Computer Science Department
Year 2010

Summary Report

Design and Development of the BlackBox Tool Kit (BBTK)

Graphic Pipelines Editor

Ricardo A. Corredor,

Enterprise:
CREATIS-LRMN INSA (Direction)
Bâtiment Blaise Pascal
7, avenue Jean Capelle
69621 Villeurbanne - France

Instructors:
Eduardo E. Dávila Serrano, software engineer, info-dev team CREATIS
Maciej Orkisz, head of info-dev team CREATIS
Béatrice Rumpler, Computer Science Department INSA-Lyon

Abstract
This project is an additional part of the Creatools platform developed in CREATIS laboratory. It is conceived as an extension of the existing Black Box ToolKit (BBTK) framework that allows to design, to implement, to execute and to test new prototypes of different methods in the treatment of medical images. The actual script language *bbs* (Black Box Script) allows to connect different components (black boxes) describing the pipeline of these methods. The objective is to design and to implement an intuitive and user-friendly graphical editor in order to provide an easier way to illustrate BBTK pipelines, and finally to generate the *bbs* script files to be executed. With the editor it must be easy to package a set of boxes into a complex box that can be used later as any atomic black box. Additionally, the use of this visual environment will reduce the complexity and the development time of a treatment pipeline, as well as the quantity of mistakes made when the script is written.

Keywords
Visual programming environments, 2D visualization, software architecture, images processing, object oriented programming, human–computer interaction.

Résumé
Ce projet est un élément supplémentaire de la plateforme de développement Creatools du laboratoire CREATIS. Il est conçu comme une extension de l'actuel Black Box ToolKit (BBTK) qui permet de concevoir, tester et prototyper de nouvelles méthodes en traitement d'images médicales et de maillages. Le langage script *bbs* (Script Black Box) nous permet de connecter les différents composants (boîtes noires) en décrivant un pipeline de ses méthodes. L'objectif est la conception et réalisation d'un éditeur intuitif et simple afin de fournir un moyen plus simple pour illustrer pipelines BBTK, et finalement de générer le script fichiers *bbs* à exécuter. Avec l'éditeur il doit être facile à regrouper un ensemble de boîtes dans une boîte complexe qui peut être utilisé plus tard comme une boîte noire quelconque. En outre, l'utilisation de cet environnement visuel permettra de réduire la complexité et le temps de développement d'un pipeline de traitement, ainsi que la quantité d'erreurs commises lorsque le script est écrit.

Mots clefs
Environnements visuels de programmation, visualisation 2D, architecture logicielle, traitement d'image, programmation orientée objet, interaction homme machine.

Introduction

The laboratory Creatis (Center for Research in Image Acquisition and Processing for Healthcare) is a biomedical imaging research unit working in information and communication science and technology, engineering sciences and life sciences. It is divided into eight research groups emphasized in particular aspects of medical image processing. Additionally, the administrative and logistic department, and the informatics services department support many activities developed in the laboratory.

The department of informatics services (*info-team*) has two areas: networks and equipment support, and the scientific software development (*info-dev*) area. The principal *info-dev* mission is "*to put in common and maintain the software components developed in the lab, so as to minimize the programming burden of the researchers*"[1]. One important application developed to help researchers and developers is Creatools.

The Creatools software suite includes a set of open-source and cross-platform tools (libraries, applications, utilities…) for quick prototyping of medical image visualization and analysis applications. This suite uses various third party libraries (itk, vtk, wxWidgets, KWWidgets, boost...) and custom components in order to provide to the final users a set of different possibilities to create powerful image treatment applications. Moreover, Creatools is ruled by the CeCILL-B license for free software.

The Black Box Toolkit (BBTK) is one of the most important tools included in the suite. The following definition obtained from the PLUME-FEATHER project reference[2] summarizes its principal objectives. It is a flexible framework for the design, programming, testing and prototyping of applications. It provides the user with libraries of high-level components for: the construction of graphical interfaces, input/output (file management), display, interaction. These components (black boxes) can be heterogeneous. The black boxes can be assembled into pipelines, using a very intuitive script language (*bbs*), in order to realize either stand-alone applications or meta-widgets, reusable in other applications. A set of black boxes can be encapsulated in one complex box with the same characteristics of an atomic black box. Script edition and testing are made easy by a graphical environment (*bbStudio*). bbStudio automatically generates from the script a graphical representation of the pipeline, by use of the Graphviz facility.

