Providing Plasticity and Redistribution for 3D User Interfaces using the D3PART Model

Jérémy Lacoche, Thierry Duval, Bruno Arnaldi, Éric Maisel, Jérôme Royan

Noname manuscript No.
(will be inserted by the editor)

Providing Plasticity and Redistribution for 3D User Interfaces using the D3PART Model

Jérémy Lacoche · Thierry Duval · Bruno Arnaldi · Eric Maisel · Jérôme Royan

Received: date / Accepted: date

Abstract We propose D3PART (Dynamic 3D Plastic And Redistribuable Technology), a model to handle redistribution for 3D user interfaces. Redistribution consists in changing the components distribution of an interactive system across different dimensions such as, platform, display and user. Our work is based on previous models for the creation of 3D plastic user interfaces, interactive systems that can handle the context of use modifications while preserving usability. In our previous work, we proposed a task model, a device model and an application component model for the creation of plastic user interfaces that handle the 3D specificities. With D3PART, we extend these models in order to include redistribution capabilities. The final solution lets developers create applications where 3D content and tasks can be automatically redistributed across the different dimensions at runtime. The proposed redistribution process is based on a client-server architecture with a meta-user interface to control the redistribution configuration. In order to illustrate D3PART, we describe three different scenarios of redistribution between a tablet and a CAVE for a 3D application for furniture planning. Indeed, with an implementation of our model, we show how redistribution can be used at runtime to combine these platforms, to switch seamlessly from one platform to another, and lastly, how redistribution can be used to create a collaborative context of use.

Keywords Plasticity, Redistribution, 3D User Interfaces, Virtual Reality

Jérémy Lacoche
IRT b<>com, UMR CNRS 6074 Irisa - Inria Rennes, France
E-mail: jeremy.lacoche@b-com.com

Thierry Duval
IRT b<>com, UMR CNRS 6285 Lab-STIC, IMT Atlantique, France
E-mail: thierry.duval@imt-atlantique.fr

Bruno Arnaldi
IRT b<>com, UMR CNRS 6074 Irisa - Inria Rennes, INSA de Rennes, France
E-mail: bruno.arnaldi@irisa.fr

Eric Maisel
IRT b<>com, UMR CNRS 6285 Lab-STIC, ENIB, France
E-mail: maisel@enib.fr

Jérôme Royan
IRT b<>com, France
E-mail: jerome.royan@b-com.com

1 Introduction

Today, users have access to a wide variety of platforms, such as mobile devices, desktop computers and immersive systems. Therefore, users are more frequently confronted with situations where they have to move from one platform to another [12]. Moreover, combining different platforms can be considered as very interesting for end users, as it gives them new interaction prospects [18]. These possibilities directly refer to ”distributed user interfaces” (DUI) and redistribution. A DUI is a user interface whose components are distributed across different dimensions, such as platforms, displays and users [13] [27]. For instance, these components can be widgets, interactors, or content. The redistribution capability of an interactive system refers to its property to change its components distribution, statically or dynamically [7]. It can include migration and replication mechanisms. Redistribution is one means of adaptation addressed by the plasticity concept that comes from 2D user interfaces. Plasticity is defined as the capacity of an interactive system to withstand variations of both the system physical characteristics and the environment, while preserving its usability [33]. Whatever the context of use: code interoperability or usability, continuity has to be guaranteed to be considered as plastic. The second means of adaptation addressed with plasticity is recasting, which consists of locally modifying the application components in order to fit a given context of use. For instance, it can be interaction techniques adaptations, or content presentation modifications. Indeed, considering recasting is needed to handle redistribution, as the input and output capacities may vary from a platform to another. Some solutions exist for the creation of reconfigurable 3D applications [14] or 3D adaptive ones [23], and some recent approaches tend to bring plasticity to 3D with a focus on recasting [16] [21]. A complete survey of plasticity for 3D user interfaces (3DUI)'s is given in [20].

Redistribution and plasticity have already been well explored for 2D user interfaces, but less for 3D. However, in the last few years interest for 3DUI's has grown. These kinds of interactive systems address Virtual Reality (VR) and Augmented Reality (AR) applications. This new trend can be explained by the improvement in graphics performance of devices, such as PC's or smartphones and also thanks to the generalization of VR and AR devices. The wide variety of existing interaction devices and the daily emergence of new ones increases the need for 3D plastic user interfaces with redistribution capabilities. In order to ease the implementation of such applications, our approach proposes to consider plasticity and redistribution for 3DUI's.

Our contribution is D3PART (Dynamic 3D Plastic And Redistribuable Technology), a new model for developers to help them in the creation of 3DUI's that can be dynamically redistributed across different dimensions: platform, user and display. The model includes a redistribution process that consists of distributing the high level tasks and the virtual environment of a 3D application across these different dimensions. The solution is based on our previous models [21] for the creation of 3D applications, independently from concrete interaction devices and 3D frameworks. D3PART also includes an adaptation process on top of these models to support dynamic recasting. It ensures that a redistributed application will fit any local context of use. At runtime, we use a client-server architecture to automatically detect new platforms and also to synchronize the different instances of a redistributed application. The redistribution process is user-initiated. Indeed, an integrated user interface is provided to the end user to enable the selection of the new distribution of the system. To illustrate our solution, we present three different scenarios of redistribution between a tablet and an immersive multi-display system, for a furniture planning application. This prototype is developed with a toolkit that implements the D3PART model. In these examples, we show how the virtual environment and the tasks can be distributed across the two platforms in order to combine them, to switch seamlessly from one platform to the other, and also to create a collaborative context of use.

This paper is an extended version of [22]. Here, we further develop the related work part, give more details about the model implementation and we give additional examples of possible uses of D3PART. This paper is structured as follows: first we review the details of the redistribution concept and we present some related work. Next, we describe the models used for the creation of plastic 3DUI's and how these models have been extended with D3PART to support redistribution. Then, we present the three examples of redistribution between a tablet and an immersive multi-display system for the furniture planning application, developed with our solution. Finally, we give some direction for future work and concluding remarks.

2 Related Work

A DUI is a user interface whose components are distributed across different dimensions [13]. For 3DUI's we consider three dimensions of distribution, from those described in [13] and [27]:

- **Display**. The application content is displayed on one or multiple devices. Common examples in 3D for this kind of distribution are multiple display systems.
- **Platforms**. The application runs on a single computing platform, or is distributed across multiple platforms. These platforms may be heterogeneous (operating system, computing power, plugged devices). For 3D applications, this category encompasses cluster approaches that combine connected homogeneous computers to run a VR application with high performance. It can also include interactive systems where the interactors of a same application are distributed across different platforms.
- **Users**. The application is shared by multiple users. This dimension is directly linked to the other two as different participants can use different displays and platforms. In 3D, this dimension directly refers to Collaborative Virtual Environments (CVE). The field of CVE includes concepts for sharing virtual worlds between different platforms and users.

Redistribution consists of changing the distribution of an interactive system on these different dimensions.

