HAL open science

Improving modularity and usability of interactive systems with Malai

Arnaud Blouin, Olivier Beaudoux

To cite this version:

Arnaud Blouin, Olivier Beaudoux. Improving modularity and usability of interactive systems with Malai. EICS'10: Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems, 2010, Berlin, Germany, Germany. pp.115-124. inria-00477627

HAL Id: inria-00477627

https://inria.hal.science/inria-00477627v1

Submitted on 29 Apr 2010

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.

Improving Modularity and Usability of Interactive Systems with Malai

Arnaud Blouin
INRIA
Rennes - France
arnaud.blouin@inria.fr

Olivier Beaudoux
GRI - Groupe ESEO
Angers - France
olivier.beaudoux@eseo.fr

ABSTRACT
In this paper we present Malai, a model-based user interface development environment. Malai is dedicated to the conception of post-WIMP (Window, Icon, Menu, Pointing device) interactive systems. Malai aims at gathering together principles from Norman's action model [19], instrumental interaction [3], direct manipulation [25], the interactor concept [17] and the DPI model (Documents, Presentations, Instruments) [4]. It completes works on data manipulation techniques used to link source data to user interfaces. We show how Malai can improve modularity and usability of interactive systems by considering actions, interactions and instruments as reusable first-class objects. Malai has been successfully used for the development of several post-WIMP interactive systems. We introduce each Malai component using the same example: a vector graphics editor.

Keywords
Interaction, action, user interface, instrument, interactive system, MDE

1. INTRODUCTION

With the diversification of platforms such as notebooks, IPhones, or PDAs, an interactive system (IS) dedicated to a given platform needs to be updated to work on another platform. Model-based user interface development environments (MB-UIDE) propose to [27, 7]: separate elements (*e.g.* presentations, users, *etc.*) during the conception of IS; grade the conception of an IS using different abstraction levels, as illustrated in Figure 1. In such a process, developers start by defining *the concepts and tasks* of an IS. An *abstract user interface*, modality-independent and free of graphical information, is then specified. The *concrete user interface* remains modality-dependent and contains graphical information. The *final user interface* corresponds to the code generated from one concrete user interface to a given development platform such as Swing. This process allows the factorization of development steps of one IS dedicated to several platforms.

Recent MB-UIDEs, such as UsiXML [28], TERESA [23] or ICOs [18], focus on transformations from one abstraction level to another, on human interface devices (HID) management, and/or on the use of multimodal interactions within IS. However, these MB-UIDEs neither use nor define elements such as instruments, interactions and actions that would improve the modularity and the usability of IS: (1) modularity by considering these elements as reusable first-class objects; (2) usability by being able to specify feedback provided to users within instruments, to abort interactions, and to undo/redo actions. Instrumental interaction defines an instrument as a metaphor of the tool or device that users handle to carry out actions (*e.g.* an eraser, provided by a graphics editor, that deletes shapes) [3]. Instrument concept can be used to fill the gap between user interaction and the resulting action applied to the IS.

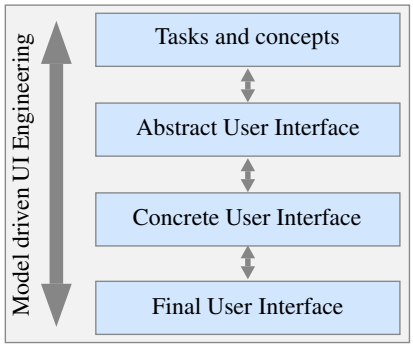
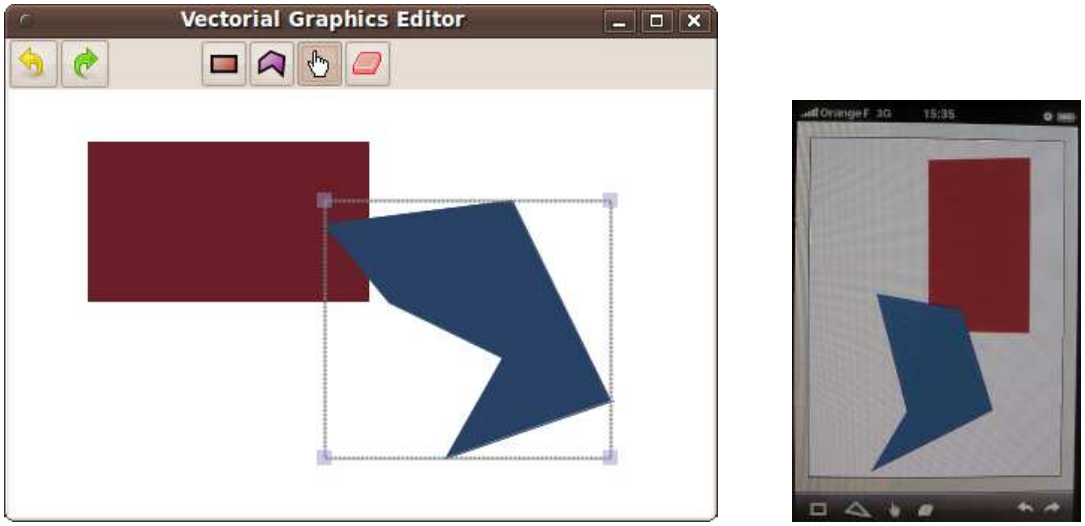


Figure 1. Model-driven UI engineering

The contribution of this paper is to propose a MB-UIDE, called Malai, dedicated to the conception of post-WIMP IS. Malai is based on the process described in Figure 1 and divides IS into six elements: the *source data*, the *user interface* itself composed of *presentations* and *instruments*. An instrument links *interactions* carried out by users to *actions* that modify the presentations. Malai gathers principles of Norman's action model [19], instrumental interaction, direct manipulation [25], the interactor concept [17], and the DPI model [4]. Malai aims at reaping the benefits of these

Author version.



