31: Multi touch support implementation in Pharo

François Lepan
Benjamin Van Ryseghem

Author: Stéphane Ducasse

24th May 2013

Thanks

We would like to especially thank Stéphane Ducasse who proposed the subject and also guide us along the whole project.

We would like to thank Igor Stasenko for the work he did about events, and the explanation he gave us to be able to continue his work.

We also would like to thank Esteban Lorenzano for his tips about design and his help for understanding and modifying the virtual machine.

We would like to thank the whole team. It was a real pleasure to work with you.

We also would like to thank the reviewers: Camillo Bruni and Damien Cassou.

Résumé

Dans le cadre de notre Projet Individuel (PJI), nous avons eu l'opportunité d'implémenter un support pour les événements multi-touches dans l'environement de développement Pharo, au sein de l'équipe RMoD d'Inria. Ce projet s'inscrit dans la continuité du travail effectué lors du précédent semestre au cours du Projet Encadré (PJE) qui était l'implémentation simplifié de gestion de gestes en Java.

Le but de se projet est de fournir un support modulaire et évolutif pour la reconnaissante et l'analyse de geste en Smalltalk, afin de faire de Pharo le premier Smalltalk supportant les événements multi-touches.

Abstract

During our Projet Individuel (PJI) at Inria in the RMoD team, we had the chance to implement a multi-touch support for the Pharo environment. This project continues the work done during last semester in the lecture Projet Encadré (PJE). This project consisted of a basic implementation of gesture recognition in Java.

The goals of our project are to propose an evolving and modular support for gesture analysis and recognition in Pharo to make it the first Smalltalk implementation supporting multi-touch events.

Contents

Chapter 1

Introduction

Currently student at the *Université Lille 1* in Villeneuve d'Ascq in the second semester of the computer science master degree, we had to do a project proposed by a researcher along the whole semester. We have done the project proposed by Stéphone Ducasse, head of the RMoD team at Inria. We had previously been in contact with Dr. Ducasse since Benjamin Van Ryseghem has done several internships at his team. We submitted to this project since it was close to what we had done during the last semester and we wanted to continue on this interesting topic. Moreover the Pharo community is really looking forward a multi-touch solution as the industry is in need of such technology.

Context: Smalltalk is an object-oriented language whose first implementation was in 1972 and leads to major user-interface improvements like multi-threaded UIs, the MVC pattern, a UI framework based on the composite pattern (with Morphic) and a lot more. But now Smalltalk lays behind compared to what other languages and integrated development environment propose. To tackle this situation, Pharo is bridging this gap between what the other IDE proposed and what Pharo proposes in term of user interaction since year 2009 to propose a powerful development environment. *Cthulhu* , as we have named this project, aims to provide a multi-touch support for Pharo.

Problems: The main problem is that the whole event infrastructure of Pharo was designed far before ideas such as multi-touch appeared. Due to this, the events infrastructure was not meant to support multi-touch. In addition, the handling of basic system events like scrolling are handled directly by the virtual machine. Because of that, it is difficult to modify the handling of events. Moreover, the Smalltalk philosophy is to have a dynamic system. Since the virtual machine needs to be recompiled each time a change is done, it is impossible to change its behaviour on the fly.

Another problem is that information about events available at image side provided by the virtual machine is not in a dedicated object but in a low-level array. Due to that only few people have the knowledge about the construction and how to interpret the data stored in those arrays.

Goals: The goal of the *Cthulhu* project is to provide a dynamic, evolutive and modular support for gesture analysis and recognition in Pharo. We wanted *Cthulhu* to be modular to allow one to add his or her own gestures easily without needing to have deep knowledge about our implementation. In addition, part of the Pharo philosophy is that a system should always be evolving in order not to die. That is the main reason to make our project infrastructure as clear as possible to ease the *Cthulhu* evolution. In a nutshell the goals are:

- fix the system events infrastructure;
- add multi touch events to the current infrastructure;
- implement a modular gestures analyser covered by automated unit tests;
- plug the virtual machine events to this new infrastructure.

