ADESSO

SCIENTIFIC SOFTWARE DEVELOPMENT ENVIRONMENT

Rubens C. Machado[1], *Roberto de A. Lotufo*[2], *Alexandre G. Silva*[2] *and André V. Saúde*[1,2]

[1]Renato Archer Research Center, P.O.Box 6162, 13089-120, Campinas, SP, Brazil
rubens.machado@cenpra.gov.br
[2]DCA–FEEC, University of Campinas, P.O.Box 6101, 13083-970, Campinas, SP, Brazil
{alexgs,lotufo,andrevit}@dca.fee.unicamp.br

ABSTRACT

This paper presents the Adesso, a computational environment for the development of scientific software. The Adesso environment leverages the reusable software component programming model to support the development and integration of components to several scientific programming platforms. The Adesso system is based on an XML component database and a set of XML document transformation tools for the automatic generation of component code, documentation and packaging. An authoring tool, built with the help of the Adesso transformation system itself, is provided to assist the user in the creation of components. The Adesso system has been used for the development of several image processing toolboxes and applications. Some of them are presented briefly in this paper.

1. INTRODUCTION

Research projects in areas like image processing, computer vision, pattern recognition and others have its main concern on the validation of new algorithms and methods. To achieve this goal, the researcher usually develop computer programs. These programs, while not being the main result of the research project, are fundamental pieces and, many times, constitute the basis for the developing of new, innovative products.

A successful scientific software application involves, besides high quality algorithms, a robust and up-to-date implementation conforming the current practices of the software field. In the current state-of-the-art, these are very demanding: sophisticated user interfaces, integration among applications, high quality user documentation and so on. In addition, the fast evolution of the modern computer platforms poses another problem to the maintenance of these applications.

All these factors lead to software models where the reutilization of programs has a fundamental role. One of such models, the component/solution model, has been successfully applied for years since the advent of the Visual Basic *controls*. In this model, software components are created and packaged independently of the integration platform where, in a subsequent step, they are glued together to form an application. Typically, the integration platform takes care of the tasks related to the integration with the underlying operating system, so that the component developer can be concerned mainly with the algorithms and methods implemented by the component.

Today, there are a number of such *integration platforms* suitable for scientific software development. Scripting languages play an important role here. The MATLAB scripting language has become a standard for scientific programming; open-source scripting languages like Perl, Tcl and Python are other actors in this play. Scripting languages like these are known to be important productivity tools, often speeding development by a factor of ten in relation to systems languages like C/C++ or Java [1]. Moreover, those languages can be easily extended with programs developed in high performance languages like C and C++, useful when dealing with problems in image processing and related areas.

The system presented in this paper provides tools to support scientific software development in this framework of extensible scripting languages, emphasizing the two-language approach for the scientific software development. To this end, our system provides a set of tools to automate the process of creating extension packages in languages C and C++ and integrating them to the target scripting language.

Another core issue in scientific software development is the documentation. Reutilization of components is closely tied to the quality of the component documentation. Many times, the documentation efforts in scientific software development consumes the bulk of the project cost. Initiatives like the *literate programming* [2] tries to couple the creation of computer source code and the associated documentation, stating that the programmer main task is to explain his code to other programmers, not only to comput-

This work was supported by FAPESP, Brazil.

ers. The programmer's main deliverable is a *document* from where code can be generated.

As we will see in the course of this work, our system tries to incorporate the main ideas of the literate programming model, although changing the emphasis to the documenting of the *component interfaces* instead of documenting the component implementation.

Software deployment is another source of troubles to the developer. Packaging, installation and maintenance of software systems are important issues when deploying applications. Our system address these common needs in an integrated fashion, considering them from the beginning of the development process.

This paper is organized as follows. The next section presents an overview of the Adesso system and the rationale for the selection of its main base technologies. The authoring tools and the underlying information model is the matter of the following section. Section 4 deals with the Adesso transformation processor and its stylesheets. Finally, some applications developed with the Adesso are briefly described and we summarize our plans for the near future and draw some final comments.