(a) UI for the workstation

(b) UI for the mobile phone

Figure 2. The vector graphics editor for two platforms

models and at improving modularity by: considering source data, presentations, actions, interactions and instruments as reusable first-class objects; easily plug interactions and actions into instruments. Moreover, Malai improves IS usability: users can see interim feedback [17] of their actions, and can abort interactions and actions they carry out. This paper completes our earlier work on Malai [5] by notably improving the action and instrument definitions, and by introducing the static and dynamic parts in the six elements.

The remainder of this paper is structured as follows. Section 2 introduces an example, a vector graphics editor, in order to illustrate our proposition. Section 3 outlines the global architecture and methodology of our proposition. Sections 4, 5 and 6 describe the three main elements that compose Malai: the user interface, the presentation and the instrument. The graphics editor for the mobile phone platform is used to illustrate these elements. The adaptation of the editor for the workstation platform is then discussed in Section 7. Section 8 compares Malai with related works. A quantitative and qualitative evaluation of our approach is presented in Section 9. Finally, Section 10 concludes this paper.

2. MOTIVATING EXAMPLE

2.1 General Description

As a motivating example, we consider the conception of a simplified vector graphics editor for two platforms. The first platform is a tactile mobile phone while the second is a standard workstation. With the editor, a user must be able to create, delete, select, move and resize shapes (rectangles and polygons). Figures 2(a) and 2(b) show, respectively, the UI of the graphics editor for the workstation and the mobile phone platforms. Both UIs contain a palette to select an instrument (*e.g.* the eraser to delete shapes), and a direct manipulation drawing area. The main difference is that shape resizing is carried out by a bimanual interaction for the mobile phone, and by using handles for the workstation.

2.2 Conception Issues

Implementing both versions of the graphics editor requires the development of the first version, followed by its adaptation to the second. Such a process is error-prone: a modification in the common part shared by the two editors must be manually propagated into both versions.

To avoid this issue, UsiXML follows the model-driven UI engineering previously introduced. However, UsiXML does not allow the conception of interactions (*e.g.* bimanual and drag-and-drop interactions): tasks are carried out by using predefined interactions on widgets (*e.g.* a simple click) or by vocal commands. UsiXML provides feedback for vocal commands that consists of textual messages such as "You said red". This feature is insufficient to provide users feedback while carrying out interactions. A developer should have an explicit way to define any kind of feedback that would help users to understand the state of the IS. Moreover, UsiXML does not provide a process that would allow a user to abort an interaction or to undo/redo an action. These issues are also shared by TERESA and ICOs, two MB-UIDEs dedicated to the conception of IS using a wide variety of platforms and/or interaction modalities.

To summarize, we come across two issues in the current MB-UIDEs:

1. *Modularity.* Interactions and actions are not considered as reusable first-class objects.

2. *Usability.* No sufficient process exists to: (1) specify feedback provided to users during interactions; (2) allow users to abort interactions; (3) undo/redo actions.

3. THE MALAI MODEL

3.1 Global Overview

Figure 3 describes the organization of Malai. Malai divides an IS into the following fundamental elements. A *user interface* is composed of presentations, instruments and widgets. A *presentation* is composed of an abstract presentation and a concrete presentation. An *abstract presentation* is created by a Malan mapping (link ①) from source data. *Source data* represent domain objects of the IS. Malan is a mapping language that defines mappings between source data and their presentations [6]. A *concrete presentation* is created and updated by another Malan mapping (link ②) from the abstract presentation. *An interaction* consumes events produced by HIDs (link ③). *An instrument* transforms (link ④) input interactions (concrete part) into output actions (abstract part). The *interim feedback* of an instrument provides users with temporary information that describe the current state of interactions and actions they are carrying out (link ⑤). *An action* is executed on an abstract presentation (link ⑥); the abstract presentation then updates the source data throughout a Malan mapping (link ⑦).

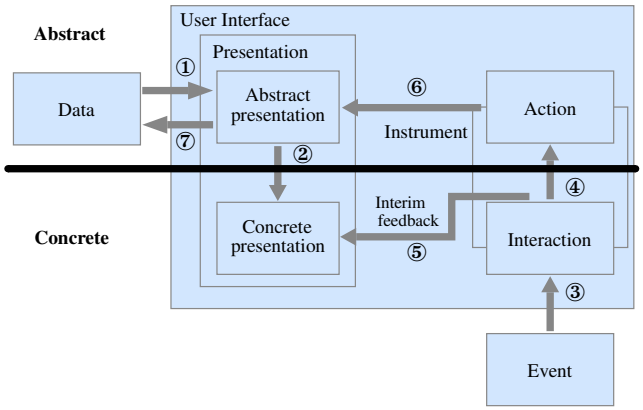


Figure 3. Organization of Malai

All elements have a *static part* described by a class diagram. Instruments, actions and interactions have a *dynamic part* described with the Malai language and state machines. Figure 4 gives the formalisms used to describe each static, dynamic, abstract and concrete part of our elements.

3.2 Methodology

Malai follows a methodology based on model driven user interface engineering (see Figure 1). However, the tasks and concepts level is not considered yet. The process starts by defining the abstract user interface composed of the abstract presentation, the source data, the Malan mappings ① and ⑦ (see Figures 3) and the actions.

	Data	User Interface					
			Presentation		Instrument		
						Action	Interaction
	abstract	*concrete*	*abstract*	*concrete*	*concrete*	*abstract*	*concrete*
static	class diagram	class diagram	class diagram	class diagram	class diagram	class diagram	class diagram
dyn.				Malan language		Malai language	State machine + Malai lang.

