SOFA 2 Component Framework and Its Ecosystem

Michal Malohlava, Petr Hnetynka, Tomas Bures [1,2]

Charles University
Faculty of Mathematics and Physics
Department of Distributed and Dependable Systems
Malostranske namesti 25
Prague, Czech Republic

Abstract

Component-based software development represents a common practice to assemble various kinds of systems using well-defined building blocks called components.
SOFA 2 is an advanced component framework providing a rich set of features including hierarchical architectures, multiple communication styles, behavior specification, transparent distribution, etc. The framework also introduces a well-defined development methodology supported by a rigorous component model and rich set of tools enabling application design, deployment, and execution. Furthermore, SOFA 2 is suitable for development of systems for multiple application domains. To allow this, SOFA 2 offers a concept of "profiles", which extend the core of the framework to be suitable for a particular domain. Currently, profiles for Java, Java ME and embedded C-based systems exist, utilizing a common development methodology and tooling.
This tutorial shows the SOFA 2 component framework and its ecosystem including tools for architecture modeling, component implementation, deployment, execution, and runtime monitoring. Furthermore, it demonstrates development workflow stressing the advanced features of the used component model.

Keywords: CBSE, component system, component model, component, sofa, ecosystem, development tool.

1 Introduction

Component-based Software Engineering (CBSE) has become a widely used development technique for all kinds of applications. It addresses develop-

[1] This work was partially supported by the Czech Science Foundation grant 201/09/H057, by the Grant Agency of the Czech Republic project P202/11/0312, and by the Charles University grant SVV-2012-265312.
[2] Email: {surname}@d3s.mff.cuni.cz

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

ment complexity by introducing well-defined building blocks called *components*, which participate in all stages of application development. Components as well as other core development assets (i.e., modeling and deployment tools, execution environment) are typically defined by so called *component framework*.

SOFA 2 [1] is a component framework enabling application development by offering hierarchical components and a rich set of additional features including behavior specification and verification, dynamic architectures, UML-based design, and transparent distribution based on automatically generated software connectors. Contrary to other hierarchical component systems, SOFA 2 supports a complete component life-cycle, i.e., from a design stage till execution and maintenance.

The objective of the tutorial is to present the SOFA 2 framework with emphasis on its ecosystem including demonstration of the application development cycle and corresponding tools.

2 SOFA 2 Component Framework

SOFA 2 supports all stages of the system development process by providing three elementary assets – a component-model, execution environment, and development and deployment tools. These are described in the rest of the section.

2.1 Component Model

The component model of SOFA 2 is defined via its meta-model, which specifies a rich set of features permitting modeling of components, assembling applications, their deployment, and execution.

Components are defined by the component type (called *frame*) and component implementation (called *architecture*). The component frame provides a black box view of the component by declaring provided and required interfaces, their properties such as communication style, and additional information such as behavioral description of communication. The component architecture represents a glass box view of component by declaring component internals – it specifies either a primitive or composite architecture. The former is directly implemented in a target programming language, while the latter is defined as a set of subcomponents and their interconnections.

The meta-model does not support only modeling of functional concerns of components, but also the extra-functional properties (EFP) are captured. Every component has its own control part which manages components' EFPs. The control part is composed of a set of *micro-components* (i.e., special kind of restricted fine-grained components) grouped into so called control aspects [5].

Connections among components are realized by *software connectors*. At design time, they are just links with an associated communication style and EFPs. At deployment time and runtime, the connectors are realized by an automatically generated infrastructure mediating communication which reflects the properties and components distribution [2].

Dynamic architecture reconfigurations are supported via well-defined reconfiguration patterns. Currently, three reconfiguration patterns are provided: *factory pattern*, *removal pattern*, and *service access pattern* [4]. The factory pattern serves to create a new component based on a pre-defined template, while the removal pattern is utilized to destroy such a component. With the help of the last pattern, components can access to external services and also they can provide externally accessible services.

Behavior of components can be formally specified by *behavior protocols* [6] and verified.

2.2 Runtime Support

SOFA 2 provides multiple runtime environments. Nowadays, Java, JavaME, and C-based runtime environments are supported by sharing a common base, while their differences are managed by SOFA 2 *profiles*.

The common runtime environment defines a concept of *SOFAnode*, which consists of a dedicated *component repository* and multiple component containers called *deployment docks*. These docks reside on physical deployment nodes (e.g., computers, virtual machines). A deployment dock provides necessary infrastructure for deploying, launching, and executing components. The dock can also ensure additional capabilities and services (e.g., OSGi services, persistence, transaction management). Furthermore, the overall runtime infrastructure includes additional technical elements such as a *deployment dock registry*, which supervises running docks, and a *global connector manager*, which controls inter-docks connections and transparent distribution.

2.3 Development Process

SOFA 2 adopts the development process, which is typical for component-based applications [3] and covers all the phases of the development.

In SOFA 2 (and in CBSE in general), there are two development processes – an *individual component development process* and *system (application) development process*. In the former, the individual reusable components are designed (their interfaces) and implemented. In the latter, the application architecture is designed together by composing selected components. These two cycles meet in the *assembly phase* where the complete application is composed together. Then, during *deployment*, application's components are assigned to

deployment docks and the actual implementation of connectors is automatically generated. Finally, the application is launched and executed.

A central part of the described development process is a repository of components bridging individual development stages. In the case of SOFA 2, its implementation is automatically generated based on the meta-model and it is used for storing both the component implementation and meta-data.

2.4 Development Tools

The tool support is a crucial part of every component framework in order to allow efficient and user-friendly development of components and applications. Currently, the SOFA 2 component framework provides the following development tools:

Cushion is a command-line tool supporting all development phases. Furthermore, it enables repository management including component interchange.

