PHARO'S VISION:
GOALS, PROCESSES AND DEVELOPMENT EFFORT

Stéphane Ducasse
stephane.ducasse@inria.fr

Marcus Denker
marcus.denker@inria.fr

January 29, 2012

Abstract

This document presents the goals, processes, architectural vision and current and future development efforts in Pharo Core. It will serve as a working document to structure the effort of Pharo Core's next versions. By Pharo Core we mean the essential parts of the system such as the compiler, basic libraries and key infrastructure such as canvas, events... We hope that it will also bring good energy to achieve some of the items described in the roadmap. This document should be read as a proposal and explanation of some of our effort. Now by no means what is described is curved into stone. We are really interested by critiques, comments and suggestions. In addition, people wanting to participate to the definition of this document are welcome. Finally, we are convinced that the best core is nothing if it has not cool libraries. So we encourage people to build libraries and we will support them.

CHAPTER 1

GOALS

It may happen that people misunderstand the goals that drive Pharo and our effort on the core of Pharo. This section will make some of these goals obvious. We will work to make them real during the next five years (may be less if more people help). This is why we believe that this is important for people interested in Pharo to read at least this chapter. We sketch some possible scenarios of use cases for Pharo. Such scenarios are important since they shed light on the reasons behind some of our actions.

Now it is important to make clear that we are talking about the core of Pharo. We are convinced that the best core is nothing if it has not cool libraries. Seaside is a typical and successful example. Therefore we strongly encourage people to build libraries and we will support them. The distribution infrastructure we want to put in place is clearly an important point to help library developers.

1.1 Goals of this document

The goals of this document are:

- Explain our goals and actions.
- Describe the processes we want to put in place to achieve our goals.
- Expose our current plans and action list: some people can be too busy to read the mailing-list discussions. Sometimes people think that because Pharo does not do or has XXX is because we are not interested in that. Usually it

is not that the reason but the lack of help, not interest. This document is the map for our goals. In addition, this document puts in perspective our actions. They are concerted and centered on our vision of what an excellent system should be.

- Gather feedback and share a common vision. If you want to help improving this document or give feedback do it in the pharo mailing-list. Tag the mail with [Vision]
- Build a momentum. We believe that we were not really good at getting the exact action map we have in mind. We hope that this document will put a good light on what we want to achieve and share with the community. We also hope that it will make clear what the system we want and that we can build it together.

1.2 General goal

Pharo's goal is to deliver a clean, *robust*, *innovative*, free open-source Smalltalk *inspired* environment. We want to provide a stable and small core system, excellent dev tools, and maintained libraries and releases. Pharo is an attractive platform to build and deploy mission critical Smalltalk applications.

robust. We want Pharo to be a really robust and tested system. We have already developed an infrastructure based on an integreation server that automatically generates new images, test them and notify its results. We will also put in place another level of checks based on integration servers as well.

innovative and Smalltalk *inspired*. We want to stress the fact that Pharo will certainly derive from ANSI and other Smalltalks. We will not do that for the sake of changing. What we want to say is that if there is something that can be improved but does not conform the ANSI Smalltalk, we will do it anyway. We favor innovation and better systems. We want, for example, first class instance variables because they will open a wide range of possibilities for metamodeling, optimization at compiler level, expressing in a more compact form all kind of information...

Strong and Powerful Infrastructure as a Motto

We do not believe that we can build a nice system on top of a weak infrastructure. We are strong believers that the infrastructure should be improved so that the complete ecosystem improves. Not only we want that people build their own libraries on top of Pharo, but also we want the core of Pharo to be built on top of an

improved infrastructure. We want that both users and the system general quality, benefit from the best libraries.

We believe in software engineering tools:

- we want to have validated packages running check rules and automated tests automatically validated.
- We want Virtual Machines to run regression tests and identify possible regressions. In addition, we would like to run benchmarks and see how each commit affects the performance.
- We want to have a good interaction with the rest of the world.

1.3 Some possible future applications for Pharo

The success of Seaside is an important factor in the success of Pharo and in general a large contributor to the renewal of Smalltalk. However, it should not make us forget to think about the future of Pharo as an innovation and business platform. Here we would like to sketch some possible future use cases for Pharo. Doing so we want to share with you some of the requirements that are needed to achieve such case and that are driving some of our current efforts.

Some of the points raised below affect the core of Pharo, its Virtual Machines and others the libraries built on top. We will mainly focus on the requirements that we identified as important on our roadmap for the next two years. The list if not exhaustive but should allow one to structure the development effort.

New IDES, new interaction frameworks

We believe that Smalltalk is a good language and environment for building new advanced user interfaces. By user interfaces we mean: new widgets and UI frameworks but also new way of handling events and notifications, new devices and their specific interactions (multitouch, gestures...), etc.

In addition, this is important not only to be able to develop applications using such new devices but also to invent new abstractions to handle them. Therefore the possibility to explore the design space is crucial. DrGeoII running on iPad or android is a typical example of such a need.