However, the complexity of a particular pipeline in terms of number of boxes increases the lines of code (LOC) in the *bbs* descriptor file. The graphical representation is only a non-editable static PNG picture. Gradually the high quantity of text becomes a restriction to understand and edit applications. The advantages to manage prototypes disappear when the user does not have a direct control over the objects.

A visual representation of the boxes and their connections described by the *bbs* script language gives an overview of the pipeline. However, a direct manipulation of black boxes and its connections is desired, in order to provide user-friendly and intuitive interactions, reducing time to modify the diagram. Furthermore, the number of typing and syntax errors should be reduced if the script is automatically generated from the graphical representation. The use of graphical metaphors would also hide a programming and logic background that is not necessarily mastered by all the possible BBTK users, such as scientists, medical researchers, mathematicians …

The objective of this project is to design and to implement a usable, extensible, and highly modifiable graphical editor for the BBTK (BBTK GEditor), which supports the principal functions of the current scripting language. In addition, the editor has to be easily integrated to Creatools, using useful components already implemented.

In the next section will be discussed the principal system requirements to be developed according to the user needs. Next, a revision of some existing applications which provide interesting interaction techniques will be made and the option of reusing some components or libraries found will be discussed. Later, a solution proposal will be presented, beginning by a high-level component description and finishing with the description of the BBTK GEditor main classes. Afterwards, some important technical aspects and the state of the actual version will be described. Finally, the conclusions and future work section will make a revision of the objectives and the tasks to continue the development of BBTK GEditor.

[1] This information is taken from Creatis web site. For more information about the laboratory and its units go to: http://www.creatis.insa-lyon.fr/site/en. Accessed 10 June 2010

[2] Projet Plume. Creatools Project Reference. Online: http://www.projet-plume.org/en/relier/creatools. Accessed 10 June 2010

It is important to note that all the tables, diagrams, or important figures that could be difficult to understand due to the two-columns style will be placed in the web site of the project and a reference will be added before their analysis.

System requirements

In the development of every new application it is necessary to describe the principal needs of the users, in terms of the functionalities and the system. A good method to describe the functional requirements is to fill a Use Case UML diagram.

The Use Case Diagram is an UML definition which shows a summarized description of the system actors, and their functional requirements or use cases.

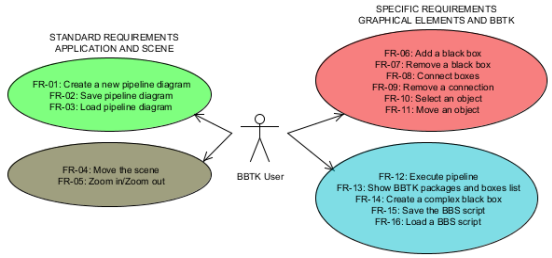

Figure 1. BBTK GEditor Use Case Diagram

The use cases diagram of BBTK GEditor (Fig. 1) presents four groups of functional requirements. The groups on the left bring together the basic requirements, or what it was called, standard requirements. They are functions that the application and the workspace must support as a visual editor compared to the other existing tools. These requirements are:

Application Requirements
FR-01: Create a new pipeline diagram
FR-02: Save pipeline diagram
FR-03: Load pipeline diagram

Scene Requirements
FR-04: Move the scene
FR-05: Zoom in/Zoom out

On the other hand, the two groups on the right join the actions over the objects designed in base to the BBTK definitions, and also the most specific requirements to support all the functionalities included in BBTK. The list of specific requirements is:

Objects Requirements
FR-06: Add a black box
FR-07: Remove a black box
FR-08: Connect boxes
FR-09: Remove a connection
FR-10: Select an object
FR-11: Move an object

BBTK Specific Requirements
FR-12: Execute pipeline
FR-13: Show BBTK packages and boxes list
FR-14: Create a complex black box
FR-15: Save the BBS script
FR-16: Load a BBS script

Additionally to the functional requirements description, it is necessary to present the restrictions that the system must solve. A standard form in software engineering is to identify the quality attributes [4] that principally affect the system architecture and design. To propose a solution for the BBTK GEditor it was selected the following collection.

Reusability: The editor can reuse code and components already made in Creatools libraries. The architecture must reflect the integration with other modules.

