HighFrame: An Integrated Solution for High-Level Development and Automatic Deployment of Component-Based Distributed Systems

Felipe Oliveira Carvalho
Pontífica Universidade Católica do Rio de Janeiro
Rio de Janeiro - Rio de Janeiro - Brasil
fcarvalho@inf.puc-rio.br

Sandra Costa Pinto Hoenstch Alvarenga
Instituto Federal de Sergipe
Aracaju - Sergipe - Brasil
sandra.costa@ifs.edu.br

Saulo Eduardo Galilleo Souza dos Santos
Instituto Federal de Sergipe
São Cristóvão - Sergipe - Brasil
saulo.galilleo@ifs.edu.br

Tarcísio da Rocha
Universidade Federal de Sergipe
São Cristóvão - Sergipe - Brasil
tarcisiorocha@gmail.com

Abstract

This paper presents HighFrame, an integrated solution for high-level development and deployment that aims to reduce the complexity of developing heterogeneous component-based distributed systems. The developer keeps the focus on the business of the system (generic components in POJO style + Fraclet annotations) and uses a graphical model to define the architecture of the distributed system. HighFrame performs the deployment process of the system by automatically generating the technical code for component models and remote bindings, distributing, instantiating and making the system ready for use.

Categories and Subject Descriptors C.2.4 [*Distributed Systems*]: Distributed Applications; D.2.12 [*Software Engineering*]: Interoperability and Distributed objects

Keywords Component-Based Development, Components Generation, Automatic Deployment

1. Introduction

Distributed systems have become increasingly more complex, being composed of dynamically interconnected heterogeneous parts in the composition of richer systems [1]. This has been a major challenge for the new generation of middleware designers and other solutions for the development of distributed systems.

A promising approach that has been adopted in the development of complex distributed systems is the Component-Based Software Engineering (CBSE). The use of a component model enables the development of reusable components and facilitates the construction of dynamic systems. However, despite these benefits, the adoption of this development model may represent the introduction of more complexity in the software development process.

In addition to this challenge, the development process of component-based distributed systems involves other obstacles, such as *(i)* remote communication between distributed components; *(ii)* deployment of components in distributed nodes; and *(iii)* the interconnection of components developed in heterogeneous models. These difficulties can make the task of developing a distributed system more complex, even when this system has simple functional requirements.

Given the emerging niche of distributed systems based on components, a important challenge is to reduce the development complexity of these types of systems. In this paper we present HighFrame, a tool that allows to reduce the complexity of developing distributed systems based on heterogeneous components.

Some published studies also identify the need for solutions that facilitate the development process of component-based systems. For example, a solution for the development of embedded systems with automatic generation of specific components in SaveCCM model is presented in [5]; an MDA (Model-Driven Architecture) tool for building service-oriented component-based applications in which high-level models are transformed into specific components in the OSGi component model is presented in [3]. Fraclet is presented in [4] as a standard programming model for diverse component models. However, besides these works are only limited to the generation of components, they also do not deal with the problems of scenarios involving distributed environments.

2. HighFrame

HighFrame incorporates a set of technologies which together provide a high-level development of distributed systems based on heterogeneous components. HighFrame proposes an integrated solution that includes: *(i)* component development based on generic implementations focused on the business code of an application; *(ii)* definition of the architecture of a system based on a high-level

EATIS'16, April 27–29, 2016, Cartagena de Indias, Bolivar, Colombia.

http://dx.doi.org/10.1145/

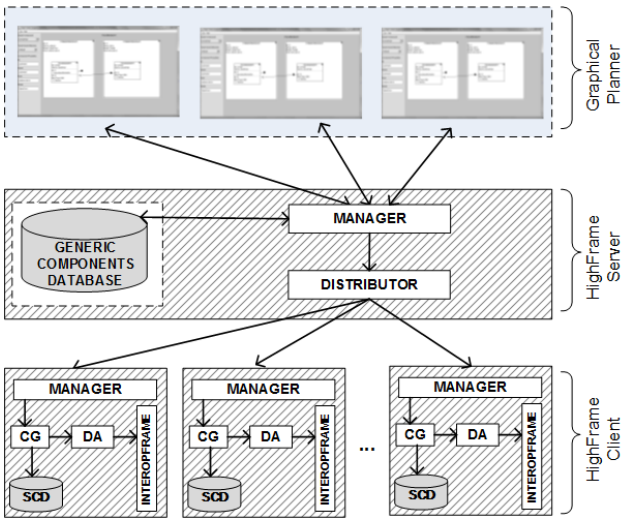

Figure 1. HighFrame Architecture.

graphical model; *(iii)* automatic deployment of the architecture in distributed nodes, providing the system in its functional form.