Requirements

- Zoomable interface. We should be able to adapt to sizes and configuration of new devices, as well as support exploration of new kind of IDEs, desktop metaphor and

- Extensible Event Infrastructure. We should be able to add new events or totally replace them. In addition, the event loop handling and interpretation (semantics of mouse event for example) should be either replaced or simply extended.
- Pluggable UI frameworks and interaction. The infrastructure should make sure that two UI frameworks should be able to coexist (probably in separate worlds) so that we can use Morphic while designing new framework. Early refactoring made in Pharo were driven by this vision and Alain Plantec's prototype of a Cairo based multi-windowing with its own event loop proved that it was worth.

Mariano ►*Here I think you can also mention Mars (there is a team now doing the gtk par and Esteban de cocoa one) and Gaucho*◄

Large applications

We believe that Pharo is a good infrastructure to support the development of large applications. Moose is a typical example of a large and complex applications built out of several large frameworks and manipulating hundreds of thousand of objects.

Requirements

- Fast data structures and rich libraries. Obviously everybody prefer to use a fast library than a slow one. We should develop rule checking that include speed regression testing.
- Fast object loader. Manipulating a lot of data often requires to save and load large amount of objects. Having an efficient and fast object serializer is a plus.
- Solid and multithreaded FFI connection. Connecting to external libraries such as databases is a key aspect. Therefore we need a robust and multithreaded FFI library.
  Mariano ►*should callbacks support mentioned here?*◄
- 64 bits. For managing large amount of data, 64 bit architecture is also important.
- A VM capable of running large images. Even if theoretically with a 32 bits VM we should be able to run images of at least 2 GBs, this is not the case of current available VMs. For example, the Windows VM does not support images bigger than 512 MB.

Large applications

We believe that Pharo is a good infrastructure to support the development of large applications. Moose is a typical example of a large and complex applications built out of several large frameworks and manipulating hundreds of thousand of objects.

Requirements

- Fast data structures and rich libraries. Obviously everybody prefer to use a fast library than a slow one. We should develop rule checking that include speed regression testing.
- Fast object loader. Manipulating a lot of data often requires to save and load large amount of objects. Having an efficient fast object serializer is a plus.
- Solid FFI connection. Connecting to external libraries such as databases is a key aspect. Therefore we need a robust FFI library
- 64 bits. For managing large amount of data, 64 bit architecture is also important.

Advanced Multi Media Applications

OpenSophie proved that Smalltalk and Squeak in particular, the ancestor of Pharo, are good platforms for building advanced multimedia applications. We believe that Pharo can be used in such situation.

Requirements

- Fast data structures and rich libraries. As above.
- Solid FFI connection. Communicating with multimedia libraries such as Gstream.
- Solid I/O and network libraries are needed.
- Strong and flexible widget library.
- Fast object loader. Multimedia applications manipulates large objects. Having an efficient fast object serializer is a plus.
- Supporting large amount of memory.
- Fast and feature wide stream implementation. Streams should be fast but also support special features such as compression, encryption, etc.
- Ways of communicating with different clients (in case they are not built in Pharo): SOAP, web services, object serialization to other languages using JSON or similar.

- Excellent error handler. If there is an error in a server, there should be always a log, a trace, a dump, etc. As much information as possible. It cannot happen that the server just goes down and there is "no log". This impacts not only in the language side but also in the VM.
- Load balancer. It would be nice a tool that let us run different images in different VMs running in different CPUs or cores.

Language and Remote IDEs for small devices

Stef ►*marcus I need your input here*◄

There is new range of devices named Plug Computers http://plugcomputer.org/. There is a need for IDEs to develop, deploy and debug such computers. Pharo can really position itself as a strong IDE for such computers. We can imagine to deploy and remotely manage applications on such devices.

Requirements

- Small core.
- Customizable compiler.
- Remote and network tools.
- Reliable serializer to move objects between images.

Scripting Systems and Glue

Pharo has the potential to get the best of two worlds: Smalltalk interaction (efficiency of direct manipulation) and scripting. We imagine that we can write scripts and get a debugger on demand, edit the script in an interactive mode and save it. Similarly we could imagine that an image can act as a smart cache.

Requirements

- Small core. Having a small core is important to be able to startup fast Pharo and to have multiple executions.
- Scripting syntax. Coral proposes a scripting syntax for Pharo.
- Good C interaction. Being able to invoke C libraries is required.
- Good Shell interaction. Having a good integration with shells is also important.
- Libraries with scriptable API. In addition we need to have a rich set of libraries to connect to a large numbers of protocols and others (XML, JSON, REST....).

- A VM that can perfectly run headless and that the images run in such a mode can also work properly. Ideally the vm should be just a C application that can be embedded in any C application.

Robotics

Stef ►*Noury luc I would like to get your input*◄ There is a demand for good IDEs for robot application development. Being able to debug robot while the robots are running and on the fly fixing the code is needed. Pharo can be the basis for building IDEs and runtime engine for robots.