Contributions: Our contributions to this project was to:

- initiate the project;
- fix and improve Igor Stasenko's work about system events;
- implement a gesture analyser with a state machine;
- add test coverage for gesture analysing (with possibility to record gesture and replay them in the context of a unit test);
- generate complex events according to the gesture analysis.

Chapter 2

Context

In this chapter we introduce the environment we worked in during our project in three parts, the team we were working with, the language we used and the current situation of Pharo about events.

2.1 The RMoD team

Presentation: The goal of the RMoD team is to help remodularisation of object-oriented applications. This goal follows two complementary lines: reengineering and definition of new constructs for programming languages. To help reengineering, new analyzation techniques are proposed in order to understand and remodularise big applications (specialised metrics, adapted visualisations, *etc*). In the context of programming languages, constructors for the modularity features and new systems modules validation are performed. The team is also working on a secured kernel for Pharo, an IDE for Smalltalk used and maintained by the team.

Application remodularization: The evolution of an application is limited by strong dependencies between its inner parts. That is why it is crucial to answer the following questions: *"How can we substitute a part by another one with minimal impact?"*, *"How to identify reusable elements?"* and *"How to modularise an application when there are bad references?"*. Answering those questions is the goal of Moose, the software analysis environment used by the team. Moose provides a set of analyzation techniques. This work is divided in tree parts :

- Tools to understand big applications (packages/modules);
- Analysis for remodularization;
- Software quality;
- Meta-tools to help build new analyses and new tools.

Semantic elements for modularity. This second line focuses on the definition of new semantic elements for languages in order to construct flexible and

reconfigurable software. The team continues its efforts on Traits and Classboxes and also works on new areas such as security in dynamic languages. RMoD works on:

- the definition of a *Traits-only* language and;
- reconciliation between reflective languages and security.

2.2 Pharo in a nutshell

Pharo is the development environment used by the team as part of the community, therefore this is the language we used for our project. To understand the challenges we faced we briefly present Smalltalk and its main characteristics.

2.2.1 What is Smalltalk ?

Smalltalk is an object-oriented, reflective, dynamically-typed programming language.

- Object-oriented: Smalltalk programs are made of objects which communicate using messages (like in Java or C++)
- Reflective: any object can inspect and modify its own structure at runtime (like Java, but to a much greater extent)
- Dynamically typed: variables do not have a type at compilation, only values have types
- *Everything is an object*: objects are the sole kind of runtime value

2.2.2 Smalltalk Basics

Smalltalk is based on two classes which constitute the conceptual core of this system, Object and Class. In Figure 2.1 you can see that each element cannot exist alone. The bootstrap[1] is the process which leads to this state. However, since Class and Object need other objects such as strings, characters, streams, numbers. The real bootstrap is more complex.

The most important thing to know is that a bootstrap is a process where a system is initialising itself via its own execution. It is close to the *Chicken or egg dilemma* where each one deeply depends on the other one.

2.2.3 Some Basic Code Lines

Here are few examples of Smalltalk code to know how to read further examples:

```
"Variables declaration"
| variable1 variable2 |

"Instance creation"
```

[1] the term *bootstrap* is often attributed to Rudolf Erich Raspe's story *The Surprising Adventures of Baron Munchausen*

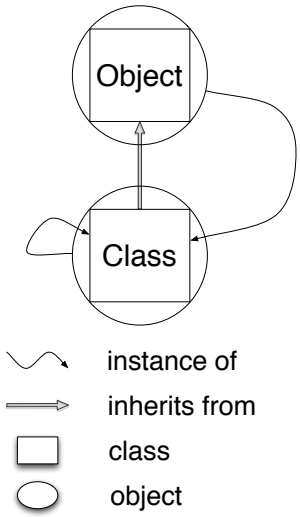

Figure 2.1: Class and Object bootstrap

```
variable1 := Point new.
"Instance setting"
variable1 x: 1.
variable1 y: 2.

variable2 := Point new.
variable2 x: 1.
variable2 y: 2.

"Equality versus Identity"
variable1 = variable2. true
variable1 == variable2. false
```

Here, we can see three things :

- new : a class method that creates a new instance of the receiver. E.g. "`Point new`" sends the message #new to the class Point which is also an object.
- = : tests if two objects represent the same object, it is a *logical* equality. It is a message, asking the receiver (before =) whether it is the same object as the argument (after the =).
- == : tests if two objects point to the same reference, it is a *physical* equality *i.e.,* identity. It is a message as well.

