Plasticity for 3D User Interfaces: new Models for Devices and Interaction Techniques
Jérémy Lacoche, Thierry Duval, Bruno Arnaldi, Éric Maisel, Jérôme Royan

To cite this version:

HAL Id: hal-01175000
https://hal.archives-ouvertes.fr/hal-01175000
Submitted on 10 Jul 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Plasticity for 3D User Interfaces: new Models for Devices and Interaction Techniques

Jérémy Lacoche
IRT b<>com, UMR CNRS 6074 Irisa - Inria Rennes
ejeremy.lacoche@b-com.com

Thierry Duval
UMR CNRS 6285
Lab-STIC, Telecom Bretagne, IRT b<>com
thierry.duval@telecom-bretagne.eu

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

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

Jérôme Royan
IRT b<>com
jerome.royan@b-com.com

Figure 1: Different ways to complete a Selection and Manipulation task with the proposed interaction model (a) A 3D-ray based interaction controlled by a razer hydra is used (b) Without this device connected, the ray is now controlled with the mouse (c) With the leap motion, another interaction technique is chosen, a 3D-cursor controlled with the user's hand.

ABSTRACT
This paper introduces a new device model and a new interaction technique model to deal with plasticity issues for Virtual Reality (VR) and Augmented Reality (AR). We aim to provide developers with solutions to use and create interaction techniques that will fit to the needed tasks of a 3D application and to the input and output devices available. The device model introduces a new description of inputs and outputs devices that includes capabilities, limitations and representations in the real world. We also propose a new way to develop interaction techniques with an approach based on PAC and ARCH models. These techniques are implemented independently of the concrete devices used thanks to the proposed device model. Moreover, our approach aims to facilitate the portability of interaction techniques over different target OS and 3D framework.

Author Keywords
3D User Interfaces; Virtual Reality

ACM Classification Keywords
H.5.1 Information interfaces and presentation: Multimedia Information SystemsArtificial, augmented, and virtual realities; H.5.2 Information interfaces and presentation (e.g. HCI): User Interfaces

INTRODUCTION
In the last years, interest in 3D user interfaces has grown [2]. For instance, 3D user interfaces include Virtual Environments (VE), serious games or Computer-Aided Design (CAD) applications. They differ from traditional graphical user interfaces (GUI) by including a third dimension to present content and by using a wider range of interaction devices than the traditional set: mouse, keyboard and touch-screen.

With the wide variety of existing interaction devices and the daily emergence of new ones, it is difficult and time consuming for developers to adapt their applications to each possible...
configuration. For instance, in these new devices we can find the Microsoft® Kinect™, the Leap Motion or the Oculus rift. More than using each time a new SDK, these devices lead developers to redevelop parts of their applications, particularly interaction techniques in order to fit each capability.

Developing an application that takes into account the plasticity property is one solution to deal with these issues. Plasticity is the capacity of an interactive system to withstand variations of both the system physical characteristics and the environment while preserving its usability [17]. Code interoperability and usability continuity has to be guaranteed to be considered as plastic. Plasticity has already been well explored in the field of 2D user interfaces, however it has been reported in [12] that the problem is even larger for 3D and that no solution meets all the plasticity requirements.

Our goal is to propose new solutions for developers to handle plasticity for 3D user interfaces. In this paper, we focus on interaction techniques. In [3], an interaction technique is defined as a method allowing a user to accomplish a task via the user interface. It includes both hardware (input/output) and software components. We want to be able to define interaction techniques that will work whatever the target context of use. This context consists in a description of the available devices through a new device model and a high level description of the tasks that have to be performed. This device model aims to describe devices in terms of data they can collect and provide. It differs from classic approaches by also including information such as intrinsic properties, limitations and representation in the real world. Then, the interaction technique model based on PAC and ARCH models gives the possibility to developers to implement interaction techniques independently of devices and of a target 3D framework.

This paper is structured as follows: first we recall the plasticity requirements for 3D that our models have to meet. Then, we introduce our new device model used in the context description. Next, we present an extension of the PAC model [7] for interaction techniques implementation that ensures independence from the devices and from the target VR framework. Last, we conclude and give directions for future work.

REQUIREMENTS

To handle the plasticity property for interaction techniques, our models have to take into account a set of 3D requirements such as some reported by Lacoché et al. in [12]:

R1 Ensure code portability. Interaction techniques must be available on many Operating Systems (mobile and desktop). Moreover they need to be possibly integrated or implemented with the main 3D frameworks (for instance, a game engine) and not dependent on a particular one. Indeed, each developer may have his own code database or 3D content in a particular framework.

