3DPlasticToolkit: Plasticity for 3D User Interfaces

Jérémy Lacoche[1,2], Thierry Duval[3,4], Bruno Arnaldi[5,6], Eric Maisel[4,7], and Jérôme Royan[2]

[1] Orange Labs, Rennes, France
[2] IRT b<>com, Rennes, France
[3] IMT ATlantique, Brest, France
[4] Lab-STICC, UMR CNRS 6285, Brest, France
[5] Irisa, UMR CNRS 6074, Rennes, France
[6] INSA de Rennes, Rennes, France
[7] ENIB, Brest, France

Abstract. The goal of plasticity is to ensure usability continuity whatever the context of use. This context must be modeled into the system and possibly taken into account to adapt the final application. The difficulty to handle plasticity for 3D applications comes from the lack of solutions for developers and designers to model and take these constraints into account. This paper introduces new models designed to deal with plasticity for Virtual Reality (VR) and Augmented Reality (AR). These models are implemented in a software solution: 3DPlasticToolkit. It aims to provide a solution for developing 3D applications that can automatically fit any context of use. This context of use includes a set of 3D hardware and environmental constraints, such as user preferences and available interaction devices. 3DPlasticToolkit includes tools for modeling this context and for creating application components independently from it. We propose an adaptation engine based on a scoring algorithm to dynamically create the most suited 3D user interfaces according to the context of use at runtime. We use a furniture planning scenario to show how these adaptations can impact interactions and content presentation.

Keywords: Plasticity, 3D User Interfaces, Virtual reality

1 Introduction

Plasticity is the capacity of an interactive system to withstand variations of both the system physical characteristics and the environment while preserving its usability [20]. Code interoperability is a necessary condition but is not sufficient for an interactive system to be considered as plastic. Usability continuity has to be guaranteed too, and performances and capabilities have to remain at least constant.

Today, there is a growing interest in 3D user interfaces, daily announcements of new interaction devices and the emergence of new kinds of users. In parallel, ubiquitous computing and continuity of access to information are widespread uses in everyday life. Regarding these new constraints, end-users need to have access to plastic user interfaces. Some tools already exist for developing user interfaces that take into account the plasticity property, especially for 2D. However, the problem is even larger for 3D

and for now no solution meets all the plasticity requirements [14]. Indeed, during the development of 3D user interfaces, designers and developers have to handle a lot of input and output devices [2], a lot of interaction techniques [2, 10], a lot of possible kinds of target users and a lot of ways to present content. Manually developing a version of an application for each possible configuration has an important combinatorial complexity and therefore is not a very flexible way toward adapting it to various features.

In this paper, we introduce new models for plasticity that fit 3D constraints and take into account most of the possible adaptation sources and targets as well as static and dynamic adaptations. These models have been implemented in a software solution for 3D application developers: 3DPlasticToolkit, to make possible the development of multiple applications that must be usable in a wide variety of possible contexts of use. We will describe how 3DPlasticToolkit helped us to develop such an application consisting of laying-out an empty room with furniture. It must be available on a wide variety of platforms, from desktop ones to immersive systems through mobile devices. It also must be independent of any concrete interaction devices and new devices must be easily integrated. Last, this application may be used by expert and novice users and needs to be adapted to each particular user's capacities and preferences.

This paper is structured as follows: in section 2, we recall the plasticity requirements for 3D and some related work. In section 3 we present an overview of our solution and in section 4 how it integrates the application model and device model from [13], to which we have added a device model and a task model to enhance the representation of the context. In section 5 we present how the application components are instantiated according to the context with a scoring algorithm and how this adaptation behavior can be modified at runtime. Last, in section 6 we describe how we used 3DPlasticToolkit to develop a furniture planning application in order to show its advantages compared to a state of the art solution.

2 Related Work

2.1 Plasticity requirements for 3D

Considering plasticity for the development of a 3D User Interface can induce a lot of advantages for the developer such as the reduction of the development and maintenance times and costs and the possibility to distribute the application widely. It also benefits the end-user as he/she will get an application that corresponds to his/her needs and that provide him/her usability continuity on his/her different interaction platforms. This is particularly interesting in the field of VR/AR as most users are still novices regarding these interfaces, and as the diversity of platforms is still important. To handle the plasticity property, a 3D toolkit must take into account a set of 3D requirements such as those reported in [14]. These requirements are the following ones:

R1 Deal with the main adaptation sources or be extendable to do so. It includes users, data semantic and hardware characteristics. To be considered as adaptation sources they must be modeled in the system.

R2 Deal with the main adaptation targets or be extendable to do so. It can refer to the presentation of the application content such as a parameter of a 3D object (color, size, etc.), the structural organization of a 3D world, or an interaction technique.

R3 Support the two means of adaptation of plasticity: recasting (modifying locally the application components) and redistribution (changing the distribution of its components statically or dynamically [4] across different dimensions such as platforms, displays, and users). In this paper we focus on local adaptations (recasting), as we already discussed about redistribution in [15].

R4 Ensure code portability. The library must be available on many operating systems (mobile and desktop). Moreover, a toolkit needs to be possibly interfaced with the main VR frameworks and not dependent on a particular one.

R5 Perform dynamic and static adaptations. To ensure usability continuity the system needs to be able to detect a context modification such as a device plugged or a new user configuration at runtime and to update the interface accordingly.

R6 Handle user and system adaptations. The system automatically chooses the best 3D user interfaces according to the adaptation process, this is adaptivity. However, the user must be aware of which adaptation occurs and to be able to modify the aspect of the interface with a set of predefined parameters, this is adaptability.

R7 Be flexible, easy to use and to extend for developers and designers. According to Myers et al. [19], a toolkit and its authoring tool must have a low threshold (easy to use and to learn) while having a high ceiling (how much can be done using them).

2.2 2D solutions

The most common approach for dealing with plastic 2D user interfaces consists of using Model-Driven Engineering. The **CAMELEON** conceptual framework [3] proposes to structure the development process of a user interface into four steps where each step is represented by models: task and concepts (T&C), abstract UI (AUI), concrete UI (CUI), and final UI (FUI). The reconfiguration of the user interface consists in applying transformations at each of these steps according to the context of use to ensure usability continuity. **UsiXML** [17] is an XML based markup language for the development of plastic user interfaces which conforms to the CAMELEON framework and can be used by designers and developers. It proposes a language for the creation of the different models at each development step of CAMELEON. It also introduces a language for the creation of rules for transforming the models according to the context of use.

