COMODI: Architecture for a Component-Based Scientific Computing System

Zsolt I. Lázár[1] and Lehel I. Kovács[2]

[1] University College Dublin, Belfield, Dublin 4, Ireland
Zsolt.Lazar@ucd.ie,
WWW home page: http://phys.ubbcluj.ro/~zlazar
[2] Babeş-Bolyai University, Faculty of Mathematics and Computer Science, 400084 Cluj-Napoca, Romania

Abstract. The COmputational MODule Integrator (COMODI) [1] is an initiative aiming at a component based framework, component developer tool and component repository for scientific computing. We identify the main ingredients to a solution that would be sufficiently appealing to scientists and engineers to consider alternatives to their deeply rooted programming traditions. The overall structure of the complete solution is sketched with special emphasis on the Component Developer Tool standing at the basis of COMODI.

Brute computing force growing at exponential rate is not the answer to the problem of complexity as signaled already in the late sixties when the term “software crisis” got coined [2]. Recently, scientists find similar obstacles standing in the way of large scale scientific software projects [3, 4], and argue for a change of paradigm. Our work ([5], [6]) comes to same conclusions in the context of small and medium size projects that make up the bulk of the activity in the community. It is pointed out that there is an acute need for shifting towards a reuse oriented paradigm which would improve the efficiency and quality of computational research.

The COmputational MODule Integrator (COMODI) [1] is an initiative aiming at a component based framework, component developer tool and component repository for scientific computing. It emerges from a set of premises that are the conclusions of a preliminary survey made with a mixed group of computational scientists on the occasions of conferences, workshops and via an on-line form on the COMODI website [1]. These translate into a series of recommendations that will guide the design process. A distinction is made between component developers and end-users: the former design and implement new components while the latter are primarily involved in assembling these into projects.

Since the two activities require different skills and work methods the requirements set for the employed tools in each case also differ. For user satisfaction the solution has to be endowed with the features listed below:

- user friendly graphical interfaces;
- intuitive, high-level representation of data and processes such that the elements of low-level programming can also be clearly identified;

- possibility for low-level control;
- support for most popular hardware and software platforms;
- comprehensive component repository;
- open source.

In order to fully support developers it is imperative that no compliance criteria are set for the computational code neither in terms of structure nor used data types. In other words, any valid code written in the supported programming languages should automatically be ready for COMODI. Therefore several restrictions apply to the process of adapting regular code to COMODI:

- no change in the source code, the interface or the implementation;
- no extra coding - connectivity is achieved by supplementing author provided source-code with automatically generated glue-code;
- no need for the author to know other languages/standards then the ones used for implementing the code;
- no platform dependence - the capabilities of the system the development is carried out on is extended by on-line servers providing compilation as web service;
- no language dependence - all present and future languages should be able to communicate seamlessly;
- low performance overhead;
- support for both open source and commercial components.

The complete solution should include the following elements:

- visual programming environment for computational projects;
- standardized scientific component descriptor language (CDL);
- component developer tools for adapting regular code to the framework;
- distributed component repository;
- compilation web service.

Figure 1 shows the COMODI architecture. The responsibilities of each part are summarized in Table 1.

The developer layer contains a user friendly *Graphical User Interface* (GUI), a *Component Developer Tool* (CDT) with a *Parser*. The CDT, after semi-automatically collecting information pertaining to the content, behavior, and representation of the component, generates a *component descriptor file* (CDF) in the XML based *Component Descriptor Language* (CDL) and the source of the *glue-code* that will intermediate the communication of the component within the COMODI framework. At this stage the CDF will contain all communication related information such as exported functions and data types. It describes both syntactically and semantically the component, supports the programming style of computational scientists as far as data structures, and it is extensible. Its complexity is expected to grow together with the user community and the number of application areas. By semi-automatic we mean that the *Parser*, which stands at the basis of the tool, analyzes lexically and syntactically the source file and generates a primary CDF. Using the GUI, the developer only has to confirm