Extensibility: Including new functionalities in the editor has to be easy and low-cost. The solution must be scalable and low coupled.

Modifiability: Due to the constant rotation of developers in *info-dev*, changes must be easy to perform; without a big effort. A complete documentation is essential.

Portability: Creatools can run in different platforms. For that, it is important that BBTK GEditor works in all the platforms supported by Creatools, i.e. currently: Windows, Linux and Mac OS.

Usability: The user interface has to be simple, coherent and intuitive.

Context

This context review was looking for easy reusable and available implementations that could be modified to accomplish the objectives of BBTK GEditor. The principal restrictions were imposed by the non-functional requirements described in the last section. All these factors affect the decision of using an external component, because it makes dependence to a code conceived by other group. However, the revision was not only searching a general library with the basic functions to interact with black boxes, but also the mainly used ideas in terms of objects and GUI widgets distribution, interaction techniques, colors, and so on. In fact, a conclusion of this analysis will show the

advantages and disadvantages of visual environments, as well as the difference between BBTK GEditor and other similar applications.

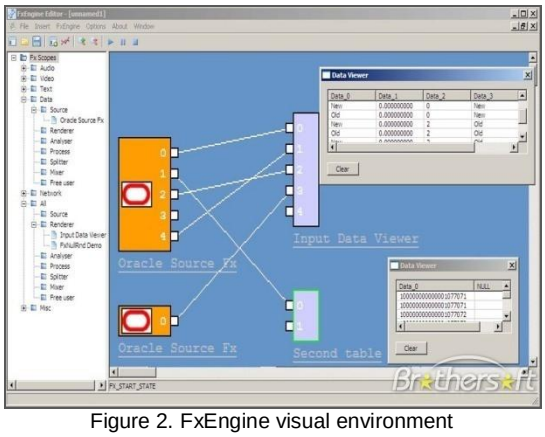

Figure 2. FxEngine visual environment

After searching possible visual programming environments, libraries or applications, a set with the most relevant characteristics was selected and evaluated according to the quality attributes already stated. Principally, the Extensibility (EXT), Modifiability (MOD), Portability (POR), and Usability (USA) could affect the final decision and could show the differences among them. The result of this study was published online[3] in a table that has the description, the analysis of Pros and Cons, and finally the quality attributes that are included in each implementation.

After comparing the alternatives, Open Inventor and FxEngine (Fig. 2) were the nearest frameworks to the requirements and specifications. However, after consulting the web sites and servers of both projects it was very difficult to get a trustful and stable version. They didn't have a complete documentation and the projects had not been updated for a long time.

One factor that affected the final decision was that *info-dev* needs fully control of its tools. This means that the introduction of external technologies was an important risk, and the specificity of the editor with BBTK and its integration, will require many changes to reach all the functional requirements.

Another significant reference as a visual programming environment is Mevislab. It is a development environment for medical image processing and visualization. It has a very powerful and consistent tool to design networks, but it has some constraints. For example, it is difficult to follow a very big diagram and it is not possible to modularize the diagrams. It is also commercial software and it was not possible to access the code.

System Architecture and Design

The BBTK GEditor Analysis Diagram[4] proposes a high-level extraction of the principal functional and non-functional requirements. It is important to note that this diagram does not reflect the implementation, but the principal concepts to develop a good detailed design and the relation between all the concepts. Additionally, the information flow can be inferred in order to have a functional overview of the program.

Regarding the bases of the detailed design, some design patterns were used in order to assemble functionalities with restrictions. Wikipedia propose a very concise definition of a design pattern which is "*a formal way of documenting a solution to a design problem in a particular field of expertise*" [3].

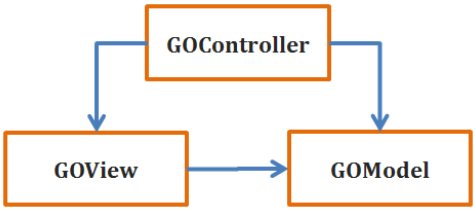

Figure 3. Graphical Objects Model-View-Controller

One pattern highly used in GUI development is the Model-View-Controller (MVC) pattern (Fig. 3) that makes a division between the GUI and the logic domain. In BBTK GEditor, this pattern is adapted in two levels: in the application and the BBTK graphical objects. Inside the application exists a division between control and view components, such as browsers, panels, buttons, menus, etc., and the BBTK graphical objects module. For the graphical objects, the same MVC design made for the Contours module of *creaMaracasVisu* library is reused. The model of each object keeps the state, the spatial position, and some individual characteristics. The view has the visual actors and the necessary information to paint them. The controller responds to user actions and updates both, the model and the view.