Let us see a basic method of the Integer class:

```
Integer>>plus: integer1 andPlus: integer2

    ^ self + integer1 + integer2
```

This method can be invoked using the following snippet:

```
5 plus: 3 andPlus: 7
```

and returns *15*.

In the fictional example, we learn three new things:

- the colon : : the way to specify parameters to most methods.
- self : the receiver of the method (similar to this in Java).
- the caret ^ : it allows you to return a value.[2]By default a method returns self *i.e.,* the receiver.

This method is equivalent to `public int plusAndPlus(int x, int y)` in Java.

A method is often referred to by the notation Class≫#selector to have a unique way to refer to a method. So the method we just saw is noted Integer≫#plus:andPlus:. One more example to see the last syntax elements, a method of class Class:

```
Class>>copyMethodDictionary
  "This method answers a copy of my method dictionary"

  | result |
  result := SortedCollection new sort: [:m1 :m2 | m1 selector < m2 selector].
  self methodDictionary do: [:method |
     result add: method.
     Transcript
        show: method selector asString, ' added.';
        cr].
     ^ result
```

Here we have :

- "some text" : a comment.
- [:argument | code] : a block (a λ-expression). They act like anonymous methods where arg is an argument of the block that is used to execute the code. In addition it captures its creation environment. It is a lexical closure.
- receiver m1; m2 : a cascade of messages. It means that the receiver of the second method (m2) is the same that the receiver of the first method (m1) receiver, in this case receiver.

Now, you know the syntax of Smalltalk.

[2]You can sometime see ↑ instead.

2.2.4 The Virtual Machine

As in some other languages (especially Java), Smalltalk's methods are converted and interpreted by a VM. In fact, the Smalltalk compiler analyses the code then creates a CompiledMethod which is a representation of the method but including more information ready to be executed by a *bytecode* interpreter or JustInTime translator.

- the *bytecode* : the source code converted into a language that the VM can interpret
- the *literals* : represent low-level objects such as numbers true, false and strings that are referenced and read by the scanner at compilation time. Other kinds of *literals* store pointers to the classes referred into method's source code

Method

Let us see an example, String≫#copy :

```
copy

    | string |
    string := String new: (self size).
    self doWithIndex: [:character :index |
            string at: index put: character].
    ^ string
```

First, let us explain what this method does :

- | string | : declares a new variable named string.
- string := String new: (self size) : creates a new instance of the class String with its size set to the size of the receiver and then stores the created string in the variable named string.
- self doWithIndex: [:character :index | : iterates over the receiver and for each character we store the element in the variable character end the index of the element in the variable index.
- string at: index put: character : at the index index of string, we put character.
- ^ string : we finally return the variable string.

In a nutshell, this method basically parses the receiver (which is a String) and fills up a new String with the same characters at the same indices.

2.3 Pharo and Events

Currently the Pharo event handling is mainly done by the VM. Indeed the virtual machine is currently responsible of transforming the operating system events into an event representation forwarded to the image. In addition, the

Pharo object responsible for dispatching events to the correct graphical widget is the widget corresponding to the mouse pointer.

Since the historical implementation, the whole system infrastructure has evolved a lot. Efforts have been made to take profit of this infrastructural changes and to make the event handling properly dispatched and managed.

But right now, this events are still represented in the system as an array with magical values corresponding to mouse position, or pressed button. By example the first item of the array correspond to the *World*[3] in which the event should be propagated. Even so the notion of multiple worlds has been dropped a long time ago. In addition, the knowledge about how to convert this array into a real event object belongs to the mouse pointer widget which is not really object-oriented since it induces a double responsibility for the mouse pointer widget class.