2. OVERVIEW

The Adesso, *Scientific Software Development Environment*, [3] is our response to the issues outlined in the previous section. It is a support system for the development of scientific software components. The Adesso underlying model embodies concepts like the component/application paradigm and the emphasis on documentation and testing. Like literate programming, it enforces the joining of code and documentation, but, unlike literate programming, the documentation is aimed at the component interface instead of the implementation.

The Adesso architecture is based on some main concerns: (a) the component information is represented in a centralized and structured way, (b) *code generators* are applied to build releases of the software and (c) *authoring tools* are used for the creation of components.

The Adesso user deals with the creation of component sets, called *toolboxes*. A *toolbox* is made of all the information needed for the building of a *product*. A *product* comprises the executable code and the documentation of the *toolbox* for one *integration platform* running on a *computational platform*. These *products* are built through transformation tools called *code generators*. As an example, an "Image Processing Toolbox" may be used to create a product for use with MATLAB on Windows platforms; the same toolbox may be the base for a Python/Linux product. The development process is illustrated in Figure 1.

Our implementation of the Adesso environment is heavily based on the *Extensible Markup Language*,

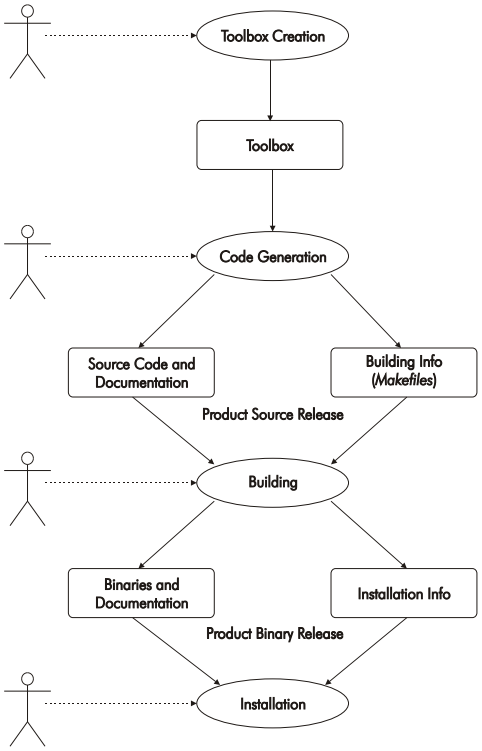

Fig. 1. Toolbox Development Process

XML, [4] and its related standards. XML is derived from the *Standard Generalized Markup Language*, SGML. SGML has long been used as a means of document management in large corporations. XML is meant to be a simpler language aiming widespread use on Internet systems.

Some XML features were crucial for our choice: (a) XML is application-neutral and cross-platform. It is a human-readable plain text language and there are an increasing number of XML parsers and other tools available for nearly all computer platforms. (b) XML documents are structured and the markup is domain-specific. This enables us to define the data model to be used by the toolboxes and allows the validation of a specific instance of the model. (c) The transformation model stated by the XSLT standard (*Extensible Stylesheet Language*, a companion XML standard [5]) is specially useful for our purposes – automatic code and document generation.

Having chosen these technologies to make up the foundation of our system, we defined the architecture shown in Figure 2 for the Adesso.

