journal homepage: www.intl.elsevierhealth.com/journals/cmpb

AIBench: A rapid application development framework for translational research in biomedicine

D. Glez-Peña[a,*], M. Reboiro-Jato[a], P. Maia[b], M. Rocha[b], F. Díaz[c], F. Fdez-Riverola[a]

[a] Dept. Informatics, University of Vigo, Campus Universitario As Lagoas s/n, 32004 Ourense, Spain
[b] Dept. Informatics, Universidade do Minho, Campus Gualtar, 4710-057 Braga, Portugal
[c] Computer Science Dept., University of Valladolid, Escuela Universitaria de Informática, Plaza Santa Eulalia 9-11, 40005 Segovia, Spain

ARTICLE INFO

Article history:
Received 13 June 2009
Received in revised form
11 November 2009
Accepted 10 December 2009

Keywords:
Scientific software development
Biomedical informatics
Open software
Reusable component model
Application framework

ABSTRACT

Applied research in both biomedical discovery and translational medicine today often requires the rapid development of fully featured applications containing both advanced and specific functionalities, for real use in practice. In this context, new tools are demanded that allow for efficient generation, deployment and reutilization of such biomedical applications as well as their associated functionalities. In this context this paper presents AIBench, an open-source Java desktop application framework for scientific software development with the goal of providing support to both fundamental and applied research in the domain of translational biomedicine. AIBench incorporates a powerful plug-in engine, a flexible scripting platform and takes advantage of Java annotations, reflection and various design principles in order to make it easy to use, lightweight and non-intrusive. By following a basic input–processing–output life cycle, it is possible to fully develop multiplatform applications using only three types of concepts: *operations*, *data-types* and *views*. The framework automatically provides functionalities that are present in a typical scientific application including user parameter definition, logging facilities, multi-threading execution, experiment repeatability and user interface workflow management, among others. The proposed framework architecture defines a reusable component model which also allows assembling new applications by the reuse of libraries from past projects or third-party software.

1. Introduction

Over the last few years, numerous authors have discussed potential and critical needs regarding the application of computing to biomedicine, emphasizing the necessity of exploiting the synergies between both disciplines to address current limitations and to promote relevant developments in both biomedical discovery and translational medicine [1]. In this context, an interesting summary of computing opportunities and challenges motivated by biomedical research and healthcare needs was summarized in the CRA-NIH 2006 Computing Research Challenges in Biomedicine Workshop Recommendations.[1] This report revealed a significant need to support both the development of new software tools and to provide the necessary support for software infrastructure and software engineering for biomedical researchers and healthcare professionals.

In response to these challenges, the clinical and translational research informatics domain is rapidly evolving from sparse and unrelated initiatives to two major well-established areas: (i) clinical research informatics (CRI), dedicated to

* *Corresponding author*. Tel.: +34 988 387015; fax: +34 988 387001.
E-mail address: dgpena@uvigo.es (D. Glez-Peña).
URL: http://sing.ei.uvigo.es/ (D. Glez-Peña).

[1] http://www.bisti.nih.gov/docs/CRA-NIH-Workshop-Recommendations-Final.pdf.

0169-2607/$ – see front matter © 2009 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.cmpb.2009.12.003

the development, use and evaluation of standards, models, processes and systems to improve the design, conduct and dissemination of clinical research [2], and (ii) translational research informatics (TRI), more concerned with the application of informatics theory and methods to translational research [3]. Although both areas overlap considerably, the former is more focused on developing practical applications for computer-aided medicine while the main goal of the latter is to provide fundamental support to translational research.

However, the rapid development of successful feature-rich applications containing advanced functionalities in the field of biomedical and clinical research still remains a major demand for smaller institutions lacking both human and financial resources. The situation worsens if we consider the software development effort required to deliver highly specialized applications usually demanding sophisticated user interfaces. Moreover, developing applications in an interdisciplinary and applied research context also presents a large number of particular requisites ranging from computational requirements to usability. Specific issues include (i) sharing of heterogeneous data, (ii) integrating third-party or previously developed algorithms, (iii) cross-platform compatibility, (iv) ability to repeat workflows while changing a few parameters or input data, (v) extensive use of logging messages to monitor the progress of long processes, (vi) establishing values for a high and variable number of parameters before running experiments and (vii) taking the maximum advantage of multithreading capabilities in highly demanding tasks, among others.

In the global context of computer science and software development, a typical approach to cope with these kinds of problems is to make use of an application framework, which can be seen as a semi-finished application and a reusable architecture design [4]. Therefore, in recent years frameworks have become very popular, especially in web application development where Ruby on Rails,[2] Symfony,[3] Spring,[4] JSF[5] or Apache Struts[6] are examples of some of the most successful alternatives for deploying scientific applications as web services [5]. Nowadays, there are general frameworks for almost any kind of software including object-oriented desktop applications (MFC,[7] Netbeans,[8] Eclipse[9]), software testing (JUnit[10]), compiler generation (Bison,[11] Javacc[12]), multimedia (WindowsMedia,[13] ffmpeg,[14] GStreamer[15]), virtual reality (Vega Prime,[16] VR Juggler,[17] CAVELib[18]) and middleware (CORBA,[19] EJB[20]). Nevertheless, the actual benefits of applying such general scalable software environments to the development of specific biomedical applications are clearly insufficient, mainly due to the special requirements of computer-assisted biomedical and clinical research areas.

With the aim of giving a more adequate support to the particular needs of several areas belonging to the theoretical and clinical biomedicine domain, different focused frameworks were also successfully developed in the C++ language during the last few years. This endeavour was particularly evident in the area of medical imaging. In this context, the Medical Imaging Interaction Toolkit (MITK) implements a free open-source software system for the development of interactive medical image processing software [6]. MITK combines the Insight Toolkit (ITK[21]) and the Visualization Toolkit (VTK[22]) for currently offering functionalities for data visualization, processing and interaction. In the same line, MeVisLab [7], a development environment for medical image processing and visualization, as well as IGstk [8], a high-level component-based framework providing common functionality for image-guided surgery applications, make use of ITK and VTK toolkits (among others) for giving support to specific biomedical applications. Another example of a successful software platform in this area is JULIUS [9], an extensible framework for medical data processing and visualization. From a different perspective, and more focused in the development of research based image-guided navigation software, is the SIGN framework [10], which provides the developer with a platform specifically designed for image-guided therapy and aids the rapid development of new applications.

From a broader perspective, but also related with the goal of giving support to the development of software techniques and processes used to manage images of the human body for clinical purposes, there are two successful application frameworks: the Multimod Application Framework (OpenMAF) [11] and MARVIN [12]. OpenMAF implements an open-source framework for rapid development of multimodal applications coded in the C++ language. OpenMAF supports several types of biomedical data where its interactive visualization approach helps the user to interpret complex datasets. In addition, the framework supports different input–output hardware devices being based on a collection of portable libraries. A different approach, but also coded in the C++ language, is the MARVIN project. MARVIN implements a medical research application framework where different modules can be plugged together in order to provide the functionality required for a specific scenario. In the MARVIN framework, application modules work on a common patient database that is used to store and organize medical data. As in the case of OpenMAF, MARVIN

[2] http://rubyonrails.org/.
[3] http://www.symfony-project.org/.
[4] http://www.springframework.org/.
[5] http://java.sun.com/javaee/javaserverfaces/.
[6] http://struts.apache.org/.
[7] http://msdn.microsoft.com/en-us/library/d06h2x6e(VS.80).aspx.
[8] http://platform.netbeans.org/.
[9] http://wiki.eclipse.org/.
[10] http://junit.org/.
[11] http://www.gnu.org/software/bison.
[12] https://javacc.2dev.java.net/.
[13] http://www.microsoft.com/windowsmedia/.
[14] http://ffmpeg.mplayerhq.hu/.
[15] http://gstreamer.freedesktop.org/.
[16] http://www.multigen.com/products/runtime/vega_prime/.
[17] http://www.vrjuggler.org/.
[18] http://www.mechdyne.com/integratedSolutions/software/products/CAVELib/CAVELib.htm.
[19] http://www.corba.org/.
[20] http://java.sun.com/products/ejb/.
[21] http://www.itk.org/.
[22] http://www.vtk.org/.

supports many standard file formats as well as interfaces to different tracking hardware.