According to Demeure et al. [12], redistribution can be system-initiated (the system automatically performs the redistribution), user-initiated (the user initiates and parametrizes the redistribution), or mixed-initiated (the user and the system collaborate to perform the redistribution). Redistribution includes migration and replication mechanisms. The concept of migratory applications was firstly defined by Bharat and Cardelli [4] as applications that can migrate from one host to another, maintaining the state of their user interface intact. In the case of replication, an application is partly or fully copied from one host to another and a synchronization mechanism ensures the consistency between the different application instances. According to Calvary et al. [8], redistribution can be performed on the fly (at runtime) or between sessions and the redistribution granularity may vary from application down to the pixel level:

- At **application level**, on the platform or user dimension, the application is fully replicated or fully migrated onto a distant platform. The application may be adapted to its new context of use, which can include platform capabilities and user preferences. Full replication implies state synchronization to maintain consistency between the different instances of the application. On the contrary, for a full migration, each platform runs its own independent version and no synchronization is performed. For instance, Bandelloni and Paterno [3] present a 2D bank application which can fully migrate from a PDA to a PC while keeping the application runtime state during the process.
- At **workspace level**, workspaces can be redistributed on the platform, display and user dimensions. A workspace is an interaction space that groups together interactors that support the execution of a set of logically connected tasks. In graphical user interfaces, a workspace can be considered as a window. For instance, the painter metaphor [30] includes two workspaces: the palette of tools on a mobile device and the drawing area on an electronic white board. Similarly, Sjolund et al. [31] propose an DUI where a media player is displayed on a desktop monitor and is controlled by a remote controller displayed on a smartphone.
- At **domain concept level**, physical interactors can be redistributed on the different dimensions. In 3D, it corresponds to the interaction techniques and widgets. Several examples of this kind of distribution can be found in the field of 3DUI's. For instance, BUILD IT [29] is a tool dedicated to the design of factories. It is composed of two projective displays. A horizontal one allows the users to have a 2D view of the factory and provides them 2D interaction for object manipulation. A vertical display provides a perspective view of the result. In the same way, for data visualization, Slice WIM [9] combines an interactive multi-touch table and a stereoscopic display in order to provide simultaneous views of the data: overview and detailed. To continue, in [26], physical interactors for navigation, pointing and application control are distributed on a tablet in order to interact with content in an immersive system. In all cases, the system distribution is hard-coded. It is not performed automatically as it has only been designed to work with these two platforms.
- At **pixel level**, view continuity is ensured across different displays thanks to a distribution on the display and the platform dimensions. In 3D, this kind of redistribution is performed for multiple display systems. In this case, an application can be distributed on a cluster of PC's and rendered on multiple displays with view continuity. For instance a CAVE system [11] consists of a room whose walls, celling and floor surround a viewer with projected images. The user feels immersed in the virtual environment thanks to a viewer-centered perspective and thanks to view continuity between these displays.

In order to handle redistribution on the different dimensions and at the different levels of granularity, solutions designed for 2D user interfaces can be found. First, "Smartphone views" [31], proposes a synchronization mechanism in order to control an application on a desktop computer with a remote controller distributed on a smartphone. Then, VIGO [19] is an architecture that supports ubiquitous instrumental interaction among multiple devices and computers. Melchior et al. [27] propose a peer-to-peer architecture for the creation of DUI's. It includes mechanisms for widgets migrations and for the adaptation of the widgets representations and interactions according to the context of use. Two model-based approaches for the creation of DUI's are proposed by Masso et al. [25] and by Melchior et al. [28]. With these two solutions application components such as widgets can be distributed on multiple platforms, displays and users. The redistribution can be static or dynamic and is performed at the concrete UI level which is independent of any rendering engine or any programming language. As detailed in [28], a new distribution of a system can be chosen with a command line interface, or with an integrated user-interface: the meta-user interface. To continue, ZOIL [35] is a software framework for the development of post-WIMP ("Windows Icons Menus Pointer") distributed user interfaces. It proposes a client server architecture with

				Dynamic Redistribution					
	Platform Dimension	User Dimension	Display Dimension	Application Level	Workspace Level	Domain Concept Level	Pixel Level	Distribution Controller	3DUI enabled
Smartphone Views [31]	Yes	No	Yes	No				Developer	No
VIGO [31]	Yes	Yes	Yes	No				Developer	No
ZOIL [35]	Yes	Yes	Yes	No				Developer	Yes
Peer-to-peer DUIs [27]	Yes	Yes	Yes	Yes	Yes	Yes	No	Not specified	No
Model-based approach [25] [28]	Yes	Yes	Yes	Yes	Yes	Yes	No	User or Developer	No
PolyChrome Framework [4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	User	No
Distributed Rendering of HTML5 Canvas [34]	Yes	No	Yes	No				Developer	Yes
VR Juggler [7] MiddleVR	Yes	No	Yes	No				Developer	Yes
CVE architectures [17]	No	Yes	No	Yes	No	No	No	User	Yes

Fig. 1: A classification of the tools for the creation of DUI's.

a transparent persistent mechanism for the synchronization between the different platforms. Media content such as 3D models can be integrated in ZOIL but the framework does not include solutions for 3D interactions yet. Likewise, the PolyChrome framework [2] supports the creation of distributed web-based applications for data visualization. The framework handles synchronous and asynchronous collaboration on multiple devices. To continue in the context of web-based application, Yokoyama et al. [34] propose a solution for the creation of DUI's on the display dimension with a parallel rendering of HTML5 canvas elements. It could be used to display 3D scenes on mutliple displays, as WebGL is supported by HTML5.

In the field of 3DUI's, solutions to create DUI's also exist but they mainly focus on specific cases and do not let the end-user change the system distribution at runtime. One specific case handled in 3D and previously cited is the case of clusters of computers that manage multi-display systems such as CAVE's [11], Holostages, or Workbenches. In these cases, the system distribution is performed on the platform and display dimensions. The VR Juggler [5] framework and MiddleVR[1] propose such solutions. The second specific case handled in 3D is the field of CVE which needs a distribution at the platform and user levels. It implies a state synchronization between the different users platforms in order to maintain a consistent application. Some architectures for CVE are reported in [15].

Regarding the different properties of redistributable user interfaces, these different solutions can be classified according to multiple criteria. First, these solutions can be classified according to the distribution dimensions they can target: platform, user and display. Then, we can also separate the solutions between those that support dynamic redistribution and those that do not. As a sub-criteria, we can classify them according to the levels of redistribution they support: application, workspace, domain concept and pixel level. Then, solutions can be classified according to who choses the distribution of the system. Last, we can differentiate between the solutions that support the creation of distributed 3DUI's and those that do not. A classification of the main solutions cited in this Section according to these criteria is provided in Figure 1. As detailed in this classification, related work is devoid of a solution for the creation of 3DUI's with redistribution capabilities at different levels. Our goal with D3PART is to fill this gap.

In this paper, we propose a solution that can handle redistribution on the platform, display and user dimensions that consider the 3D specificities. In our case, the redistribution is user-initiated and controlled with an integrated user interface. We focus on redistribution for 3DUI's at the application, workspace, and domain concept levels. The pixel level on clusters of PC's is not covered. Indeed, we consider that handling redistribu-

[1] http://www.middlevr.com/middlevr-sdk/

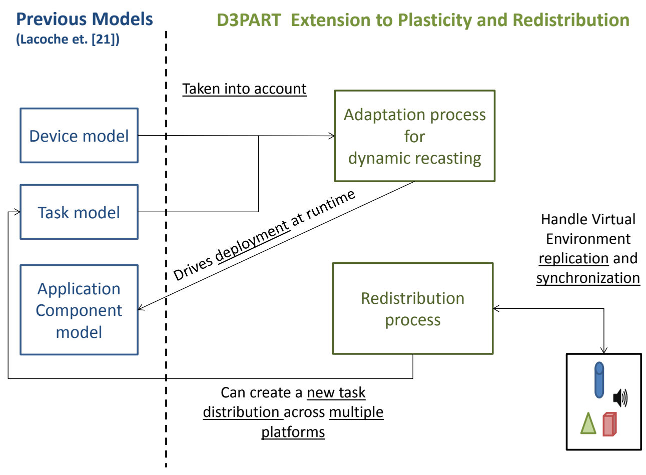
Previous Models
(Lacoche et. [21])
D3PART Extension to Plasticity and Redistribution
Taken into account
Device model
Task model
Application Component model
Adaptation process for dynamic recasting
Drives deployment at runtime
Handle Virtual Environment replication and synchronization
Redistribution process
Can create a new task distribution across multiple platforms

Fig. 3: The D3PART architecture is based on previous models from [21]. Dynamic redistribution and recasting are handled by our solution.

tion at the pixel level with high performance expectations is already a mature field of research, while the other levels are less explored in 3D. The proposed solution can be interfaced with modern 3D frameworks, especially game engines in order to be easily integrated into the 3D developers and designers work-flow. One of the advantages of our approach is that any application developed with our model automatically benefits from redistribution capabilities.