The Adesso provides tools for the creation of *toolboxes*, transformation stylesheets and derived *products*. The user makes use of authoring tools to create a *toolbox* in conformance with the data model established by the Adesso. For each supported integration platform (scripting language), the system supplies *stylesheets* to drive the transformation machine, *style processor*, in order to create a product source

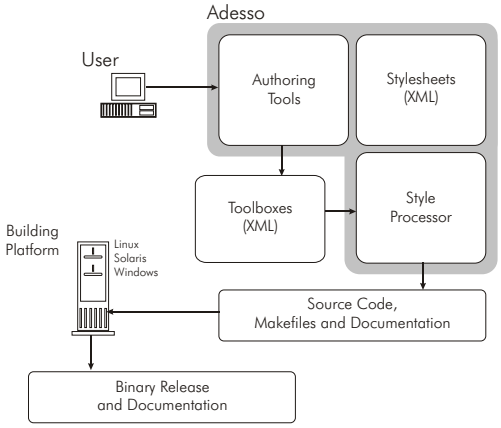

Fig. 2. Adesso Architecture

release. This source release is independent of computational platforms (operating systems). Once the source release is generated, a binary (executable) release can be built on several computational platforms. This stage makes use of common tools like compilers and linkers available in the target platform, in addition to the Adesso provided cross-platform building tool (a kind of *make*).

The Adesso system was developed with the *Tcl/Tk* [6] scripting language and its extension for XML processing *tDOM* [7].

In the next sections, we will describe the available Adesso tools for the main stages in toolbox development: toolbox authoring and code/document generation, building, packaging and installing.

3. TOOLBOX AUTHORING

Using the authoring tools provided by the Adesso system, the author writes the source code and documentation for each component of the toolbox, following the *toolbox data model*, presented next. The whole information entered in this stage is independent of the integration and computational platforms.

Since the user is editing an XML file, he can choose among the several generic XML editors currently available. However, a specialized editor is a better choice as it can enforce the correctness of the information entered by the user and optimize some usual procedures. A specialized XML editor is one that has embodied the underlying XML data model and is capable of validating the XML document being created.

To define the structure and data types of an XML document, there is a standard named *XML Schema* [8]. This standard describes a language, based on the XML syntax, to define the structural relationships among the elements of an XML document and the data types represented in document instances. The *XML Schema* document enables the validation of instances of the target document, much like the DTD, *Document Type Definition*, described in the XML standard, does.

Observing that the schema is itself an XML document, we are able to use the Adesso transformation tools to generate useful code to create the graphical user interface elements for a toolbox editor. The *toolbox XML Schema* document can, thus, drive the creation of specialized editors for the Adesso. We have explored this framework in a project called *XML Contextual Editor*.

The *XML Contextual Editor* project is aimed at creating a framework for the generation of XML editors based on transformations of an XML Schema correspondent to the targeted XML documents. In other words, it creates XML editors to edit a determined XML document structure.

An XML Schema consists, basically, of datatype and structure definitions for validating a class of XML documents. Every datatype definition is derived, by restriction or extension, from a primitive datatype or from another derived datatype. Although inheritance and other object-oriented modeling concepts are not directly used in these derivation mechanisms, the specification allows direct mapping of an XML Schema to such a model.

The *XML Contextual Editor* project utilizes the transformation infra-structure of the Adesso to automatically generate implementations of the XML Schema object-oriented model. For each datatype implementation, a small graphical component is also generated with specific properties. The whole set of these graphical components, together with the defined structures, makes up a high-level Graphical User Interface (GUI), assuring easy edition of only valid XML instance documents. A screenshot of an automatically generated GUI for toolbox editing is shown in Figure 3. The editor was implemented in the Java language.

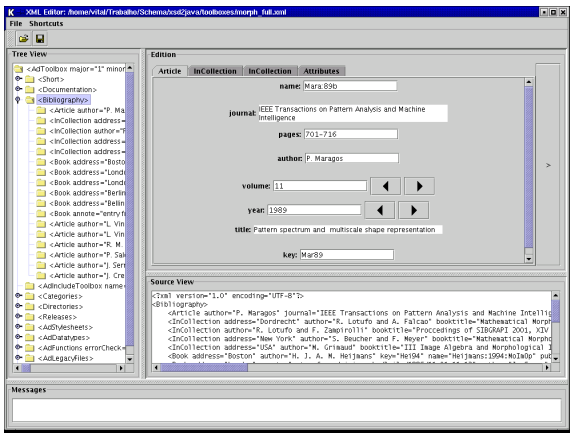

Fig. 3. Adesso Contextual Editor

The results of this project are currently being used to create a distributed authoring tool intended for the AdessoWeb project (described in the following).