Taking into consideration the current state of the art, we conclude that it is characterized by the availability of both (i) very general software architectures for large-scale developments (including object-oriented desktop applications, software testing, compiler generation, multimedia, virtual reality and middleware) and (ii) more specific application frameworks offering biomedical-related libraries and specific hardware handlers coded in the C++ language (some of them including ITK and VTK toolkits for image processing and visualization and others supporting standard file formats and hardware drivers). As a consequence, there is no general and adaptable framework able to directly cope with the specific issues of the broad and particular nature of developing software for research applications in a translational biomedical domain. In particular, existing frameworks do not give central support to dynamic graphical user interface (GUI) generation, customization of default behaviour and application aspect, design of a clear application workflow, automatic script construction for supporting workflow repeatability, update service for automatically deploying software upgrades and automatic generation of technical documentation. In parallel with this situation, Java has found increased adoption in the scientific community due to the huge amount of freely available APIs and open-source scientific developments, regardless of its other native benefits such as language inter-operability, cross-platform nature, built-in support for multi-threading, networking, etc.

In this context, we present AIBench (*Artificial Intelligent workBench*), an open-source Java desktop application framework, specifically intended to improve both quality and productivity in the development of specialized applications for computer-assisted biomedical and clinical research. The main objective is to cover the gap between general desktop application frameworks and the specific requirements of scientific software development, by allowing the rapid production of high quality application prototypes with minimum effort into problem-unrelated functionalities and with the maximum reusability level of previously coded algorithms. We believe that whenever the core algorithms for solving a problem become available, it should be almost mandatory to deliver an acceptable application prototype without affecting the code of the domain specific routines. This prototype may subsequently evolve to a real final application by the use of more advanced capabilities of our rapid application development (RAD) framework.

The aim of this article is to provide an in-depth description about the AIBench framework architecture, present the development infrastructure and demonstrate its application to different unrelated example problems belonging to the broad scope of biomedical research.

2. The AIBench application framework

The AIBench platform was particularly conceived to facilitate the development of a wide range of research applications based on general input–processing–output cycles where the framework acts as the glue between each executed task. In order to provide the basis for supporting rapid application development, our framework manages the three key concepts that are present in every AIBench application: *operations*, *datatypes* and *views*. The developer only needs to concentrate on how to divide and structure the problem-specific code into objects of these three entities. The framework will carry out the rest of the work to generate a completely runnable final application. These tasks include:

- Producing a GUI under which the user is allowed to select and execute the implemented functionality.
- Automatically retrieving the user parameters of a given operation whenever it is needed. The parameters could be both primitive values (numbers, strings, booleans) or any complex data-type previously created by an operation.
- Running operations, gathering the results and keeping them available for further use.
- Displaying the results through custom (or default) views.
- Keeping track of all executed operations together with the information needed to repeat the same (or modified) workflow in the future.

2.1. *Framework architecture*

In order to accomplish the final goal of covering the gap between final-user applications and internal algorithms managed by research developers, AIBench incorporates two advanced internal modules: the *Clipboard* and the *History*. The Clipboard is a complex data structure repository which contains the outputs of the executed operations classified by their type (class). This structure allows the final user to examine what was produced during the current session and the possibility of forwarding these objects to subsequent operations. The History keeps track of what operations were executed and which objects were used as inputs. This structure allows the framework to entirely reconstruct the current session in order to re-execute it in the future. Fig. 1 shows a screenshot of an example AIBench application. Although applications developed with AIBench do not necessarily have a GUI, it is the most common layout.

By default, the main window of any AIBench application contains five zones (see Fig. 1). All the implemented operations are located in the menu bar of the application. A *Clipboard tree* displays the AIBench Clipboard contents, that is, all the objects generated by the executed operations. A *History tree* shows the AIBench History, that is, all the operations executed together with their inputs and generated outputs. The central panel of the application is used to display the contents of the objects using the AIBench views (default or custom). Finally, the bottom area is used to arrange available tools or add-ins. One tool included by default with the framework is the *Shell*, which can be used to run previously generated scripts in order to automate the execution of preconfigured operations.

From an architectonical perspective, AIBench is structured in several layers, as shown in Fig. 2. Our AIBench framework runs over a plug-in engine able to define a straightforward reusable component model where both, the framework native components and the application-specific functionalities, are divided and packaged into plug-ins. AIBench plug-ins are isolated by default, increasing the modularity and ensuring

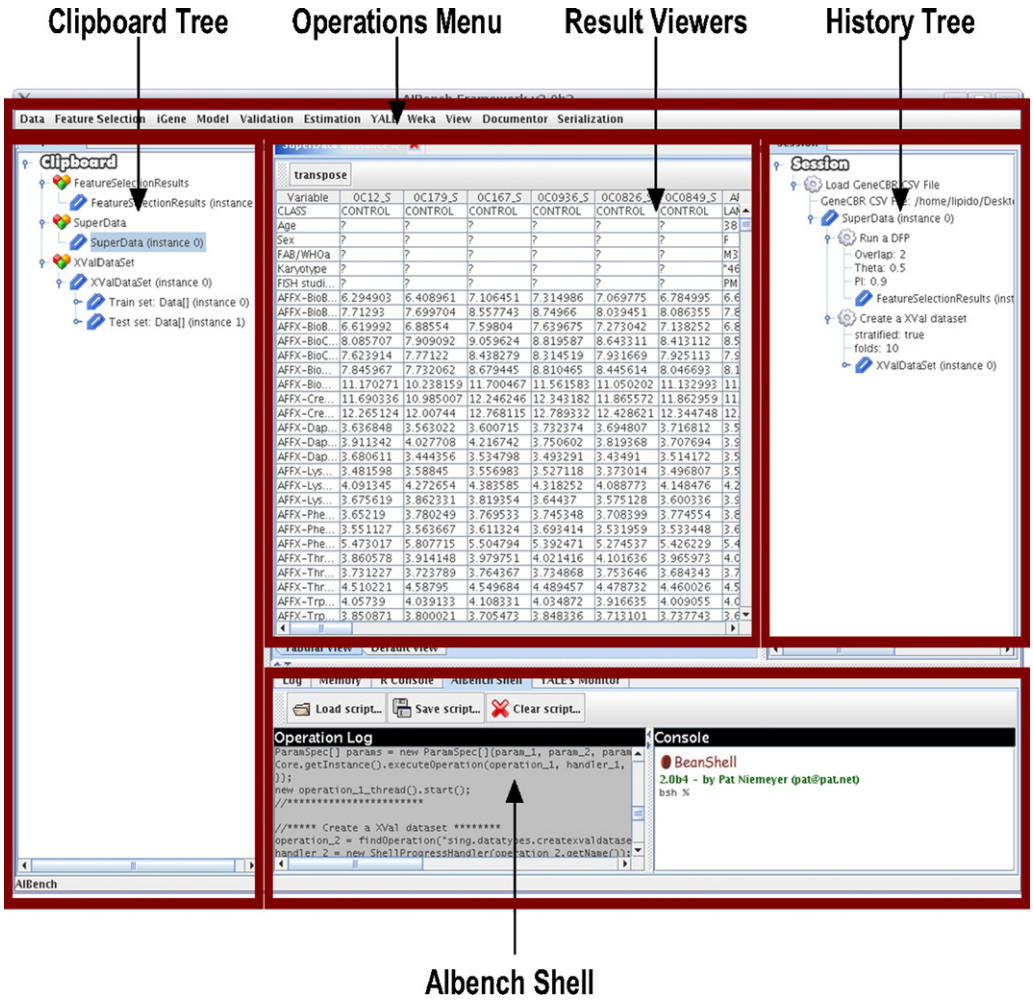

Fig. 1 – Visual appearance of an AIBench application with default layout showing five active areas. This main window layout can be adapted by defining which components are present in the main window and specifying their relative position.