3 Conceptual Modeling of 3DUI's in D3PART

Most approaches to handle redistribution and plasticity are dedicated to 2D user interfaces and do not address new issues introduced by 3DUI's. Indeed, 3DUI's include a wider range of possible interaction devices and interaction techniques for interacting with more complex content. For instance, this content includes 3D meshes with complex materials and behaviors. That is why, in order to design 3D applications that handle plasticity, recasting and redistribution, as shown in Figure 3 taken from [22], our solution D3PART extends our previous plasticity models [21]. In this previous work we presented three models for the implementation of plastic 3DUI's. These models are shown in Figure 2.

First, in order to represent the device context of use, we use the device model presented in [21] that can describe the devices used for 3DUI's. It defines a platform as a hardware environment composed of input and output devices and computing units. The goal of this device model is to precisely describe most of the devices that can be used for interaction purposes at runtime. The model includes device capabilities, limitations and representations in the real world. As shown in Figure 2, each device is composed of input units, output units and physical objects for its representation in the real world. Input units can be trackers, vocal commands, camera streams, etc. Output units can be visual displays, sound outputs, force feedbacks, etc. Each device corresponds to a class that inherits from the basic device class. In this class, the developer has to complete some functions to fulfill the input data, trigger the outputs and tell the system when a new instance of the device is plugged or unplugged. These steps can be done with a SDK dedicated to a particular device. Another XML description file is used to describe the device properties that corresponds to the device and its device units file. These properties can also be reported in the device SDK by the developer to perform its configuration.

Second, we use a task model briefly introduced in this previous work. These tasks represent the behavior of the application independently from any concrete application component. They represent the features that the developer wants to integrate into the application with a high granularity. For 3DUI's, according to Hand [17], these tasks belong to three categories: selection and manipulation, application control, and navigation. They represent the features that the developer wants to integrate into the application with a high granularity. Therefore, a task is represented by a name that describes its role in the final application. For instance, it can be "Selection", "Navigation", etc. A task can define different functions (the task events) that constitute the application logic, such as adding an object into the scene, or loading a new scene configuration, etc. Dependencies between the tasks can also be described by the developer. For instance, an application control task with a menu needs a selection task, therefore the two tasks are defined as dependent.

Therefore, as detailed in Figure 4, in D3PART an application is defined with a set of high level tasks and with a description of the virtual environment. In our implementation, these needed tasks and the dependencies must be provided by the application developer or the designer in an XML configuration file. An example of an XML configuration file is given in Listing 1 for the application described in Section 5. Four high level tasks are needed in this application, selection and manipulation, navigation, application control (named furniture control) and redistribution. The furniture control task is defined as dependent on the selection and manipulation task, which is expressed by the topTask="0" in the file. The redistribution task is parametrized with the redistribution server IP. An application in D3PART is also described by its virtual environment. This virtual environment is composed of visual (3D content) and

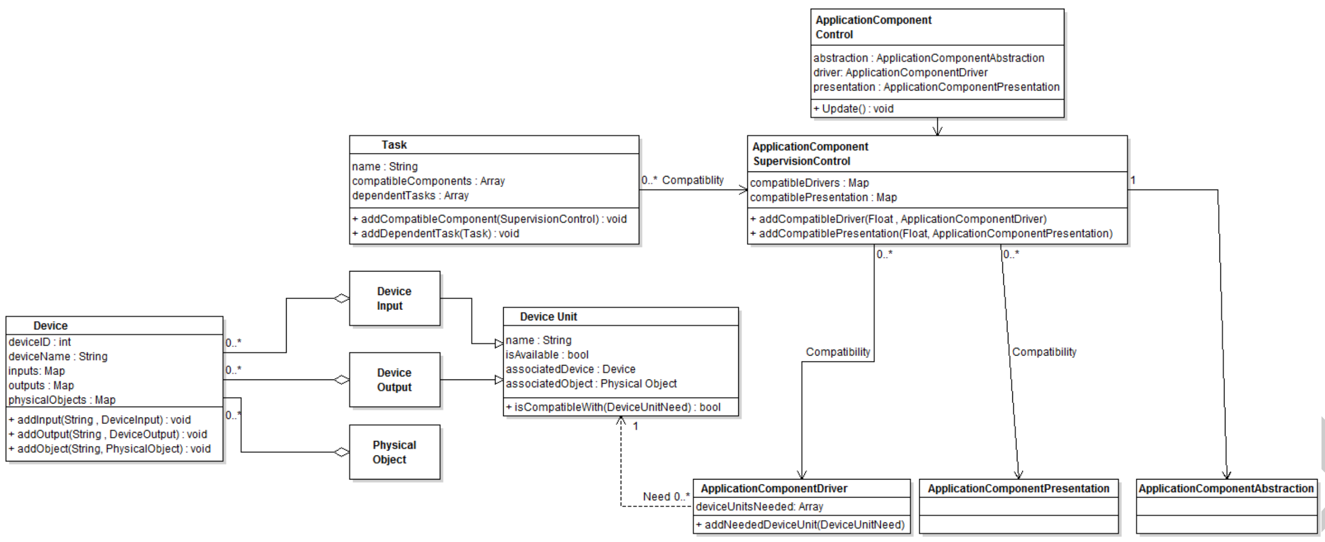
ApplicationComponent Control
abstraction : ApplicationComponentAbstraction
driver: ApplicationComponentDriver
presentation : ApplicationComponentPresentation
+ Update() : void
Task
name : String
compatibleComponents : Array
dependentTasks : Array
+ addCompatibleComponent(SupervisionControl) : void
+ addDependentTask(Task) : void
0..* Compatiblity
ApplicationComponent SupervisionControl
compatibleDrivers : Map
compatiblePresentation : Map
+ addCompatibleDriver(Float , ApplicationComponentDriver)
+ addCompatiblePresentation(Float, ApplicationComponentPresentation)
1
0..*
0..*
Compatibility
Compatibility
Device
deviceID : int
deviceName : String
inputs: Map
outputs : Map
physicalObjects : Map
+ addInput(String , DeviceInput) : void
+ addOutput(String , DeviceOutput) : void
+ addObject(String, PhysicalObject) : void
0..*
0..*
0..*
Device Input
Device Output
Physical Object
Device Unit
name : String
isAvailable : bool
associatedDevice : Device
associatedObject : Physical Object
+ isCompatibleWith(DeviceUnitNeed) : bool
1
Need 0..*
ApplicationComponentDriver
deviceUnitsNeeded: Array
+ addNeededDeviceUnit(DeviceUnitNeed)
ApplicationComponentPresentation
ApplicationComponentAbstraction

Fig. 2: The three models used in D3PART, a task model, a device model and an application component model.

```
<TaskConfig>
<NeedTask taskName="SelectionManipulation"
    taskId="0"/>
<NeedTask taskName="FurnitureControl" taskId="1"
    topTask="0"/>
<NeedTask taskName="Navigation" taskId="2"/>
<NeedTask taskName="Redistribution" taskId="3">
<ParamTask serverIp="127.0.0.1"/>
</NeedTask>
</TaskConfig>
```

Listing 1: The XML task configuration file of the furniture planning application.

sound assets. It can be edited separately, for instance in a game engine editor, or loaded with an X3D file depending on the implementation of the models used.

