Multiplatform framework for managing windows

Michal Kevický*
Supervised by: Silvester Czanner†

Department of Applied Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University
Bratislava / Slovakia

Abstract

We introduce a multiplatform C++ framework for managing application windows respectively displayed content. The framework provides an API for handling display devices, representation and presentation of displayed content, managing input events and other actions. Furthermore, objects are extended with semantic context which defines their functional properties. Semantics allows more accurate classification of entities, choosing more accurate reactions for incomming signals. The framework provides platform, which does not strictly rely on classical WIMP (windows, icons, menu, pointer) paradigm and gives ability to develop solutions non-conventional way (e.g. zoomable UI, tangible UI).

Keywords: Graphical user interfaces, Windowing systems, Frameworks, Scalability

1 Introduction

Lately we experience vast expansion of electronic gadgets with many different graphical user interfaces. Each new interface introduces new philosophy of control, thus confusing users and leading to deformation of user's learning curves of such systems in a negative way. It increases the costs of the development of multiplatform applications and brings new possibilities for various errors.

Graphical user interface usually depends on 4 parts. The first part, an *operating system* (**OS**), is not the one which should be important for real layout. OS is here to provide communication with *input/output* (**I/O**) devices, management of resources, communication between processes, and handling permissions, and entire user model. OS defines a "platform". The second part is a window managing application (i.e. *window manager*, **WM**). It manages the content and defines how should this be displayed on attached displays. Windows management is the main contribution of our work. The third part is represented by the content of windows. It usually consists of a canvas containing set of well positioned widgets. Since there are many multiplatform widget sets (e.g. wxWidgets, Qt, GTK, Swing, AWT, ...), there should be no problem of designing and implementing portable layout of window. The fourth part is represented by input devices, i.e. physical interaction, is the part which can not be modified software way and hence also is not field of our interest.

WMs are either part of OS (like in Windows series from Microsoft) or can be separate application like in many of *NIXes. As a bonus, in *NIX X Window system, each application can act like a WM, what caused there are really a plenty of WMs now.

WMs of present days usually do not provide interface for customization/extension in non-layout properties (i.e. in behaviour, input devices, representation of windows) using predefined APIs, leading to workaround solutions. Our solution implements modular framework model which is easy to extend and modify. An end user for our framework is primarily WM or presentation logic programmer.

The framework is here to define and implement abstract interfaces for the end user to ease writing of portable respectively multiplatform code. The implementation of the framework for a specific platform consists of *core* code of the framework and set of the *platform specific modules* or *wrappers* providing functionality defined by the abstract interfaces mentioned above.

The structure of this paper is organized as follows:

Section 2 introduces few existing projects implementing similar functionality or providing relevant informations to our work.

In Section 3, we describe used technologies, reasons for their selection and their pros and cons.

Section 4 describes architecture of framework, object structure and application flow.

Section 5 presents advantages of framework and possibilities it brings for window manager / application programmer and end user of final window manager.

Section 6 sumarizes our results, section 7 presents our plans for future work and section 8 provides conclusion of

*kevicky@gmail.com
†s.czanner@warwick.ac.uk

entire article.

2 Related work

In the world of WMs, there is a large scale of a different applications or application packages, from lightweight programs to complex desktop environment packages. Especially in world of *NIXes, there are hundreds of options to choose WM. Here is an overview of few works interesting to us:

The *Compiz-fusion* [3] project is alpha for recent Linux world in field of desktops and visual effects. The Compiz-fusion project implements using AIGLX/XGL extensions to the X Windows system an compositing windows manager introducing different visual effects (wobbly windows, transparency, desktops mapped on rotatable cube, ...) to the world of Linux.

The *Metisse* [8] is French project extending FVWM project which has brought their own implemention of composite extension and has come up with their modification of X server. The goals of the project are similar to ours - easing the design, implementing and testing of new *human computer interaction* (**HCI**) approaches and techniques. What is worse, the Metisse is primary X Window System oriented. On the other hand, one of goals of our framework is the development of unified multiplatform base for managing windows.