that accidental coupling is not introduced, but they can also interact by establishing dependencies or extension points. A *dependency* between plug-ins allows one plug-in to require other plug-ins to be present at runtime and to be allowed to access their classes and/or resources. An *extension point* declares a place where some plug-in can be extended by another plug-in (*extension*), usually providing a specific interface implementation.

The Core layer contains two native plug-ins: the *Core* and the *Workbench*. The AIBench Core detects and registers the application-specific operations, executes them upon request, keeps the results in the Clipboard structure and stores the session workflow in the History. The graphical user interface aspects are implemented in the Workbench plug-in, which creates the main application window, composes a menu bar with all the implemented operations, generates input dialogs when some operation is requested for execution, instantiates the registered results viewers, etc. All additional services bundled with AIBench belong to the Services layer and are also implemented via independent plug-ins that can be easily removed to meet application-specific needs. The Core and Services layers are maintained by the *AIBench team* and constitute all the code built-in and distributed with the framework, being the starting point of every development.

The application layer is placed on the top of the architecture and contains the application-specific code (operations, data-types and views) provided by applications developers (AIBench users). In this sense, when an applications developer starts using the framework, there are no operations, data-types or views available, because these components are problem-specific items. However, operations, data-types and views can (and should) be shared among applications related to the same area, especially when they are developed inside the same team. These higher level components, along with other third-party libraries, are also packaged in one or more plug-ins. Finally, from the most abstract point of view, an AIBench application can be seen as a collection of operations, data-types and views, reusable in more than one final application.

For the implementation of AIBench, some open-source third-party libraries were used including the Platonos Plugin Engine, BeanShell, Apache Log4J and Apache Ant. Table 1 shows a brief description of these libraries and their role in AIBench.

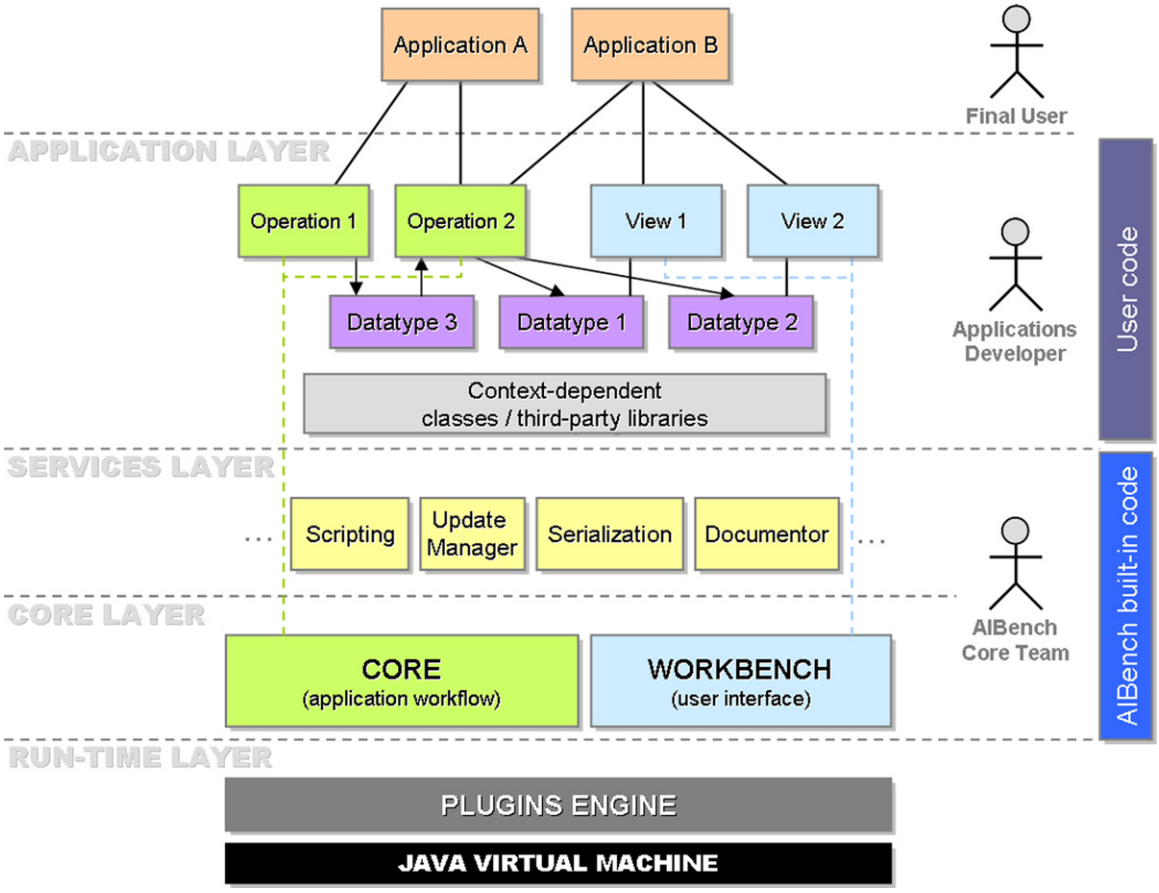

Fig. 2 – Four-layer architecture of AIBench. The first three layers include the Runtime, the Core and the Services layer, which are currently implemented in the framework. The application layer is context-dependent and contains the custom operations, data-types and views as well as third-party libraries needed to obtain the final functionality.

Table 1 – Third-party open-source projects supporting AIBench.

Third-party library	Description	Role
Java SE 5.0+ http://java.sun.com/j2se/1.5.0	The Java Runtime Environment (JRE).	AIBench is fully developed in Java language and needs JRE 5.0 or later due to its strong dependency on Java annotations.
Platonos Plugin Engine http://platonos.sourceforge.net	A plug-in engine for Java. It allows applications to be modularized via packages containing code, resources and a plug-in descriptor where relationships between plug-ins can be established.	The main foundation of the AIBench runtime. The *Core*, the *Workbench*, the built-in services as well as the final application functionalities are all plug-ins, running on top of the Platonos Plugin Engine.
BeanShell http://www.beanshell.org	A scripting language for the Java Virtual Machine. It allows an application to execute scripts without the need of compiling them.	The *Scripting* service plug-in was implemented using BeanShell which monitors the user workflow and generates a script that can reproduce all steps in a future session without user interaction.
Apache Log4J http://logging.apache.org/log4j	Library to manage log messages inside Java applications.	AIBench includes this library to output its log messages. In addition, the *Workbench* plug-in contains a logging panel, placed by default in the bottom of the main window. All logging messages coming from the plug-ins are displayed in this area. The log detail level can be tuned in a standard Log4J configuration file.
Apache Ant http://ant.apache.org	Utility to compile and package Java applications.	The AIBench SDK includes an Ant script file intended for developers using this tool instead of Eclipse or any other IDE to build their applications.