The tasks and the virtual environment are the static representation of the application. This representation is independent of the context of use encountered at runtime. In D3PART, the concrete implementation depends on application components that are deployed to achieve each high level task. Indeed, as shown in Figure 2 each task exposes a list of compatible application components that can be deployed to achieve it. These lists are also edited in an XML configuration file. To develop these components we also use the model presented in [21], a model for developing concrete application components independently from any 3D framework or 3D devices. An application component can correspond to an interaction technique,3D widget, a visual effect, etc. The proposed model for the creation of these components is a modification of PAC [10] and ARCH [1] models. This model is shown in Figure 2 and an example of a component is given in Figure 5. It corresponds to

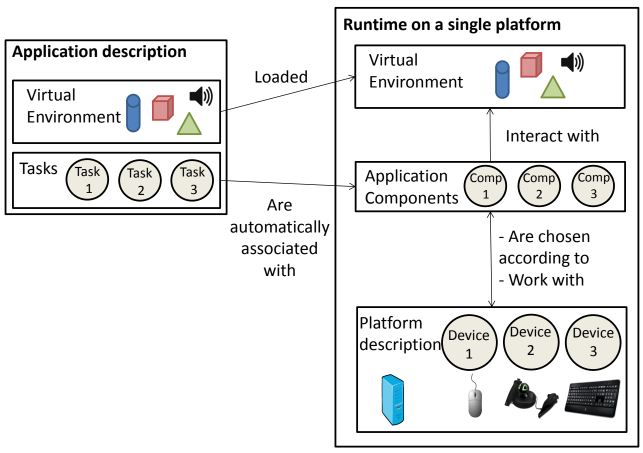

Fig. 4: The description of an application and its execution on a single platform that we presented in [22].

the application component of a 3D-ray based interaction technique for the selection and manipulation task. As shown, an application component is divided into five facets. These facets decouple its different features:

– The Abstraction: it describes the semantics of the component and the function it can perform,
– The rendering presentation facet is the only facet depending on a 3D framework. It handles graphics output and physics. In our case, in the examples given, these facets are developed with Unity3D[2]. For a given application component, this facet can also define its representation in the virtual world. For

[2] https://www.unity3d.com/

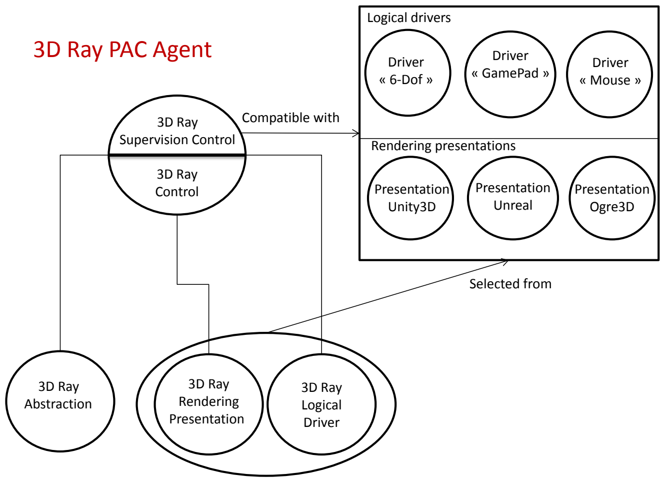

Fig. 5: One example of application component given in [21]. The figure represents the application component of a 3D ray-based interaction technique.

instance, the 3D aspect of a widget will be defined in this facet.

– The logical driver handles input and output devices management. Its main use is for the development of interaction techniques. It implements the way the interaction technique is controlled according to a set of abstract interaction devices. In this facet, the developer describes all required input and output units according to a set of parameters taken from the device model.
– the Control: it ensures the consistency between the rendering presentation, the logical driver and the abstraction.
– the Supervision Control: it receives the context modifications at runtime and then is able to determine if it is still possible to use a particular logical driver. It also contains all the types that can be instantiated as a logical driver or rendering presentation facet for the current application component.

As detailed in Figure 5, this technique has multiple compatible logical drivers in order to be possibly driven by different kinds of devices such as a 6-Dof tracker, a mouse or a gamepad. Similarly, multiple rendering presentations can be developed to make the technique available in many 3D frameworks. For the same selection and manipulation task, we could also use a 2D cursor for selecting and moving the objects on the screen plane. Different logical drivers can control this technique based on devices such as a mouse and a multi-touch screen. Other examples with more details are provided in [21].

4 Deployment of 3DUI's in D3PART

4.1 Adaptation Process

In the previous Section we described the static representation of an application implemented with D3PART. In this Section we describe the adaptation process included in D3PART that uses this representation in order to dynamically adapt the application at runtime on a single platform. At runtime, as detailed in Figure 4 high level tasks are automatically associated with concrete application components according to the encountered context of use (the devices connected to the platforms) in order interact with the virtual environment. For these components, the rendering presentation and the logical driver facets are also chosen according to the context of use. The control and abstraction facets do not depend on this context. The association is performed with an automatic adaptation process that we included in D3PART on top of the device and task models as shown in Figure 3 in order to support dynamic recasting. The association is made with a scoring system that takes into account the platform capabilities and the list of compatible components exposed by each task. Its goal is to maximize the usability of the application. We won't give a full description of this scoring mechanism because it is not in the scope of this paper. The association process is performed at each context change in order to detect any no longer usable application components or more adapted ones. The association process that we propose can be described as follows:

1. A context modification is detected. For example, it can be the connection of a new device or the addition of a task. It can also be the disconnection of a device or the suppression of a task.
2. For each deployed application component, we check if the association with the current logical driver is still possible in the current context of use. This association is still possible if the devices that it uses are still plugged in and available. If not, the application component is destroyed and the associated task is classified as ”not in progress”.
3. For each task classified as ”not in progress”, we create a list of all possible triplets (application component, logical driver, rendering presentation) that can achieve the given task. A triplet is possibly instantiable if device units needed by the logical driver can be found in the list of connected devices and if they are available. The rendering presentations that do not correspond to the current used 3D framework are omitted. A compatibility score is attributed to each triplet. The one with the best score is deployed. The devices units, associated with the logical driver,

are set as not available. The task is classified as ”in progress”.
4. For each task ”in progress” that has not been processed in the previous step, we check if we can find a triplet more adapted than the current one. This optimization is not performed at the same time as the previous step. Indeed, the priority is given to the association of application components to the tasks classified as ”not in progress”. To perform this optimization, we create a list with all triplets that get a better score than the current one. If the list is empty, the current one is still the most adapted. Conversely, we destroy the current application component and we deploy the new best choice.

So, this adaptation process supports the dynamic recasting of the application and always ensures its optimal usability whatever the context of use. It will make it possible for the application to handle the different context changes encountered during the redistribution process presented in Section 4.2.

4.2 Redistribution Process

With the plasticity models and the dynamic recasting mechanism introduced in the previous section, a developer can create an application that can be adapted to the capabilities of a wide variety of platforms. As shown in Figure 3 and described in [22] D3PART also includes a redistribution process that makes the integration of redistribution capacities totally transparent and automatic for the developer. The process consists of distributing the high level tasks and the virtual environment across the different dimensions: platform, display and user. The developer's work is to create high level tasks, and implement the compatible application components with the help of the models from [21], described in the previous section. With the implementation of multiple compatible components for each task and multiple logical drivers, which use different kinds of devices, for each component; the developer ensures that his application will be usable on a wide variety of platforms.