The *SphereXP* [5] is Slovak project by Dušan Hamar, and it has been one of the pioneers of Microsoft Windows XP window managing customization. SphereXP provides extension of standard flat rectangle desktop to space inside sphere. Together with KDE project and Blackbox, SphereXP is the one that proves, that customization of the presentation layer of Windows XP is really possible.

Last but not least there are *Blackbox* [1] and *Fluxbox* [4], the fork of Blackbox. Both are very lightweight WMs implementing stackable WM concept, providing spartan visual interface. For minimum functions they provide, and MIT license, they are released under, they has became inspiration and basic code of our framework.

The Quartz-compositor as part of MacOS X by Apple or the Aero from Windows 7 by Microsoft are well known technologies so they will not be discussed.

3 Used technology

Response times of the visual interfaces are one of their critical features, so it is expected to use speed efficient language. WM can also be considered as feature from "system programming" category. On the other hand, framework is here for other users to simplify them implementation. The C sometimes appears to be one of the synonyms for system programming, but also it is not so simple as Java or C#. Compromise between the power and the userfriendliness seems to be combination of the *C* and the *C++* with possible *inter process communication* (**IPC**) with modules written in different languages.

However *Portable Operating System Interface [for Unix]* (**POSIX** IEEE 1003 standard [6]) is primarily oriented to the *NIXes, and it should be considered as a guideline for the multiplatform computing.

We have used *Boost*[2], because it is set of peer-reviewed C++ libraries which are almost the part of the upcoming C++ standard. It also comes here to provide rapid application development features.

Ontologies and *description logics* are concepts known from semantic web. Main reason for using them is a possibility of describing objects by their properties, where classification of objects using taxonomies faces problems. These entities should act like good candidates for multiple branches of a tree. Except for the problem with multiple occurances in the taxonomy tree[1], description logics solve also the problem with resolving context and scalability. For example, there is no need for launching a particular multimedia player[2], there is only need for a player capable of playing 7.1 sound, respecting DRM[3] as well as the possibility of controlling with a mobile phone using a bluetooth technology.

4 Architecture

WMs can be divided into 3 categories: the tailing WM, the stacking WM, the compositing WM.

The *tailing* WM splits planar screen into the disjoint zones (with no visual overlaping between applications). This concept has been used in earlier versions of Microsoft Windows (e.g. version 1.0).

The *Stacking* WM introduces the layers of windows, allowing overlaping of windows. This concept can be found in classical WMs running on X Windows System, including Blackbox and Fluxbox, KDE to version 4, also Microsoft Windows prior to Microsoft Aero technology, and Mac OS prior to OS X. Main difference between the *stacking* and the *compositing* WMs is, that stacking ones draw content of windows directly meanwhile compositing ones let application to render their output to buffers and the WM work with content from WM's buffers. Representatives of the compositing WMs are the Apple's Quartz-compositor, the Compiz-fusion project or the Microsoft's Aero. The primary goal of the framework was to develop the compositing window manager, but it

[1] one object could belong to multiple tree branches, which should be disjoint

[2] such a player might not even exists on target system

[3] digital rights management

is not problem to customize it for the old school methods mentioned above.

Most of the recent WMs which are implementing standard WIMP paradigm user interface suffers with several limitations. The first to mention is, that the object's structure has fixed amount of properties (e.g. window: dimensions, position, title, scalability, min/max/normal state and that is almost complete listing). There is no native way to communicate any different properties between the applications.

As WMs assume 2D windows, recent controls has also been designed for 2D interfaces. 3 (and more) dimensional interfaces are just part of sci-fi or experiments, but there is not native way for integration into the WM's system. Similar situation occures while considering displays - 2D is standard, 3D is sound of future.

The framework solves this problem with *dynamic amount of properties* of the "window" object. Window object itself is quite confusing notation in the context of the framework. The framework itself contains "content" entity which the most important property is the binary blob. The binary blob contains only "some kind of data". Other relevant property is dataType which determines, which interpreter to use (does binary blob represent pixmap, plain text, h264 video stream, set of 3D instructions?).