2.2. AIBench programming concepts: operations, data-types and views

As previously stated, every AIBench application is divided into three kinds of components: *operations*, implementing the algorithms and data processing routines, *data-types*, storing relevant problem-related information and *views*, rendering data-types obtained from executed operations.

AIBench *operations* define high-level problem-oriented processes. Each operation is implemented through only one Java class (which can delegate its internal behaviour to other classes). Generally speaking, one operation is a unit of logic with a well-defined input and output specified via a set of *ports*. A port is a point where some data can be defined as an input to the operation or can be defined as an output. A method of the operation class is associated with each port. There are three types of ports: IN (for input data, where the associated method must have one parameter of the type of the incoming data), OUT (for output data, where the associated method must not have any input parameter and its return type must be of the type of the output data) and IN–OUT (for both input and output data, where the method must have one parameter for the input data and a return type for the output data). Every time an operation is executed, one instance of its class is allocated in memory and all the methods associated to the ports are invoked in a predefined order, with the parameters already retrieved from the user. The framework is responsible for providing the input values, in the case of IN (or IN–OUT) ports, and to gather the output values, in the case of OUT (or IN–OUT) ports. To specify which methods of a given class are ports, how they are ordered and some other specific details, the class should be annotated with a set of predefined annotations.

AIBench *data-types* are intended to support problem-specific data structures (in-memory representation of some stored data, models, results, etc.). Although they are as important as operations, AIBench data-types are very simple to implement. Any Java class can be a data-type without additional code, only the fact of being the input or output type of some operation is required to be considered as an AIBench data-type. However, to attain more extensive control over data-types, the programmer can create *explicit* data-types. These data-types are also classes, but with additional information about their internal structure in order to enable AIBench to use them for displaying their parts to the final user in the GUI, or be selected as input in different operations.

Related to AIBench data-types and operations, the framework also defines the concept of a *transformer*. These abstractions are very useful when an AIBench application invokes other third-party components (a typical task is the need to adapt some data structures). A transformer is a method of some class which takes an object of one type as input and returns an object of another type. If such a method is declared, AIBench can perform automatic data-type conversions.

AIBench *Views* are intended to visualize the results of the executed operations in a friendly way. In this sense, one view is associated with a given data-type and is implemented through a Java class ensuring two requisites: (i) it must extend the standard class *JComponent* and (ii) it must have a constructor with only one parameter of the same type of the class it visualizes. Every time the framework needs to render a data-type, AIBench automatically generates an instance of the view class and passes the data through its constructor. Once the component is rendered, it is displayed to the user. Views can be as complex as needed in order to add extra interactivity in final applications. AIBench views are not mandatory: a default view is provided based on the standard *toString* method and also including a bean property inspector.

2.3. Main features

Given the singularity of developing software in a scientific context, AIBench provides a set of key features that are summarized and briefly described in this section. These are organized in (i) native functionalities, (ii) additional services and (iii) design principles.

AIBench's native functionalities provide the programmer with a set of core capabilities, which are summarized in the following items:

- *Dynamic GUI generation*. This is one of the most productive features, since AIBench implements not only a graphical user interface skeleton providing a basic workspace, but also dynamically generates complex input dialogs for every operation (see Fig. 3), giving a valuable aid in a time-consuming task present in every desktop application development cycle.
- *Application workflow*. Once all components are implemented, AIBench gives a classic full-useable application, which allows the user to execute operations, define the input parameters when needed, analyze the results in the main window and keep all the generated information in the Clipboard in order to be forwarded to subsequent operations.
- *Customization*. Although AIBench provides a default behaviour and application aspect, it can be deeply configured to meet the requisites of final applications. Such customizable aspects include main window layout (component presence and placement), appropriate icons, toolbar visibility for data and operations, splash screen, custom input dialogs, application help files, etc.

In addition, the framework is also made available including the following service plug-ins, which are specially useful in scientific applications:

- *Automatic script construction service*. In order to support workflow repeatability, AIBench provides a scripting plug-in which monitors the user workflow and generates a script that can reproduce all steps in a future session without the user interaction. This is a very valuable feature for scientific applications allowing the user to transparently create *execution macros* that can be easily modified and applied to different input data. The generated scripts are implemented in BeanShell and can also be modified by any text editor.
- *Update Manager service*. Since developed applications are structured and distributed in one or more plug-ins, this service gives the capability of managing a customizable remote plug-in repository to any AIBench application, where the final user can download or update the components needed.

```
@Operation(description = "Compare between ontologies,
genes and annotations")
public class AnnotateGenList {

    ...

    @Port(direction = Direction.INPUT,
            name = "Ontology data")
    public void setOntologyData(OntologyData od) {
            ...
    }

    @Port(direction = Direction.INPUT, name = "Ensembl Genes")
    public void setBioGene(BioGen[] bg) {
            ...
    }

    @Port(direction = Direction.INPUT, defaultValue = "true",
            name = "Cellular component")
    public void setOntologyCC(boolean cellular_component) {
            ...
    }

    @Port(direction = Direction.INPUT, defaultValue = "true",
            name = "Biological process")
    public void setOntologyBP(boolean biological_process) {
            ...
    }

    @Port(direction = Direction.INPUT, defaultValue = "true",
            name = "Molecular function")
    public void setOntologyMF(boolean molecular_function) {
            ...
    }

    @Port(direction = Direction.INPUT,
            name = "Operation")
    public void setOperationType(operationsTyupe ot) {
            ...
    }

    @Port(direction = Direction.OUTPUT,
            name = "Operation result")
    public AnnotatedGenResult getAnnotatedGenes() {
            ...
    }
}
```

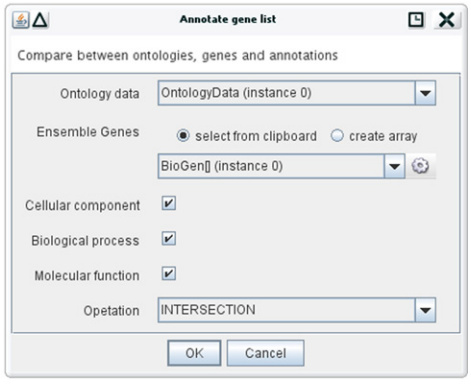

Fig. 3 – Dynamically generated input dialog for a given operation. The source code of the operation can be seen on the left side and the generated dialog with all input ports rendered with suitable components appears on the right-side.

This functionality dramatically eases the software maintenance task by allowing final users to download updates and bug fixes directly from their desktop.

- *Serialization service*. In order to facilitate saving and loading of final application data structures, this service manages all data-types which implement the standard java interface *Serializable* by automatically providing operations to support these functionalities from hard disk.
- *Documentor service*. During the development cycle of any application, technical documentation about the implemented components should be also carried out. The Java language natively provides this functionality via the *javadoc* utility, but not at the AIBench abstraction level. By taking into account the separation in operations, views and data-types, the *Documentor* service generates a full HTML technical report with a detailed description of all components currently coded inside the AIBench application. Fig. 4 shows an example of the technical documentation automatically generated.