3.1. Toolbox Data Model

The structure of a toolbox is modeled by an XML Schema document. Toolboxes are component collections. There are a variety of component types like, for example, functions written in the MATLAB language, functions in C, demonstration scripts, etc. The toolbox XML files are intended to hold (or point to) all the code and documentation about these components. Next is a brief presentation of the XML elements of a toolbox.

Toolbox identification: toolbox name and version, CVS strings.

Global documentation: generic documents, README files, installation instructions, bibliography etc.

Component classifiers: tags used to group components for documentation, organization in directories and releases.

Components: this constitutes the core of the toolbox. There is a variety of component types, like MATLAB functions, C functions, demonstrations etc. Each component contains the following element groups:

- **Identification:** name of the function.
- **Documentation:** documentation about the function.
- **Prototype definition:** definition of the function prototype (signature).
- **Implementation:** this element contains or points to the function source code.
- **Dependencies:** enumerates possible dependencies on other toolbox components.
- **Testsuite:** code to exercise the component features in order to validate its behavior.

External dependencies: enumerate toolbox dependencies on external libraries and other kinds of files.

Toolbox-specific stylesheets: these elements allow the customization of the existing stylesheets to take into account toolbox peculiarities.

4. CODE GENERATION

In the code generation stage, the transformation tools are used to build a *source distribution* for a chosen integration platform. This distribution contains all the sources and documentation of the components, along with generated interfaces for use with the integration platform and building instructions for the creation of *binary distributions* (Makefiles). The source distribution is independent of the computational platform.

The process of transforming the toolbox data into a source distribution is driven by a set of *stylesheets* and implemented by a *style processor* as shown in Figure 4. Each supported integration platform has a corresponding set of stylesheets.

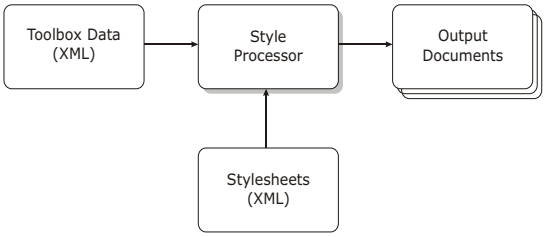

Fig. 4. Adesso Transformation Model

The transformation model depicted in Figure 4 is basically the same used by the *XSLT*. In fact, it was inspired on that. The stylesheets used as input by the Adesso style processor are written in a reduced language that mix the structure of the XSLT stylesheets with fragments of Tcl code. The resulting transform specification is quite flexible, allowing kinds of processing not possible, at least in the usual way, in the more formal language of the XSLT processor. The stylesheets use XML for its syntax, taking advantage of the underlying XML parser. To avoid the direct writing of XML, we have developed a simple preprocessor based on indentation (an idea borrowed from the Python language, where the indentation level of the statements is used to group them in code blocks). The following is a very simple stylesheet to show the language appearance:

```
stylesheet toolbox_index.html:

    template AdToolbox:
        <html><body>
            <h1>
                Component Index for Toolbox
                [sty:value @name]
            </h1>
            <dl>
                [sty:apply AdFunctions/AdFunction]
            </dl>
        </body></html>

    template AdFunction:
        <dt><b>[sty:value @name]</b></dt>
        <dd>
            [sty:value Short]
            (implemented in [sty:value Source/@lang])
        </dd>
```

The stylesheet creates an HTML page containing an index of the toolbox components. The page title indicates the name of the toolbox, obtained from the attribute *name* of the XML top element (*AdToolbox*) via the processor command `sty:value`. The command `sty:apply` causes the firing of the templates that match the elements of type *AdFunction*, where lives the function data. For each toolbox function the stylesheet emits one HTML definition list item formed by the name of the function, its one-line

description (XML element *Short*) and a note about the component implementation language(s) (attribute *Source/@lang*).