We added a built-in high level task and its corresponding application component in order to allow any developer to add redistribution capability to his application. The application component for redistribution is also defined with the extension of the PAC and ARCH models described in Section 3. No logical driver is defined as no specific interaction device is needed by this component. The abstraction facet contains the redistribution logic and the rendering presentation facet contains the parts that are dependent on the target 3D framework. Regarding the process, redistribution needs a connection mechanism between the different platforms. This is needed for platform registration and state synchronization. To do so, we use a client/server architecture to which the different platforms can register. Once registered, these platforms are available for the redistribution process. For now, this feature is implemented with the network capabilities of the target 3D framework. Therefore, it is integrated into the rendering presentation facet. We chose this solution in order

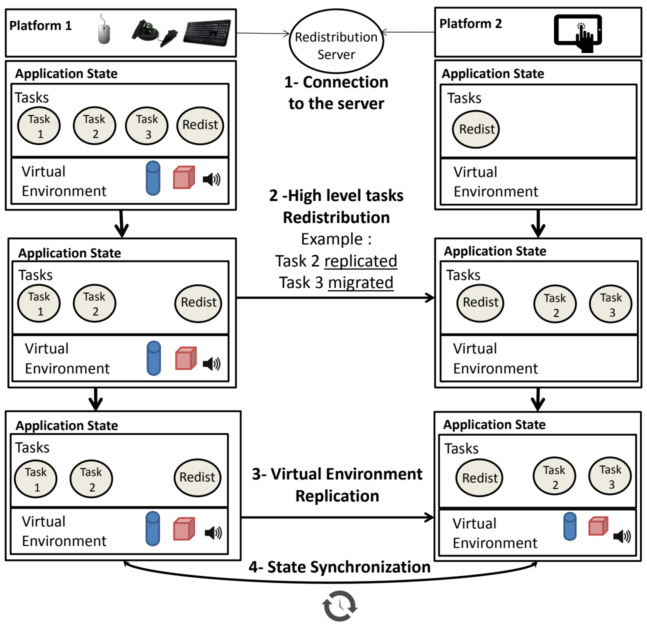

Fig. 6: With D3PART, the redistribution process is performed in four steps as shown in [22]. First, the different platforms connect to the redistribution server. An empty application runs on these distant platforms. It only contains the redistribution task. Then, the user initiates a new distribution of the system with the meta-user interface. Here he chooses to replicate the task 2 and to migrate the task 3 to a second platform. For these two tasks, thanks to the dynamic recasting mechanism described in Section 4.1, compatible application components are automatically deployed on the second platform that fits its capabilities. The third step consists in replicating the VE from the first platform to the second one. It includes 3D meshes, their materials, and sound assets. With this step, we transmit the current state of the application in order to keep it consistent on the different platforms. Then in the last step, we maintain this consistency during the execution of the application. Indeed, the redistribution server ensures state synchronization between the two platforms while the application is running. For now, the 3D objects transform and the tasks events are synchronized.

to rapidly create prototypes. However, as future work, this mechanism could become independent of the 3D framework and implemented in the abstraction facet. Our implementation does not show apparent latency but being independent from the 3D framework would allow us to optimize the network load. As proposed in the model-based approach introduced by Melchior et al. [28], this component implements an integrated user interface for platform registration and for the control of the redistribution process: the meta-user interface. In our case, the redistribution is performed at runtime and is user-initiated. Indeed, the meta-user interface is proposed for the end-user of the application. The interface can be shown and hidden at runtime with a graphical button or a device button depending on the context of use. The redistribution process is then performed in four different steps as proposed in [22], and shown in Figure 6.

The first step consists in connecting to the redistribution server. The IP address of the server can be given in the meta-user interface, or the XML task configuration file, as shown in listing 1. This step must be performed on the platform where the application is running and on each platform that must be available for redistribution. An empty application runs on these distant platforms. It contains the framework that can run an application developed with the D3PART model. The only task defined as "needed" on these distant platforms is the redistribution one. Therefore, the corresponding application component is deployed on these platforms.

The second step consists in configuring the desired redistribution with the meta-user interface. First, the user chooses the platform on which the application will be redistributed from a list of available ones. These available platforms are the ones that have registered to the redistribution server. When a new platform registers, it is automatically added to this list. In our case, the basis of the redistribution process is made on the platform dimension. However, as each platform may manage another display and may be used by another person, user and display dimensions can also be targeted. Then, the user configures the high level tasks distribution across the two platforms. As shown in Figure 7: multiple choices are given to the user in the menu:

- Full migration: all tasks migrate. Each platform runs an independent version of the application. It can be performed when the user wants to switch to another platform.
- Partial migration: the user chooses which task(s) will migrate to the distant platform. The application is distributed and so shared between the two platforms. It can be performed to combine different platforms.
- Partial replication: the user replicates some tasks to the distant platform. He will be able to perform these tasks on the two platforms within the same shared application. In the same way, it can be used to combine multiple platforms.
- Full replication: all tasks are replicated and can be performed on different platforms in the same shared application. This kind of redistribution can be used to start a collaboration with a user on a different platform.

Dependent tasks must be redistributed together. Therefore, they are grouped into the menu as shown in Figure 7. In this figure the furniture control task is dependent to the selection and manipulation task. In the meta-user interface we associate a warning icon to a high level task if it cannot be performed on the distant platform. To do so, we ask the distant platform if an application component can be deployed for each task according to the platform capabilities. The goal of this feature is to warn the end user that the application can be degraded if this task is redistributed. On the other platform, thanks to the adaptation process included in D3PART, described in the previous section, an application component is automatically associated with each redistributed task. As said, these components are chosen in order to fit the platform capabilities in terms of device availability.

When the redistribution of tasks has been done, the third step consists in fully copying the virtual environment to the distant platform. The goal is to maintain the application state during the redistribution to the target platform. This virtual environment includes 3D meshes, their materials, and sound assets. To perform this copy, we consider three options:

- Assets are known in the distant platform. Only the names are transmitted.
- Assets are not known but can be downloaded from a distant server. In this case, URLs are provided.
- Assets are not known. For instance in a case of a 3D painting application, the user is editing new 3D content. Here, assets can be streamed over the network.

For now, our implementation only includes the first one.

The last step consists in synchronizing the different platforms. As for CVE's, a synchronization is performed in order to keep a consistent state between the instances of the same application, running on different platforms. In the case of a full migration, no synchronization is performed because each platform runs an independent version. The synchronization is performed as long as all platforms are connected to the redistribution server.

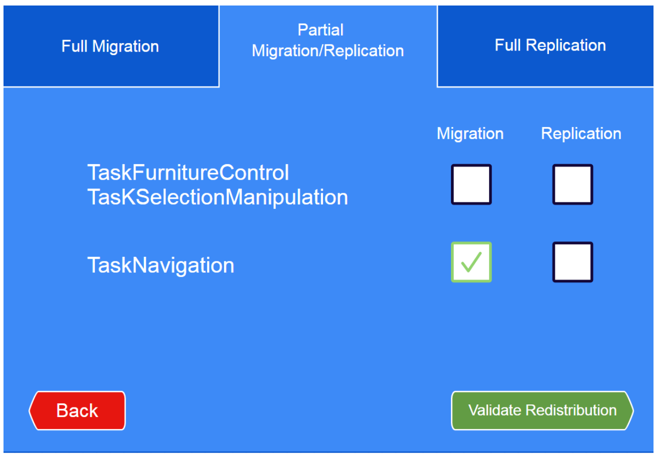

Fig. 7: The meta-user interface is an integrated user interface designed to control the redistribution process. Here, the user chooses the high level tasks that will be redistributed to the distant platform. In this example, also presented in [22], three tasks can be redistributed: navigation, selection/manipulation and furniture control. The last two ones are dependent. The user selects a partial migration, only the navigation task will be redistributed. The other two tasks remain on his current platform.