Finally, AIBench follows several design principles (in order to ensure more development productivity), taking ideas from our previous Java developments such as geneCBR [13], from which we adopted the input–process–output application model, and some of the most successful web frameworks such as Struts, Spring, Ruby on Rails and others. These design principles include:

- *MVC design*. The framework is based on three main concepts (data-types, views and operations) following the well-known Model-View-Controller design pattern, encouraging application developers to decouple visualization and data processing code in every AIBench-based application. AIBench operations (*Controller*) are automatically triggered when the user requests them and define the code of the algorithms along with their I/O interface. These operations take as input, and produce as output, instances of AIBench data-types (*Model*), which can be any kind of Java class, carrying the problem's specific data. Finally, data-types are presented to the user through the AIBench views (*View*).
- *Problem independence*. Although AIBench is specially intended to support scientific and biomedical software workflows, the framework internal architecture does not contain any concept related to any specific discipline like Physics, Maths, Data Mining, Vision, etc. providing developers with a general and flexible framework.
- *Non-intrusive*. The application-specific implementation (algorithms, data structures and viewers) should contain the minimum amount of framework-related code. This allows the developer to keep his own application core

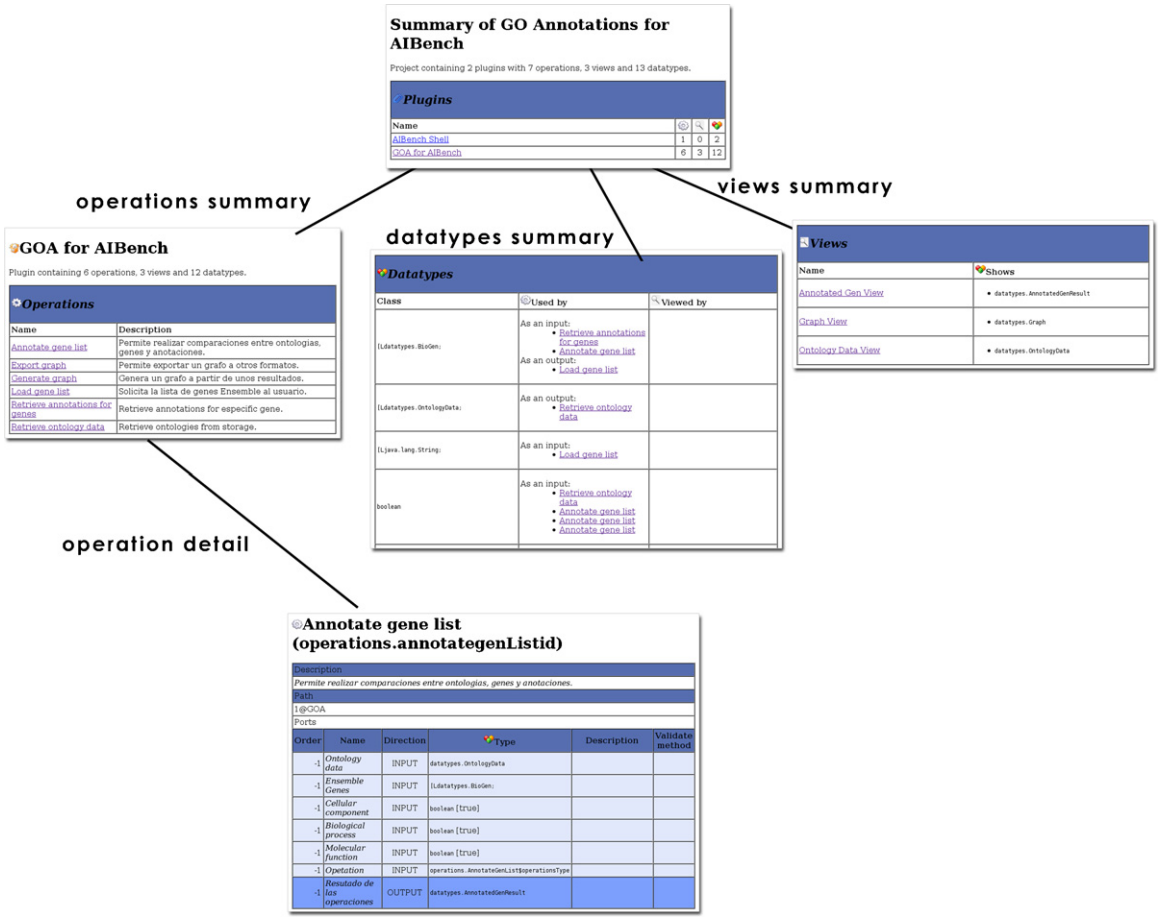

Fig. 4 – HTML technical documentation generated by the Documentor plug-in for an example AIBench-based application (described in Section 4.3). The screenshot shows an application summary containing two plug-ins, a list with several implemented operations, data-types and views together with a detailed description of an operation.

functionality clean and subsequently, to easily reuse it in other programming environments or applications.

- *Smart defaults*. In order to allow the developer to deploy applications as quickly as possible, the framework requires only minimum configuration to execute implemented techniques. Whenever possible, AIBench makes a default decision in the absence of a custom value for a given parameter using a preconfigured setup.

3. Application deployment process

The proposed AIBench development infrastructure consists of three major components: (i) a framework development kit (SDK), (ii) a version control system and (iii) a community platform.

The AIBench SDK is the starting point of every application, since the source code, the resources and the configuration files are added and/or modified in its file folder structure until the application is complete. Fig. 5 shows the file structure of the SDK and describes the purpose of the most important configuration files and folders.

Developing an application with AIBench requires operations, data-types and views (with the essential classes and resources) to be bundled into a plug-in that can consist of a folder or a .jar file stored in the *plugin_src* directory. A special XML file, named *plugin.xml*, must be included (see Fig. 5). This file contains essential information and customization parameters like operations, views, transformers, custom input dialogs or icons distributed within the plug-in, dependencies on other plug-ins, etc.

There must be at least one plug-in developed by the programmer in every AIBench application, but it is also possible to split the application into more than one plug-in and, especially, reuse other plug-ins by establishing dependencies between them. In terms of plug-ins, the operations extend the Core plug-in while the views extend the Workbench plug-in.

The AIBench SDK is distributed by periodically making available stable releases and also by nightly builds automatically generated with Apache Ant. The framework is ready to work with Eclipse[23] IDE and detailed information explaining how to start programming with AIBench framework is available through the project website.[24] In addition to using Eclipse,

[23] http://www.eclipse.org/.
[24] http://www.aibench.org/.