Requirements

- Network.
- Small Core.
- C integration.

Mariano ►*I would merge this section with the one of "Language and Remote IDEs for small devices"*◄

Web/Deployment Infrastructure

We can imagine systems whose interfaces is the web but that should store a lot of data and manage a lot of load requests.

Requirements

- Network infrastructure.
- Object serialization.
- New generation of OO databases. New object header.
- The VM could support the famous object immutability.

More secure languages

We believe that Pharo should provide a good infrastructure for inventing the next languages generation and in particular in the context of secure dynamic languages.

Requirements

- Modifiable VM.
- Customize Compiler.
- Flexible IDE.

- Small core.

Mariano ►*I am not sure if Pharo could be the best language for security....the fact of being able to play as god goes in contrary to security also. I would let all this security stuff for the javascript guys.* ◄

Support for Language Innovation

Helvetia [] and PetitParser are typical examples showing that language innovation can be done in Smalltalk. Pharo should be the environment to support such type of activities. In addition, Pharo should benefit from research on type inferencer, traits, ...

Requirements

- Smalltalk is a simple language: therefore its compiler should be simple and allows us to make crazy extensions.
- Tools chain should be parametrizable.

CHAPTER 2

PROCESSES

In this section we present some processes to support the generation of a small kernel and a set of validated packages.

2.1 Towards a small kernel

As mentioned in the goals Section above we want and we are steadily working *since years* step by step on code cleaning to obtain a small core. We are getting there but it requires a lot of rewriting.

In the long-term, we expect to have a minimal core with a binary loader that only load package on demand and where an image is just a smart cache.

Process. We want a process that will be centered around this small core. The path to arrive there is not trivial and not as fast as we would like. It requires a lot of cleaning because there are many dependencies and not modular concepts. We made already a lot of progress with for example the Settings framework usage. We expose some of the non technical problems we are facing in the following.

Figure 2.1 shows that we want to have a small core (See Section 3.5) and use some specification (probably in terms of Metacello configuration) to loaded a set of certified packages (The package certification is the responsibility of the catalog builder described in Section 2.2). Note that in presence of a fast binary loader (such a Fuel) we could avoid to create images and delay the creation of image.

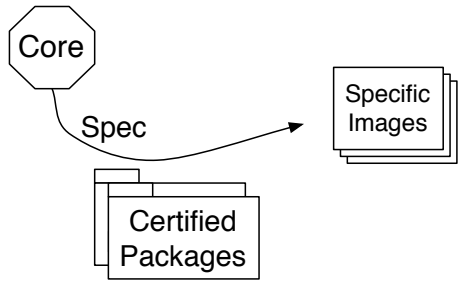

Figure 2.1: Given a Core and a Spec we should load a set of Certified Packages to produce intermediate (or not) systems expressed as full images.

Benefits. There are obvious benefits to have a small core and this is why also we are working on bootstrapping the system (See Section 3.5). We discovered and fixed already a lot of problems.

Dealing with changes. One key question is how do we deal with changes. This question is important because it has some requirements and challenges. Figure 2.2 shows that given a Core, a Spec (or set of spec), a set of Packages and a change we should get potentially a new core, a new set of specs and new set of packages.

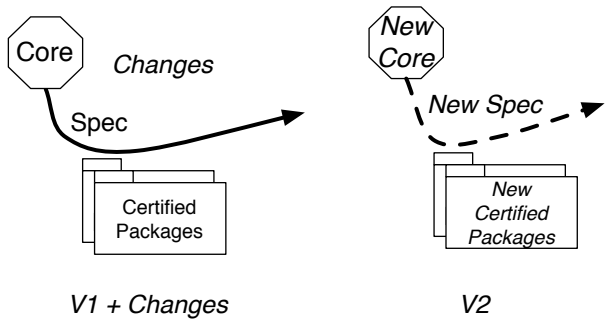

Figure 2.2: Given a Core, a Spec, a set of Packages and a change we should get a new core, a new spec and new set of packages.

Problems. We currently face some problems.

- Loading all the packages should take a reasonable amount of time because it is not reasonable to wait couple of 10 of minutes waiting that an integration engine build the system. Already publishing changes takes too much time. So recreating in addition a working system is not possible.
- Maintaining a core image and an image with some non-core but important package that are shipped is *really* difficult for the following reasons:
  - In general, we do not have the knowledge or time to maintain externally developed packages.
  - When we do a refactoring in the core it may touch parts used by the tools and since we are working on alpha branches it means that we should modify the non-core packages to reflect our changes.
- Not loading certain packages such as Morphic Example, Sounds ... is a good way to kill them.
- Since not everybody is using Metacello we have to maintain Configuration for packages we do not know, we cannot simply maintain stable and development branches.
- Having good tools to work cleaning the system is important. Right now we cannot load fast refactoring engines, change the system without impacting the refactoring tools and save.
- All the packages that are required in the core or to be loaded by the process (for example the AST package) should be owned by Pharo to be able to evolve when needed. For example to bootstrap the compiler in itself (and avoid to have to maintain two compilers) we need to have the compiler tool chained parametrized (there are other really good reasons - Read Section 3.6).