R2 Independence over the devices used. Interaction techniques must work whatever the concrete devices available. The devices needed by an interaction technique must be defined with a high level description. Alternatives must be possibly defined in case of a compatible device is not found. Moreover, interaction techniques must be aware of the device properties.

R3 Handle user and system adaptations. Interaction techniques must be possibly instantiated with an automatic adaptation process, this is adaptability. However, the user has to be able to modify the interaction techniques with a set of predefined parameters, this is adaptability.

R4 Interaction techniques must be configurable at runtime (dynamic adaptation) and between sessions (static adaptation). To ensure usability continuity, context modifications, such as a device plugged or a new task needed, must be recorded. The interaction techniques have to be dynamically adapted according to the context modifications.

A NEW DESCRIPTION FOR INPUTS AND OUTPUTS

In order to perform hardware adaptations and fulfill the second requirement (R2), an accurate description of the input and output devices is needed: this is the device model. Indeed, the adaptation process has to know which devices are available, their capabilities and limitations. This model is needed to perform device selection and adaptation. In the model, inputs devices are considered as the devices that can collect data from the real world. For example, a position, a pressure on a button or a sound acquisition. Regarding output devices, they restitute computer generated values to the real world, for instance an image on a screen or a vibration.

In the literature we can find several classifications about how to perform static or automatic input device selection, such as the Buxton taxonomy [4] extended later by Mackinlay et al. [15] and also DEVAL [16] a device abstraction layer for VR and AR applications. They consider a device as a composition of several input units where each unit is responsible to acquire one kind of data. This kind of classification aims to describe the data provided by input devices units such as the number of degrees of freedom (DoF) of an input value or the property sensed (position, motion, pressure). DEVAL has the advantage to describe a wide variety of device units. It includes more recent devices that can also have output capabilities. For instance, trackers, buttons, haptic feedback, speech and gesture recognition. Less common sensors are also included such as light and temperature sensors. This classification hierarchically structures inputs and outputs units according to their properties. Anyway, this model and the previous ones only define devices with input and output data while it would be interesting to also expose their physical properties in the real physical workspace or their internal properties like refresh rate or accuracy. The graph representation of devices proposed by Lipscomb and Pique [14] and the DEVAL extension introduced by Lindt [13] expose this kind of meta-data. Lipscomb and Pique [14] give the physical characteristics of each device and so a more precise description on how the data is acquired and their limitations. For example, it differentiates bound and unbound inputs as well as isotonic, isometric and elastic ones. In the DEVAL extension [13], three kinds of meta-data can be added to a device unit. First, static devices properties do not change over the time such as the weight of a HMD. Next, configurable devices properties depend on the device setup like the smoothing factor of a tracking de-
device. Finally, runtime properties include performances and device states. The classification into three categories can be discussed because the associated category of a property may change over different devices. For instance, the resolution of an image acquired by a camera can be configurable for some devices such as the Microsoft® Kinect™ while for most of them it is a static value. Moreover, the set of properties introduced in the model does not include enough properties to precisely describe the capabilities of each device. For instance, values boundaries and devices position are not included.

Our device model aims to solve these issues with an accurate description of the devices capabilities, limitations and of their representation in the real world. The model must be extendable because new devices will still appear and they might include new properties not yet included in the current model. The model is described with UML class diagrams, so it is totally editable by any developer who wants to add new properties or a new input or output type. At runtime the properties of a device instance are fulfilled with an XML description file edited with a graphical tool that takes into account the UML diagram. Then, the developer has to complete some functions to fulfill the input data, trigger the outputs when needed and update the dynamic parameters. Moreover, a function has to be implemented in order to tell the system when a new instance of the device is plugged or unplugged. These steps can be done with a device SDK. Static properties have to be fulfilled into the description before execution if needed, for instance the position of a device in the real world. These properties can also be reported into the device SDK to perform its configuration.

In the model, we consider an interaction device as a complex entity that may acquire input(s) and render output(s) and that has a representation in the real world. As shown in Figure 2, the model represents a device as a collection of inputs, outputs and physical objects entities. A device is defined by its name, the name of the SDK used to manage it, its index in the case we use different instances of the device, and a boolean that indicates if it is plugged or not.