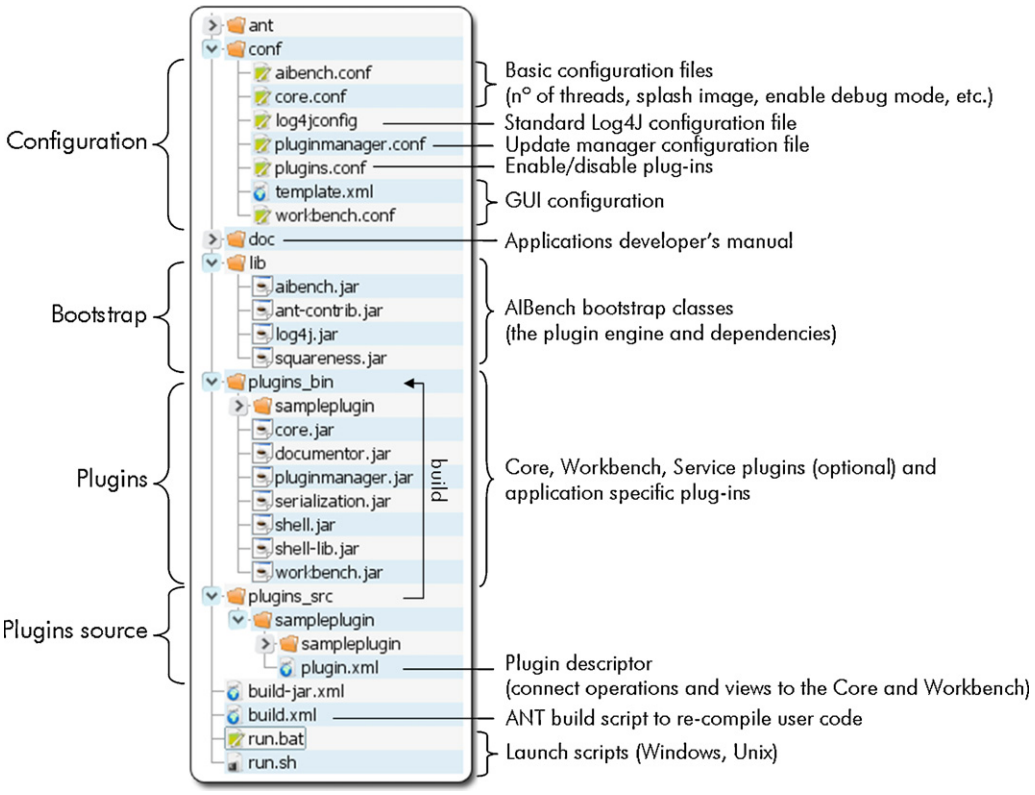

Fig. 5 – AIBench SDK build environment directory structure. There are five main directories containing both framework base files and developer oriented documentation.

final applications can be built and deployed in a script-based environment by using Apache Ant.

The AIBench project is maintained and distributed using the concurrent CVS version control system.[25] The community platform is based on Joomla[26] and provides wiki support through MediaWiki,[27] forum assistance by phpBB,[28] bug tracking facilities using Bugzilla[29] and an interface to the version control system. Using such a highly integrated development environment, it is straightforward for the AIBench Core Team to add more built-in plug-ins and services to the common code base, thereby making it directly available to other researchers. In addition, plug-ins developed by AIBench users are also welcomed, especially reusable data-types and views related to any biomedical field.

4. Examples of AIBench applications

This section presents several successful applications that have been implemented using AIBench in the field of metabolic engineering, biomedical text mining and functional genomics. These examples show real final applications, all providing full featured user interfaces that were rapidly developed by exploiting different framework capabilities.

4.1. *In silico metabolic engineering with OptFlux*

The Metabolic Engineering (ME) field is devoted to the design of microorganisms with enhanced capabilities, regarding the production of a target compound or any other relevant industrial goal [14]. The challenge is to design strains by reaching the ideal set of genetic modifications to apply to the wild type in order to optimize a given objective function.

In these tasks, distinct strategies are employed to make use of available models of metabolism together with mathematical tools to identify targets for genetic engineering. However, the application of such optimization algorithms and even the use of metabolic models for phenotype simulation is currently limited to the developers of the techniques or experienced programmers, since a computational open-source platform that provides a user standard interface is not available.

The OptFlux application (available at http://www.optflux.org) aims to become a reference for the community, including a number of tools to support *in silico* ME. The user can load a genome-scale model (for instance using the standard format SBML) that will serve as a basis to simulate the wild type and mutant strains (i.e., with a set of selected gene deletions). The simulation of these strains will be conducted using a number of approaches (e.g. Flux-Balance Analysis, Minimization of Metabolic Adjustment or Regula-

[25] http://www.nongnu.org/cvs/.
[26] http://www.joomla.org/.
[27] http://www.mediawiki.org/.
[28] http://www.phpbb.com/.
[29] http://sing.ei.uvigo.es/cgi-bin/bugzilla/.

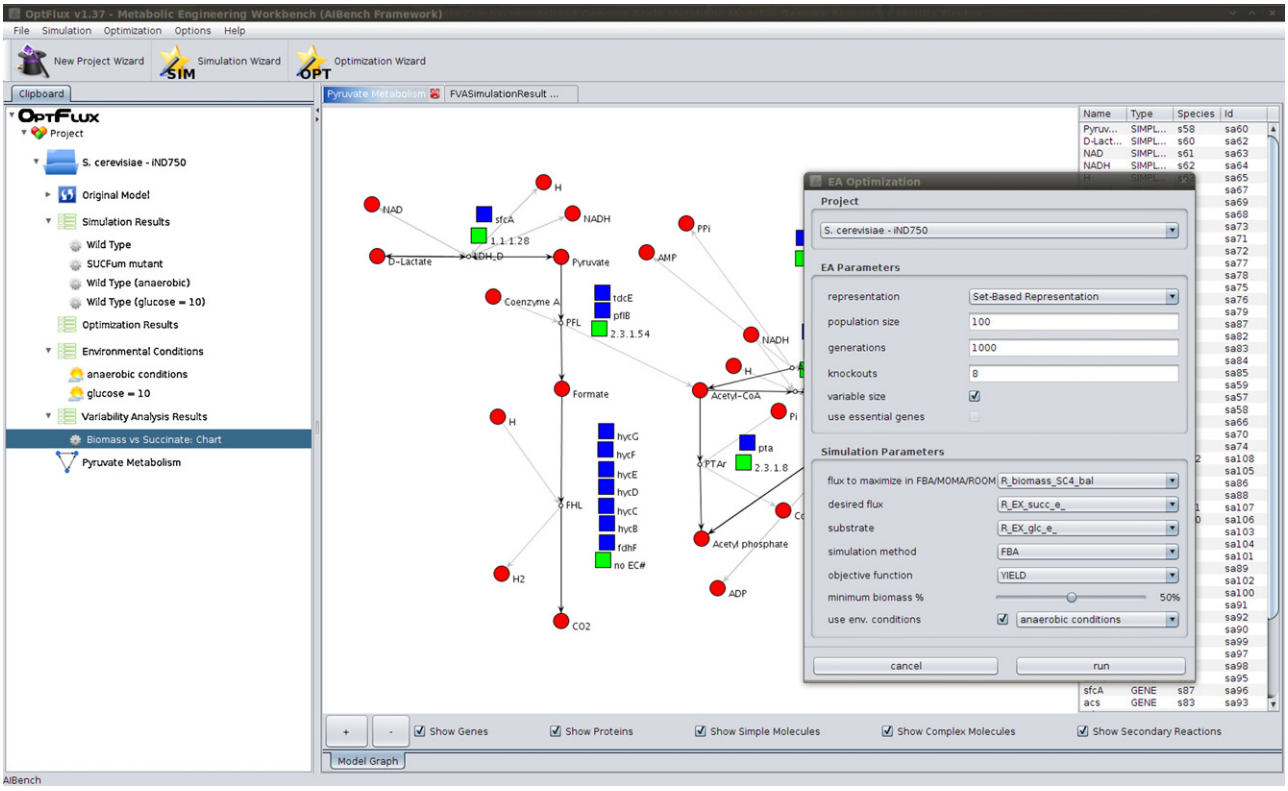

Fig. 6 – Screenshot of the OptFlux application. In the image some tests using the *Saccharomyces cerevisiae* genome-scale model iND750 are represented. Several simulation results as well as predefined environmental conditions can be observed in the Clipboard area. On the right-side, a graphical representation of *pyruvate* metabolism is shown along with the overlapped EA optimization operation GUI.

tory On/Off Minimization of metabolic fluxes) that allow the set of fluxes in the organism's metabolism to be determined, given a specific set of environmental conditions.

OptFlux also includes a number of methods for strain optimization (i.e. metabolic target identification), including metaheuristics such as Evolutionary Algorithms and Simulated Annealing. These are used to reach the best set of gene deletions that optimize a given objective (e.g. the Biomass-Product Coupled Yield). OptFlux incorporates a visualization tool for analyzing the model structure and simulation results that is compatible with the layout information of CellDesigner.[30] An illustration of the main OptFlux functionalities developed in the AIBench framework is given in Fig. 6.