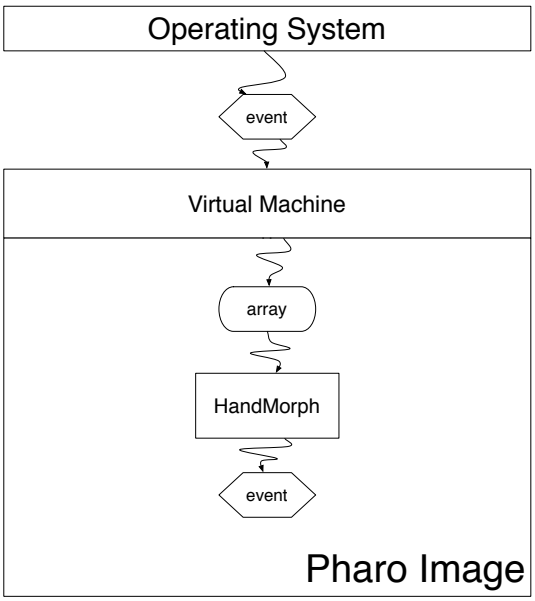

Figure 2.2: The propagation form a system event to Pharo

In the Figure 2.3 you can see how an event from the operating system is propagated to Pharo. First the event is caught by the virtual machine which does some computation. The result of this computation is stored in an array and forwarded to the image. Finally, the HandMorph class[4] compute the virtual machine array. This computation results in the generation of a Pharo events which is then propagated through the widgets.

One problem of this infrastructure is the double computation where only one computation is needed. Moreover, the fact that the part of the computation is done inside the virtual machine reduces the access to the computation logic, and limits the computation logic modifications as well. Pharo boasts about

[3]Equivalent to workspace in Linux or Mac OSX.
[4]The class of the mouse pointer widget

being self documented since all the source code is directly accessible, all the logic encapsulated in the virtual machine breaks this assumption.

Another issue with this infrastructure is that it makes difficult to simulate the generation of an event since the computation and the dispatch are encapsulated inside only one class. Having those two steps separated would ease the simulation of events propagation.

In a nutshell the current event implementation is for old, difficult to understand because of structural issues, and should be fixed in order to provide an object-oriented way to handle events.

Chapter 3

Challenges

In this chapter we first list the problems encountered in Pharo related to events and then define the goals of our project.

3.1 Current Problems in Pharo

The current Pharo system has problems for easily handling multi-touch gestures at different levels:

- The virtual machine is partially interpreting the operating system events while the interpretation should only be done on image side to allow modification;
- The fact that the virtual machine provides on image side an array with magic values reducing the understanding of generated events;
- The lack of decoupling in the event array computation and generated event propagation;
- The fact that the event generation on instance side is done in the inappropriate object.

3.2 Goals of the Project

Reimplementing the event infrastructure. Starting from Igor Stasenko's work, we need an image-side event hierarchy that can be easily extended to fit gesture triggered events. The infrastructure also needs to provide a good abstraction of the virtual machine events array in order to offer a better understanding of Pharo low-level events manipulation.

Implement a gesture analyser. Once the event is caught by the image, it then needs to be analysed in order to trigger the system event fitting best the gesture performed by the user. The analyser needs to be extendable to provide the opportunity for future users to extend the field of possible gestures recognised.

Virtual Machine modification. The virtual machine has to be modified to not transform operating system events but to forward them directly to the image in order to delegate the understanding and analyses of the events to the image. This way the event computation will be in only one place and the underlying computation logic will be able to be modified at run time.

Chapter 4

The Cthulhu project

Cthulhu[1] is a project whose goal is to implement a multi-touch event support for Pharo. The projects has been splitted in 5 different parts: refactoring of the events hierarchy, getting operating-system events from the virtual machine, gestures recognition itself, implementation of a state machine and virtual machine modification.

4.1 Refactoring of the System Event Hierarchy

Goal: The goal is to have an event infrastructure that can be extended to support evolution.

Problems:

- How to modify the events implementation without breaking the image?
- How to make the infrastructure extendable?