The set of physical objects describes the representation of the device in the real world. For example, it gives properties such as a 3D representation of the device if we want to represent it in the virtual world. Its position gives the possibility to a developer to automatically adapt the coordinate system of the tracking values. Moreover, all inputs and outputs are associated to a physical object, this information may help to select an input or an output unit used by an interaction technique. Indeed, for an interaction technique, inputs and outputs that correspond to the same physical object can be preferentially selected. The goal would be to minimize device switchings as well as the homing time of the GOMS model [6].

Regarding input description, the goal is to describe all possible acquired signals that are currently used in 3D applications. We established a list of three categories that gather these possible inputs:

- real values: the most common value type used. It refers to continuous values acquired by trackers, sensors, touchscreens, etc. It can represent a position or a rotation as well as more original values such as a temperature or the lighting intensity of a room.
- discrete values: taken into a set of predefined ones. For example a button pressed or not, a gesture or a vocal command.
- Generic streams: they continuously provide array of values. This category is divided into multiple subtypes with more specific properties (image streams, EEG signals, sound acquisition...)

In the model, these different input types include a description of the acquired data, their properties and limitations, and information about how these data are acquired by the device. To do so, we propose to describe each input into two entities. The “data description” ensures the first need, and the "technology" ensures the second one.

For the two entities of the real value input type described in figure 3, the description reuses the most important properties of previously described taxonomies, especially [14] and [15], while adding some new ones. Regarding the new properties, three booleans describe the axis on which the value is defined as proposed by Mackinlay et al [15]. A fourth axis called "none" is also included for values that are not expressed in a 3D coordinate system, for example a temperature. To continue, the semantics lets the developer know which real world data is acquired. For example, it can be the name of a body joint in the case of the Microsoft® Kinect™. In the "technology" entity, we included a reliability rank of the acquired data between 0 and 1, this property is present for all input types. This value may change at runtime, as some SDK are able to give a reliability score for each value acquired. With the "Techno Real Type" property, we differentiated the values acquired with a distant wireless sensor and with a physical object. For example with optical tracking there is no intermediary for the interaction contrary to a gamepad.
A description of output devices is also included. An output
is described by one entity that provides the information about
how the data is rendered to the real world. All output types
but visual and sound are extracted from the MPEG-V stan-
standard [10]. These types are:

- Visual outputs. They include screens such as monitors and
  HMDs. They also encompass punctual lights, for instance
  a gamepad led or a remotely controlled lighting system that
  modifies the lighting conditions of the real world.
- Sound outputs. They stimulate the auditory sense and in-
  clude devices such as speakers and headphones.
- Tactile outputs. They stimulate the touch sense, for exam-
  ple the vibration of a gamepad.
- Force-feedback outputs. They apply a force in return of
  a user interaction in order to simulate a collision with a
  virtual object. Typical force devices are robotics arms.
- Temperature outputs. They modify the temperature of the
  real world or of a contact point. An example is to control
  the temperature according to the weather conditions in a
  virtual world.
- Wind output. They change the air speed and direction dy-
  namically.
- Scent outputs. They stimulate the olfactory sense.
- Sprayer outputs. They can throw water on a user. It can be
  used to simulate a virtual rain effect in the real world.

Three properties are common to all output and input types:
the name, the refresh rate and a possible associated relative
object. For instance, the force feedback type properties are
listed in Figure 4. The first three properties are extracted from
the description of force feedback devices given by Florens et
al. [8]. First, the continuous force is the maximum force
that can be applied for an unlimited period without damaging
the device. Secondly, the peak force is the maximum feasible
force. Thirdly, the force resolution gives the quantization step
of the force that can be applied. All these values are expressed
in newtons. Then, we have extended this description with a
boolean for each DoF on which the device can apply a force.

In this section we have introduced an extendable device
model to perform device selection and adaptation. Two con-
crete examples of descriptions of device units types have been
given, real values for inputs and haptic rendering for outputs.
The other listed inputs and outputs types are also described
in the model. **We need such a model to adapt the interaction
techniques to the hardware configuration (R2). In order to fit
R4 the model can be configured before a session and updated
at runtime.** In the next section, we show how this model is
used to perform automatic selection of device units and for
device intrinsic parameters adaptations.

**PAC AGENTS FOR INTERACTION TECHNIQUES**
Having modeled input and output devices, we now focus on
the interaction techniques that work independently from any
concrete device thanks to this model. Interaction techniques
are created in order to complete a high level task needed in the
target application. For 3D user interfaces, according to Hand
[11], these tasks belong to three categories: selection and ma-
nipulation, application control and navigation. According to
the plasticity requirements, there is a need for a model for in-
teraction technique description that ensures a good portability
over the possible target 3D frameworks and an independence
of techniques over the interaction devices.