4.2. *Biomedical Text Mining support with @Note*

The field of Biomedical Text Mining (BTM) has been growing rapidly over the last few years, providing a number of valuable methods for the automated extraction of useful knowledge from the biomedical literature. A number of new algorithms have been recently proposed for the main BTM tasks and those were evaluated over several benchmarks. However, we are still far from achieving an acceptable level regarding the transfer of this knowledge to the biomedical research community.

@Note [15] is a Biomedical Text Mining platform that aims to bridge this gap. It copes with the major Information Retrieval and Information Extraction tasks and promotes multi-disciplinary research, providing support to three different usage roles: biologists, text miners and application developers.

The workbench is meant for both BTM research and curation. On one hand, it supports regular curation activities, providing an intuitive interface requiring no prior knowledge of the specific technique's implementation. On the other hand, it is also meant for people with programming skills who might wish to extend the workbench capabilities.

Regarding its main functionalities, @Note provides the ability to process both abstracts and full-texts; an information retrieval module enabling PubMed search and journal retrieval; a pre-processing module with PDF to text conversion, tokenisation and stopword removal; a named entity recognizer based on dictionaries; a manual curation environment and a data mining module specific for BTM tasks. Therefore, @Note sustains the general workflow of BioTM, fully covering the main activities performed. The main functionalities of @Note are illustrated in Fig. 7. The complete software is freely available in http://sysbio.di.uminho.pt/anote/wiki.

[30] http://www.celldesigner.org/.

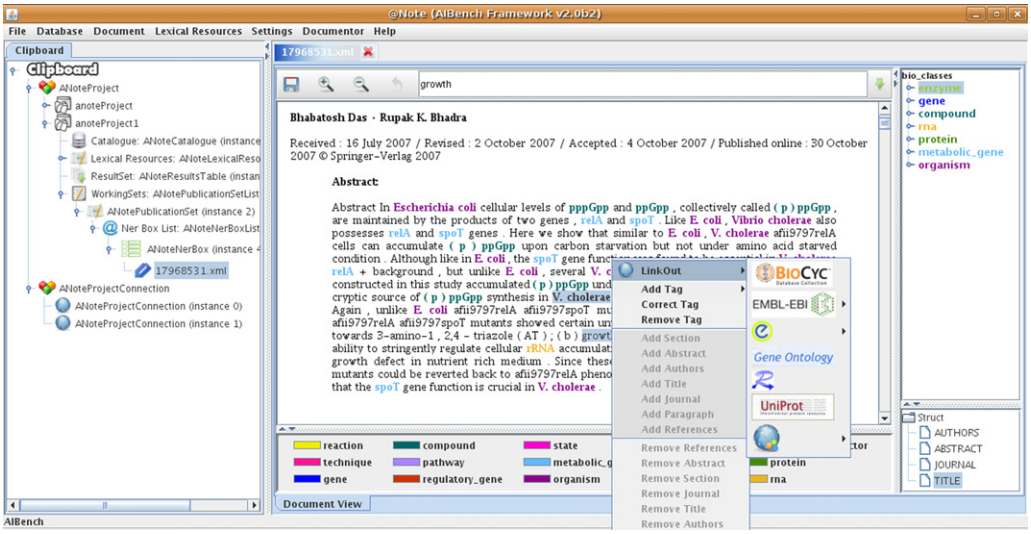

Fig. 7 – Screenshot of the @Note application. Multi-project capabilities are shown in the Clipboard area. The expanded project represents a full cycle of operations using the @Note application including: PubMed search, Information Retrieval, Document Structuring and Named Entity Recognition with automatic annotation. On the right-side, a manual curation process is being applied to an automatically annotated document by using an advanced and interactive custom view.

4.3. *Retrieve and manage Gene Ontology annotations with GOABench*

Biologists currently spend a lot of time and effort in searching for all the available information about each small area of research. This fact is hampered further by the wide variations in terminology that may be common usage at any given time. This inhibits effective searching by both computers and people. For example, one database can associate a gene product with the term 'translation', whereas another could use the

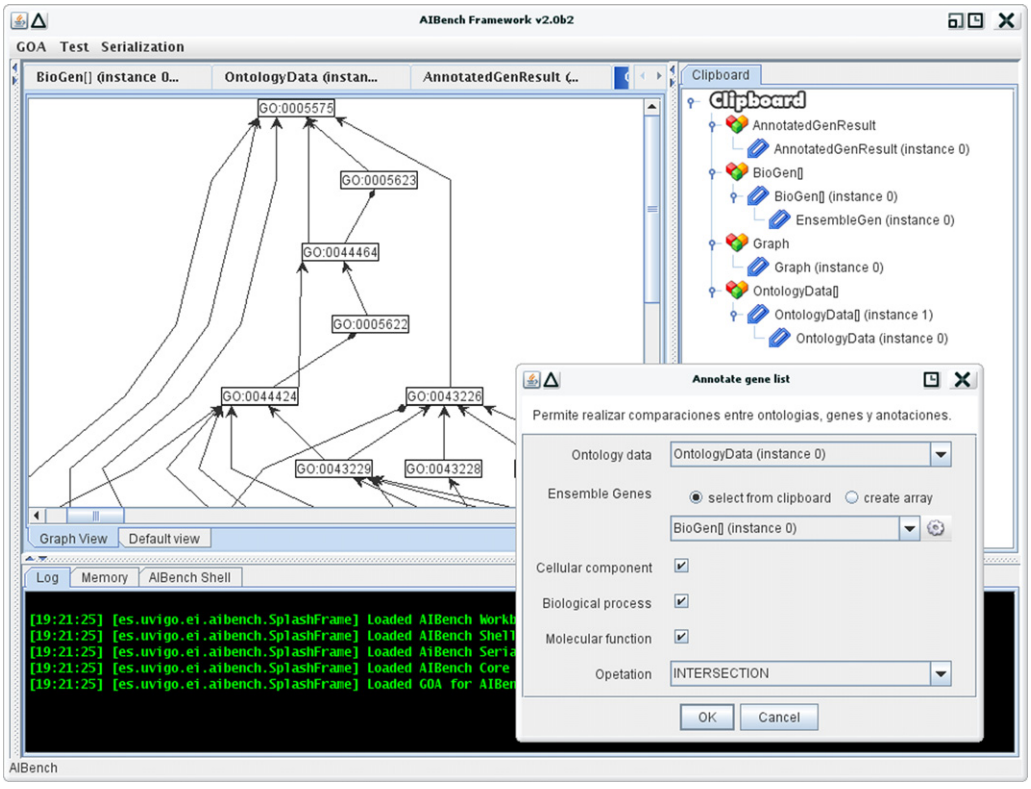

Fig. 8 – Screenshot of the GOABench (*GO Annotations for AIBench*) application. The screenshot shows a results view with a rendered graph together with a fully generated annotation process dialog. Annotation results and graphs are indexed by the clipboard tree showed on the right-side of the main window.

phrase 'protein syntheses', making it difficult for researchers and even harder for a computer to find functionally equivalent terms.

In order to address these terminological issues, the Gene Ontology (GO) project provides a controlled vocabulary to describe gene and gene product attributes in any organism [16]. In this context, GO defines three sub-ontologies (cellular component, biological process and molecular function) with a direct acyclic graph structure, where the main relationships belong to the type 'is_a'. A gene product might be associated with or located in one or more cellular components being active in one or more biological processes, during which it performs one or more molecular functions. Collaborating databases annotate their genes or gene products with GO terms, providing reliable references and indicating what kind of evidence is available to support existing annotations.