Solutions:

- The solution proposed should be able to modify the event implementation without breaking the image. The solution should also provide an alternative infrastructure pluggable in place of the current one, and to revert the change if any error happens.
- To make the infrastructure able to evolve along with Pharo it has been decided to use a *command* pattern for the gestures. Indeed if one wants to add a new event he or she will only have to add a new class extending the class named *SystemInputEvent*. The Figure 4.1 shows the class hierarchy of *SystemInputEvent*. All subclasses of *SystemInputEvent* should implement the method #accept: used for a *double dispatch* pattern. Here is an example of such a #accept: method:

```
SystemKeyboardInputEvent>>#accept: anObject
    ^ anObject handleKeyboardInputEvent: self
```

[1] Because of the multiple tentacles

If one wants to add his or her own event inside this hierarchy, only an #accept: method is needed as long as the method properly dispatch its behaviour.

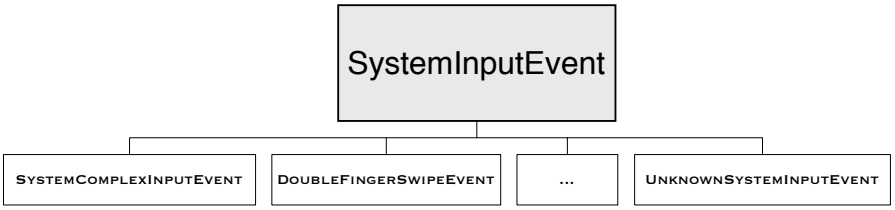

Figure 4.1: The SystemInputEvent hierarchy

4.2 Getting Operating System Events

Goal: The goal is to retrieve events performed by the user to be able to analyse them and to generate the according gesture.

Problems:

- How to retrieve events from the operating system without modifying the Pharo virtual machine?
- How to inject these events into the Pharo event handling mechanism?

Solutions:

- To retrieve event from the operating system we used TUIO which is a protocol for multi-touch events. TUIO retrieves the multi-touch gestures from the hardware and then generates events for each finger on the device. We used Tongseng[2], a software that generates TUIO events from the trackpad.
- To inject the received events into Pharo, we used a TUIO client implementation made by Simon Holland[3] that parses TUIO events and creates Smalltalk objects for cursors and blobs. Then we were able to use those objects to simulate operating system events.

4.3 Gesture recognition

Goal: The goal is to recognise gestures performed by the user. Theses gestures can be a click, a swipe, a scroll, etc.

[2]Fajran Iman Rusadi - https://github.com/fajran/tongseng
[3]http://mcl.open.ac.uk/sh/squeakmusic.html

Problems:

- How to make to distinguish two different gestures?
- How to easily define a new gesture?

Solutions:

- To make the difference between two gestures we analyse the blob related to the event. Then we use the *Naive Bayes*[4] algorithm to categorise the event as the most probable gesture.
- To let one add a new gesture we used a *command* pattern to dispatch the responsibility of managing the received event to the correct analyser. The Figure 4.2 presents the class hierarchy of *GestureAnalyzer*. The following code shows how the correct analyser is chosen when it comes to update a cursor:

```
updateCursor: aBlobCursor
    | matchingHandlers events |

    matchingHandlers := self handlers select: [:analyzer | analyzer handleUpdate ].
    events := matchingHandlers collect: [:each | each updateCursor: aBlobCursor ].
    events do: [:each | each ifNotNil: [ self eventAnnouncer announce: each ]]
```

For each analyser (named `handler` in this method), it is asked if the analyser can handle the current event. Then the analyser matching the current event interpret the event and its blob to produce events. Finally the generated events are propagated through the system. IF one want to add here a new analyser, he or she needs to subclass *GestureAnalyzer* and to implement the method #handleUpdate to make it returns `true` in the needed situation. Here is an example of #handleUpdate:

```
OneBlobGestureAnalyzer>>#handleUpdate
    ^ self numberOfBlobs == 1
```

This method means that the OneBlobAnalyzer is the analyser to use when only one blob is invalided *i.e.,* only one finger is used.