A convenient model could be PAC [7] (Presentation-
Abstraction-Control), a multi-agent model that ensures a
good decoupling between user interface semantics and its
concrete implementation. It decomposes an interaction com-
ponent into three facets:

- the Presentation: it is the concrete implementation of the
  component in charge of its input and output management,
- the Abstraction: it describes the semantics of the compo-
nent and the function it can perform,
- the Control: it ensures the consistency between the presen-
tation and the abstraction.

However, this model has not been designed to perform con-
text adaptation: a PAC agent will be the same whatever the
target platforms and users. Therefore, Calvary et al. intro-
duced Compact [5] (COntext of user Mouldable PAC for plas-
ticity) a specialization of PAC. Compact divides each facet
into two parts, the physical part that is dependent from the
current context and the logical one which is always the same.
For instance, an algorithm used by the abstraction facet can
be replaced according to the target platform performances by
just replacing the physical part. However, keeping the se-
manitics and algorithms constant whatever the context was the
main interest of the PAC model with its presentation facet
that deals with it. PAC does not provide a good decoupling
between the input and the output management because the
Presentation facet includes both. This decoupling is needed
in 3D because of the larger set of possible devices. To solve
this issue Grappl [9] describes an interaction technique with
a base class that performs semantics and presentation compu-
tax. This class is extended by other classes that implement
different ways to control the technique with several sets of
interaction devices. Compared to PAC, the method does not
ensure a good decoupling between semantics and presenta-
tion and a good independence to the target 3D framework.
Indeed, Grappl uses its own internal scene graph system.

To solve all of these issues we represent the interaction tech-
nique model with a PAC model enhanced with ARCH con-
cepts [1]. ARCH proposes to add adaptor components be-
tween the different facets of PAC-like models. This model
is also considered as a meta-model for other software mod-
els by proposing a generic separation between facets of inter-
active components. ARCH represents an interactive compo-

---

Figure 4: The description of a force feedback output.
Fly Navigation
«Joysticks»

By using this PAC/ARCH approach we ensure a good de-
t ruth value to adapt the gain of a 3D cursor displacement.

The work of the developer is to choose these device units in
order to drive correctly the interaction technique. The logi-
cal driver describes all required inputs and outputs units ac-
cording to a set of parameters taken from our device model.

Some can be optional if they are not needed for a good us-
ability. The logical driver can be instantiated if these units
can be found at runtime. At runtime, the logical driver
receives the input data that it needs and it can trigger the
outputs. The device units may come from different con-
crete devices in order to perform device composition. An
example of logical driver is given in Figure 5.

Getting the data independently from concrete devices is one
of the possibilities given by the logical driver. Nevertheless,
it does not ensure the same behavior over all the devices be-
cause all do not have the same capabilities and the same in-
trinsic properties. As the logical driver is associated with
device units, it can access each property of our device model.

Thus, the possibility to perform adaptation to the intrinsic pa-
rameters of each device unit is given to the developer. For
instance, in case of a 3D cursor interaction technique for se-
lection and manipulation controlled with a 6-DoF device, if
the virtual environment is bigger than the device tracking vol-
ume, the user will not be able to reach all parts of the virtual
world. One solution consists in using the boundaries of the
tracking value to adapt the gain of a 3D cursor displacement.

By using this PAC/ARCH approach we ensure a good de-
coupling between the interaction technique semantics and its
concrete implementation, the independence of the technique
over the target 3D framework and OS (R1), over the con-
crete devices used as well as over the interaction modality
used (R2). Indeed, as we can develop multiple compatible
logical drivers for the same interaction technique, this inter-
action technique can be controlled with different modalities.
Moreover, in a case of a context modification at runtime, any
presentation facet can be exchanged with another one if it is
compatible with the interaction technique. This property
ensures the possibility to perform static and dynamic adap-
tations (R4). To do so, as shown in figure 6 a facet is imple-
mented on top of the control facet: the supervision control.
This facet is not present in the classic PAC model. This com-
ponent contains all the types that can be instantiated as a pre-
sentation facet for the current interaction technique: a list of
compatible logical drivers and a list of all rendering facets.
It also receives the context modifications at runtime and then
is able to determine if a presentation facet is still possible in
the current context and may ask the adaptation engine for a
replacement. For instance, if a device is unplugged from the
system, the supervision control may detect that the current
logical driver is unusable and therefore ask the system to re-
place it by another one into its list of compatible ones.