A midterm solution to get the process working is:

- to load some predefined tools or packages into the "Core" and we take the responsibility to maintain these even at the cost to make mistakes;
- modify these elements as part of the package required by the system;
- make sure that we do not introduce new dependencies.

A better solution would be that:

- All the important tools are managed by their developers which works with us and test the changes with us and maintained with stable and development Metacello tags.
- Get a really fast loader.
- Do some steps in the direction of been able to reapply changes.

We could record some changes to be applied on the packages that were not managed, but so far we do not have the right infrastructure and experience to do that.

Business Value. Having such goal and process can help getting solid, scalable systems that can also be deployed on constrained devices.

2.2 Towards a verified package catalog

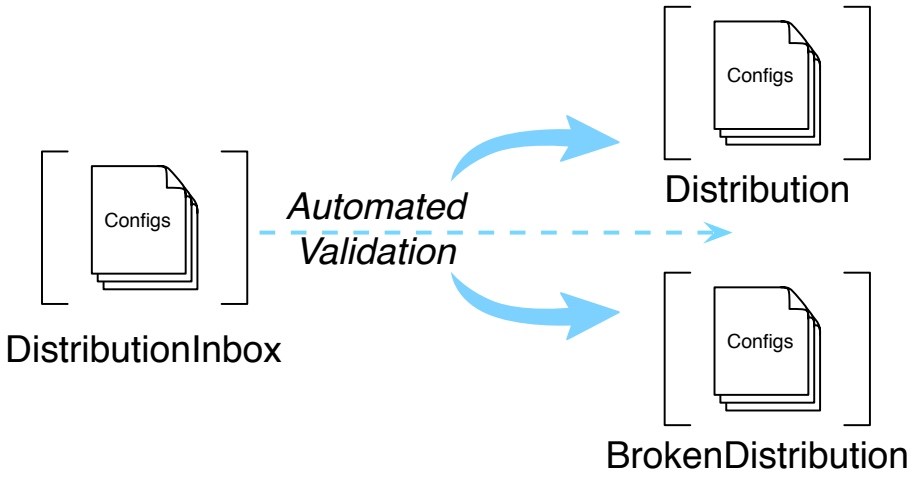

Figure 2.3: ConfigurationOfProject published in the DistributionInbox, the Configurations will be loaded and their tests and checks will be executed.

We want to provide a more robust package distribution to the end-programmer. For this we plan to put in place the following process.

Process. Figure 2.3 illustrates the principle. For each Pharo distribution we want to have one repository containing the validated configuration of projects. We foresee a validation process that works as follows: (1) configurations are published in the distribution inbox. (2) an automated process: loads, run the tests and the rules of the projects. (3) depending on the success of the previous steps the configuration is included in the distribution or moved in a broken repository for study.

In a second step, the transitive closure of all the needed packages will be copied automatically in the distribution repository to create a self contained distribution.

The act of publishing a configuration will then means something more than simply copying the distribution.

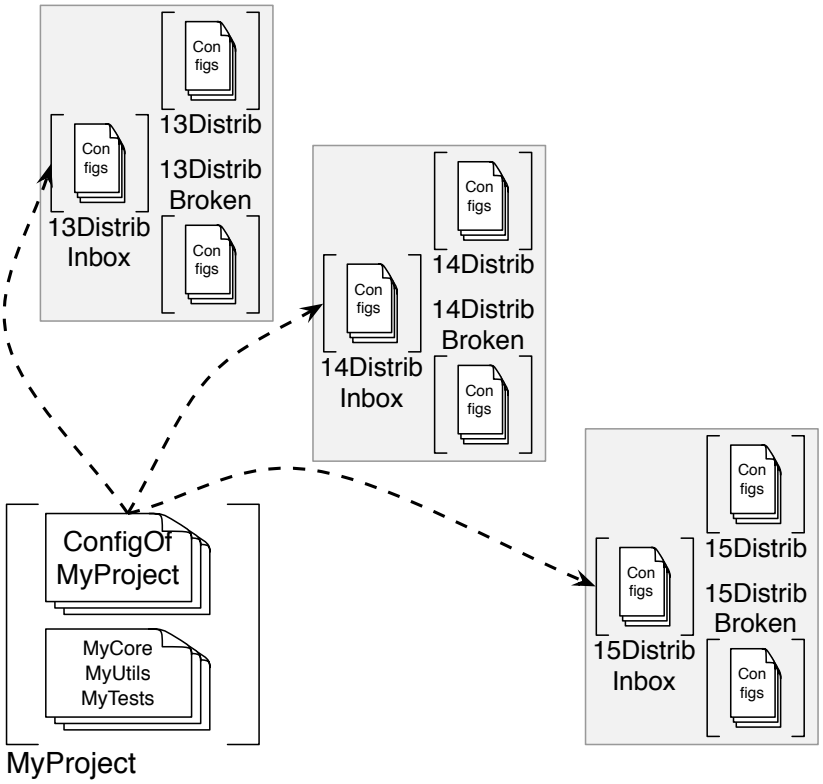

Figure 2.4: You should save your configuration in your repository. Then publish it when ready in the different repositories for validation.