We have implemented several sets of stylesheets for supporting the code and documentation for the following integration platforms: C libraries, MATLAB, Tcl/Tk and Python. For each platform we are able to create the core toolbox library, the interfaces for using the code from the platform, demonstration scripts, building and installation rules and documentation in HTML, online help systems and PDF.

An interesting feature of the documentation generation stylesheets is the generation of figures from scripts. The figures are specified through the scripts that will create them. This feature enables the *Reproducible Research* [9] approach to scientific publications, where all results are presented along with the corresponding software piece.

To help in the generation of *binary distributions* for different computational platform, the Adesso system provides a platform-independent building tool that interprets the building rules created in the previous stage and launches the appropriate tools in the target platform. The target platform is supposed to provide tools like compilers and linkers.

5. APPLICATIONS

5.1. SDC Morphology Toolbox for MATLAB

As stated in its homepage[1], the SDC Morphology Toolbox for MATLAB is a powerful collection of latest state-of-the-art gray-scale morphological tools that can be applied to image segmentation, non-linear filtering, pattern recognition and image analysis.

The roots of the Adesso system are closely related to the development of the SDC Morphology Toolbox. This was the first application built with the help of the Adesso. The project was fairly large and the data model and transformation styles of the Adesso was largely influenced by the structure of this morphology toolbox.

The SDC Morphology Toolbox web site illustrates much of the capabilities of the Adesso system, as it was almost fully created with our system.

5.2. Python Image Processing Toolboxes

Recently, the *Python* language [10] is having a great growth. It is a modern scripting language, with refined concepts of object orientation and modularization. The *Numeric* package [11] is an extension that brings to the Python world strong support for multi-dimensional matrix computation. Python, associated with the Numerical package, is a generic language, extremely portable and efficient enough for image processing tasks. Python has the flexibility of Perl, associated with the numerical power and ease of use of MATLAB, but is available as an open source environment. The source code is generally small when compared to compiled languages by several reasons: high-level data types and operations, no type declarations (dynamic typing), automatic memory management, and command blocks marked by indentation. In C/C++, equivalent data structures and functionalities with the same optimization would cost considerable more programming time. There is also a great native set of libraries implemented in C/C++ (built-in) that practically discard the process of compilation / correction / re-compilation (except for API extensions of the language). These characteristics generate a high productivity gain.

The Adesso has full support to the creation of extensions to the Python language, automating the development of C, C++ and pure Python modules. Supported by the Adesso, we have developed an image processing toolbox for the Python language. This image processing toolbox [12] is intended to be used as a practical resource in image processing courses. The Python toolbox is compatible to the equivalent MATLAB image processing toolbox being used and developed.

The image processing toolbox is called `ia636` as a reference to the code of the computer vision graduate course taught at the Faculty of Electrical and Computer Engineering, UNICAMP. The last version of the toolbox can be seen at the ia636 homepage[2].

5.3. ProntoVideo

ProntoVideo [13] is an interactive tool for object video segmentation (video masking). It allows users to select, extract (delineate), and track objects from arbitrary backgrounds of image sequences (digital video).

ProntoVideo was developed with the language Tcl/Tk and supported by some image processing extensions written in C and developed in the Adesso environment.

5.4. AdessoWeb

The ubiquity of the World Wide Web in these days increasingly stress the excellence of the component based software – the Internet is a huge component integration environment. As the Adesso is heavily based on a Web core technology, the XML, it is particularly suited for operation via the Internet. On the other side, as the Adesso is used by a growing group of developers and makes use of many utility tools from different sources, the difficulties related to the installation and maintenance are becoming significant.