Our framework consists of *core*, which defines basic structure split into the four layers (*figure 1*): *Input layer* (**IL**), *Application logic layer* (**ALL**), *Presentation layer* (**PL**) and *Client application layer* (**CAL**). Other important parts of core are abstract interfaces and functions for management of modules and communication between them and the platform specific wrappers.

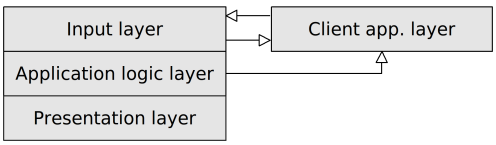

Figure 1: Framework's core

The *input layer* (*figure 2*) handles input events received from the OS, the client applications, or events generated inside the system. Events are being classified, passed to the priority queue, subsequently routed with respect to the security policies to different queues - high priority for realtime applications and slower one with ontology mapping for future pass into *application logic layer*.

The *application logic layer* (*figure 4*) is the brain of entire framework. Here is the place for modification of appli-

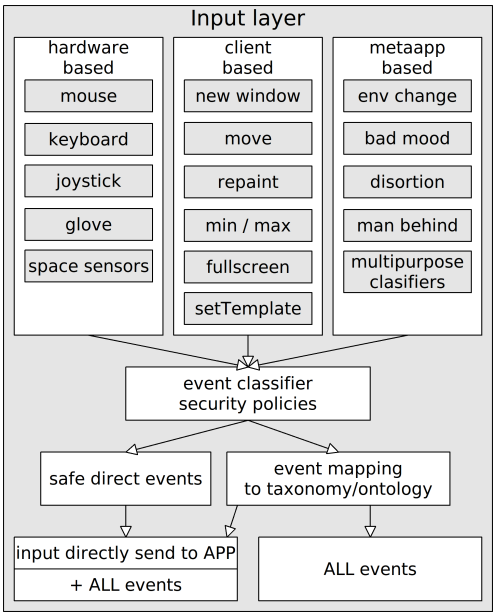

Figure 2: Input layer

cation logic, write own rules, generate events and change configuration of state machine. ALL configures actions for PL, passes commited events to client applications (i.e. "windows"). Aplication logic is designed to be fully modifiable and portable, as far as it operates with the abstract entities. At the time, only conservative processing engine is fully implemented. It processes events by executing code in written form (i.e. C++). Another available concept for processing unit is using visual modelling - in our case there is the modified *timed petri network* (m**TPN**).

TPN consists of *places* and *transitions* connected with *directed arcs* (each arc connects one place with one transition). Places acts as accumulators of tokens - requests. Transitions represents actions or parts of code. They can be enabled or disabled. Directed arcs represents set of conditions that have to be satisfied (accumulated sufficient amount of tokens) to enable transition. If conditions are not satisfied, transition is disabled. Enabled transition have to fire (execute code) in defined time conditions (i.e. it have to fire from t_{from} to t_{to} time units since transition has been enabled). There are various modifications of Petri Networks, introducing different enhancements like mentioned time, different types of tokens, spetial additional conditions. Petri Networks are also designed for verification of algorithms, detecting deadlocks, infinite loops etc., which are not considered in this stage.

Modeling with TPN is illustrative enough for people not experienced in programming but with mathematical aparate. Visual modeling utility for mTPN is part of future work.

ALL can be also extended with any other visual mod-

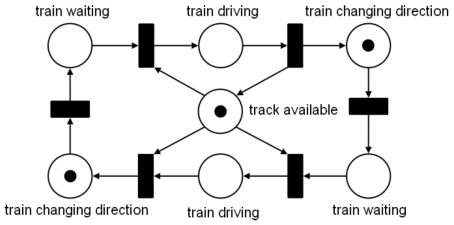

Figure 3: Petri network sample: places (rings), transitions (rectangles), tokens (dots) [7]

eling system[4], but we are not planning to implement any others.