4.4 Implementation of a State Machine

Goal: The goal is to let the user perform a gesture and reduce the noise introduced by variations in gesture parameters which could lead to a wrongly analysed result. A state machine is used to keep the context of the currently

[4] https://en.wikipedia.org/wiki/Naive_Bayes_classifier

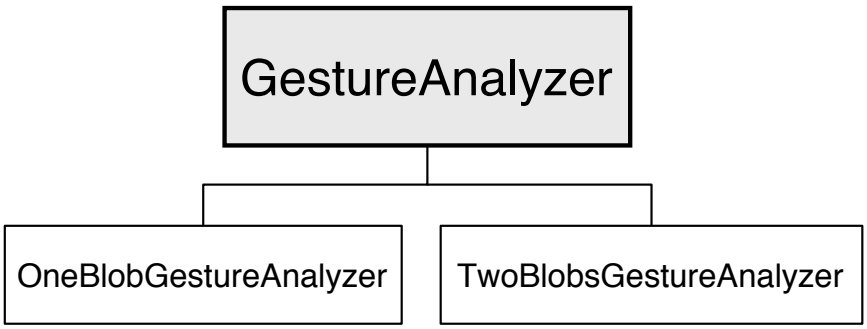

Figure 4.2: The GestureAnalyzer hierarchy

performed gesture to improve accuracy while categorising the event into a gesture. Moreover, for accuracy reasons, while a state has been set, the state machine sticks to this state until one blob disappears *i.e.,* a finger is removed from the input hardware.

Problems:

- What is the threshold before actually switching to a definitive state?
- How to ensure the correctness of the analysis?

Solutions:

- In order to switch to another state we first go through discovering-states showing the will to switch to another state. If enough same discovering-states have been consecutively triggered then the state machine switch to the according state. By example, while the user wants to perform a swipe gesture, it may happened that because of the human factor, an event is analysed as a rotating gesture. In this case, the state machine instead of directly switching to the ROTATING state first go through a discovering-state named PREROTATING pointing that the user may would like to do a rotation. Then if the same discovering-state is computed five[5] times in a row, the state machine switch to the real state: ROTATING.
- To ensure the correctness of the analysis we implemented unit tests covering the analyses. This way we were able to reproduce exactly the same scenario until the result actually matches the expected result. To get more precise tests, we implemented a gesture recorder and a gesture player. The first one to record a real gesture, and the latter one to replay this gesture during a unit test.

[5]This value is a setting chosen by experimentation.

4.5 Virtual machine modification

Goal: The goal is to analyse events not in the virtual machine but directly in the Pharo image where the behaviour can be modified on the fly. This part is related to the retrieval of operating system events addressed in section 4.2.

Problems:

- How to modify the virtual machine to make them forward the events to the image?
- How to access the hardware events from the virtual machine to prevent the operating system gesture analyses?

Solutions: This part has not been done yet since the four people in the world knowing the Pharo virtual machine were too busy to help us doing this. We approached Esteban Lorenzano who told us that it is doable and we will work with him during this summer in order to validate this approach.

Chapter 5

Conclusion

5.1 Technical Results

We have learned a lot of technical details during our project at many levels. Here is a non-exhaustive list of things we have learned or deepened.

Programming language. We already knew Smalltalk before the beginning of this project, but we have improved our knowledge about the Smalltalk language. Especially because we borrowed a lot of books from the lab, in particular the ones about dynamic languages. Thanks to Smalltalk, and the fact that you can inspect living objects, we now understand more deeply object-oriented languages.

Pharo. By analysing the Pharo system event architecture then the whole system, we have read a lot of code and this way we learned a lot about the Pharo internal structures. The *Cthulhu* project allows us to think at the precise definition of event hierarchy. Thanks to this analysis, we now have a better comprehension of modularity.

English. The team being multi-cultural with people from several countries, English is used all the time for internal communication. Moreover, all the mails shared on the Pharo mailing list are in english too.