In the field of pervasive computing, the **Dynamo Framework** [1] uses proxy models and interaction models to maintain a mediation chain that defines multimodal interaction techniques. The system can check the context (services, devices) at runtime and reconfigure itself dynamically. These models let developers focus on interaction techniques development independently from the concrete devices used. However, to avoid wrong associations between interaction techniques and devices, designers or developers have to create pre-defined mediation chains (interaction models). It needs *a priori* knowledge on how the devices will be used and is a lesser automatic approach than describing at a fine grain each device to perform the associations. Moreover, the framework does not include yet the possibility for the user to reconfigure the system and to express his/her preferences.

2.3 3D solutions

Model-driven engineering can also deal with the development of 3D Virtual Environments [7]. The configuration of the 3D content and the deployment on various software and hardware platforms can be automated through code generation. However, the configuration is static, it does not address dynamical context changes.

2D model-based user interface development can be extended to 3D to handle plasticity [8]. User and hardware adaptations are integrated into the user interface development process with model transformations rules described with UsiXML, but the solution focuses on the creation of adaptable 3D widgets and the final user interface is generated as a VRML or X3D file. Therefore, the interaction part is limited.

Viargo [21] proposes to abstract devices by units which provide events to interaction metaphor components. They process the events to update the state of the application. If a device is exchanged at runtime, the interaction metaphor is not disturbed while the new device events are compatible with it. Nevertheless, Viargo only considers hardware as adaptation sources and the interaction techniques as adaptation targets.

The **Grappl library** [9, 16] adapts the interaction techniques to the hardware context. It proposes an association between a high-level task and a set of compatible interaction techniques. To refine the choice, a set of parameters is associated with each task by the designer or by the developer. However, Grappl does not solve the conflict that may happen when different interaction techniques provide the same interaction possibilities. Furthermore, Grappl does not take into account any user preference while a user could prefer an interaction technique to another one for a specific task. Even if the user interface is constructed at runtime, Grappl does not give any solution to deal with context modifications such as a device unplugged.

As Grappl, the **CATHI framework** [18] also aims at adapting the application according to the hardware configuration. It also creates the best user interface according to a set of high-level needed tasks and to the current context of use. It represents a 3D user interface as a graph of interconnected components. The designer selects the high-level tasks to add to this graph. Then, according to the encountered context at runtime, the most suited low-level tasks are connected to the graph. A low-level task represents an interaction technique. It is based on an interaction modality and determines a set of device units that are needed for this interaction modality to be usable. This low-level task is instantiable if all these units are available at runtime. A scoring system is used to avoid conflicts between equivalent possible low-level tasks. The scores are customized by developers or designers with rules that can take the context into account. At runtime, the graph with the higher level score is selected as the current 3D user interface proposed to the user. CATHI handles context modifications at runtime by recreating the interface graph when a modification happens, but it only takes into account device configuration and weather conditions as context information. It does not take user adaptation into account yet, the only possibility given to the user to change the adaptation behavior is to modify the set of rules, which can be difficult for non-expert users.

MiddleVR [11] is a generic VR toolbox that supports many interactions devices and VR displays. Configuration files make it possible to adapt an application to many VR systems without any new coding. Anyway, it manages neither recasting nor redistribution and doesn't provide anything for high-level task description or user preferences.

All these different solutions lack the feature, for the end-users, to check and control adaptation behavior. Even if the created application is considered as the best one by the system, the user may want to try another interaction technique or another device. He must be able to modify the application adaptations at runtime.

3 Solution Overview

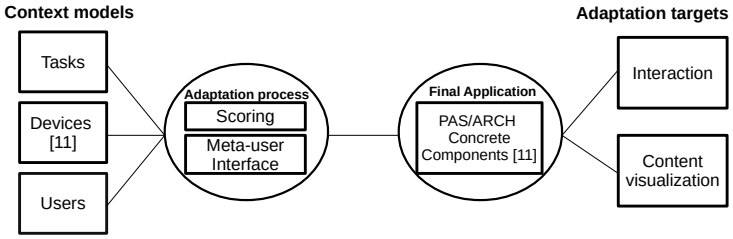


Fig. 1: 3DPlasticToolkit overview. We propose models for three kinds of adaptation sources. The adaptation process can be configured at runtime by the end-user through an integrated user interface: the meta-user interface.

Our toolkit implements new models that aim at satisfying the requirements exposed in Section 2.1: to deal with the main adaptation sources and complete **R1**. It proposes models for these adaptation sources and exposes them to the adaptation process. They correspond to the context models represented in Figure 1. These models can be edited and extended by any developer. For now, data semantic is not modeled in the system and not taken into account as an adaptation source, so it is not addressed in this paper.

First, the device model represents the hardware configuration at runtime. We describe its implementation in [13]. It exposes the available devices' input and output capabilities. This model does not only describe the data acquired or provided by devices, but it also exposes their properties, limitations, and representations in the real world. This model is described with UML class diagrams. It can be edited by any developer who wants to add new properties, input, or output types. To add a new device into 3DPlasticToolkit, the developer must create a new class that inherits from the basic device class and then complete some functions of this class. These functions must fulfill the input data, trigger the outputs and tell the system when a new instance of the device is plugged or unplugged. The device properties corresponding to the device and the description of its units are fulfilled at runtime with an XML description file. The developer can create and edit this XML file with a dedicated graphical tool. He can also edit the dynamic properties at runtime in the device class.

Second, a task model represents at a high level the application behavior and possibilities through a collection of high-level tasks. The application developer or designer provides this collection in a configuration file. Tasks can also be added and removed at runtime using the toolkit API. Tasks expose compatible concrete application components that will be possibly instantiated in the final application. A concrete applica-

tion component is a software element that can be instantiated in the final application to accomplish a task. For instance, it can correspond to the code for a 3D widget or an interaction technique. The compatibility between a task and a concrete application component is ranked: each compatible component is associated with a score. This score can be modified at runtime according to the context. Additional properties can also be included in the task descriptions in the configuration file.

Third, each task is associated with a user to perform user adaptation, especially for taking user preferences into account. This user model can also be edited with a configuration file. It includes the user profile with different properties such as age, size or skill to define a profile for each possible user. It also includes preference scores for the concrete application components, such as a specific interaction technique.