HighFrame architecture is fully distributed. This architecture is shown in Figure 1.

HighFrame Server provides through the Manager module a catalog of existing components on the Generic Components Database. With these components the developer can compose the architecture and the deployment plan of the distributed system. The composition process of a system can be done graphically with a graphical planner – with it the developer defines the distributed nodes, drag and drop the components to the respective nodes and creates the interconnections between them. Moreover, the developer also defines which specific component models are used in the deployment and which means of remote binding are used between them (eg. Web Services SOAP/REST, RMI). The high-level model of the system is mapped to generate the HighADL, a specific ADL of HighFrame. A deployment plan is also generated automatically with HighADL. The Manager module of HighFrame Server processes the ADL and then selects each referenced generic component to perform the distribution to the respective distributed nodes.

After that, HighFrame Client (which is located on each distributed node) uses the Component Generator (CG) module to generate the specific components in a given model (eg. Fractal, OpenCOM). Then the Component Generator triggers the Deployment Agent (DA) that makes the installation, activation and interconnection (*local binding*) of the components, and also promotes the *remote binding* between components of distributed nodes. A solution called InteropFrame [2] is used in the process of remote binding. InteropFrame was developed to provide automatic interconnection between possibly heterogeneous distributed components.

2.1 HighFrame Use

The steps to be followed for development with HighFrame are: i) development of generic components; and ii) defining the architecture through the graphical planner (HighFrame Designer). With the completion of these steps, HighFrame generates the technical code of specific components, remote distributed communication, communication between heterogeneous components and automatically performs the deployment of the distributed system.

2.1.1 Development of generic components

So that the developer can focus on the business of the system being developed, HighFrame provides an abstraction layer for the development of the involved components. This layer is based on the Fraclet annotation model [4], which deals with concerns of component-based development that are independent of any component technology. For example, developing a Java component with this model would be equivalent to developing a class in POJO (Plain Old Java Object) style including Fraclet annotations in the source code: *@Component* denotes component, *@Interface* denotes an interface of the business of the system, *@Requires* marks a required interface and *@Provides* marks a provided interface. The annotated source code represents a generic component and is ready to be inserted in HighFrame's database of generic components. This database centralizes all generic components in a server so that they become available for reuse in any new architecture. An example of the definition of a generic component is shown in Source Code 1.

Source Code 1. Generic Component Example.

```
@Component(provides = @Interface(name = "r",
signature = Runnable.class))
public class Receiver implements
Runnable {
    private final Logger log = getLogger("comanche");
    @Requires
    private Scheduler s;
    @Requires
    private Handler rh;
```

In the example from the Source Code 1, we have the definition of the component *Receiver* that provides an interface "r". This interface refers to the implementation of Java's *Runnable* class. The *Receiver* component also defines two required interfaces, the first one is "s", which requires a component that provides *Scheduler* and the second is "rh", which requires a component that provides *Handler*.

2.1.2 Definition of the architecture through the graphical planner

The process of defining the architecture of the system allows to perform various settings for the use of existing components in the database of generic components. The framework provides a high-level graphical tool, called HighFrame Designer, for the composition of an architecture. Figure 2 presents the architecture of the *Comanche Web Server*, planned with HighFrame Designer.

This process of architecture definition allows the developer to signalize the creation of components for different component technologies that will be deployed in different network nodes, communicating through one of the communication methods offered by the framework. The developer does not need to know technical details nor destine efforts to develop technical code, so he can keep the focus on the business of the application. The whole process of creating the architecture can be performed with the "drag and drop" pattern. The result of this high-level modeling are two files that are generated in XML. One is the HighADL and the other one is the deployment plan, both to be sent to HighFrame Server.