Figure 4. Static/dynamic parts and abstract/concrete parts

Then, depending on the platform, the concrete presentation, the instruments, the interactions and the Malan mapping ② are defined to create a concrete user interface. Finally, a final user interface can be generated from a concrete user interface for a given development platform (*e.g.* Swing or .Net). These steps are developed by using our Malai Eclipse plug-in (see Figure 5).

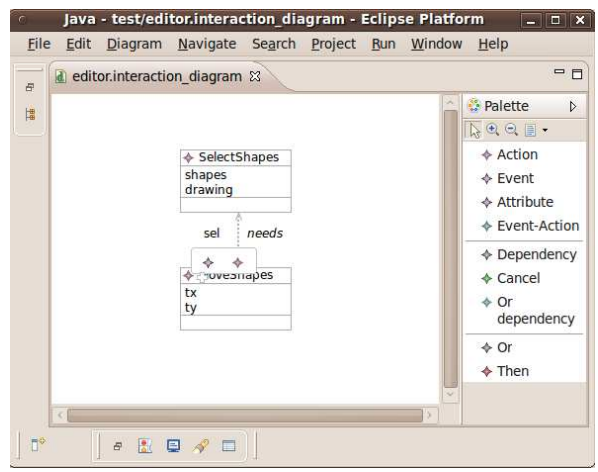


Figure 5. Screen shot of our Malai Eclipse plug-in

4. USER INTERFACE

A user interface (UI) is composed of presentations and instruments. Presentations allow users to view data from given viewpoints. Instruments are manipulated by users to carry out actions on presentations. Instruments can be part of presentations. For instance, handles in our graphics editor may be associated with a shape to resize it. Such handles are instruments contained in the presentation. A UI can also have a set of widgets, such as windows or panels.

Figure 6 describes the UI model of the graphics editor for the mobile phone platform. The UI contains a *window* composed of a container and a canvas. The *container* corresponds to the editor palette. It contains buttons (*hand*, *rect* and *polygon*) used to select an instrument. The *canvas* corresponds to the presentation of the editor that consists of a drawing area. The UI also contains instruments: the *pencil* creates rectangles

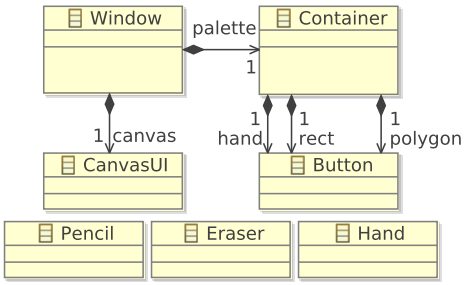


Figure 6. User interface model of the graphics editor

and polygons; the *eraser* deletes shapes; the *hand* selects, moves and resizes shapes.

5. PRESENTATION

A presentation is composed of an abstract presentation and of a concrete presentation. An *abstract presentation* defines the presentation of the IS without any graphical information. A *concrete presentation* is the graphical representation of the abstract presentation. The abstract presentation thus represents the model of MVC (*e.g.* a Swing *TableModel*), while the concrete presentation represents the view of MVC (*e.g.* a Swing *JTable*).

The abstract and concrete presentations are linked by a *Malan mapping*: when the abstract presentation is modified, the mapping applies the modification to the concrete presentation. Malan mappings are not described in this paper to keep focus on the interaction part of Malai.

Figure 7 describes the abstract presentation of the graphics editor: a *Drawing* contains shapes; a *Shape* is defined by its line *thickness*, a *color* and a set of *points*. A shape is either a *Rectangle* or a *Polygon*.

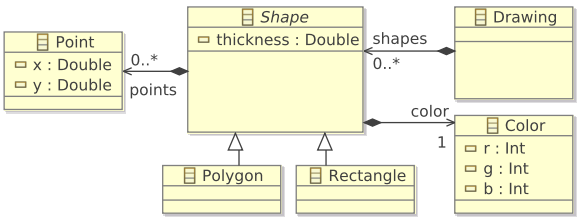


Figure 7. Abstract presentation of the graphics editor

Figure 8 describes the concrete presentation. It defines the *CanvasUI* in which shapes can be drawn. A canvas contains concrete representations of shapes (class *ShapeUI*). *ShapeUI* is composed of a *thickness*, a *color* and a set of *points*, and is either a *RectangleUI* or a *PolygonUI*. Association *selection* refers to the selected *ShapeUI* of the *CanvasUI*. A *CanvasUI* also contains a box (class *SelectionBox*) used to outline the selected shapes.

In this example, the abstract and concrete presentations are relatively similar. The reason behind such a

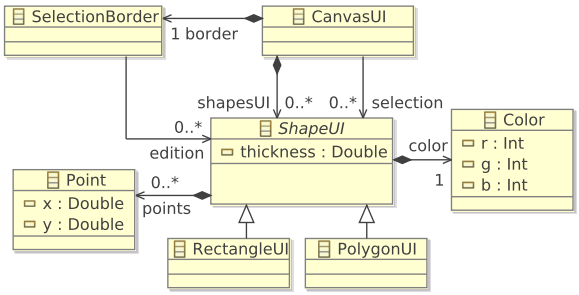


Figure 8. Concrete presentation of the graphics editor

similarity is that a shape is by definition a graphical object. The abstract presentation thus contains graphical details usually only defined in the concrete one.

6. INSTRUMENT

An instrument is divided into two parts, as shown in Figure 9: the abstract part defines the *actions* that can be executed by the instrument; the concrete part defines the *interactions* that a user can carry out to execute actions. An instrument thus defines the *links* between its actions and user interactions.