SOFA IDE offers a graphical designer for SOFA 2 applications architectures. It shares the core functionality with the *Cushion* tool. However, it provides user-friendly environment and additional features including UML-based modeling, repository visualization and migration.

MConsole is a plugin to SOFA IDE for launching and monitoring SOFA 2 applications. It allows visualization of the runtime environment, monitoring, and management of executed applications.

3 SOFA 2 Availability

All the described tools, corresponding source code, and documentation are available at the SOFA 2 homepage http://sofa.ow2.org/. Furthermore, SOFA IDE is published via an Eclipse update site located at http://sofa.ow2.org/update-site/. The SOFA 2 component framework and related tools are distributed under the LGPL license. [3]

4 Tutorial

4.1 Requirements

To develop and execute a SOFA 2 application in Java, a common computer with the Java Development Kit (JDK) installed is required. To run graphical designer *SOFA IDE* and monitoring tool *MConsole*, the installation of Eclipse is required. It is recommended to download the newest version of the Eclipse workbench. [4]

[3] http://www.gnu.org/licenses/lgpl-2.1.txt
[4] http://www.eclipse.org/

The SOFA 2 binary packages including the runtime environment, repository, and *Cushion* tool can be downloaded from http://sofa.ow2.org/. To install them it is necessary to unpack the downloaded packages. To install the graphical designer *SOFA IDE* and management tool *MConsole* the best option is to use the Eclipse update site located at http://sofa.ow2.org/update-site/.

4.2 Workflow

The tutorial demonstrates development of a simple *ping-pong game* with help of SOFA IDE and Cushion tools. The starting point is a UML model of the game depicted on Figure 1. The ping-pong game is played by two players (components `Ping` and `Pong`). Each player controls a paddle and tries to return a ball. The referee controlling the game is represented by the `Game` component. To develop an executable application, the tutorial follows the SOFA 2 development process – the UML model is decomposed into components, which are implemented and assembled into a ping-pong application. The assembled application can be later deployed and executed.

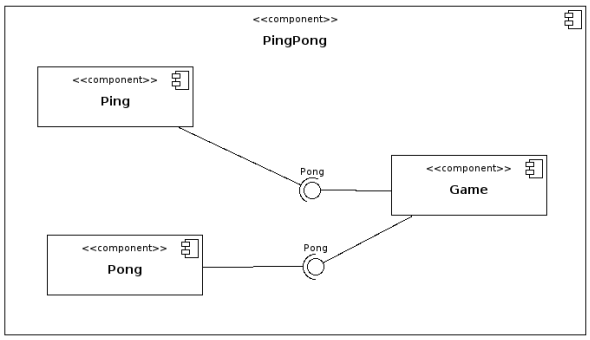

Fig. 1. UML design of the ping-pong game.

Design and Development. During the stage the UML model is decomposed into SOFA 2 components – the SOFA IDE identifies component frames and related communication interfaces, and proposes primitive (`Ping`,`Pong`,`Game`) and composite architectures (`PingPong`). A developer only needs to implement primitive components and commit all artifacts into a repository.

Assembly. Based on the UML design, SOFA IDE proposes an assembly of components. In this case, the top-level composite component `PingPong` serves as an assembly template, which is filled by particular realization of sub-components. The complete assignment is specified by an assembly descriptor.

Deployment. During the deployment, application's components are assigned to deployment docks where should be executed. Based on the assignment, the connectors are automatically generated. A deployment plan according which the application is launched, is the result of deployment.

Execution and maintenance. The last step of the development process is application execution by launching a selected deployment plan. It can be done either by the *MConsole* or by command-line tool. The result of the implemented application is shown on Figure 2.

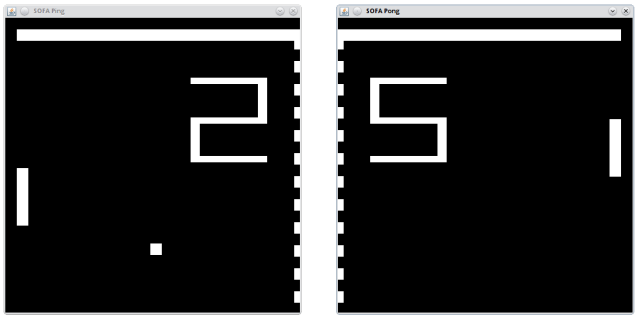

Fig. 2. The resulting ping-pong game for two players.

References

[1] Bures, T., P. Hnetynka, and F. Plasil, *SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model*. In Proceedings of SERA'06, Seattle, USA, 2006.

[2] Bures, T., and F. Plasil, *Communication Style Driven Connector Configurations*. In Lecture Notes in Computer Science, vol. 3026, pages 102–116, 2004.

[3] Crnkovic, I., M. R. V. Chaudron, and S. Larsson, *Component-Based Development Process and Component Lifecycle*. In Proceedings of ICSEA'06, Tahiti, 2006.

[4] Hnetynka, P., and F. Plasil: *Dynamic Reconfiguration and Access to Services in Hierarchical Component Models*. In Proceedings of CBSE'06, Vasteras, Sweden, LNCS 4063, 2006.

[5] Mencl, V., and T. Bures: *Microcomponent-Based Component Controllers: A Foundation for Component Aspects*. In: Proceedings of APSEC'05, Taipei, Taiwan, IEEE CS, Dec 2005.

[6] Plasil, F., and S. Visnovsky: Behavior Protocols for Software Components. IEEE Transactions on Software Engineering, vol. 28, no. 11, Nov 2002.