoli, Italy, 2004. Sattelite workshop of Advanced Visual Interfaces (AVI) 2004, Expertise Centre for Digital Media. 2.1, 4

[16] Kris Luyten and Karin Coninx. Distributed User Interface Elements to support Smart Interaction Spaces. In *Seventh IEEE International Symposium on Multimedia*, pages 277–286, December 2005. invited paper. 1, 2.1

[17] Kris Luyten, Tom Van Laerhoven, Karin Coninx, and Frank Van Reeth. *Computer-Aided Design of User Interfaces*, volume 3, chapter Specifying User Interfaces for Runtime Modal Independent Migration, pages 283–294. Kluwer Academic, 2002. 2.1

[18] Kris Luyten, Tom Van Laerhoven, Karin Coninx, and Frank Van Reeth. Runtime transformations for modal independent user interface migration. *Interacting with Computers*, 15(3):329–347, 2003. 2.1

[19] Guido Menkhaus and Sebastian Fischmeister. Dialog model clustering for user interface adaptation. In *Proceedings of the International Conference on Web*

Engineering 2003 (ICWE 03), volume 2722 of *LNCS*, pages 194 – 203. Springer Verlag, 2003. 2.1

[20] Giulo Mori, Fabio Paternò, and Carmen Santoro. Design and development of multidevice user interface through multiple logical descriptions. *Transactions on Software Engineering*, 30(8), August 2004. 2.1

[21] Jun Rekimoto and Masanori Saitoh. Augmented surfaces: A Spatially Continuous Work Space for Hybrid Computing Environments. In *Proceedings of Computer Human Interaction 1999 (CHI 99)*, 1999. 2.1

[22] Robert Steele. A Web Services-based System for Ad-hoc Mobile Application Integration. *ITCC*, 00:248, 2003. 2.2

[23] Peter Tandler, Thorsten Prante, Christian Müller-Tomfelde, Norbert Streitz, and Ralf Steinmetz. ConnecTables: Dynamic coupling of displays for the flexible creation of shared workspaces. *Proceedings of User Interface Software and Technology 2001 (UIST 01)*, pages 11–20, 2001. 2.1

[24] Peter Tandler, Thorsten Prante, Christian Müller-Tomfelde, Norbert Streitz, and Ralf Steinmetz. ConnecTables: Dynamic coupling of displays for the flexible creation of shared workspaces. *Proceedings of the 14, Annual, ACM Symposium on User Interface Software and Technology (UIST'01)*, pages 11–20, 2001. 2.1

[25] Tom Van Laerhoven and Frank Van Reeth. Real-time simulation of watery paint. In *Journal of Computer Animation and Virtual Worlds (Special Issue CASA 2005)*, volume 16:3–4, pages 429–439. J. Wiley & Sons, Ltd., October 2005. 4

[26] Chris Vandervelpen and Karin Coninx. Towards model-based design support for distributed user interfaces. In *Proceedings of Nordic Conference on Human-Computer Interaction 2004 (NordiCHI 04)*, pages 61–70. ACM Press, 2004. 2.1

[27] Chris Vandervelpen, Geert Vanderhulst, Kris Luyten, and Karin Coninx. Lightweight Distributed Web Interfaces: Preparing the Web for Heterogeneous Environments. In *Proceedings of the International Conference on Web Engineering 2005 (ICWE 05)*, September 2005. 2.1, 3.4

[28] Matthias Wagner, Thorsten Liebig, Olaf Noppens, Steffen Balzer, and Wolfgang Kellerer. Towards Semantic-based Service Discovery on Tiny Mobile Devices. 2004. 2.2

[29] Jim Waldo. *The Jini Specifications*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000. 2.2