As shown in Figure 1, such an application component can be an interaction technique, a 3D menu, a 3D metaphor for data visualization, etc. This model is an extension of PAC [6] to create application components independent of any concrete interaction devices and that can support alternative concrete representations. It separates the original presentation facet into two different facets. First, the rendering facet handles graphics output and physics. It depends on a particular 3D framework. It can also define its representation in the virtual world, such as the 3D aspect of a widget. Second, the logical driver facet handles input and output devices management for the development of interaction techniques. It describes the way the interaction technique is controlled according to a set of interaction device units described with the device model. The developer must choose these device units to drive correctly the interaction technique. The logical driver can be instantiated if these units are available at runtime. It receives the input data that it needs from concrete devices and it can trigger the outputs. By using this approach we ensure a good decoupling between the application component semantics and its concrete implementation, the independence of the component over the target 3D framework and OS, over the concrete devices used. Moreover, as multiple rendering presentations can be implemented, the same component can have different 3D aspects in the final application. We use a C# implementation of this model but the presentation facets can be easily developed in the language required by the target 3D framework or OS. In that case, a simple wrapper is needed to ensure the interface.

To take into account the adaptation sources and impact the different adaptation targets, we propose to use a scoring algorithm that will drive the application component instantiations and modifications. As shown in Figure 1, this scoring algorithm is one part of our adaptation process. This adaptation process handles local adaptations to support Recasting and partially covers **R3**. This core component of our system receives the different events corresponding to the changes in the context of use and can react accordingly at runtime. Therefore dynamicity is supported and **R5** is covered. Thanks to this mechanism, the optimal usability of the application is always ensured.

One important missing capability in the state-of-the-art solutions is the possibility for the end-user to check and modify the adaptation behavior at runtime. To solve this issue and therefore cover **R6** our toolkit contains a built-in configuration user interface that can be shown and hidden at runtime: the meta-user interface. As shown in Figure 1, this is the second part of our adaptation process. For instance, the meta-user interface allows the end-user to update his/her profile, change concrete application components

or switch from an interaction device to another one. As shown in [15], such an interface can also be used to control the redistribution process of an application developed with our models. Therefore, **R3** can be completely covered.

For now, we only partially cover **R6** by providing a graphical authoring tool, a collection of implemented interaction techniques and some integrated devices.

4 Context representation

In this section, we will only focus on the task and user models, as our device model has already been introduced in [13].

4.1 Task Model

As in Grappl [9] and CATHI [18] we represent a 3D user interface as a composition of high level tasks. Both consider a high-level task component as a self-contained constituent of a 3D user interface. Both solutions focus on interaction tasks. An interaction task corresponds to an action performed by a user via a user interface to achieve a certain goal. In Grappl, each interaction task has a corresponding coding interface that compatible interaction techniques must respect. In the CATHI framework, high-level interaction tasks are connectable components connected to the application logic and low-level interaction tasks.

Our task model does not only focus on interaction tasks. They can refer to interaction techniques, widgets or application logic components. They are elementary tasks that represent the behavior of the application independently from any concrete application component. For now, it does not include the notions of sequences of events and actions that can occur in the application. At runtime, each task is associated with a concrete application component. Each task in the system derives from a basic task class and contains a list of compatible application components developed with the model proposed in [13]. This compatibility is ranked. Indeed, some application components can be considered more suited than others. Therefore, a compatibility score is assigned to each application component. This compatibility list has to be edited in an XML file. For a given task, the developer has to give the list of the control facets names that correspond to the compatible application components. The names of these control facets are associated with the compatibility scores. This XML file also contains the compatibility scores between control facets, rendering presentations and logical drivers for the application model. Indeed, for a given application component, some rendering presentation and logical driver can be considered more suited than others by the application developer. An excerpt of this XML file for the creation of this list is given in Listing 1.1. It corresponds to the compatible application components for the selection and manipulation task given in Figure 2. To associate the component names with concrete code instances, we use a factory design pattern in which the developer has to register his/her components. As the compatibility can also depend on the context of use, these scores can be edited at runtime, which can result in modifications in the final application. This compatibility list is exposed to the system to allocate the best application components according to the desired tasks and the current context of use. To illustrate this task

```
1 <TaskCompatibility taskName="TaskSelectionManipulation" componentName="
      Ray3DC" score="1.0"/>
2 <TaskCompatibility taskName="TaskSelectionManipulation" componentName="
      Cursor3DC" score="0.8"/>
3 <TaskCompatibility taskName="TaskSelectionManipulation" componentName="
      Proximity3DC" score="0.5"/>
```

Listing 1.1: Excerpt of an XML file describing compatibilities between tasks and application components.

model, Figure 2 and Listing 1.1 give an example of a selection and manipulation task. This task is compatible with three interaction components: a 3D ray component with a score of 1.0, a 3D cursor component with a score of 0.8, and a 3D proximity component with a score of 0.5. This association can let the system perform user adaptation.

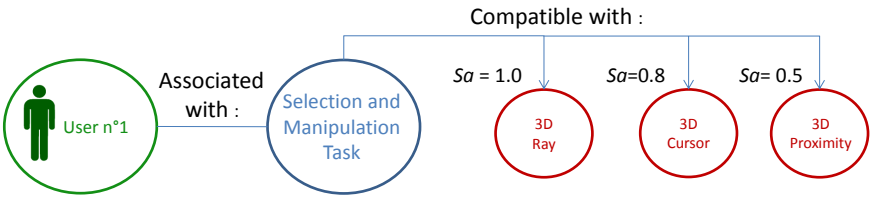


Fig. 2: An example of high level task: selection and manipulation, compatible with three concrete application components.

The developer can also include some parameters into the task as key-value properties. At runtime, an application component will have access to its corresponding high-level task and therefore its parameters. In the example of the manipulation task given in Figure 2, we can parametrize the degrees of freedom on which objects can be manipulated. An application control task could be parameterized with a tree that defines the possible choices of a menu. Dependent tasks can also be defined by the developer or the designer to indicate if a task needs another task to be completely performed. For example, a 3D menu would need a selection interaction task to be achieved.

To represent an interaction task as an action performed by a user via a user interface, we associate each task to a particular user (see Figure 2). This user is described according to the model presented in the next subsection.