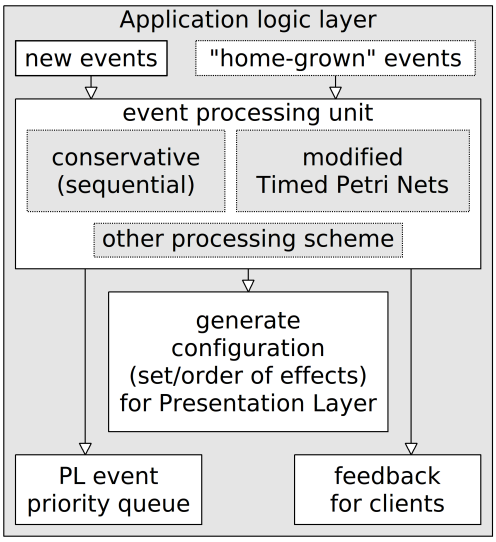

Figure 4: Application logic layer

The *client application layer* provides interface between the applications outside the framework (clients) and framework functionality. This should be done in two ways. The first is done by the wrapper, which does not need cooperation from client, wrapper just parasites on its interfaces and translate framework's communication to platform native codes and vice versa. The second is to handle communication native way, i.e. write "window" application using framework's API, what makes the client application framework dependant. Using framework's API may be useful for example if a user needs quickly gain existing feature. As example should be simple image viewer, which only initializes framework's communication object, sets dataType to "image/jpeg", and loads image into the attached binary blob property. The rest is work of PL and framework's interpreters/renderers.

The *presentation layer* is processing configuration prepared by the ALL. PL is the one responsible for visual interpretation binary blobs (by renderers), preparing scene,

[4]like timed automata etc.

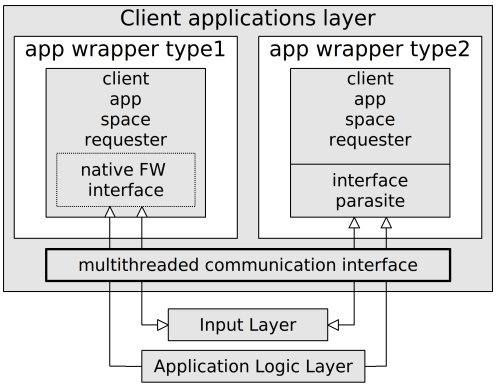

Figure 5: Client application layer

caching scenes, work with hardware acceleration, buffers, decorating output with special effects, finalization of rendered scene. PL is also place for user's own modules for special effects and visual data interpretation.

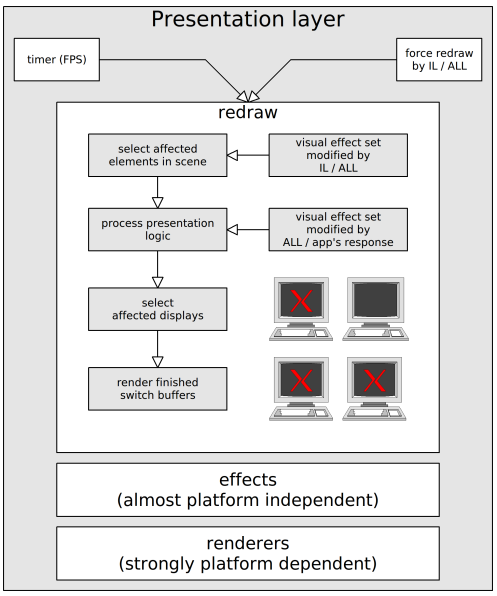

Figure 6: Presentation Layer

5 Possible applications

In fact framework separates output from application, what gives us flexibility for further manipulations. Several examples are mentioned here, but they are just examples, they should look simple but are there mentioned just for illustration. Not every of modules is implemented at present time, few are part of future work.

5.1 Image processing

As mentioned above, a visual output from application is just a binary blob, which could but does not have to be

passed for further graphical/data processing. For example, there is need to display histogram of a 24bpp[5] bitmap. Once a module implementing that functionality is implemented, programmer just need to load it and add to processchain for specific window and application developer does not need to implement it again (i.e. reuse of code).

5.2 Distributing output

Can be useful for viewing and recording video at same time - one machine can act as a high quality display, another as a recorder/encoder and last one as a streaming server for the internet (i.e. load balancing).

Another example is sharing applications by extending paradigm of a remote desktop. Machine A is running application and renders output, machines B and C are displaying same output window for two different users and are receiving inputs from them, both B and C are passing input signals back to A (i.e. collaboration).