About your project. Figure 2.4 shows the process we think that projects should follow.

- Always publish the configuration of your project in your project repository. Think about modularity.
- Push to the given distribution inbox when ready for it.

CHAPTER 3

CURRENT DEVELOPMENT EFFORT

Now we explain all current development efforts and why there are done that way and their impact on the resulting system. Scenario presented in the first section are also a reason why we are doing them. The list is not exhaustive but we are conscious that this is important to finish a task before passing to the next one.

Note that some efforts are simpler to integrate or require less development effort. We take a pragmatic approach: if a change is ready to get integrate we do it. Now we are conscious that we should finish some work because else we risk to lose them.

3.1 A Robust and Extensible System Events

We need an infrastructure to easily be able to plug multitouch, Genie like behavior (Genie was a hand recognition system developed by Nathanael Schaerli in 2002) and experiment with new devices or UI metaphors.

Context. Before Pharo, the VM filled up a buffer for each new event and the system checked at regular interval if there was something present (See Figure 3.1).

Then different UI frameworks (Morphic and MVC) were calling the InputSensor which had a polling logic to see if a new events was present in the buffer. In the case of Morphic, HandMorph interprets itself this buffer and creates and interprets events. Notice that there was confusion and code in Morphic was directly

calling InputSensor logic and not the HandMorph one, breaking layers. In addition HandMorph handles events using an explicit case statement logic which hampers extensibility.

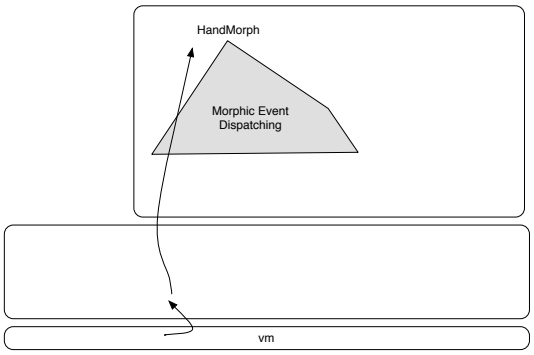

Figure 3.1: Old days: VM filled up a buffer and the system checked at regular interval if there was something present. HandMorph interprets this buffer and creates and interprets events.

Current ongoing effort. Since Pharo 1.0, several actions were made:

- An effort to use an event-driven structure was made first by Michael Rueger then followed up by Igor Stasenko and Stéphane Ducasse. An input fetcher and a input handler were introduced. This new infrastructure was already validated by the fact that it is possible to build an independent UI event interpretation loop.
- Igor Stasenko built a wait-free collection so that the events are managed in a solid data-structure.
- We systematically fixed layer violations. Morphic code should not shortcut anymore the HandMorph logic and should not access low-level input sensor.
- Improved modal logic. We introduced abstraction to HandMorph to simplify the handling of modal interaction (dropping certain event while a certain condition holds). It simplifies the logic and encapsulates it.

Pharo 1.4 can now drop the polling semantics. All the VMs support now the event infrastructure. This moves could have been done earlier but the Windows VM did not integrate a fix adding a semaphore input during one and half year. With the new VM infrastructure (jenkins and GIT/CMake) put in place by Igor Stasenko. We can now compile daily all the platforms.

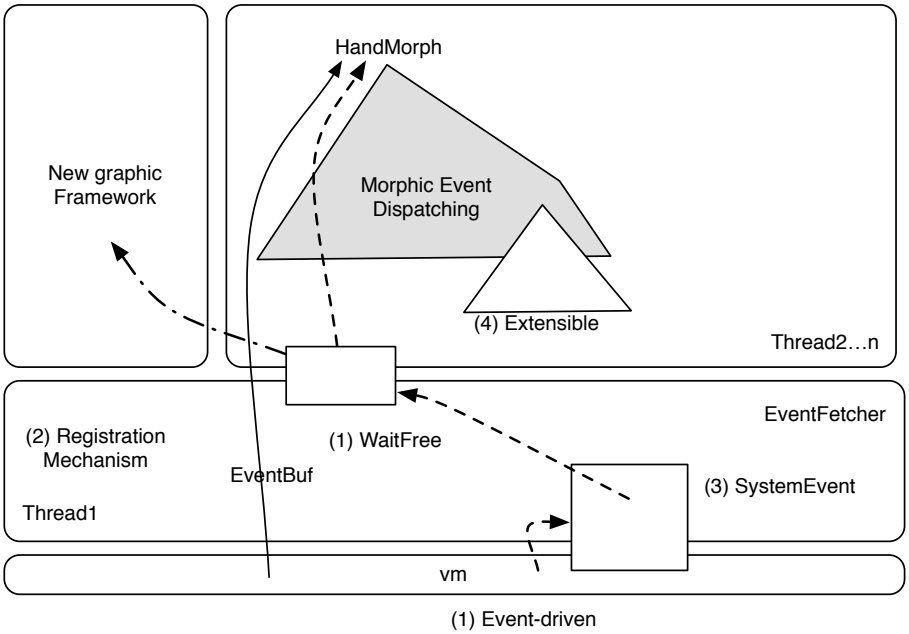

Figure 3.2: Current and future Pharo effort.