2.2 Automatic Deployment

After the process of defining the architecture, the files are sent to HighFrame Server. Manager module interprets HighADL and extracts the architecture of the distributed subsystems. Then it selects the generic components and creates the packages to be sent to the distributed nodes. These packages contain the system subarchitecture, the deployment plan and the generic components. To

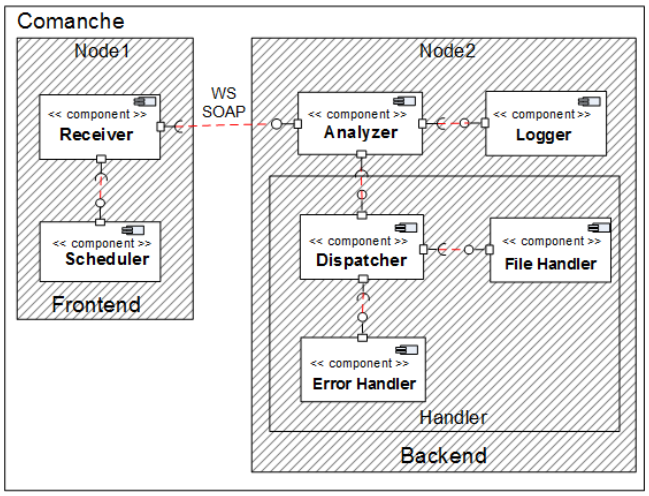

Figure 2. *Comanche Web Server* architecture.

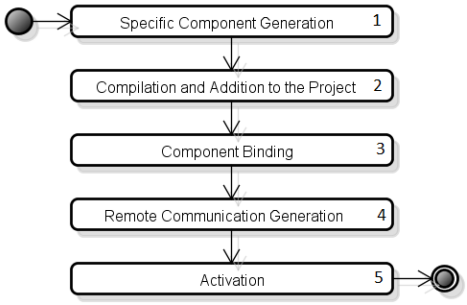

Figure 3. Activity diagram of HighFrame Client.

understand the tasks performed when a package is delivered to the HighFrame Client, Figure 3 presents a simple activity diagram.

HighFrame Client interprets the received sub-architecture and generates specific components from generic components (1). After the generation of specific components, the compilation and addition to the project step is performed at runtime (2). Thus, these components become available for use by the runtime of these component models. Then, HighFrame Client interconnects the components (3). When a connection between distributed components is detected, HighFrame Client triggers the InteropFrame module to automatically generate the technical code for distributed remote communication (4). Lastly the system is activated (5) and automatically executed.

HighFrame allows the generation of OpenCOM and Fractal components, as well as InteropFrame provides interoperability between these two component models. Furthermore, both HighFrame and InteropFrame solutions are flexible and extensible. This extensibility allows the development of new plugins (Java classes + Velocity templates) for the support of new component models and also new methods of remote communication between heterogeneous components.

3. Scenario for Evaluation

To evaluate the HighFrame proposal, we present a case study based on the *Comanche Web Server* [4], which is a simple web server composed of the following components:

- *Receiver* - Performs the incoming HTTP requests;
- *Scheduler* - Responsible for scheduling the HTTP requests for analysis;
- *Analyzer* - Performs the analysis of HTTP requests;
- *Logger* - Logs all incoming requests;
- *Dispatcher* - Responsible for the interpretation of incoming HTTP requests;
- *FileHandler* - Performs the handling of the request for a file;
- *ErrorHandler* - Responsible for handling errors.

The proposed model of the *Comanche Web Server* is being distributed in two nodes. This architecture is shown in Figure 2. In the first node is the *Comanche Frontend*, which consists of the components *Receiver* and *Scheduler*. In the second node is the *Comanche Backend*, which is composed of the components *Analyzer*, *Logger*, *Dispatcher*, *FileHandler* and *ErrorHandler*. To measure the behavior of HighFrame in a complex environment, the *Frontend* is generated in the OpenCOM model while the *Backend* is generated in Fractal. The component *Receiver* of the *Frontend* communicates with the component *Analyzer* in the *Backend* through *SOAP Web Service*. Every definition of component models, distributed nodes and methods for distributed remote communication is performed at a high-level with HighFrame Designer.