The remainder of this section describes the three elements that compose instruments: interactions, actions and links. The same example of the instrument *Hand* is used throughout this section. The instrument *Hand* allows users to directly select, move and resize shapes through the drawing area.

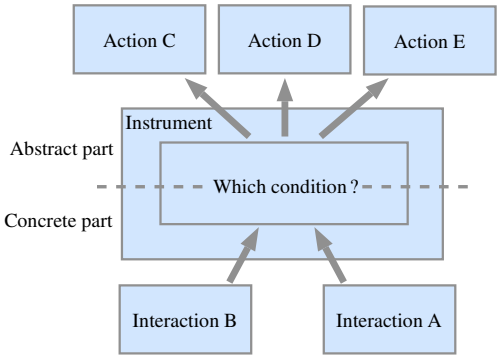


Figure 9. Instrument principle: from an interaction to an action

6.1 Interaction

This section describes an interaction composed of a static part and a dynamic part.

Static part

The static part defines *interaction data*, and *events* produced by HIDs. Instrument *Hand* resizes shapes using a bimanual interaction: the user selects a shape with a first finger and then resizes the shape with the first finger and a second one.

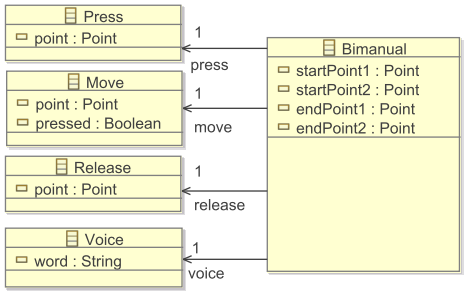


Figure 10. Static part of the bimanual interaction

Figure 10 describes the static part of the bimanual interaction. Class *Bimanual* defines the interaction data: attributes *startPoint1* and *startPoint2* correspond to the first and second finger starting positions; attributes *endPoint1* and *endPoint2* define their respective final position. Classes *Press*, *Move*, *Release* and *Voice* define the events used by the interaction: they correspond respectively to a pressure, a move, and a release of a pointing device. Their attribute *Point* defines the position of the pointing device when the event occurred. Attribute *pressed* of class *Move* defines if the button used is pressed or not. Class *Voice* corresponds to the pronunciation of a word. Attribute *word* of class *Voice* specifies the word spoken.

Dynamic part

The dynamic part of an interaction describes its behavior using a *finite state machine*. Defining interactions using finite state machines allows the specification of complex interactions [1]. Defining interactions independently from actions and instruments mainly aims at providing a set of predefined interactions that can easily be reused in different IS. Such a set can be extended to define new interactions.

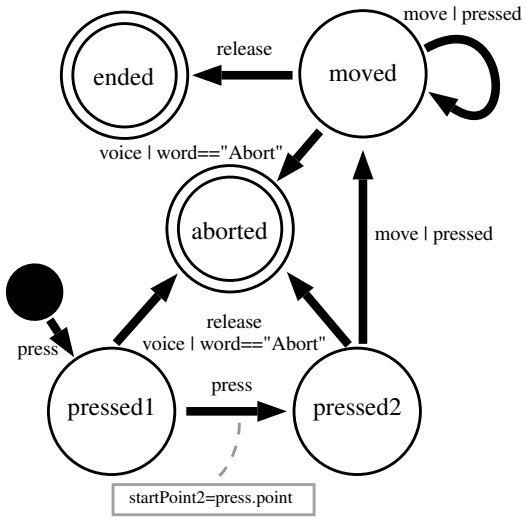


Figure 11. Dynamic part of the bimanual interaction

Finite state machine transitions have two parts. The first part specifies the name of an event defined in the static part (*e.g.* *press* for the bimanual interaction). The optional second part defines a predicate used as a filter on the specified event. For instance, transition *press / button==1* only concerns an event *Press* when its attribute *button* equals 1. There exists three kinds of transition: (a) a *terminal transition* is an event with a terminal target state; (b) an *aborting transition* is an event with an aborting target state; (c) a *non-terminal and non-aborting transition* is an event that links a source state to a non-terminal and non-aborting target state.

Figure 11 describes the state machine of the bimanual interaction. The state machine starts with the first finger pressure (state *pressed1*). State *pressed2* is reached when a second pressure occurs. If a *release* event occurs in the states *pressed1* and *pressed2*, the interaction is aborted (state *aborted*). In state *pressed2*, both fingers can be moved so that the state *moved* is reached. The interaction ends when one of the fingers is released (state *ended*). The interaction can be aborted if the word "Abort" is spoken (transitions *voice / word="Abort"*). Allowing the definition of aborting states follows the direct manipulation recommendation which claims that a user must be able to abort any interaction he carries out [25]. Each transition defines code that set up the interaction attributes. For instance in Figure 10, code is associated with the transition *press* (*cf.* the rectangular box). This code specifies the value of the attribute *startPoint2* by using the event *press*.

6.2 Action

This section describes an action composed of a static part and a dynamic part. Our action model is based on the design pattern *Command* [12] since we consider an action as an object which can modify data.

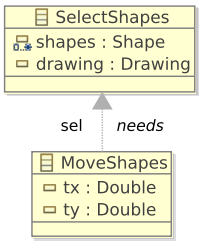


Figure 12. Static part of two actions

Static part

The static part of actions specifies their data and the relations between them through a class diagram. For example, Figure 12 defines the static part of two actions that the instrument *Hand* can produce. Class *SelectShape* contains two attributes: *shapes* corresponds to the shapes to select; *drawing* is the drawing that contains the selected shapes. Attributes *tx* and *ty* of class *MoveShapes* specifies the translation. The *needs* rela-

tion between these two actions is also defined: a shape must already be selected in order to be moved.