The GOABench tool implements a user-friendly desktop application with the goal of facilitating the retrieval and automatic annotation of custom gene lists obtained by the wet-lab users during their experiments. GOABench software allows the user to perform several tasks including (i) retrieve and download an up-to-date version of the Gene Ontology database, (ii) render the ontology via interactive direct acyclic graphs, (iii) highlight relevant terms in the ontology given a gene list of interest and (iv) export both the graphs and the annotation results to external formats. An example of the GOABench software is illustrated in Fig. 8.

5. Conclusions

This paper has presented AIBench, a new Java desktop application framework oriented to the scientific and biomedical domain which was born inside a research group interested in increasing its software development productivity.

AIBench provides the programmer with a proven design and architecture. Following the MVC (model-view-controller) design pattern, the applications developed with AIBench are divided into three types of well-defined objects: operations, data-types and views, which identify units of work with a very high coherence that can be easily combined and thus reused. This common programming model has also a direct benefit in the application maintenance carried out by other developers in a research team, especially by those programmers who are familiar with AIBench.

Every AIBench-based application automatically inherits several functionalities which are independent of the problem scope, but useful for every application like input dialog generation, application context management, experiment repeatability, concurrent execution of operations, etc. The programmer can spend more time in the problem-specific requirements rather than in the low level details. These built-in capabilities are also highly customizable in order to meet the final application needs. Custom input dialogs, views, icons, and components can replace defaults without changing the AIBench code base. In addition AIBench incorporates many configuration options, such as component placement in the main window, operations visibility, optional toolbar, etc.

The plug-in based architecture of AIBench allows applications to be easily developed by adding new modules, each one containing a set of AIBench objects. The coarse-grained integration between functionalities is carried out by establishing dependencies between these plug-ins. This also facilitates reusing and integrating functionalities of past and future developments based on this framework.

The main limitation of AIBench is related to its native input–process–output application model. In this sense, the proposed design approach could be sometimes very coarse-grained in order to develop highly interactive applications. In this context, coding a new operation able to handle every user interaction does not seem to be the best design decision. However, for most final applications the user interactive work to be done is related to results rendering, such as sort tables, manipulate graphs, etc. In these situations, and given the fact that AIBench allows the programmer to provide any Java component in order to render a specific data-type, views are the best place to handle these minor events, and thus, to provide the final application with more interactivity. In addition, we have cleaned and made public many methods of the Core and Workbench plug-ins. Through their API, the programmer can interact with the AIBench kernel in order to trigger operations, listen to operation start and finish events, directly fetch data from the Clipboard, etc. The so-called *Service* plug-ins make extensive use of the Core and Workbench API.

AIBench is free software (under the terms of the GNU Lesser General Public License) and both the source code and the SDK can be downloaded from its website. Recently we have opened the CVS access to the source code and installed some other collaborative tools such as a wiki and a discussion forum.

6. Mode of availability

The AIBench SDK and source code are freely available from the project homepage on http://www.aibench.org and licensed under the terms of the GNU Lesser General Public License.

Acknowledgements

This work was partially supported by the Integrated Action *Development of computational tools for cancer diagnosis using gene expression data* (HP2006-0125) between Portugal (University of Minho) and Spain (University of Vigo), the project *Development of biomedical applications* (09VIB10) from University of Vigo and the project MEDICAL-BENCH: Platform for the development and integration of knowledge-based data mining techniques and their application to the clinical domain (TIN2009-14057-C03-02) from Ministry of Science and Innovation. D. Glez-Peña acknowledges Xunta de Galicia (Spain) for the program María Barbeito. We would also like to thank all those involved in the implementation of the *OptFlux*, *@Note* and GOABench applications referred to in Section 4, namely: Isabel Rocha, Anália Lourenço, Eugénio Ferreira, Pedro Evangelista, Rafael Carreira, José P. Pinto, Rubén Romero, Pablo Ferreiro and José R. Méndez.

REFERENCES

[1] J.C. Wooley, H.S. Lin, Catalyzing Inquiry at the Interface of Computing and Biology, The National Academies Press, Washington, DC, 2005.

[2] P.J. Embi, P.R. Payne, Clinical research informatics: challenges, opportunities and definition for an emerging domain, Am. Med. Inform. Assoc. 16 (2009) 316–327.
[3] J. Nakaya, The Translational Research Informatics (TRI), Int. J. Comput. Sci. Netw. Secur. 6 (2006) 7A.
[4] M.E. Fayad, D.C. Schmidt, R.E. Johnson, Building Application Frameworks: Object-Oriented Foundations of Framework Design, John Wiley & Sons, New York, 1999.
[5] G. Benson, Web server issue, Nucleic Acids Res. 36 (2008), doi:10.1093/nar/gkn381, W1.
[6] I. Wolf, M. Vetter, I. Wegner, T. Böttger, M. Nolden, M. Schöbinger, M. Hastenteufel, T. Kunert, H.P. Meinzer, The medical imaging interaction toolkit, Med. Image Anal. 9 (6) (2005) 594–604.
[7] J. Rexilius, W. Spindler, J. Jomier, M. Koenig, H. Hahn, F. Link, H. Peitgen, A framework for algorithm evaluation and clinical application prototyping using ITK, in: MICCAI Workshop on Open Science (MICCAI'05), The Insight Journal, 2005.
[8] A. Enquobahrie, P. Cheng, K. Gary, L. Ibanez, D. Gobbi, F. Lindseth, Z. Yaniv, S. Aylward, J. Jomier, K. Cleary, The Image-Guided Surgery Toolkit IGSTK: an open source C++ software toolkit, J. Digit Imaging 20 (Suppl. 1) (2007) 21–33.
[9] B. von Rymon-Lipinski, T. Jansen, Z. Król, L. Ritter, E. Keeve, JULIUS—an extendable application framework for medical visualization and surgical planning, in: Computer Assisted Radiology and Surgery (CARS'01), International Congress Series, Elsevier, 2001, pp. 184–189.
[10] E. Samset, A. Hans, J. von Spiczak, S. DiMaio, R. Ellis, N. Hata, F. Jolesz, The SIGN: a dynamic and extensible software framework for image-guided therapy, in: MICCAI workshop on Open Source and Data for Medical Image Computing and Computer-Assisted Intervention (MICCAI'06), The Insight Journal, 2006.
[11] M. Viceconti, C. Zannoni, D. Testi, M. Petrone, S. Perticoni, P. Quadrani, F. Taddei, S. Imboden, G. Clapworthy, The multimod application framework: a rapid application development tool for computer aided medicine, Comput. Methods Programs Biomed. 85 (2) (2007) 138–151.
[12] T. Rudolph, M. Puls, C. Anderegg, L. Ebert, M. Broehan, A. Rudin, J. Kowal, MARVIN: a medical research application framework based on open source software, Comput. Methods Programs Biomed. 91 (2) (2008) 165–174.
[13] D. Glez-Peña, F. Díaz, J.M. Hernández, J.M. Corchado, F. Fdez-Riverola, geneCBR: a translational tool for multiple-microarray analysis and integrative information retrieval for aiding diagnosis in cancer research, BMC Bioinformatics 10 (2009) 187.
[14] G. Stephanopoulos, A. Aristidou, J. Nielsen, Metabolic Engineering, Academic Press, San Diego, 1998.
[15] A. Lourenço, R. Carreira, S. Carneiro, P. Maia, D. Glez-Peña, F. Fdez-Riverola, E.C. Ferreira, I. Rocha, M. Rocha, @Note: A workbench for biomedical text mining, J. Biomed. Inform. 42 (4) (2009) 710–720.
[16] Gene Ontology Consortium, The Gene Ontology (GO) database and informatics resources, Nucleic Acids Research 32 (Database issue) (2004) D258–261.