To add a new task to the system, the developer must create a new class that inherits from a basic task class. To select which tasks will have to be performed in a particular application, the developer must fulfill an XML configuration file. This file also contains the parameters of the tasks. As for the device model, a graphical tool is provided to perform this configuration to make it usable by any designer. For now, key-value properties in a string format can be extracted from the configuration file for each task. Custom

```
1 <TaskConfig>
2 <NeedTask userId="1" taskName="SelectionManipulation" taskId="0"
      ScoringModule= "User" >
3 </NeedTask>
4 <NeedTask userId="1" taskName="FurnitureControl" taskId="1" topTask="0"
      ScoringModule="Default"/>
5 <NeedTask userId="1" taskName="Navigation" taskId="2" ScoringModule="
      User"/>
6 </NeedTask>
7 </TaskConfig>
```

Listing 1.2: Task configuration file with 3 high level tasks

properties can also be included in XML nodes in the configuration file. The parsing of these nodes must be implemented in the corresponding task class by the developer. As an example, the configuration file for the furniture planning application detailed in Section 6 is given in Listing 1.2. The application is composed of three tasks. The first one is a selection and manipulation task for menu selection and moving 3D objects into the scene. he second one is an application task for adding furniture into the room with a menu. A menu needs a selection mechanism so this task is defined as dependent on the first one in the configuration file. Last, the third one is a navigation task that is needed to move the user's point of view into the room. For each task, there is a parameter named "Scoring Module", it allows the developer to choose how the compatibility scores will be taken into account. The possible choices and the impact on adaptations are described in Section 5.

4.2 User Model

The goal of the user model is to describe the users who will interact with the application and therefore perform user adaptations. This user model must contain two kinds of information.

First, the user profile contains the different properties that characterize the user. These properties can be for example the user age, his/her gender, his/her level of expertise. In the user model, they are represented as key-value properties.

Second, the user model contains user preferences. These preferences are represented as scores that will be taken into account by the adaptation process, described in Section 5, to instantiate the application components at runtime. Multiple scores can be contained in the user model:

- As proposed in Section 4.1, a score *Sa* represents the compatibility between a high-level task and a concrete application component. The user can also express how he/she perceives this compatibility by including a score *Sau* in his/her model. For instance, it can tell the system which interaction techniques the user prefers.
- In the application model, the scores *Sld* represent compatibility between an application component and a logical driver. The user model includes scores *Sldu* to expose

```
<UserConfig>
<User userId="1">
<Name value="Bernard"/>
<Age value="35"/>
<Gender value='M'/>
<Expertise value="Novice"/>
<UserPrefComponent Name="3DRay" Task="SelectionManipulation" score="1.0"
    >
   <UserPrefDriver Name="3DRayGamePadDriver" score="0.3" />
   <UserPrefDriver Name="3DRay6DofDriver" score="1.5" />
   <UserPrefDriver Name="3DRayMouseDriver" score="0.5" />
</UserPrefComponent>
<UserPrefComponent Name="3DCursor" Task="SelectionManipulation" score="
    0.8">
  (...)
</UserPrefComponent>
<UserPrefComponent Name="3DProximity" Task="SelectionManipulation" score
    ="0.5">
  (...)
</UserPrefComponent>
</User>
</UserConfig>
```

Listing 1.3: User configuration file with only one user described.

preferences for this compatibility. For example, the user may prefer using some kinds of devices to control a specific interaction technique.

– In the same way, the scores *Srp* represent compatibility between an application component and a rendering presentation facet. The user model can also expose preferences for these facets with scores *Srpu*. For instance, it can express a preference for a particular representation of a 3D widget.

As for the task model, an XML configuration file is used to edit the profiles and preferences of the users. One example of an XML configuration file for one user is given in Listing 1.3. The file contains the user properties as well as his/her preferences. An authoring tool makes it possible to perform this configuration. At runtime, 3DPlasticToolkit API allows the developer to add dynamic properties and to modify the user preferences. Listing 1.4 gives an example of code with some modifications for this user. Here, the level of expertise of the user is updated, he is now considered as an expert. Moreover, his/her preference score for the 3DCursor component increased.

5 Adaptation process

The adaptation process consists in taking into account the description of the context of use to adapt application components developed with the model proposed in [13]. Its

```
int indexUser = 1 ;
IndividualUser user = UserManager.Instance().getUser(indexUser) ;
user.setProperty("Expertise" , "Expert") ;
float newScore = 2.0 ;
user.setPreferenceComponent("3DCursor", "SelectionManipulation",
    newScore) ;
```

Listing 1.4: Properties and preferences modified at runtime with the user API.

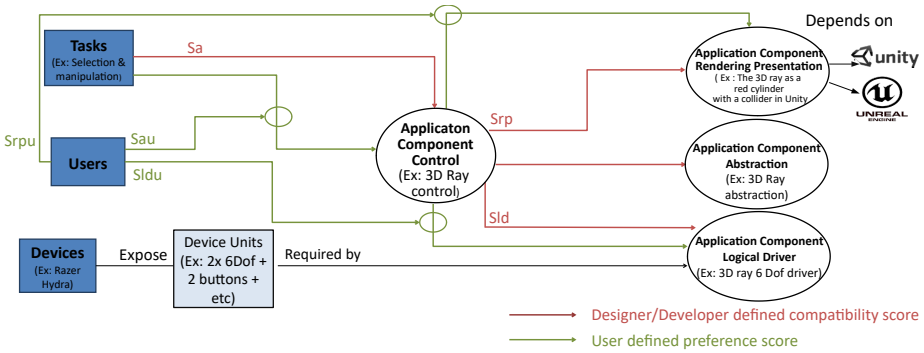


Fig. 3: A summary of the application model and the context models.

goal is to provide always the best application to cover **R4** and **R5**. It is divided into two parts. First, there is an adaptation engine on top of all application components, which continuously checks the current context and drives the instantiation and modification of these agents through communication with all supervision control facets. Second, a graphical meta-user interface lets the end-user check and modify the adaptation behavior at runtime.

5.1 Adaptation engine

We propose to use a scoring system combined with our application model. This scoring system uses the compatibility scores given by the developer or the designer and exposed in the application model and the task model. It also uses the preferences scores contained in the user model. The goal is to use these different scores to maximize the usability of the final application.