Dynamic part

The dynamic part of an action defines the different steps of its life cycle using our Malai language. The life cycle of an action is depicted by Figure 13. It extends the usual action life cycle where actions occur *after* interactions. Once created, the action can be executed (transition *do*) and updated several times. The action can also be aborted while in progress. It can then be recycled into another action. Once the action ends and if its execution has side effects, it is saved for undo/redo purposes; otherwise, the life cycle ends. A saved action can be removed (state *Action removed*) from the system (*e.g.* because of the limited size of the undo/redo memory).

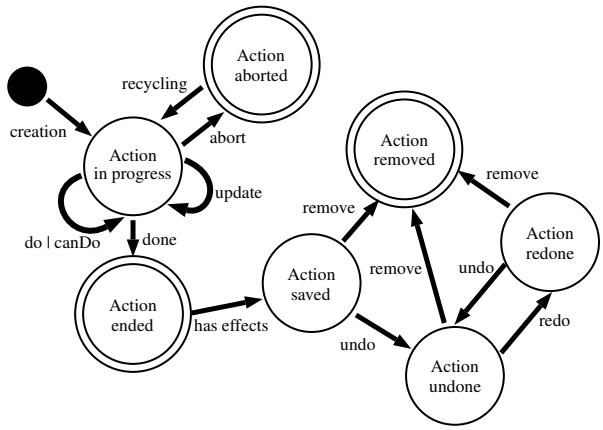


Figure 13. Action life cycle

For example, Listing 1 describes the dynamic part of the action *MoveShapes* in the Malai language. It specifies different transitions of the action life cycle. Function *canDo* (line 1) checks if shapes have been selected. Method *move* (line 7) moves the selected shapes; it is called in the methods *do* and *redo* to perform the translation. Method *undo* calls the method *move* to cancel the translation.

During an interaction, a user may want to change or to abort the action in progress. For example, he can begin the resize of shapes and say "Abort" to abort the interaction (see Section 6.1) and thus the corresponding action in progress. This process is carried out by instruments that link the interaction and the action life cycles, as explained in the following section.

```
1 boolean canDo() {
2  return sel.isDone()
3 }
4 do()   { move(tx, ty)   }
5 undo() { move(−tx, −ty) }
6 redo() { move(tx, ty)   }
7 move(double tx2, double ty2) {
8   foreach Shape shape in sel.selection
9     shape.move(tx2, ty2)
10 }
```

Listing 1. Dynamic part of action *MoveShapes*

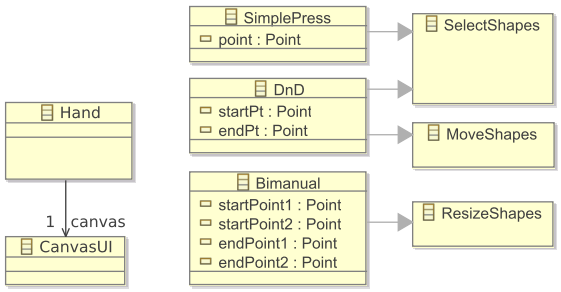
The main difference between a Malai action and a task, such as a *ConcurTaskTrees* task [22], is that an action describes its process (through the methods *do*, *undo*, *redo* and *canDo*). Following the process of Figure 1, actions can be partially derived from tasks. Therefore, process descriptions can be added to these derived actions.

6.3 Links between interactions and actions

The main goal of an instrument is to link input interactions to output actions, and to manage their life cycle. An instrument has a static part and a dynamic part.

Static part

The static part of an instrument defines: (1) the instrument data; (2) the input interactions and their related output actions. For instance, Figure 14(a) defines the data of the instrument *Hand*, which simply defines the *canvas* where the hand operates. Figure 14(b) declares the links between the *Hand* interactions and actions: a finger pressure (interaction *SimplePress*) selects a shape; a drag-and-drop (*DnD*) moves the selected shapes or selects shapes; a bimanual interaction resizes the selected shapes.



(a) Class diagram (b) Interaction to action links

Figure 14. Static part of instrument Hand

Dynamic part

The dynamic part of an instrument describes the working of interaction-action links using our Malai language. An interaction-action link is composed of: (1) a condition that must be respected to link an interaction to an action; (2) an optional interim feedback. Interim feedback provides temporary information related to the current interaction and action carried out by the user [26].

```
1 Links Hand hand {
2   SimplePress sp −> SelectShapes action {
3     condition: getObj(sp.point) is ShapeUI
4   }
5   DnD dnd −> SelectShapes action {
6     condition: getObj(dnd.endPt) is CanvasUI
7     feedback : Rectangle rec = new Rectangle(
```

```
8                    dnd.startPt, dnd.endPt)
9                hand.setTmpShape(rec)
10    }
11    //...
12    default: hand.setTmpShape(null)
13 }
```

Listing 2. Dynamic part of Instrument *hand*

For instance, Listing 2 describes three interaction-action links of the *Hand* declared in Figure 14(b). A *SimplePress* interaction creates a *SelectShapes* action if the targeted object is a *ShapeUI* (line 3). A *DnD* interaction selects shapes only if the targeted object of the interaction is a *CanvasUI* (line 6). This link also defines an interim feedback (lines 7 to 9) that consists of a rectangle that uses the start and the end positions of the *DnD*. This rectangle gives the user temporary information related to the selection he is carrying out. When the interaction is ended or aborted, the rectangle is removed from the canvas (line 16).

The dynamic part of an instrument also controls the action and interaction life cycle. For instance, when an interaction is aborted, the corresponding action is aborted too. Similarly, when an interaction is modified, the corresponding action is updated or executed. Such a process is automatic for each link and is not visible nor editable by developers.