5.3 Space

Most of recent WMs are still 2D and single machine oriented. Framework does not rely on any number of dimensions or machines. If there is need of 3D interpretation of a content, there is possibility to add property[6] of depth and Z coordinate of position to each window and write logic, which operates in 3D space. If there is need of union of vritual space of multiple machines, there is possibility join them into virtual domain.

5.4 Application classification

Each person has it's own organizational structure for objects by their characteristics. Dishes, food and things associated with eating are usually stored in kitchen. Also applications can be grouped to categories - work, social interaction, relax, etc. . This can also lead to organizing them in space, giving several groups of applications different priorities in interaction and machine time. Switching between applications (usually by pressing *Alt+Tab*) can also be extended with groups, like keys on different keyrings and user can quickly switch between "windows" in specific group and does not have to traverse unimportant ones.

Frequently popuping applications can also receive less priority by classifiers in ALL to surpress disturbing effect. Multiple behaviours of entire interaction systems can be introduced based on user's settings or for example user's mood (switched on requests or by receptors).

5.5 Kiosk mode

Or restricted (interaction) environment is often used for single purpose applications installed in a public or open space, where a machine should serve only one specific application and should be resistant again attempts to gain unprivileged control of a machine. Building restricted environment is not a simple task, which usually requires sanitizing of input systems. Customization via framework is simple task, because can be switched to mode when no other application can be displayed or receive input from user.

[5] color depth 24 bits per pixel

[6] dynamic properties vs. limitation of existing WMs

6 Results

We have developed the core and basic parts (i.e. systems wrappers) of framework. Actual port is working on 32 / 64bit Linux versions, port to Windows XP is in progress at the moment. In actual state of development, porting the framework to a specific platform is mainly work of porting wrappers to OS + platorm specific utilities to handle communication with drivers (including renderers, wrapper for CAL as shown on figure 7). At the moment, framework is not oriented to high performance.

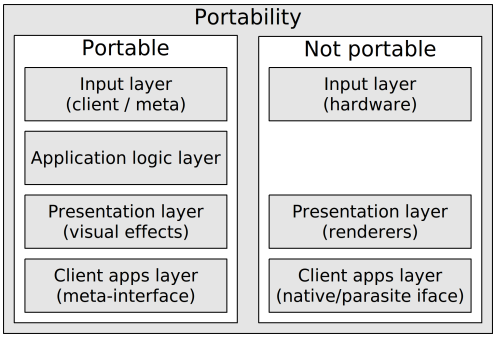

Figure 7: Portability

7 Future work

Future development can be divided to

- a visual modeling tool for mTPN as well as pre-fail verification for modelled mTPN structures
- visual configuration utilities
- porting to mobile devices
- network functionality, visual domain - a space of a rendered output shared between multiple workstations in domain
- improving performance
- improving security

8 Conclusion

Basic structures of concept were successfuly implemented, but there is still place for improvements. Our interest for future work focuses to mobile platforms, improving configuration interface and implementing mTPN visual interface to ease interaction for common people.

References

[1] Blackbox, February 2011.
http://blackboxwm.sourceforge.net/.

[2] Boost c++ libraries, feb 2011.
http://www.boost.org.

[3] The compiz-fusion project, February 2011.
http://www.compiz.org/.

[4] Fluxbox, February 1. 2011.
http://fluxbox.org/.

[5] Dušan Hamar. Spherexp, feb 2011.
http://www.spheresite.com/.

[6] IEEE. *IEEE Standard Interpretations for IEEE Std 1003.1-1990 and IEEE Std 2003.1-1992*. IEEE, 1994.

[7] Jan Krumsiek. Petri nets for network analysis, 2007.
http://www2.bio.ifi.lmu.de/lehre/SS2007/SEM_Advanced/Folien/petrinets_slides.pdf.

[8] Nicolas Roussel Olivier Chapuis. *Metisse is not a 3D desktop*. UIST '05 Proceedings of the 18th annual ACM symposium on User interface software and technology, ACM New York, NY, USA , 2005.
http://insitu.lri.fr/metisse/.