Current ongoing effort.

- We are introducing a registration mechanism, so that it can be easy to build a record mechanism of events or other broadcasting systems.
- We are introducing a major refactoring (dashed line in Figure 3.2): the event fetcher interprets the eventBuf and creates SystemInputEvents. Such events fully encapsulate the logic and behavior of low level events (avoid complex logic spread in HandMorph, duplicated code, ...) are then passed to the handler queue. Then HandMorph manipulates such objects and can decide to republish them.

What is left to do.

- We are rewriting the HandMorph logic to not use anymore hardcoded case statements. In fact there are at least two case statements (one for building morphic events from raw buffer, and one for defining the way morphic works). We have nearly fixed the first one using double dispatch.
- We should migrate HostMenus processing to an adequate place, the pasteUpMorph should be responsible to publish its menu. Now we may simply

deprecate the host menu facilities because of resources development reason.
- We should probably make sure that the libraries code that has interaction do it via Morph and not direct input sensor. For example, Rectangle fromUser should be packaged with Morph and use HandMorph API and not directly invoke input sensor.
- The second case statement logic should be rethought to support a smoother extensibility.
- We should validate the extensibility by evaluating how easy it is to introduce multitouch events and Genie in the system.

Open questions. Fernando Olivero proposed a new way to manage event and devices at the level of Morphic. Basically it acknowledge that HandMorph has too much responsibilities that the logic of the event processing is complex, brittle and difficult to extend. The model proposed by Fernando is based on decomposing responsibilities and using state pattern to identify double-click and other state-oriented situations. One question is do we propose a state-machine that can be integrated in HandMorph in a compatible way or is it better to define a separate infrastructure and after a moment to remove HandMorph. Frankly we do not have the answer (yes no magic ball here). Fernando already integrated his model with the refactoring we are performing. A possible path is to integrate our refactorings, finish the first two steps of the previous sections and evaluate how the solution of Fernando is a good alternative.

Business Value. Touch Event, MultiTouch and others like new user interface development can radically benefit from our effort. We should be able to plug easily multiple touch devices.

3.2 Rewrite of filesystem/stream

Context. Current file support is rather old, cumbersome and counter intuitive. Pharo as a scripting language should have a good and easy to use file library.

Possible Solutions. We want to use the FileSystem library. However, the state of FileSystem is not totally at the level required to support a replacement of the existing one.

- We started an effort to add comments in the code.
- We improved the FileSystem library.

- We started to write a documentation in a form of a chapter for the Pharo by Example volume two book.

What is left to do.

- Evaluate if the complete API is implemented (for example rename was missing).
- Migrate existing code to use the FileSystem library.
- Improve or add support for the Stream related API (newFileNamed, oldFileNamed:, readOnly...). A possible solution is to use the Xstream library but it should be discussed on the mailing-list.
- Finish to write a solid documentation to be part of Pharo by Example 2.

Potential Problems. Streams may be a problem or require a larger effort. It may be wise to proceed in a two step process: (1) first focus on the replacement of the file directory implementation - implement some backward compatible streams API using the existing stream library. (2) Assess whether Xtreams is the way to go. (3) Decide and migrate to Xtreams if decided.

3.3 Announcements and Ephemerons

The general question is: How can we get better weak structure and also announcement frameworks? In particular since infrastructure of the system will be based on Announcement, it is important that: (1) error in user code cannot break the dispatch of system announcement, (2) announcement registration does not impact garbage collector reclaiming of objects.

Context. In some cases announcement users need to pass a block that will be executed. Such block may refer to objects and s such it may be a problem because this blocks the garbage collector to reclaim such objects. This is why Announcements introduced a weak form but this is a just an way to provide a possible solution. This approach has the disadvantage that the user needs to decide upfront the kind of registration he wants to perform.

Possible Solutions. An Ephemeron is an interesting data structure: it is a pair whose key holds strongly an element and a value which holds weakly an element. When the value is reclaimed by the garbage collector, the key reference in turned into a weak reference. Using ephemerons we can then simplify the Announcement and make sure that clients do not have to worry about weak or not registration.

In addition,

3.4 UI Canevas for Zoomable Interfaces

Building decent graphical engines is now difficult in Pharo. Javascript often offers between rendering. How can we get decent zoomable canvas that can support 2010 UI technologies like openGL and Cairo?

Context. The current UI canvas is a bit old and shows its age. In addition the class hierarchy starts to get complex and messy.