[1] http://www.mmorph.com

[2] http://marahu.dca.fee.unicamp.br/course/ia636.html

The AdessoWeb project is aimed at operating the Adesso in a distributed environment and publishing the interfaces through the WWW. This framework will avoid the installation problems and facilitate the maintenance. The AdessoWeb is based on *distributed objects* cooperating on a CORBA bus and operated primarily through World Wide Web browsers. Physically, the system is a conglomerate of computers connected by a local area network that provides services through the Internet. Figure 5 shows the main components of the AdessoWeb. The AdessoWeb is currently under development.

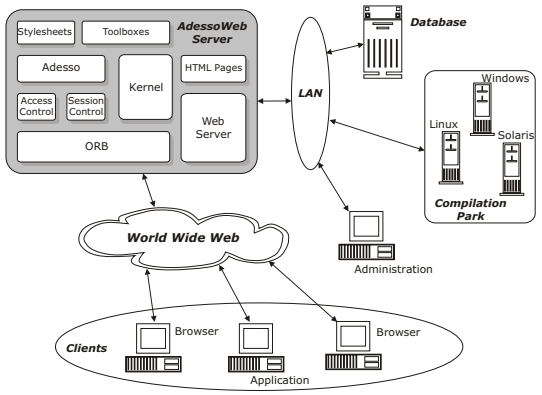

Fig. 5. AdessoWeb Organization

6. CONCLUSION

This paper presented the Adesso system for scientific software development and outlined some application developed with its help. After a couple of years using the Adesso, we can testify the benefits of its development process, mainly in the software reusability, documentation and maintenance. For the near future, we plan to extend the Adesso with distributed systems capabilities, intending its use through the World Wide Web. Also, the utilization of the Python language for image processing systems will deserve our attention in the next months.

7. REFERENCES

[1] J. K. Ousterhout, "Scripting: Higher level programming for the 21st century," *IEEE Computer*, 1998.

[2] D. E. Knuth, "Literate Programming," *The Computer Journal*, vol. 27, May 1984, Issue 2.

[3] R. C. Machado, "Adesso - Ambiente Para Desenvolvimento de Software Científico," M.S. thesis, Electrical Engineering Faculty - UNICAMP, Brazil, jun 2002.

[4] W3C Recommendation 6-Oct-2000, *eXtensible Markup Language (XML) 1.0 (Second Edition)*, http://www.w3.org/TR/REC-xml-20001006.

[5] W3C Recommendation 16-Nov-99, *XSL Transformations (XSLT) Version 1.0*, http://www.w3.org/TR/xslt.

[6] B. B. Welch, *Practical Programming in Tcl and Tk*, Prentice Hall PTR, 3rd edition, 1999.

[7] J. Loewer, *tDOM - A fast XML/DOM/XPath package for Tcl writen in C*, http://sdf.lonestar.org/ loewerj/tdom.cgi.

[8] W3C Recommendation, 2 May 2001, *XML Schema Part 0: Primer*, http://www.w3.org/XML/Schema.

[9] Jonathan B. Buckheit and David L. Donoho, "Wavelab and Reproducible Research," http://www-stat.stanford.edu/ donoho/Reports/1995/wavelab.pdf.

[10] M. Lutz and D. Ascher, *Learning Python*, O'Reilly and Associates, April 1998.

[11] D. Ascher, P. F. Dubois, K. Hinsen, J. Hugunin, and T. Oliphant, "Numerical Python," September 2001, http://numpy.sourceforge.net/numdoc/numdoc.pdf.

[12] A. G. Silva, R. A. Lotufo, and R. C. Machado, "Toolbox of Image Processing for Numerical Python," *Sibgrapi*, October 2001, Florianópolis, Brazil, IEEE.

[13] R. Lotufo, R. Machado, F. Flores, A. Falcão, R. Koo, G. Mazzela, and R. Costa, "Prontovideo – an image sequence segmentation tool applied to video edition," *Sibgrapi*, October 2001, Florianópolis, Brazil, IEEE.