For a given task, multiple components can be compatible, so the developer or the designer can rank each compatible application component with a score *Sa*. In the same way, for a given application component, a compatible rendering presentation will be ranked with a score *Srp* and a logical driver with a score *Sld*. Theses scores are edited separately in an XML configuration file. In the user model, each user has his/her own preferences for these compatibilities and therefore the model contains the corresponding scores: *Sau*, *Srpu*, and *Sldu*. This ranking is illustrated Figure 3.

At runtime, the construction of the 3D user interface consists in associating to each task the best triplet (application component, logical driver, rendering presentation). The

goal of the adaptation engine is to find the triplet which maximizes the score of compatibility to create the application that uses the most suited devices and with the most suited content presentation. For a given triplet, its score is computed according to the compatibility scores and the user preferences scores. As said in Section 4.1, for each task there is a parameter named "Scoring Module". The score assigned to a triplet for a given task will depend on its associated scoring module. A scoring module corresponds to a software component that implements a particular score computation according to the scores exposed to the system. Three built-in modules can be chosen:

- **Default module**: this module only considers the compatibility scores provided by the designer or the developer. The goal is to provide them the maximum control over the adaptation process. The score for a given triplet is: $S = Sa + Sld + Srp$
- **User module**: this module only uses the scores extracted from the user preferences. The goal is to provide the application that corresponds as much as possible to the user's needs. The score for a given triplet is: $S = Sau + Sldu + Srpu$
- **Combination module**: this module proposes to combine the developer's scores with the user's ones. It provides a good compromise between the user's preferences and the developers' and designers' choices. The score for a given triplet is: $S = (Sa + Sau) + (Sld + Sldu) + (Srp + Srpu)$

This system gives to designers and developers a good control over the adaptation process to build a final application that will best fit the user needs. Moreover, the 3DPlasticToolkit API gives also the possibilities for developers to integrate new scoring modules. Therefore, it gives them the flexibility to deeply control the adaptation process if they have specific needs.

The scoring process is performed at every context modification, it ensures to detect application components that are not available or suited any longer or more suited ones:

1. Context modification is detected. For example, it can be the connection of a new device, the add of a task, or the association of a task to a newly detected user.
2. This modified context is transmitted through all supervision controls currently instantiated. For each one, we check if the associated logical driver is still possible in the current context of use. It is still possible if the devices that it uses are still plugged and available. If not, the application component is destroyed and the associated task is classified as *not done*.
3. For each not done task, 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 the logical driver needed device units 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. We attribute a score for each triplet according to the previously described computation. Then, if multiple triplets are found, the one that obtains the best score is used to instantiate the PAC agent with the suited logical driver and rendering presentation. The devices units associated with the logical driver are set as *not available*. The task is classified as done
4. For each done task that has not been processed in the previous step, we check if we can find a triplet more suited than the current one. To do so, 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 suited. Conversely, we destroy the current application component and we instantiate the new best choice. For now, this choice is directly applied but it could be only suggested to the user to produce a less disturbing effect.

To illustrate this process, a usage scenario is given in Figure 4. It shows the different steps of the adaptation process in 3DPlasticToolkit when context changes happen. In this example, we focus on the instantiation of application components for the following tasks: navigation and object manipulation. This adaptation scenario can correspond to the application described in Section 6. In this example, the context change consists of modifications of the available devices.

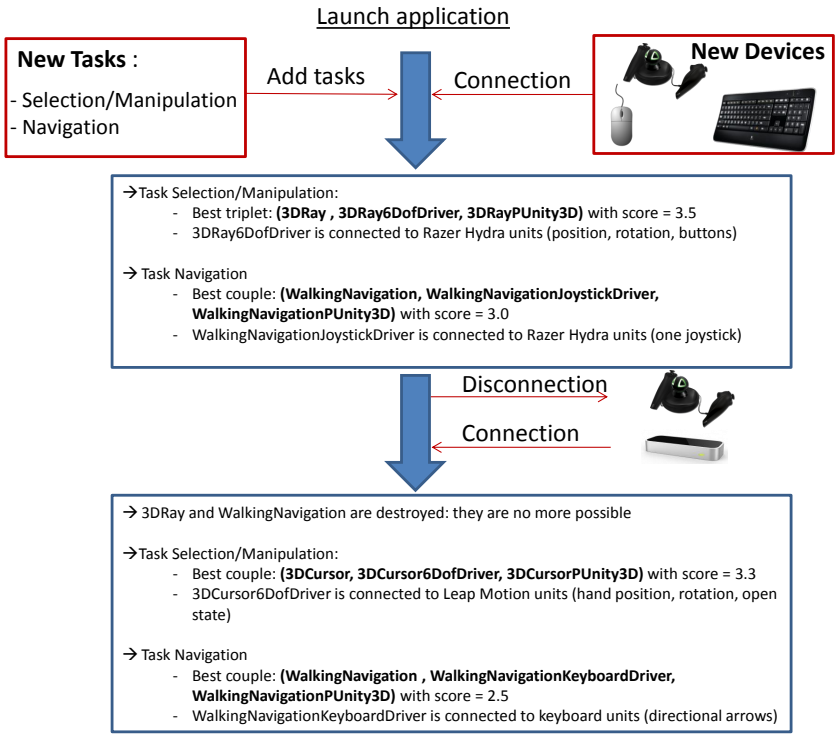


Fig. 4: Example of adaptation process.

In this scenario, when the application is launched, three devices are connected to the computer: a keyboard, a mouse, and a Razer Hydra. The Unity3D game engine is used. In the XML configuration file of the application, the Oculus Rift HMD is chosen as the main display. Therefore, the associated device class configures the device SDK and the 3D rendering accordingly. The launch of the application induces the instantiation of a

component for each task. The score assignation for the components is made with the user module according to the scores detailed in Listing 1.3:

1. For the selection and manipulation task, according to the devices available the possible choices are:
   - (3DRay, 3DRay6DofDriver, 3DRayPUnity3D) with $S = 1.0 + 1.5 + 1.0 = 3.5$
   - (3DCursor, 3DCursor6DofDriver, 3DCursorPUnity3D) with $S = 0.8 + 1.5 + 1.0 = 3.3$
   - (3DProximity, 3DProximity6DofDriver, 3DProximityPUnity3D) with $S = 0.5 + 1.5 + 1.0 = 3.0$
   - (3DRay, 3DRayMouseDriver, 3DRayPUnity3D) with $S = 1.0 + 0.5 + 1.0 = 2.5$
   - (3DRay, 3DRayGamePadDriver, 3DRayPUnity3D) with $S = 1.0 + 0.3 + 1.0 = 2.3$
   - (3DCursor, 3DCursorGamePadDriver, 3DCursorPUnity3D) with $S = 0.8 + 0.2 + 1.0 = 2.0$
   - (3DProximity, 3DProximityGamePadDriver, 3DProximityPUnity3D) with $S = 0.5 + 0.2 + 1.0 = 1.7$