Possible Solutions. Design a new canvas that is designed to talk to vectorial frameworks and have a default back-end as fallback. Migrate all the system to use the new API and canvas.

What have been done so far. Igor Stasenko did the following:

- Designed and implemented a new canvas API and all the necessary abstractions to generate adequate code for different backends.
- Deep integration to be able to inject frames into existing VM structure (Igor can you confirm this?)
- Designed and implemented a new textMorph, paragraph and other classes.
- Defined a default back-end.
- Defined a Cairo back-end.
- Defined an openGL back-end (using an OpenGL calling framework).

What is left to do.

- Finish the Cairo back-end for the canvas API.
- Finish the openGL back-end for the canvas API.
- Rewrite the existing Morph drawOn: methods to use the new API.
- Rewrite the existing call to the old API to the use the new API.
- Deprecate the old Canvas api.

Business Value. Applications such as Moose, Roassal, Mondrian can really benefit from it. Tablets application can also benefit from the zoomable and vectorial aspects of it.

3.5 Bootstrap of the Core

Context.

Possible Solutions. cf bootstrap paper

- Making sure that the system is executable
-
-

3.6 Fully parametrized compiler tool chain

Context. The current compiler chain is dated. It is difficult to extend it. For example, we need to be able to support the following typical use cases:

- Compiling towards another byte code sets: for example compiling specific programs to legoOS.
- Supporting a bootstrappable compiler: we should be able to recompile the existing compiler with itself to be able to modify it.
- To support, the creation of new kernels, we should be able to specify the classes of literal objects like true false nil.

Possible Solutions.

3.7 Package as Real Objects

Context. objects instead of string matching (aka cache for everybody)

Some browsers have cache but all the infrastructure should benefit from a fully robust implementation.

Senders should show packages

Possible Solutions. RPackage

3 iterations. Full tests.

Potential Problems. We will get more packages if we map category to packages. Doru suggested

Stage 1: Get RPackage in the image and pretend that it acts like a smart cache for categories. Everything is expressed in terms of categories and mirrored in RPackage. Tools can now be built on top of RPackage. At this stage, no modification is needed and the new tools will work next to the old ones.

Stage 2: Leave the logic of Monticello loading as it is now (n-to-1 Categoeies-to-MonticelloPackage), but modify Monticello publishing to rely on 1-to-1 RPackage-to-MonticelloPackage. This will basically force people to start changing the configurations.

Stage 3: Change the Monticello loading to only allow 1-to-1 RPackage-to-MonticelloPackage. At this point, we can safely remove the Categories.

RPackage supports the notion of tags. A class can belong to any number of tags.

3.8 Package Meta data

Context. Since we want to validate package using specific rules and tests. We need a mechanism to describe false positives. No developer will use a tool that report endlessly the same errors. Currently SmallLint relies on adding pragma to method body but this solution does not scale. Nobody want methods with 15 pragmas about rules. In addition, we need a way to store package documentation and other meta-data.

Possible Solutions. We are currently experimenting with the introduction of a per package manifesto class. Once a first validation on real cases will be performed we will announce it and gather feedback.

3.9 Less Model Clutter and Duplication

Context. We should stop to have models for program elements each time duplicated all the time. Example are: PseudoClass, FilePackage, MC* packages, MethodReference, RBEnvironment.

Possible Solutions. We designed Ring (a source code meta model) to serve as common basis for a large set of tools [3]. Ring is used instead of the MethodReference and CommentReference. It supports a code model that is polymorphic with Smalltalk run-time objects. Therefore a browser such as Nautilus is able to manipulate runtime classes and compiled methods as well as Ring objects (representing classes remotely accessing, on a disc...).

Ring is an important effort. Veronica Uquillas-Gomez spent several months designing, implementing and testing it. This work has a real value for the community.

What is left to do.

- Analyze the current models uses and learn how they can be replaced by ring. We did such analysis for pseudo class.
- Convert use of models into ring ones.

Business Value. Using one single code meta model or as few as possible (and polymorphic with runtime objects) will bring less code to maintain, a large opportunity for reuse - for example browsers can be reused for runtime objects, files on disc or remote objects.

3.10 Building and Reusing UI Logic

Context. Building user interface is tedious. Now existing approaches to offer builders only focus on the declaration (drag and drop) of the UI elements. Code is generated: it is not sure that the builder can be reopened on modified interface. Some UIBuilders rely on specific event mechanisms. Then more important the reuse of the logic is not supported. For example, how can we reuse that an input field selects the elements of a list?

Stef ►*More here/ mention Glamour*◄

Possible Solutions. First class objects representing the descriptions of the graphical elements like in VisualWorks or Microsoft is an interesting path.

with Specs: one model can propose multiple UI presentations.

Serializing morphs is not a good idea since the class can change.

Second we need to have models and compose them and reuse their logic. ValueHolders are a path to explore.

3.11 New Network Layer

Zinc is the large effort and we value it. Ocean is rewriting test first the socket library.