Two kinds of information are synchronized between the different instances of the application. First, the 3D object's transforms are synchronized in order to maintain consistency between the different 3D worlds. In the case of collaboration, to handle concurrency when moving objects, the priority to move an object is given to the first user who grabs it. Then, other users cannot grab and move this object until the first user has released it. Other mechanisms could be integrated as well. As detailed by Margery et al. [24], it could also be possible to let the user simultaneously modify independent parameters of a same object, or to let them simultaneously modify co-dependent parameters of a same object. To do so, we would have to use a more up-to-date synchronization engine, such as the #FIVE architecture [6]. To continue, the events of high level tasks are also synchronized. The events constitute the logical implementation of the application and have to be synchronously performed on each application instance. To do so, we use an observer design pattern. The redistribution application component observes all task events. When one event is triggered, it is transmitted with its corresponding parameters through the network as text messages in order to be triggered distantly. An example of an event given in the example application in Section 5 is the addition of a 3D object into the scene.

5 Examples of Redistribution

In order to illustrate the redistribution possibilities offered by D3PART, we present different use cases that are based on a furniture planning application. This application consists of laying-out an empty room with furniture. Its goal is to help people to plan the use of particular premises. At the task level, the application is composed of three tasks. First, a navigation task is needed in order to navigate within the room. Second, we need an application control task (named furniture control) for adding furniture into the room with the help of a menu. Adding an object is defined as an event into the task. Last, we need a selection and manipulation task for moving furniture, and for menu selections. These two last tasks are defined as dependent: indeed, selection possibilities are needed when interacting with the menu. In these different cases we use two platforms. First, we use a mobile device which is an Android tablet. Then, we use an immersive system, a CAVE [11] with active stereo and with dimensions: 9.6m length $\times$ 3.1m height $\times$ 3.0m width. MiddleVR is used to handle the different screens and clustering. Even if they are not present in these examples, other platforms could also be considered such as Head-Mounted-Displays (HMD) and desktop environments. In the different examples, all systems runs approximately at 25 fps. The difference in frame rates does not impact the synchronization.

As described in section 4.2, the redistribution process starts with the connection of the tablet and the CAVE system to the redistribution server. For all the presented cases, the application is first launched on a tablet. According to the automatic adaptation process described in Section 4.1, one concrete application component is deployed for each needed task. Each component is chosen in order to fit the platform capabilities. First, for the furniture control task, a 2D menu is instantiated with the list of furniture that can be added. According to its implementation the menu can be hidden if needed. For the manipulation task, an interaction technique based on the multi-touch capabilities of the tablet is deployed. With this technique the user can translate the objects onto the floor with one finger and rotate them around the up axis with two fingers. For the navigation task, a pan and zoom navigation technique is deployed. Here, this component places the camera to have a plan view of the scene on the tablet as shown in Figure 8a. With the multi-touch capabilities, the user can translate the point of view and can zoom within the scene while keeping the plan view of the room.

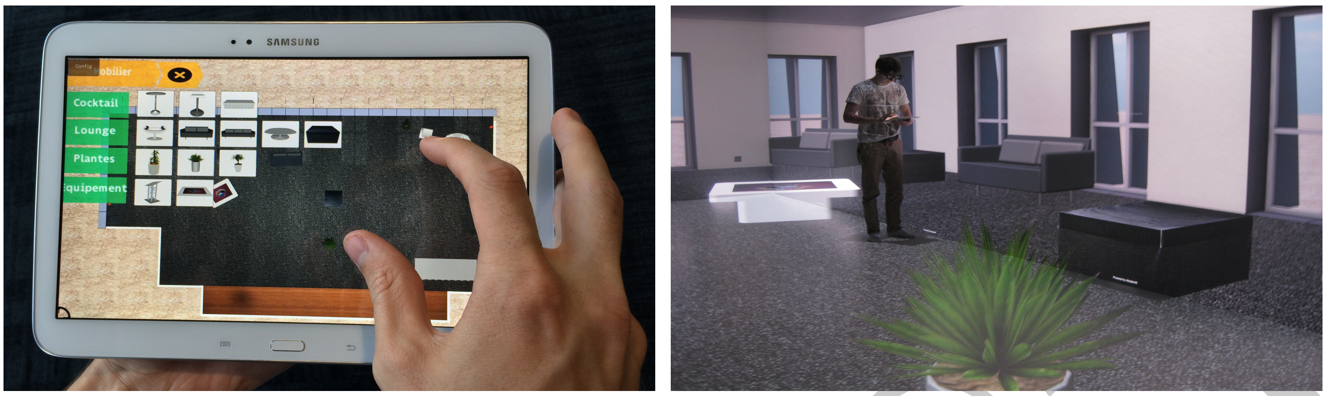

Fig. 8: The redistributed World-In-Miniature: an example of redistribution that demonstrates how it can be used for platform combination. Here the redistribution is performed between an immersive multi-display system (a CAVE) and a tablet. (a) Interaction techniques for the selection/manipulation task and the application control are deployed on the tablet. On this platform the user has a plan view of the virtual world at a reduced scale with 2D interaction capabilities. (b) An interaction technique for the navigation task is deployed in the CAVE. Therefore, at the same time, the user is immersed and can navigate at scale of one in the same shared virtual world in a CAVE.

5.1 Redistribution for platform switching

Today, users are more frequently confronted with situations where they have to move from one platform to another [12]. This is one scenario possible for our furniture planning application. Indeed, the application may be used on a wide variety of different platforms such as desktop environments, smartphones, immersive systems, touch tables, etc. All platforms do not offer the same possibilities and therefore some can be more adapted to specific needs. Therefore we want to ensure for the end user seamless transitions between these different platforms. This example demonstrates how the redistribution capabilities of our solution can ensure usability continuity during these changes of hardware environment. In this scenario, the redistribution is performed on the platform and the display dimensions and at the application level.

First, the user is interacting on the tablet at his desk. With this tablet, he can also work while being mobile. All the tasks are available, as corresponding application components are deployed, as explained in the previous section. However, the tablet only offers a 2D plan view of the result and the user would like to have a 3D view at a scale of one, in order to better perceive the volumes. To do so, an immersive system is available: a CAVE. The meta-user interface allows the user to perform a full migration of his application to this platform. The application totally migrates to the CAVE, all tasks and all contents, nothing remains on the tablet. The user can now be immersed at scale of one and continue to fine-tune the layout of the room. Usability continuity is ensured thanks to the included adaptation process. The application is adapted to the target platform. Indeed, as described in Section 4.1, application components are chosen according to the new platform capabilities. In that case, a 3D ray-based manipulation technique is deployed. The position and the rotation of the ray are set with the tracked flystick and its buttons are used for object selections and to change the ray length. For the navigation task, a walking navigation metaphor is deployed. The tracked head position, combined with the flystick joystick, are used to move the point of view. For the furniture control task, a 3D movable menu is deployed. The 3D ray is used to select the menu items, to move it and also to hide and show it.

When the user has finished his work, he may want to continue his work while mobile. Therefore, the meta-user interface is also available in the CAVE and so the inverse process is also possible to migrate back to the tablet. For example, he would show the result to a colleague who could use another kind of VR setup, such as an HMD.