   The combination chosen is (3DRay, 3DRayDriver6Dof, 3DRayPUnity3D) because it gets the best score (3.5). Therefore, the application component is instantiated. The user can now manipulate the objects with a 3D Ray-based interaction technique controlled with the Razer Hydra.
2. For the navigation task, the possibilities are:
   - (WalkingNavigation, WalkingJoystickDriver, WalkingNavigationPUnity3D) with $S = 1.0 + 1.0 + 1.0 = 3.0$
   - (WalkingNavigation, WalkingKeyboardDriver, WalkingNavigationPUnity3D) with $S = 1.0 + 0.5 + 1.0 = 2.5$

   The chosen triplet is (WalkingNavigation, WalkingDriverJoystick, WalkingNavigationPUnity3D) because it gets the best score (3.0). It corresponds to an interaction technique based on a walking navigation metaphor controlled with one of the joysticks provided by the Razer Hydra. This metaphor can be compared to the "WALK" navigation type from X3D. The joystick is used to move the point of view forward and backward as well as changing its rotation around the up axis.

The logical driver instantiated for the navigation task also describes a tactile output as an optional output. A vibration is triggered when colliding a virtual object. As this output is optional, the logical driver can be instantiated without it. Indeed, the Razer Hydra does not provide any vibration capabilities.

After a few minutes, someone asks the user for the Razer Hydra. The user does not want to stop using the application because he/she has not finished to lay the items of furniture out. In exchange for the Razer Hydra, he gets a Leap Motion. As the Razer Hydra is disconnected, the two currently instantiated application components are destroyed because they are not possible anymore. Then, with the connection of the Leap Motion the interface is rebuilt as follows:

1. For the selection and manipulation task, some triplets are not possible any longer, the list of possible ones is updated:

- (3DCursor, 3DCursor6DofDriver, 3DCursorPUnity3D) with $S = 0.8 + 1.5 + 1.0 = 3.3$
- (3DProximity, 3DProximity6DofDriver, 3DProximityPUnity3D) with $S = 0.5 + 1.5 + 1.0 = 3.0$
- (3DRay, 3DRayMouseDriver, 3DRayPUnity3D) with $S = 1.0 + 0.5 + 1.0 = 2.5$
- (3DProximity, 3DProximityGamePadDriver, 3DProximityPUnity3D) with $S = 0.5 + 0.2 + 1.0 = 1.7$

The triplet that gets the best score (3.3) is now (3DCursor, 3DCursor6DofDriver, 3DCursorPUnity3D). It corresponds to a 3D cursor controlled with a driver that implements a 6-DoF interaction. The position and rotation of the cursor are controlled through one hand detected by the Leap Motion. The open or closed state of the hand is used as the signal for selection and deselection.

2. For the navigation task, no joysticks are available any longer. One triplet is possible and therefore is instantiated: (WalkingNavigation, WalkingKeyboardDriver, WalkingNavigationPUnity3D). The walking navigation component is now controlled with the directional keys of the keyboard. The up and down keys are used to move forward and backward while the left and right ones are used to rotate the point of view around the up axis.

This usage scenario demonstrates the ability of the proposed model to handle context modification to ensure usability continuity of any 3D user interface. Moreover, with the use of a scoring system, this usability is maximized according to the current context of use. In the next section, we demonstrate how this adaptation mechanism can be configured at runtime by the end-user thanks to an integrated graphical user interface: the meta-user interface.

5.2 The meta-user interface

One of the identified drawbacks in current solutions is the lack of control given to the user over the adaptation process. Indeed, the user must be aware of any system adaptation and be able to modify the result (requirement **R5**). Therefore, our solution proposes a built-in application component that implements a graphical user interface (GUI) that satisfies this need: the meta-user interface. The aspect of the meta-user interface is chosen thanks to our scoring mechanism presented before. It can be a 3D menu placed in the 3D world for an AR or VR system, or it can be a 2D menu placed on top of the virtual world. The menu can be shown and hidden at runtime and the user can interact with it thanks to the different selection techniques integrated into 3DPlasticToolkit. The meta-user interface provides the end-user with a view of the current state of the system and gives him the possibility to modify the following aspects of the application:

- For each task, the user can see the currently associated application component and can select another one to achieve it. Only the possible ones are proposed to the user. For instance, it can be used to switch from an interaction technique to another one.
- For each instantiated application component, the user can see the associated logical driver and can select another one to control it. Similarly to the interaction techniques, only the possible logical drivers are proposed. For example, it can be used to change the kinds of devices that control an interaction technique.

- For each instantiated application component, the user can see the associated rendering presentation and can select another one. Only the possible ones are proposed. For example, it can be used to change the aspect of a 3D widget at runtime.
- For each logical driver, the user can see all associations between actions and device units. The user can change each associated device unit in a list of compatible and available ones.
- The user can edit his/her profile. It can be a modification of his personal properties such as his/her age or expertise and the modification of his/her preference scores. These modifications are taken into account as context changes by the adaptation process.

These features are used by the adaptation process to modify the user model. Indeed, the preferences and profiles modifications are directly reported and saved in the user model. In the same way, the choices for the application components, logical drivers, and the rendering presentations are also used to update the user model. Indeed, when a user changes one of these components in the meta-user interface and keeps the new one until the end of the application, the preferences scores of the two components are swapped. The goal is to learn from the user's habits to automatically adapt the application.

6 Development of a Furniture Planning Application with 3DPlasticToolkit

To demonstrate the benefits of 3DPlasticToolkit, we developed a furniture planning application. We developed this application with 3DPlasticToolkit to evaluate its capabilities. We compared the results to what could be obtained with MiddleVR, one solution introduced in Section 2 and the only one of them available online. The goal of this application is to help a customer to plan the use of premises, here a room rented for special events. As this room can be under construction or too far for a real guided tour, we propose to immerse the customer into a virtual version of the premises. The customer has the capability to layout the room with furniture (add, remove, move). These features help him/her to understand the potential of the free space and make it possible for him/her to imagine and plan how space will be used.