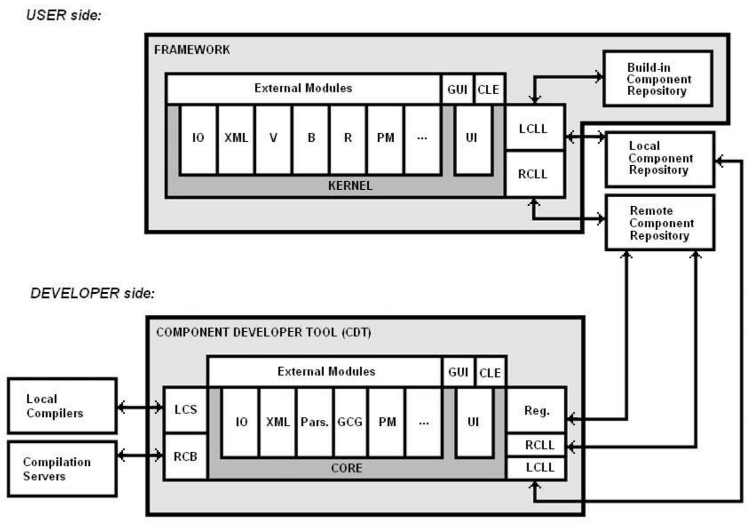

Fig. 1. On the user side: IO: *Input/Output System*, XML: *Extended Markup Language Parser*, V: *Validator*, B: *Binding System*, R: *Running System*, PM: *Project Manager*, UI: *User Interface*, GUI: *Graphical User Interface*, CLE: *Command Line Editor*, LCLL: *Local Component Locater and Loader*, RCLL: *Remote Component Locater and Loader.* On the developer side: LCS: *Local Compilation Service*, RCB: *Remote Compilation Broker*, Pars.: *Parser*, GCG: *Glue-Code Generator*, Reg.: *Registrar.*

Role of the framework	**Role of the component developer tool**
– component assembling – project verification and validation – project execution – runtime user interaction	– assist the developer in documenting the code – generate glue-code – assist the developer in compiling the component – register the component with the global repository

Table 1. Responsibilities of the two major parts of COMODI

the exported ports, provide human readable documentation for the component, set default values and add representation related information. The CDT then contacts on-line *compilation servers* and returns ready-made binaries for the platforms of the developer's choice. The compiled library together with the descriptor file is uploaded by the developer to a place where it can be accessed publicly while the CDT registers the component in the *component repository.* The deployed component is a package containing the component's source code

- if the developer chooses to make the source open - the component descriptor file, the binaries for both the computational- and the generated glue-code, and further resources. Components are packed into standard ZIP or TAR.GZ format and registered in the *Remote Component Repository.* Upon use within the COMODI framework, the component is downloaded and stored in the *Local Component Repository.*

The sources provided by the component developer suffer no changes whatsoever during the component creation process. All glue-code comes as additional functions in a separate file. Not touching the source of the developer has the benefit of the compiled component being usable both within and outside the COMODI framework making COMODI components fully compatible with traditional programming environments.

The parser is implemented such that it can recognize the majority of imperative programming language, once their EBNF definitions are available.

The paper presents the general requirements and a few design guidelines for a complete reuse oriented solution for computational scientists. We attribute the lack of impact of present solutions to code reuse to the high threshold of effort required, in many cases made worse by a closed source and restrictive copyrights. We argue that the community needs a solution that allows a smooth, effortless transition to the new paradigm.

References

1. COMODI homepage: http://phys.ubbcluj.ro/comodi/
2. Naur, P., Randell, B.: Software Engineering: Report on 1968 NATO Conference, NATO, 1969
3. Post, D.E.: The Coming Crisis in Computational Science. Proceedings of the IEEE International Conference on High Performance Computer Architecture: Workshop on Productivity and Performance in High-End Computing, Madrid, Spain, February 14, 2004
4. Post, D.E., Votta, L.G.: Computational Science Demands a New Paradigm. Phys. Today, January (2005) 35
5. Lázár, Zs.I., Pârv, B., Fanea A., Heringa, J.R., de Leeuw, S.W.: COMODI: Guidelines for a Component Based Framework for Scientific Computing. Studia Babeş-Bolyai, Series Informatica, Vol. XLIX, No. 2 (2004) 91
6. Lázár, Zs.I., Heringa, J.R., Pârv, B., de Leeuw, S.W.: COMODI: Component Based Programming in Scientific Computing: The Viable Approach. arXiv:cs.CE/0509002
7. Lázár, Zs.I., Pârv, B.: COMODI: Component Wiring in a Framework for Scientific Computing. Studia Babeş-Bolyai, Series Informatica, Vol. XLIX, No. 2 (2004) 103