5.2 Redistribution for platforms combination

This example demonstrates how redistribution can be used in order to combine different platforms. This is the example we gave in [22]. In that case, redistribution is performed on the display and platform dimensions, and at the domain concept level. Our example is based on the World-In-Miniature technique [32], which provides the user with a handheld model of the virtual environment at a smaller scale. It can be used for manipulating virtual objects, or for navigation. This miniature representation is directly rendered in the virtual world. Here, we propose to deploy this technique onto a tablet in order to control the furniture planning application in a CAVE system. The user will be able to interact with the tablet while being immersed at a scale of one in the CAVE. This use case can be useful for novice users who are not confident with 3D interactions and may prefer more common multi-touch interactions. Indeed, the user can interact with the usual and easy-to-use multi-touch capacities of the tablet, while being immersed at the same time in the 3D world with the CAVE.

The user chooses a partial migration to the CAVE, only the navigation task migrates to the distant platform. Other tasks remain on the tablet. This choice is made with the meta-user-interface, as shown in Figure 7. As described in Section 5.1, an interaction technique, based on a walking metaphor controlled with head tracking and a joystick, is deployed in the CAVE for this navigation task. It places the point of view inside the room in order to immerse the user in it. At this time the application is distributed on two platforms and displays. First, as shown in Figure 8a, a redistributed World-In-Miniature is on the tablet. The virtual world is displayed at a lower scale with a plan view. Moreover, as said in Section 5.1, a 2D menu for furniture control and a multi-touch interaction for selection and manipulation are deployed. Second, as shown in Figure 8b, at the same time the user is immersed at scale of one into the room in the CAVE and can navigate within it. Our transparent synchronization mechanism ensures the consistency between the two parts of the application. Indeed, the synchronization of the 6 DoF transforms of the objects between the two platforms ensures consistency when the user moves an object on the tablet. As well, the command for adding an object into the room is also synchronized. Therefore, when an object is added with the 2D menu on the tablet, the same object is also added in the CAVE.

5.3 Redistribution for collaboration

In this example, we demonstrate that our redistribution process can be used in order to create a Collaborative Virtual Environment (CVE). Here, redistribution is performed on the user, platform and display dimensions and at the application level. Indeed, the replication capabilities included in our solution allow any user to start at any time a collaboration with another person

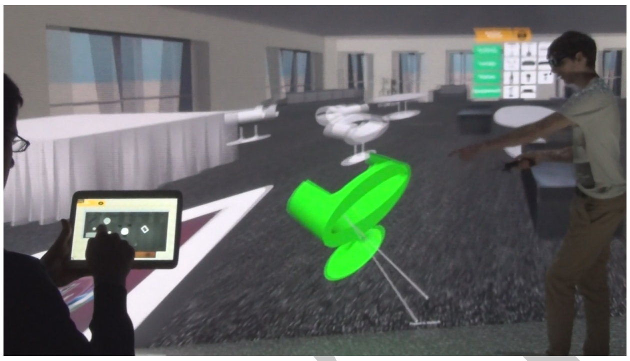
Fig. 9: An example of redistribution on the user dimension. Two users are collaborating on the same VE with two different platforms.

using a different platform. With this feature any application developed with the D3PART model, including the furniture planning one, automatically benefits from collaboration capacities.

In this scenario, the first user has performed a first configuration of the empty room with his tablet and now wants to share his room configuration and wants to finish it with another user. Therefore, he performs a full replication from the tablet to the second user platform: the CAVE. All tasks are replicated: navigation, selection and manipulation, and furniture control. Therefore, the two users now have the same interaction capabilities. The plasticity property handled by the system ensures usability continuity between the two platforms, and the interaction capabilities remain the same. Indeed, the application components deployed for these different tasks are chosen according to each platform's capabilities with the adaptation process included in D3PART. They are the same as for the two scenarios described in the two previous sections. In this case, the collaboration is asymmetric as the two people are using different platforms and different interaction techniques, as shown in Figure 9. A collaboration with two similar systems could also be performed. Here, the collaboration is co-located, both users are situated in the same place and can directly communicate about the result. However, the collaboration could also be distant. Indeed, our architecture makes it possible to have distant connections to the redistribution server. With the virtual environment replication and the synchronization performed by the redistribution process, a high consistency between the two instances of the application is ensured. Both users are interacting in the same shared virtual environment. In order to provide awareness about the activity of the distant user, for now, only the view frustum of each user is represented in the virtual environment. Future work could include different awareness mechanisms, for instance, trying to make the distant user perceive his current context of use.

6 Discussion and Perspectives

According to the classification given in Section 2, D3PART can support the creation of distributed user interfaces on the platform, user and display dimensions. Dynamic redistribution is supported at the application, workspace and domain concept levels. The examples presented in Section 5 allows us to demonstrate that these dimensions and levels can be targeted with D3PART. Redistribution at the pixel level is not handled by our solution and could be a perspective of work to complete in 3DPART. In that case, within D3PART, it would be needed to include a method to synchronize multiple stereoscopic displays and a system for placing cameras in order to ensure view continuity between these displays.

In D3PART, the controller of the redistribution is the end-user with the meta-user interface. As a perspective of our work, our goal would be to also obtain system-initiated redistribution or mixed-initiated redistribution. For instance, this kind of approach could consist in finding the right platform, or the right user for each task according to the platforms capabilities and the user preferences. Our scoring system could be used to do so, for example, by applying no longer locally, but instead, by applying it on all available platforms. Indeed, for each task we could compare the scores obtained by the applications components on each of the available platforms and instantiate the best ones. In that case we could directly perform the redistribution and we would obtain a system-initiated redistribution. With the same mechanism, we could also assist the user in the meta-user interface by telling him which task would be adapted to each platform, here the redistribution would be mixed-initiated.

D3PART is totally designed to handle distributed 3DUI's as it includes solutions for 3D interactions and for the synchronization of 3D worlds. For now, the dynamic recasting capability included in D3PART, focuses on adapting the interaction techniques to the available devices. As future work, we also want to consider level of details during the virtual environment replication. Indeed, as each platform may not have all the same computational capabilities, the rendering of the assets could be handled differently. In some cases, it would be necessary to consider adaptive assets. For instance, a very complex 3D model cannot be rendered in the same way on a PC with multiple GPUs as on a mobile device. To solve this issue, we first plan to integrate information about the computing power of the platforms into the device model. Second, we plan to give the possibility to parametrize the choice of an asset according to the computation capabilities of the target platform. During the virtual environment replication process, only the assets that correspond to the distant platform computation capabilities would be transmitted.

Regarding the literature review on plasticity presented by Lacoche et al. [20], the plasticity property is defined on four dimensions: the adaptation sources, the adaptation targets, the adaptation controller and the adaptation time. D3PART can cover most of the design space problem defined in this previous work. Indeed, with our model adaptations can target the content presentation, the interaction techniques and the system distribution. These adaptations can occur at runtime and between sessions while they can be controlled by the system (recasting) and by the end-user (recasting and redistribution). However, for now, all the cited adaptation sources cannot be targeted. For instance, some future work are needed in order to include in D3PART the users characteristics and preferences.

To finisg, two aspects of D3PART must be evaluated if we really want to demonstrate its interest and efficiency. First, we plan to evaluate D3PART with developers. We want to verify if our solution is easy to use and efficient for the development of 3DUI's with redistribution capabilities. To do so, feedback from developers has to be collected during the creation of such an application with D3PART. Second, we also plan evaluate the system to assess the interest of redistributable 3DUI's, their usability and their acceptability for end users.

7 Conclusion

In this paper we introduce D3PART (Dynamic 3D Plastic And Redistribuable Technology), a new model to handle plasticity and redistribution for 3DUI's. Based on previous work on plasticity for 3DUI's, our solution eases the development of 3DUI's interfaces with redistribution capabilities. Our approach is based on a client-server architecture. Redistribution can be performed at runtime by the user with an integrated user interface, namely the meta-user interface. Dynamic recasting is handled by D3PART, with the included adaptation process, and ensures usability continuity whatever the new distribution chosen. The distributed application will fit each of the target platform's properties. With this approach, any application developed with the D3PART model automatically benefits from redistribution capabilities.