We exploited a library of existing 3D objects (the room and the items of furniture) to implement this application. We use 3DPlasticToolkit or MiddleVR to develop its interactive capabilities. The first step consists of importing those objects into Unity3D and create a scene with the empty room and with the 3DPlasticToolkit or MiddleVR scripts. The second step in the development consists of configuring the interactive capabilities of the application. With 3DPlasticToolkit, the developers select the high-level task that will describe this application. At the task level, the furniture planning application is composed of three different tasks:

- **The "SelectionManipulation" Task.** The goal of this task is to give the possibility to the end-user to select and move furniture in the room.
- **"The "Navigation Task"** With this task the user will be able to move his point of view in the scene with a navigation interaction technique.

– **The ”FurnitureControl”.** This task has been specially built for this application. It contains different events that can be triggered by the associated application component such as adding an item of furniture, saving the current layout, and loading a pre-defined one.

The XML task configuration file of this application is provided in Listing 1.2. MiddleVR does not introduce such an abstraction for interaction techniques. Indeed, they need to be chosen in Unity3D at development time, preventing from changing them at runtime or between sessions. We then chose a 3D ray-based selection/manipulation technique and a joystick-based navigation technique.

Regarding those tasks, 3DPlasticToolkit components for navigation and selection/-manipulation were already developed as they are interaction capabilities required for 3DUser interfaces. Examples of selection/manipulation techniques included in 3DPlasticToolkit are described in [13]. MiddleVR also includes the same built'in interaction techniques for selection/manipulation and navigation. For the ”FurnitureControl” task, we had to develop a dedicated application component for both MiddleVR and 3DPlasticToolkit. This application component implements a graphical menu with the required features (adding objects, load, save). The rendering presentation facets dynamically check which kind of device is used to place this menu in the 3D space (in immersive mode) or screen space (in 2D mode). For instance, with 3DPlasticToolkit, as shown in Figures 5a and 5b, for desktop and immersive setups, the menu is placed in the 3D space and can be moved, while on a tablet it is static and it overlays the application as seen in Figure 5c.

Then, our goal is to deploy this application on multiple platforms and for users with different profiles. Here, we describe three types of platforms that can be used and how the application is adapted. 3DPlasticToolkit discovers the capabilities of the devices dynamically and changes can be handled at runtime, while MiddleVR requires a graphical user interface to make a configuration which cannot be changed at runtime.

First, the application can be used on a desktop platform (see Figure 5a). This platform is simply composed of a monitor, a mouse, and a keyboard. With MiddleVR, the basis of the ray is controlled with the mouse by simulating a 6-DoF tracker. This simulation is limited at it only allows to move the ray backward and forward and to rotate it around the Up axis. The application is then not completely usable. Selection/-Manipulation is confirmed with one button of the mouse and navigation is controlled with keyboard buttons simulating a joystick. With 3DPlasticToolkit, a 3D ray-based interaction technique is proposed for the selection and manipulation task. The logical driver uses the mouse position to control the ray extremity and the buttons for selecting and grabbing. For the navigation task, a walking navigation metaphor is deployed. The associated logical driver uses the arrows of the keyboard to translate and rotate the user's point of view. For the furniture application control task, a 3D graphical menu is deployed and placed according to the user's point of view. In such a situation, the user stands in front of his PC and he could decide to plug another device at runtime to benefit from more advanced 3D interactions. A similar adaptation scenario as the one described in Section 5.1 could be considered with the furniture planning application. Indeed, the plugging of a new device such as Razer Hydra or of a leap motion would



(a) (b) (c)

Fig. 5: Three different types of platforms on which the furniture application can be run with 3DPlasticToolkit: (a) a desktop platform (b) an immersive platform (an HTC Vive), (c) a mobile platform (a tablet).

result in modifications of the deployed application components. Such a situation cannot be handled with MiddleVR.

Second, as shown in Figure 5b, the application can be used on an immersive platform. Here, we use an HTC Vive that is composed of an HMD and two 6-DoF controllers with buttons and trackpads. With MiddleVR, the application can be easily configured. The 3D-ray interaction technique can be associated with one of the two controllers, and one of the trackpads can be used for navigation. With 3DPlasticToolkit, for the selection and manipulation task, a virtual hand is used to select and catch the scene objects. The logical driver uses the position and the rotation of one of the controllers to set the hand's position and rotation. One button of the controller is used to close the hand and grab an object. For the navigation task, the user can navigate at scale one in the area defined by the head tracking zone. Moreover, we combine it with a teleportation capability. With the second controller, the user can use a ray-based interaction technique to select a point in the scene where he wants to be teleported. As for the desktop platform, the graphical menu is also deployed in the 3D space for the furniture application control task. In this example, multiple alternatives for the two first tasks could be deployed. Indeed, a 3D-ray could also be controlled with a 6-DoF controller to select and manipulate the scene objects. As well, the teleportation navigation technique could also be replaced by a navigation technique that exploits the trackpad of one of the controllers to rotate and move forward and backward as in MiddleVR. For the "FurnitureControl" task, an alternative of the 3D menu could be to propose a concrete application component based on vocal recognition. Contrary to MiddleVR, with 3DPlasticToolkit such adaptation can be intended directly by the end-user. Indeed, he could edit his/her preferences score or he could use the meta-user interface at runtime to change the deployed interactions as detailed in Section 5.2.

Third, as shown in Figure 5c, the application can be used on a mobile platform. Here the platform is an Android tablet. With 3DPlasticToolkit, it requires a specific build of the application for Android configured in Unity3D. However, mobile platforms are not handled by MiddleVR and therefore our application cannot be deployed with this solution. With 3DPlasticToolkit, for the selection and manipulation task, a 2D cursor interaction technique is deployed on the tablet. The chosen logical driver uses the multi-

touch capabilities of the tablet. 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. The graphical menu for the application control task is also deployed on this platform and overlays the whole application in screen space. It can be shown and hidden.