7. CROSS-PLATFORM ADAPTATION

In the previous sections we described the Malai model using the graphics editor for the mobile phone platform as an example. In this section, the adaptation of the graphics editor to the workstation platform is discussed.

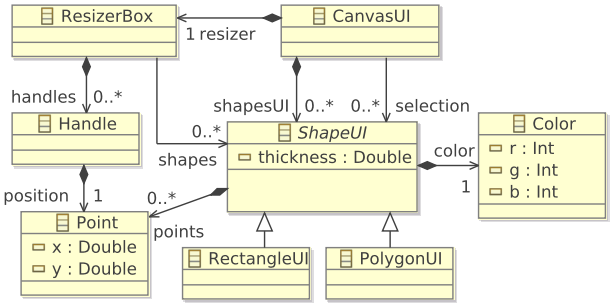


Figure 15. Concrete presentation for the workstation platform

Since our workstation platform does not provide any bimanual capability, the concrete presentation and the instrument *Hand* must be modified regarding the action *ResizeShapes*. Figure 15 describes the modifications to apply to the concrete presentation: the *SelectionBox* is replaced by a *ResizerBox* which contains *handle*s used to resize the selected *shapes*.

Figure 16 describes the modification to apply to the instrument *Hand*: the bimanual interaction is replaced

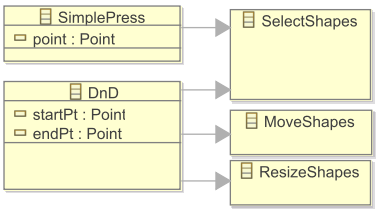


Figure 16. Links of Instrument *Hand* for the workstation platform

by a DnD interaction. The condition of the new *DnD*-to-*ResizeShapes* link states that a shape is resized when the targeted object of the DnD is a handle, as illustrated in the following code snippet:

```
Links Hand hand {
  DnD dnd -> ResizeShapes action {
    condition: getObj(dnd.endPt) is Handle
  }//...
}
```

Since the concrete presentation must be modified, the mapping between the abstract and the concrete presentations must be modified too. Moreover, this mapping must now fit the characteristics of the workstation platform, such as its screen size.

The adaptation described in this section illustrates the simplicity to modify an IS dedicated to a given platform to fit another platform. This advantage can be explained by the modularity of interactions, actions and instruments. The library of predefined interactions also allows code reuse. The abstract parts of all Malai elements remain unchanged.

8. RELATED WORK

Similarities between Malai and the Arch model [2] can be noticed, as illustrated in Figure 17. The *functional core* is close to an abstract presentation and source data since they cover the domain-dependent data and functions. *Logical interactions* correspond to Malai interactions and events. Malai does not manage *physical interactions* that may consist of describing HIDs. Instruments and the concrete presentation are related to the dialog controller: the instrument is the core of our model. It uses input interactions to execute actions as output. The *functional core adapter* corresponds to a Malan mapping between the abstract and the concrete presentations, and to actions produced by instruments that modify the functional core. However, Arch does not distinguish abstract presentation from source data and their Malan mapping.

Using a higher abstraction level than those of widget toolkits to develop IS is an old challenge: in 1985, the system COUSIN already proposed to generate code from UI specification [14]. MB-UIDEs are based on this principle. Most recent MB-UIDEs focus on the conception of multi-target IS and on post-WIMP interactions modeling. For instance, the Cameleon project proposes

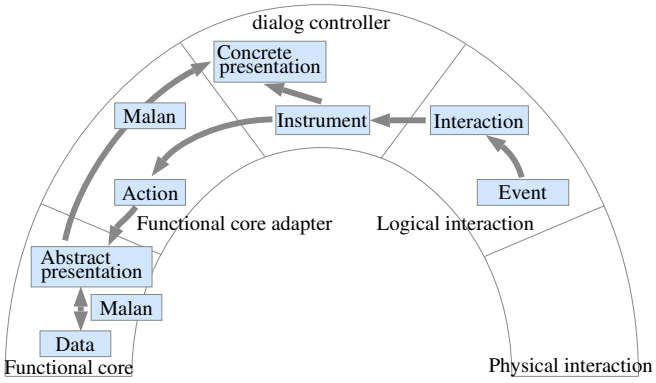


Figure 17. Arch and Malai comparison

a framework to statically or dynamically adapt an IS from one context of use to another [7]. A context of use is composed of the *platform*, the physical *environment* in which users take place, and the different kinds of *user* that use the IS. Our approach does not take account of the environment and the user parameters yet. Moreover, Malai does not consider dynamic adaptations (*i.e.* during the execution of an IS) for the moment. MB-UIDEs such as TERESA [23], UsiXML [28], and UIML [20], describe multi-platform IS and multimodal interactions using an XML formalism. Others MB-UIDEs, such as OpenInterface [24] and ASUR [13], aim to provide a better way to develop tangible UIs and ubiquitous IS. However, all these MB-UIDEs fail to consider actions, interactions and instruments as first-class objects. Moreover, these MB-UIDEs do not have a dedicated mapping language, such as Malan, that expresses complex operations between data and presentations.

ICOs is a MB-UIDE that defines IS using Petri nets [18]. Contrary to Malai, ICOs provides a formalism to describe physical interactions (*e.g.* a mouse behavior). The definition of component behavior is more complex in ICOs than in Malai: Malai defines or reuses interactions, based on low level events (*e.g.* "mouse pressed", or "mouse released" events) within instruments; ICOs component behaviors are described by high level events (*e.g.* dragging, or click events) that are composed of low level events . Moreover, ICOs does not provide a devoted formalism to allow action/interaction aborting and the definition of interim feedback. Concerning the functional core adapter, Malai provides a mapping language and an action model to link the functional core to the dialog controller. ICOs does not focus on the functional core adapter. However, ICOs has the advantage to formally describe a UI to check conception errors in safety-critical IS.