To illustrate these possibilities, we have presented three examples of redistribution on different dimensions and at different levels for a furniture planning application. These examples show how redistribution can be used to switch from a mobile platform to an immersive one, to combine these two platforms, and finally to create a collaborative context of use between them.

References

1. A metamodel for the runtime architecture of an interactive system: The uims tool developers workshop. SIGCHI Bull. **24**(1), 32–37 (1992)
2. Badam, S.K., Elmqvist, N.: Polychrome: A cross-device framework for collaborative web visualization. In: Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces, pp. 109–118. ACM (2014)
3. Bandelloni, R., Paternò, F.: Migratory user interfaces able to adapt to various interaction platforms. International journal of human-computer studies **60**(5), 621–639 (2004)
4. Bharat, K.A., Cardelli, L.: Migratory applications. In: Proceedings of the 8th annual ACM symposium on User interface and software technology, pp. 132–142. ACM (1995)
5. Bierbaum, A., Hartling, P., Morillo, P., Cruz-Neira, C.: Implementing Immersive Clustering with VR Juggler. In: ICCSA 2005, pp. 1119–1128. Springer-Verlag, Berlin, Heidelberg
6. Bouville, R., Gouranton, V., Boggini, T., Nouviale, F., Arnaldi, B.: # five: High-level components for developing collaborative and interactive virtual environments. In: Proceedings of Eighth Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS 2015), conjunction with IEEE Virtual Reality (VR) (2015)
7. Calvary, G., Coutaz, J., Dâassi, O., Balme, L., Demeure, A.: Towards a new generation of widgets for supporting software plasticity: The "comet". In: R. Bastide, P. Palanque, J. Roth (eds.) Engineering Human Computer Interaction and Interactive Systems: Joint Working Conferences EHCI-DSVIS 2004, Hamburg, Germany, July 11-13, 2004, Revised Selected Papers, pp. 306–324. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
8. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interacting with computers **15**(3), 289–308 (2003)
9. Coffey, D., Malbraaten, N., Le, T., Borazjani, I., Sotiropoulos, F., Keefe, D.F.: Slice wim: A multi-surface, multi-touch interface for overview+detail exploration of volume datasets in virtual reality. In: Symposium on Interactive 3D Graphics and Games, I3D '11, pp. 191–198. ACM, New York, NY, USA (2011)
10. Coutaz, J.: PAC, on object oriented model for dialog design. In: Interact'87 (1987). 6 pages.
11. Cruz-Neira, C., Sandin, D.J., DeFanti, T.A., Kenyon, R.V., Hart, J.C.: The CAVE: Audio Visual Experience Automatic Virtual Environment. Commun. ACM **35**(6), 64–72 (1992)
12. Demeure, A., Sottet, J.S., Calvary, G., Coutaz, J., Ganneau, V., Vanderdonckt, J.: The 4C Reference Model for Distributed User Interfaces. In: ICAS 2008, pp. 61–69
13. Elmqvist, N.: Distributed User Interfaces: State of the Art, pp. 1–12. Springer London, London (2011)
14. Figueroa, P., Green, M., Hoover, H.J.: Intml: A description language for vr applications. In: Proceedings of the Seventh International Conference on 3D Web Technology, Web3D '02, pp. 53–58. ACM, New York, NY, USA (2002)
15. Fleury, C., Duval, T., Gouranton, V., Arnaldi, B.: Architectures and Mechanisms to Maintain efficiently Consistency in Collaborative Virtual Environments. In: SEARIS 2010 (IEEE VR 2010 Workshop on Software Engineering and Architectures for Realtime Interactive Systems). Waltham, United States (2010)
16. Gonzalez-Calleros, J., Vanderdonckt, J., Muoz-Arteaga, J.: A structured approach to support 3d user interface development. In: ACHI 2009, pp. 75–81. DOI 10.1109/ACHI.2009.14
17. Hand, C.: A survey of 3D interaction techniques. In: Computer graphics forum, vol. 16, pp. 269–281 (1997)
18. Hutchings, H.M., Pierce, J.S.: Understanding the whethers, hows, and whys of divisible interfaces. In: Proceedings of the working conference on Advanced visual interfaces, pp. 274–277. ACM (2006)
19. Klokmose, C.N., Beaudouin-Lafon, M.: Vigo: instrumental interaction in multi-surface environments. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 869–878. ACM, New York, NY, USA (2009)
20. Lacoche, J., Duval, T., Arnaldi, B., Maisel, E., Royan, J.: A survey of plasticity in 3d user interfaces. In: 2014 IEEE 7th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), pp. 19–26 (2014)
21. Lacoche, J., Duval, T., Arnaldi, B., Maisel, E., Royan, J.: Plasticity for 3d user interfaces: new models for devices and interaction techniques. In: Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 28–33. ACM (2015)
22. Lacoche, J., Duval, T., Arnaldi, B., Maisel, É., Royan, J.: D3part: A new model for redistribution and plasticity of 3d user interfaces. In: 3DUI 2016: IEEE symposium on 3D User Interfaces Summit, pp. 23–36. IEEE (2016)
23. Lindt, I.: Adaptive 3d-user-interfaces. Ph.D. thesis (2009)
24. Margery, D., Arnaldi, B., Plouzeau, N.: A General Framework for Cooperative Manipulation in Virtual Environments, pp. 169–178. Springer Vienna, Vienna (1999)
25. Massó, J.P.M., Vanderdonckt, J., López, P.G., Fernández-Caballero, A., Pérez, M.D.L.: Rapid Prototyping of Distributed User Interfaces, pp. 151–166. Springer Netherlands, Dordrecht (2007)
26. Medeiros, D., Carvalho, F., Teixeira, L., Braz, P., Raposo, A., Santos, I.: Proposal and evaluation of a tablet-based tool for 3D virtual environments. SBC **4**(2), 31 (2013)
27. Melchior, J., Grolaux, D., Vanderdonckt, J., Van Roy, P.: A toolkit for peer-to-peer distributed user interfaces: concepts, implementation, and applications. In: EICS 2009, pp. 69–78. ACM
28. 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, pp. 11–20. ACM (2011)
29. Rauterberg, M., Fjeld, M., Krueger, H., Bichsel, M., Leonhardt, U., Meier, M.: BUILD-IT: a planning tool for construction and design. In: CHI 1998, pp. 177–178. ACM

30. Rekimoto, J.: Pick-and-drop: A Direct Manipulation Technique for Multiple Computer Environments. In UIST 1997, pp. 31–39. ACM
31. Sjölund, M., Larsson, A., Berglund, E.: Smartphone views: building multi-device distributed user interfaces. In: International Conference on Mobile Human-Computer Interaction, pp. 507–511. Springer (2004)
32. Stoakley, R., Conway, M.J., Pausch, R.: Virtual reality on a WIM: interactive worlds in miniature. In: CHI 1995, pp. 265–272. ACM
33. Thevenin, D., Coutaz, J.: Plasticity of user interfaces: Framework and research agenda. In: Proceedings of INTERACT, vol. 99, p. 110117 (1999)
34. Yokoyama, S., Ishikawa, H.: Parallel distributed rendering of html5 canvas elements. In: International Conference on Web Engineering, pp. 331–345. Springer (2011)
35. Zöllner, M., Jetter, H.C., Reiterer, H.: ZOIL: A Design Paradigm and Software Framework for Post-WIMP Distributed User Interfaces. Springer London, London (2011)