@@ MORE@@

3.12 Serializers

Context. The serializers are a bit outdated, cumbersome and not that fast. Originally we thought that ImageSegment was the solution. We apply an scientific ap-

proach and Mariano Martinez-Peck proved to us that it was not. The analysis are described in a scientific publication [2].

Possible Solutions. Fuel is a fast and solid serialization framework [1]. Its development started in 2010 and the vision was that we need a good serializer to support faster package loading.

- Fully tested,
- Several years of engineering,
- Serious benchmarking,
- Clean design

Potential Problems. MC uses old serializers. One solution is to propose a converter between the two systems.

Fuel should handle well migration.

3.13 SystemChangeNotifier replacement

Context. SystemNotifier is a mechanism that we introduced with Roel Wuyts a while ago. The idea was that each tool requiring to change its state after a change of the system happens, just needs to register its interest to the SystemChangeNotifier. Historical note: before SystemChangeNotifier, tools had to identify and modify all the places in the system to get notify. It was obviously terrible. SystemChangeNotifier is a typical example of why we need a good backbone.

One of the problems we face with SystemChangeNotifier is that the event raised are not objects and the protocol is sometimes incomplete or unclear. In addition it duplicates the behavior of Announcements.

Possible Solutions. SystemAnnouncement is a new announcement-based system notifier equivalent. As shown in Figure 3.3, SystemAnnouncement currently listens to all the notifications raised by SystemChangeNotifier and republish them as Announcements. RPackage is based on them as well as some Moose tools.

The second large step to be performed is to replace SystemChangeNotifier and use instead SystemAnnouncement.

http://code.google.com/p/pharo/issues/detail?id=5174 propose a list of steps.

- 1a) Internalize state in Announcements, instead of using data from Events.
- 1b) Implement SystemChangeNotifier protocol on SystemAnnouncer, in extension protocol *SystemEvents, so external consumers will register correctly when loaded (but receive polymorphic Announcements instead of Events).

- 1c) Implementing announcement-handling methods on current Event-consumers(ChangeSet, SmalltalkImage, RecentMessageList, TestCase, TestRunner, etc.) in Core.
- 2) Migrate current Event-consumers from SystemChangeNotifier to SystemAnnouncer
- 3) Switch SystemChangeNotifier uniqueInstance to SystemAnnouncer.
- 4) Remove all contents in System-Change Notification except SystemChangeNotifier's class method.
- 5a) Make new package from SystemEvents extension protocol.
- 5b) Store SystemEvents package in external repository, remove from Core.

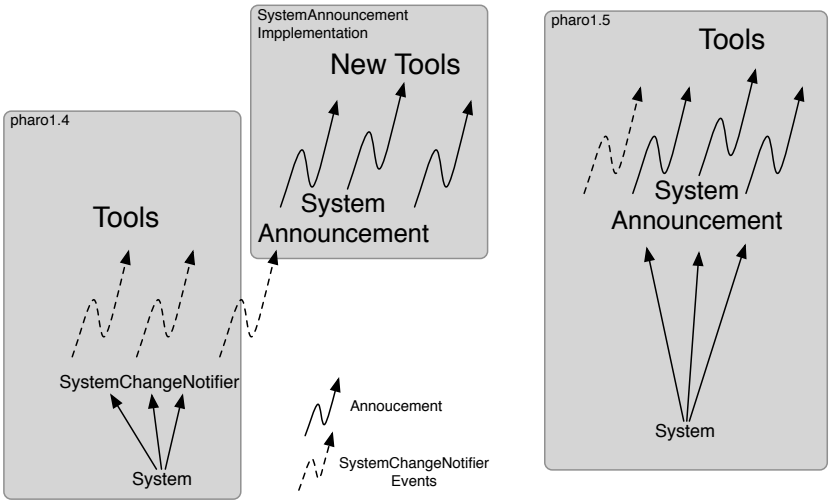

Figure 3.3: You should save your configuration in your repository. Then publish it when ready in the different repositories for validation.

3.14 Cleaning Morph

Context.

Possible Solutions.

Potential Problems.

Business Value.

3.15 One Unified FFI framework

Context.

Possible Solutions.

Potential Problems.

Business Value.

3.16 Keyboard mapping

Context.

Possible Solutions.

Potential Problems.

Business Value.

3.17 Keyboard mapping

Context.

Possible Solutions.

Potential Problems.

Business Value.

BIBLIOGRAPHY

[1] M. Dias, M. Martinez Peck, S. Ducasse, and G. Arévalo. Clustered serialization with fuel. In *Proceedings of ESUG International Workshop on Smalltalk Technologies (IWST 2011)*, Edinburgh, Scotland, 2011.

[2] M. Martinez Peck, N. Bouraqadi, M. Denker, S. Ducasse, and L. Fabresse. Experiments with a fast object swapper. In *Smalltalks 2010*, 2010.

[3] V. Uquillas Gómez, S. Ducasse, and T. D'Hondt. Ring: a unifying meta-model and infrastructure for smalltalk source code analysis tools. *Computer Languages, Systems and Structures*, 2011.