Additionally, to manage the actions in the scene it was used the Observer/Observable pattern (Fig. 4). This pattern "*defines a one-to-many dependency between objects so that*

[3] Context revision table: http://www.creatis.insa-lyon.fr/~corredor/ContentPages/Documentation.html#Context

[4] Analysis diagram: http://www.creatis.insa-lyon.fr/~corredor/ContentPages/Documentation.html#Analysis

when one object changes state, all its dependents are notified and updated automatically" [2]. When an object in the pipeline changes its state, the scene manager is notified to update the other elements.

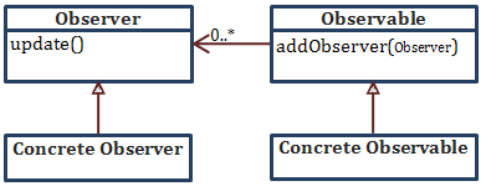

Figure 4. Observer design pattern

According to these design guidelines, it was possible to define a global architecture for the implementation. Firstly, in order to reuse the components developed with the third party libraries of Creatools, it was necessary to use them as well for BBTK GEditor. The architecture diagram (Fig. 5) presents the principal dependences among the components.

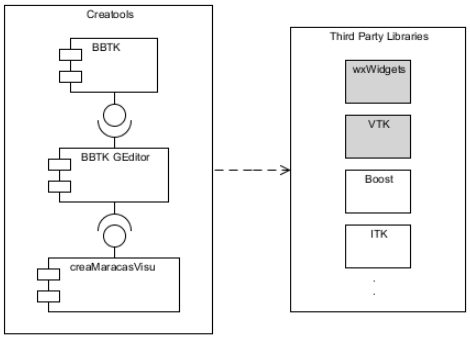

Figure 5. BBTK GEditor Component Diagram

The GUI will be implemented using wxWidgets objects and the virtual canvas will be managed by VTK [1]. Nevertheless, reusing some parts of *creaMaracasVisu* library will help to include the VTK objects in wxWidgets objects. This custom library has also a system to manage the interaction of the user with the virtual world, and the implemented MVC of some graphical objects, for example the connections between black boxes. Finally, BBTK GEditor has a total dependence with BBTK in aspects such as reusing some GUI objects to show the list of black boxes and its information, revising the possible connections between objects, and executing the pipelines translated into temporal *bbs* files.

Inside the principal BBTK GEditor is developed the first level of the MVC described before (Fig. 6). A division into three static libraries is made in order to conserve the low coupling and to delegate functions in specific modules. The *KernelBBTKGEditor* library joins together the logic definition of the graphical objects in the scene, i.e. the models in the MVC implementation for the objects. The *WxBBTKGEditor* library works as the view and part of the controller, having the objects which mainly depend of wxWidgets library. Lastly, the *VtkBBTKGEditor* library brings together the view and controllers of the graphical objects that uses the VTK external library.

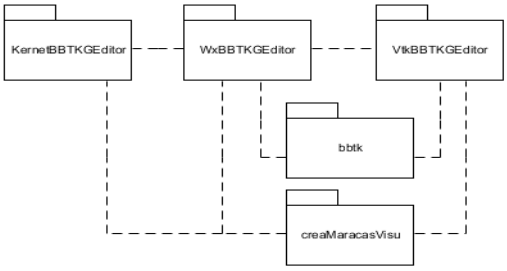

Figure 6. BBTK GEditor libraries composition

Each static library has a more detailed description presented in UML class diagrams[5], but an explanation of the main classes in each library is useful to understand them.

In the kernel diagram it is possible to see all the principal core concepts presented in the analysis diagram, but having a more clear idea about the implementation. All the graphical objects extend of a general *GObjectModel* which has the common characteristics among the elements, for example the position in the VTK virtual world. The dependence of *creaMaracasVisu* contour model to create the connection, and the emergence of a new type of objects, *GComplexPortType* are two important considerations. These new elements will be necessary when creating complex black boxes. The application will have a parameter to enable this option and to decide the inputs and outputs of the new box. They will be represented as a simple box in the scene with one port and a name, depending in the port type: complex input or complex output. After adding the complex ports to the scene it is possible to save the *bbs* file of a complex box to be reused in a pipeline.