5.2 Human results

5.2.1 Integration in Research Group

Even if we learned a lot of technical details, humanly we discovered a new working environment inside the RMoD team, where communication and autonomy are really important.

- Communication is the backbone of the research work whether written or oral. Moreover, we have often taken a seat and asked questions to team members, and spent hours sharing ideas and test them. A large part of ideas used in *Cthulhu* were born this way, and it was really pleasant to work this way.

- The autonomy in work was important too because we have to make our own schedule and to learn how to manage our time. Moreover, we were alone working on the *Cthulhu* project, so we had to set a rhythm by ourselves.

We also had multiple points of view on the work of a researcher, which is the job Benjamin Van Ryseghem would like to do. Moreover, the team being multi-cultural, we have learned some cultural parts from Argentinian culture (like Alfajores), or Ukrainian. It was really cool to practice our english with people from all over the world.

5.2.2 Pharo, a living community

Beside working for Inria, we worked as active members of the Pharo community. Benjamin Van Ryseghem developed some projects before doing this project. Those projects have been improved and integrated into the current version of Pharo. These improvements have been done with the help of other members of the community especially during sprints (coding session). The community is really reactive and any question, from the dumbest ones to the most specific ones, can be asked on the mailing list, you will always get an answer.

In a nutshell: It was really a good experience which we hope to reproduce. We really like to manage a project by ourselves, and to schedule our work alone.

5.3 Conclusion

Context: After the project we did last semester, we wanted to deepen our knowledge about event handling and how to disambiguate the gesture recognition. Moreover the Pharo community was looking forward for a real multi-touch event support made real by Stéphane Ducasse while proposing this topic. Hopefully this project will lead to the first Smalltalk implementation of multi-touch gesture recognition.

Goal: The goal of this project was to implement a gesture analyser able to be extended and to provide unambiguous gesture for Pharo events handling mechanism. In parallel, we had to fix the Pharo infrastructure to ease the events modularity.

Problems: The most important problems we encountered were:

- How to get operating system events into Pharo?
- How to fix the event infrastructure to let us generate the wanted events?
- How to make the state machine flexible enough to always have the state matching the user's will?
- How to implement an extendible architecture for gesture analysers?

Solution: After structural refactorings we implemented a gesture analyser easily supporting extensions as well as a state machine based on a known algorithm to categorise events based on a set of references.

Next Steps: The next steps will be to modify the different virtual machines to make them directly forward operating system events to the image and to delegate events analyses to the image. Another step will be to improve the categorisation algorithm by keeping track of analysed gestures as references for future categorisations.

Conclusion: As a conclusion, *Cthulhu* provides a fully tested and working support for multi-touch events as well as an extendable analyser architecture. Starting from the work of Igor Stasenko about system events refactoring, we finished the refactoring and implemented from scratch an architecture of analysers. The open source implementation is fully available on SmalltalkHub[1]. Along the development we also fixed the Pharo internal infrastructure to make it fit our needs. Additionally we also clean some old and obsolete code.

[1] http://smalltalkhub.com/#!/~BenjaminVanRyseghem/PJI

Résumé

Dans le cadre de notre Projet Individuel (PJI), nous avons eu l'opportunité d'implémenter un support pour les événements multi-touches dans l'environement de développement Pharo, au sein de l'équipe RMoD d'Inria. Ce projet s'inscrit dans la continuité du travail effectué lors du précédent semestre au cours du Projet Encadré (PJE) qui était l'implémentation simplifié de gestion de gestes en Java.

Le but de se projet est de fournir un support modulaire et évolutif pour la reconnaissante et l'analyse de geste en Smalltalk, afin de faire de Pharo le premier Smalltalk supportant les événements multi-touches.

Abstract

During our Projet Individuel (PJI) at Inria in the RMoD team, we had the chance to implement a multi-touch support for the Pharo environment. This project continues the work done during last semester in the lecture Projet Encadré (PJE). This project consisted of a basic implementation of gesture recognition in Java.

The goals of our project are to propose an evolving and modular support for gesture analysis and recognition in Pharo to make it the first Smalltalk implementation supporting multi-touch events.