As we said, a PAC agent represents an interaction technique
instantiated to achieve a high level task. To represent this
relation Grappl [9] associates each task with a set of compati-
ble interaction techniques. In the same way, in our interaction
model, each task derives from a basic task class and contains
a list of properties as well as a list of all PAC agents that can
complete it. Therefore, the developer is responsible to asso-
ciate the interaction techniques that can complete a high level
task. This compatibility list is exposed to the system to allo-
cate the best interaction techniques according to the desired
tasks. At runtime, the abstraction facet is associated to the
task description in order to access its properties. This asso-
ciation allows a developer to include some parameters into
the task that will be used by the interaction technique. For
instance, in a manipulation task we could parametrize the de-
grees of freedom on which objects can be manipulated.

To illustrate our interaction model, the Figure 6 presents a
manipulation task and its compatible interaction techniques in
which we detail the 3D ray-based PAC agent. In that case, the
presentation facet creates the ray geometry, handles collision
detection and performs scene graph modifications according
to the control facet requests. The logical driver handles how
the ray is manipulated according to different device units. The
first one is based on a 3D interaction device that can provide
a 6-DoF tracker. The ray base is controlled in position and ro-
tation by the data given by this tracker. A discrete input (such
as a button) is used to attach and detach an object to the ray
extremity. Two other discrete inputs are used to change the
length of the ray, the first one to increase it, the other one to
decrease it. It also includes an optional tactile output in or-
der to perform a vibration feedback when an object is caught.
This implementation is shown in Figure 1a, the device used
is a razer hydra. The second logical driver is based on two
2-DoF force input like joysticks. The position of the ray base
is constant and set at the center of the user view, just in front
of the main camera. The first 2-DoF input is used to control

Figure 5: An example of a logical driver and its device units
needs. It controls a fly navigation interaction technique with
joysticks. The first needs are two real values that describe the
two joysticks needed. A need is defined with some properties
extracted from the device model. As shown some properties
can be omitted, only the detailed ones will be used to select
the available device units. The last one is tactile feedback, which is optional and can give a vibration feedback when the
user collides any 3D object.

![Fly Navigation](image)

- The rendering facet is the only facet depending on a 3D
  framework. It handles graphics output and physics.
- The logical driver handles input and output devices man-
  agement. It describes the way the interaction technique is
  controlled according to a set of interaction device units.
  The work of the developer is to choose these device units in
  order to drive correctly the interaction technique. The logi-
cal driver describes all required inputs and outputs units ac-
cording to a set of parameters taken from our device model.
  Some can be optional if they are not needed for a good us-
ability. The logical driver can be instantiated if these units
  can be found at runtime. At runtime, the logical driver
  receives the input data that it needs and it can trigger the
  outputs. The device units may come from different con-
crete devices in order to perform device composition. An
example of logical driver is given in Figure 5.

Getting the data independently from concrete devices is one
of the possibilities given by the logical driver. Nevertheless,
it does not ensure the same behavior over all the devices be-
cause all do not have the same capabilities and the same in-
trinsic properties. As the logical driver is associated with
device units, it can access each property of our device model.

Thus, the possibility to perform adaptation to the intrinsic pa-
rameters of each device unit is given to the developer. For
instance, in case of a 3D cursor interaction technique for se-
lection and manipulation controlled with a 6-DoF device, if
the virtual environment is bigger than the device tracking vol-
ume, the user will not be able to reach all parts of the virtual
world. One solution consists in using the boundaries of the
tracking value to adapt the gain of a 3D cursor displacement.

By using this PAC/ARCH approach we ensure a good de-
coupling between the interaction technique semantics and its
concrete implementation, the independence of the technique

![Logical Driver](image)
The creation of inter-lacks our toolkit to cover R1 and R2. This model exposes device context changes such as the add of the removal of a device, or properties modifications. The second one is an interaction model that uses the device model, it is based on PAC and ARCH. The approach lets the developer create interaction techniques independently of concrete devices and of a 3D framework.

Future work consists in establishing and adaptation engine that will create the interaction techniques at runtime according to the current context in order to always provide the most suited application. Our models must also be extended in order to take into account different levels of adaptation such as user and content adaptation. Our perspective is to create a tool for developers and designers for the creation of plastic 3D user interfaces. Such a tool is being developed with our models in Mono C# and interfaced with the Unity3D game engine.

REFERENCES