To conclude this section, this comparison shows that MiddleVR cannot meet all plasticity requirements. In particular, MiddleVR fails to cover **R2** and **R5** as it cannot propose dynamic adaptation of interaction techniques. Usability continuity is not always ensured with MiddleVR. As well, as it is only available on Windows it cannot cover **R4**. With MiddleVR, the end-user is also partially excluded from the adaptation process, and then we cannot cover **R1** and **R6**. However, contrary to the current implementation of 3DPlasticToolkit, MiddleVR can handle clustering and can deploy to a CAVE system. That is why we developed an interface between 3DPlasticToolkit and MiddleVR to exploit these capacities as described in [15]. Moreover, we consider that MiddleVR tools are very easy to use for developers and better cover **R7** so far. Indeed, the solution only relies on a graphical user interface and on Unity3D scripts while 3DPlasticToolkit may require some coding, the use of not yet optimized graphical user interfaces and manual edition of XML files. To confirm our assumptions, we plan to set up a formal comparison between these tools with experienced developers.

7 Conclusion

3DPlasticToolkit is a toolkit that supports plasticity for 3D user interfaces, it relies on three models to represent the context of use, including hardware, task and user configurations. Its adaptation process is based on a scoring algorithm that takes into account the context of use to create the most suited 3D user interface. This adaptation engine can be configured at runtime with a built-in graphical tool: the meta-user interface.

This solution covers most of the plasticity requirements for 3D. Moreover, these models have also been extended to support redistribution as a mean for adaptation [15].

To demonstrate the capabilities of 3DPlasticToolkit and its differences with state-of-the-art solutions, multiple examples are given in the paper based on the development of a furniture planning application. This application is now totally usable and meets its plasticity requirements.

Our future work will consist in fulfilling the current lacks of our toolkit to meet all plasticity requirements.

Exploring scenario engines to complete the task model: The task model allows the developer to model the application behavior and possibilities at a high level. However, it does not let him orchestrate the sequences of events and actions that can occur in the virtual environment. This is the goal of scenario engines such as SEVEN [5] that could be associated with our toolkit.

Toolkit completion and validation: For now, our solution mainly focuses on helping developers for the creation of plastic 3D user interfaces. However, authoring tools are not completely developed and are not ready to use yet by designers. More work should be done on that point before a possible evaluation.

Exploring new solutions for learning user preferences: For now, the user's preferences scores are declared for each user. It would be interesting to analyze the current user profile and the current task parameters, to automatically determine the user preferences scores, as suggested in [12] to build general, group and individual user models, by using machine learning algorithms.

8 Reference to the official publication

Springer Nature Switzerland AG 2019
P. Bourdot et al. (Eds.): EuroVR 2019, LNCS 11883, pp. 1–22, 2019.
The final authenticated version is available online at:
https://doi.org/10.1007/978-3-030-31908-3_5

References

1. Avouac, P.A., Lalanda, P., Nigay, L.: Autonomic management of multimodal interaction: DynaMo in action. In: EICS 2012. pp. 35–44. ACM New York, NY, USA, Copenhagen, Denmark (2012)
2. Bowman, D.A., Kruijff, E., LaViola, J.J., Poupyrev, I.: 3D User Interfaces: Theory and Practice. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA (2004)
3. Calvary, G., Coutaz, J., Thevenin, D.B., L., Florins, M., Limbourg, Q., Souchon, N., Vanderdonckt, J., Marucci, L., Paterno, F., Santoro, C.: The CAMELEON Reference Framework. Deliverable D1.1 (2002)
4. Calvary, G., Coutaz, J., Dâassi, O., Balme, L., Demeure, A.: Towards a new generation of widgets for supporting software plasticity: the "comet". In: EICS 2005, p. 306–324. Springer
5. Claude, G., Gouranton, V., Bouville Berthelot, R., Arnaldi, B.: #SEVEN, a Sensor Effector Based Scenarios Model for Driving Collaborative Virtual Environment. In: ICAT-EGVE (Dec 2014)
6. Coutaz, J.: PAC, on object oriented model for dialog design. In: Interact'87 (1987), 6 pages.
7. Duval, T., Blouin, A., Jézéquel, J.M.: When model driven engineering meets virtual reality: Feedback from application to the collaviz framework. In: 7th Workshop SEARIS (2014)
8. Gonzalez-Calleros, J., Vanderdonckt, J., Muoz-Arteaga, J.: A structured approach to support 3d user interface development. In: Advances in Computer-Human Interactions, 2009. ACHI '09. Second International Conferences on. pp. 75–81 (Feb 2009)
9. Green, M., Lo, J.: The grappl 3d interaction technique library. In: VRST 2004. p. 16–23. ACM, New York, NY, USA
10. Hand, C.: A survey of 3d interaction techniques. In: Computer graphics forum. vol. 16, p. 269–281 (1997)
11. Kuntz, S.: Middlevr a generic vr toolbox. In: 2015 IEEE Virtual Reality (VR). pp. 391–392 (March 2015)
12. Lacoche, J., Duval, T., Arnaldi, B., Maisel, E., Royan, J.: Machine Learning Based Interaction Technique Selection For 3D User Interfaces. In: EuroVR 2019. p. to appear. Springer, Lecture Notes in Computer Science
13. Lacoche, J., Duval, T., Arnaldi, B., Maisel, E., Royan, J.: Plasticity for 3D User Interfaces: new Models for Devices and Interaction Techniques. In: EICS 2015. ACM
14. Lacoche, J., Duval, T., Arnaldi, B., Maisel, E., Royan, J.: A survey of plasticity in 3D user interfaces. In: 7th Workshop SEARIS (2014)

15. Lacoche, J., Duval, T., Arnaldi, B., Maisel, É., Royan, J.: D3part: A new model for redistribution and plasticity of 3d user interfaces. In: 3D User Interfaces (3DUI), 2016 IEEE Symposium on. IEEE (2016)
16. Lee, W.L., Green, M.: Automatic layout for 3d user interfaces construction. In: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications. p. 113–120 (2006)
17. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V.: Usixml: a language supporting multi-path development of user interfaces. pp. 11–13. Springer-Verlag (2004)
18. Lindt, I.: Adaptive 3D-User-Interfaces. Ph.D. thesis (2009)
19. Myers, B., Hudson, S.E., Pausch, R.: Past, present, and future of user interface software tools. ACM Transactions on Computer-Human Interaction (TOCHI) **7**(1), 3–28 (2000)
20. Thevenin, D., Coutaz, J.: Plasticity of user interfaces: Framework and research agenda. In: Proceedings of INTERACT. vol. 99, p. 110–117 (1999)
21. Valkov, D., Bolte, B., Bruder, G., Steinicke, F.: Viargo - a generic virtual reality interaction library. In: 5th Workshop SEARIS (2012)