The *WxBBTKGEditor* classes describe the frames, dialogs, menus that compose the GUI. The *wxVtkSceneManager* makes the connection between wxWidgets and VTK elements and controls the state of the objects in the scene. The creation and addition of each graphical object is made by this important class. A mechanism of drag-and-drop from the Package Browser of BBTK is used to insert an

[5] Libraries Class diagrams: http://www.creatis.insa-lyon.fr/~corredor/ContentPages/Documentation.html#Design

element in a specific position of the virtual world.

Finally, the *VtkBBTKGEditor* provides controllers and views of the graphical objects, implemented with external objects. Modifying the graphical representation of each object, its colors and size can be easily performed changing an individual class or a general constants file included in the implementation.

Implementation and Test

The implementation of BBTK GEditor was made using the object-oriented programming language, C++. To generate projects for different programming environments like Visual Studio, CodeBlocks, Eclipse, among others, the Cmake build system was used. Using this tool it is possible to compile code in different platforms.

The actual version was tested with Creatools 2.0.3 and creaThirdPartyLibraries 3.0.1, which includes the latest VTK and wxWidgets stable versions (these applications can be downloaded from the Creatools website). In this version (Fig. 7), almost all the functional requirements were implemented with a basic and simple interface. Only the requirement which produces a pipeline diagram from a *bbs* file was not implemented due to time, but the architecture supports its addition. The principal objective in the development process was implementing the functions in a way that some custom parameters could be easily changed according to the results of later user tests and validations. However, some requirements were found in the development and it was not possible to have a stable solution to these needs. For example, the complex black box creation and edition is performed, but the list of black boxes is not updated with the inclusion of the new complex box.

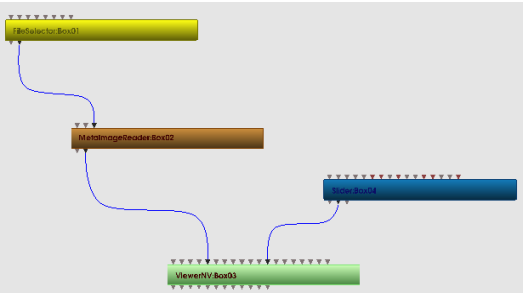

Figure 7. An example of a black boxes pipeline using BBTK GEditor

Regarding the test and validation phase, the software development team gave a feedback about the relevance of the requirements to be implemented. They also provided some important guidelines about the interactivity and visual aspects of the editor. Nevertheless, it is necessary to design a test plan to supply a complete version of this implementation that can be quickly and effortlessly downloaded, installed, and used by all the possible users of BBTK GEditor.

Conclusions

A first version of the graphical editor for BBTK library was designed and implemented. This implementation permits to design and execute black boxes pipelines The architecture presents a modifiable and extensible structure that follows some design patterns in order to resolve the principal non-functional requirements of the software.

In spite of the appearance in the process of new requirements that were not considered in the initial analysis, their implementation did not produce a high effort. These changes showed the level of flexibility and extensibility in the design, although it is indispensable to continue the development of BBTK GEditor in order to have a more stable version for the users.

Future Work

More development cycles will be needed to test all the requirements and to present a fully functional application to the users. However, a list of important changes, corrections, and additions has been made in order to help the future programmers to identify the principal problems with the actual version. This list will be published using MantisBT, a free bug tracking system. It is also essential to continue the code documentation and to create a friendly user manual that explains the functions of the editor.

References

[1] Will Schroeder, Ken Martin, and Bill Lorensen. "*The Visualization Toolkit*", Third Edition. Kitware Inc. 1997

[2] SourceMaking. "*Behavioral patterns. Observer design pattern*", Sourcemaking, teaching IT professionals, 2010. URL: http://sourcemaking.com/design_patterns/observer. [Online]. Accessed 9 June 2010

[3] Wikipedia contributors, "*Design pattern*". Wikipedia, The Free Encyclopedia, 2010. [Online]. Accessed 9 June 2010.

[4] Wikipedia contributors, "*List of system quality attributes*", Wikipedia, The Free Encyclopedia, 2010. [Online]. Accessed 12 June 2010.