The Garnet system introduced the concept of interactors [17]. The use of interactors aims at facilitating the development of IS by separating widgets from their interactions. This principle is used in toolkits to let widgets be independent from interactions and contexts of use [10].

Recent *Rich Internet Applications* frameworks, such as Flex [15], allow the direct development of both Internet and desktop applications. All of these frameworks focus on the UI definition. Thus, they do not provide an independent action, interaction nor instrument model. Moreover, some interaction techniques, such as bi-manual interactions, are not managed.

VIGO is an instrumental-based architecture devoted to the creation of distributed UIs [16]. M-CIU is an instrument-based model that proposes to unify DnD interaction techniques [8]. Both VIGO and M-CIU do not separate the action, interaction and instrument concepts. An instrument is described by a unique state machine and can be reused into different IS. In contrary to Malai, VIGO and M-CIU do not consider actions as undoable and abortable objects.

MDPC is an architecture that extends MVC [9]. MDPC externalizes the picking process that allows the picking of objects into a UI. This improves controllers modularity by being reusable for different IS. In Malai, instruments and interactions allow such a modularity: an interaction is reusable while an instrument can be used in different versions of the same IS. However, MDPC does not provide a dedicated language to map data to presentations as the Malan language.

ICON is a toolkit that configures physical interactions and connects these interactions to a UI [11]. SwingStates is a library that adds state machines to the Java Swing UI toolkit [1]. SwingStates defines interactions and replaces traditional callbacks and listeners by state machines. Both ICON and SwingStates directly connect interactions to source data contrary to Malai. However, Malai uses state machines to describe and encapsulate interactions like SwingStates does.

9. EVALUATION

Malai has been successfully applied on several examples such as the vector graphics editor presented in this paper, a calendar and an XML editor. These post-WIMP examples have been developed using our open-source implementation[1].

In this section, we compare the cost of development and adaptation of the graphics editor with the Java language, and Malai. Then, we evaluate Malai using two criteria introduced in [21].

9.1 Development effort comparison

The first stage of the evaluation deals with effort estimation in terms of time and number of lines of code (LC). Only one developer, expert in both Java and Malai, participated in the experiment. So the results must be used carefully and may not reflect the result of a broader experiment. This first stage is divided in

[1]Implementations and examples are available at the following address: http://www.irisa.fr/triskell/perso_pro/ablouin/software.html

three parts depicted in Figure 18: the *basic version* of the graphics editor *without* undo/redo actions, nor action/interaction aborting, nor interim feedback; the *full version* that allows action undo/redo, action/interaction aborting, and interim feedback; the *adaptation* of the editor from the workstation platform to the mobile phone platform.

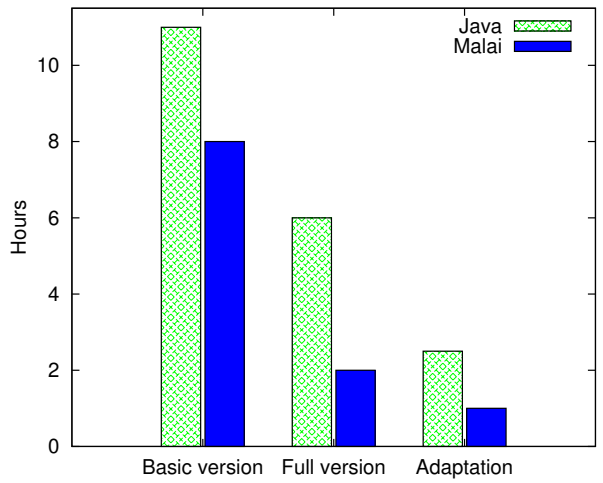


Figure 18. Development effort comparison between Java and Malai (note single developer)

The development of the graphics editor for the workstation platform using Java Swing needed 11 hours and around 850 LC. With Malai, 8 hours were needed to carry out the same task. This difference can be partially explained by the use of predefined interactions: our Malai implementation provides a set of predefined interactions that can be directly used within different IS. On the contrary, with Java Swing we needed to develop bimanual and classic interactions ourselves.

Adding undo/redo actions, action/interaction aborting, and interim feedback to the Java graphics editor needed 6 hours. This full version is composed of approximately 1050 LC, aproximatly 200 LC more than the previous version. With Malai, these improvements needed 2 hours and less than 100 LC added to dynamic parts of actions and instruments.

Adapting the Java graphics editor to fit the mobile phone platform[2] needed 2.5 hours and the modification of more than 200 LC. With Malai, this adaptation needed 1 hour and the modification of around 50 LCs (Malan mappings and instrument conditions).

In this example, our approach needs less time and LC to improve usability and adaptation of the graphics editor. The development of the graphics editor with Malai was carried out using our Malai Eclipse plug-in. This tool alleviates the definition and the maintenance of the different Malai elements. We plan to check these improvements on further more complex examples.

[2]We suppose the mobile phone supports Java applications.

9.2 Olsen criteria

We now evaluate Malai using two criteria introduced in [21]: flexibility and generality. *Flexibility* evaluates "*if it is possible to make rapid design changes that can then be evaluated by users*". The Malai flexibility consists of being able to:

1. Reuse actions, presentations, instruments and interactions of an IS for several platforms.
2. Easily plug in/out interactions and actions to instruments.
3. Reuse predefined interactions in different IS.

The development effort comparison detailed in the previous section illustrates our statements: the modularity of Malai, the predefined interactions, and the use of instruments help to quickly modify an IS.

Generality evaluates the possibility of the proposed solution to be used in different use cases. Malai has been applied for the specification of WIMP and post-WIMP IS: interactions such as bimanual and multimodal interactions can be described as well as classical interaction based on widgets. However, Malai has not yet been applied on mixed reality and tangible IS.

10. CONCLUSION AND FUTURE WORKS

We have introduced a model called Malai dedicated to the conception of multi-target and post-WIMP IS. Our approach considers actions, interactions, instruments, presentations and user interfaces as first-class objects. Such a decomposition aims to alleviate the development of IS by improving the reuse of these objects. For instance, a set of predefined interactions can be reused in different IS. Moreover, actions, presentations, *etc.* defined for one execution platform can be reused for another. Our approach also provides processes to clearly define interim feedback and to abort interactions and actions. These processes aim to improve the usability of an IS by letting a user: be aware of the current state of an IS; control interactions and actions he carries out.

Future work will consider two issues: the dynamic adaptation of an IS at runtime when the context of use changes; the user and environment context of use parameters. Works on transforming a task model to an abstract user interface will be carried out too. Experiments will be performed to evaluate the development cost of several complex case studies.

11. REFERENCES

1. C. Appert and M. Beaudouin-Lafon. SwingStates: adding state machines to Java and the Swing toolkit. *Software, Practice and Experience*, 38(11):1149–1182, 2008.
2. L. Bass, R. Little, R. Pellegrino, S. Reed, R. Seacord, S. Sheppard, and M. Szezur. The arch model: seeheim revisited. In *User Interface Developers Workshop*, 1991.

3. M. Beaudouin-Lafon. Instrumental interaction: An interaction model for designing post-WIMP interfaces. In *Proc. of CHI '00*, volume 2, pages 446–453, 2000.

4. O. Beaudoux and M. Beaudouin-Lafon. OpenDPI: A toolkit for developing document-centered environments. In *Enterprise Information Systems VII*. Springer, 2006.

5. A. Blouin and O. Beaudoux. Malai: un modèle conceptuel d'interaction pour les systèmes interactifs. In *Proceedings of IHM'09*, pages 129–138. ACM Press, 2009.

6. A. Blouin, O. Beaudoux, and S. Loiseau. Malan: A mapping language for the data manipulation. In *DocEng '08: Proceedings of the 2008 ACM symposium on Document engineering*, pages 66–75. ACM Press, 2008.

7. G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting With Computers*, 15(3):289–308, 2003.

8. M. Collomb and M. Hascoët. Extending drag-and-drop to new interactive environments: A multi-display, multi-instrument and multi-user approach. *Interacting with Computers*, 20:562–573, 2008.

9. S. Conversy, E. Barboni, D. Navarre, and P. Palanque. Improving modularity of interactive software with the MDPC architecture. In *Proc. of EIS'07*, 2007.

10. M. Crease, P. Gray, and S. Brewster. A toolkit of mechanism and context independent widgets. *Lecture Notes in Computer Science*, 1946:121–133, 2001.

11. P. Dragicevic and J. Fekete. Input device selection and interaction configuration with ICON. In *Proceedings of IHM-HCI 01*, pages 543–448. Springer Verlag, 2001.

12. E. Freeman and E. Freeman. *Design Patterns*. O'Reilly, 2005.

13. G. Gauffre, E. Dubois, and R. Bastide. Domain-specific methods and tools for the design of advanced interactive techniques. *Lecture Notes in Computer Science*, 5002/2008:65–76, 2008.

14. P. J. Hayes, P. A. Szekely, and R. A. Lerner. Design alternatives for user interface management sytems based on experience with cousin. In *Proc. of CHI '85*, pages 169–175. ACM, 1985.

15. C. Kazoun and J. Lott. *Programming Flex 2.0*. O'Reilly Media, 2007.

16. C. N. Klokmose and M. Beaudouin-Lafon. VIGO: instrumental interaction in multi-surface environments. In *Proc. of CHI '09*, pages 869–878. ACM, 2009.

17. B. A. Myers. A new model for handling input. *ACM Transactions on Information Systems*, 8(3):289–320, 1990.

18. D. Navarre, P. Palanque, J.-F. Ladry, and E. Barboni. ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. *ACM Transactions on Computer-Human Interaction*, 16(4):1–56, 2009.

19. D. A. Norman and S. W. Draper. *User-Centered System Design: New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates, 1986.

20. OASIS. User interface markup language (UIML), 2008.

21. D. R. Olsen, Jr. Evaluating user interface systems research. In *Proc. of UIST '07*, pages 251–258. ACM, 2007.

22. F. Paternò, C. Mancini, and S. Meniconi. ConcurTaskTrees: A diagrammatic notation for specifying task models. In *Proc. of INTERACT '97*, pages 362–369, 1997.

23. F. Paternò, C. Santoro, J. Mäntyjärvi, G. Mori, and S. Sansone. Authoring pervasive multimodal user interfaces. *Int. J. Web Engineering and Technology*, 4(2):235–261, 2008.

24. M. Serrano, L. Nigay, J.-Y. L. Lawson, A. Ramsay, R. Murray-Smith, and S. Denef. The openinterface framework: a tool for multimodal interaction. In *CHI '08: CHI '08 extended abstracts on Human factors in computing systems*, pages 3501–3506, 2008.

25. B. Shneiderman. Direct manipulation: a step beyond programming languages. *IEEE Computer*, 16(8):57–69, 1983.

26. B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 2004.

27. P. Szekely. Retrospective and challenges for model-based interface development. In *Proc. of DSV-IS'96*, pages 1–27. Springer-Verlag, 1996.

28. J. Vanderdonckt. A MDA-compliant environment for developing user interfaces of information systems. *Lecture Notes in Computer Science*, 3520/2005:16–31, 2005.