The environment in which the experiments were performed consists of two computers interconnected: One with Intel(R) Core (TM) i5 2.50GHz CPU, with 6GB of RAM and the 64-bit OS Microsoft Windows 8 Pro, running the HighFrame Server and one HighFrame Client for the *Frontend* deployment; and other one with Celeron(R) Dual Core (TM) 1.90Ghz CPU, with 2GB of RAM, and the 32-bit OS Windows 7 Pro, running the HighFrame Designer and one HighFrame Client for the *Backend* deployment. The source code was compiled and run from the development environment for Java applications *Eclipse 4.3 Kepler*, using the JVM (TM) SE Runtime Environment 1.7.

4. Results

To measure the performance of HighFrame the following metrics were defined: (i) time to generate the HighADL and the deployment plan; (ii) time for the generation of specific components; (iii) time to generate the distributed remote communication.

For the measurements each experiment was run 12 times, excluding the best and the worst result.

4.1 HighADL and deployment plan generation

This performance test measures the processing time that the HighFrame takes to generate the HighADL and the deployment plan files. These are generated by mapping the graphical model planned in HighFrame Designer into XML files. The results of this experiment are shown in Figure 4.

On the X axis of the graph from Figure 4 is the sequence number of the experiments, while on the Y axis is the time scale in milliseconds. The longest measured time was 5ms, while the smallest was 3ms. These measurements result in an average time of 4ms. These results show that the time for generation of the HighADL and the deployment plan is negligible compared to the benefits of using the tool, as this do not causes a significant impact on processing time.

4.2 Specific components generation

In this experiment we measured the time required for the creation of specific components of the *Comanche Web Server*. The scenario is composed by the *Frontend* and the *Backend* observed in Figure 2. Figure 5 shows the obtained results.

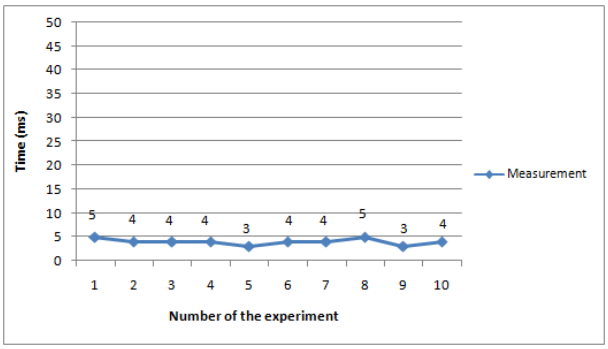

Figure 4. Time for the generation of HighADL and deployment plan.

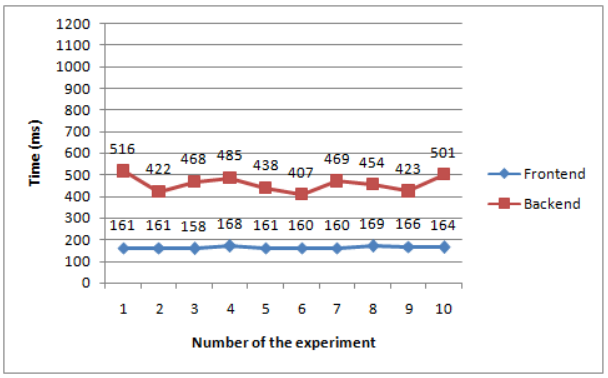

Figure 5. Time for specific components generation

In the graph of Figure 5, it is possible to see that the time spent generating components was almost constant. The average time to generate each component of the *Frontend* was 81.4 ms. The average time to generate each component of the *Backend* was 91.66 ms. The average total time to generate all the components from the *Frontend* was 162.8 ms, while for generating the *Backend* was 458.3 ms.

4.3 Remote communication generation

In this experiment we measured the time required to create and activate the components of remote communication (proxies). *SOAP Web Service* was the method used for the generation of the remote communication. The results are shown in Figure 6.

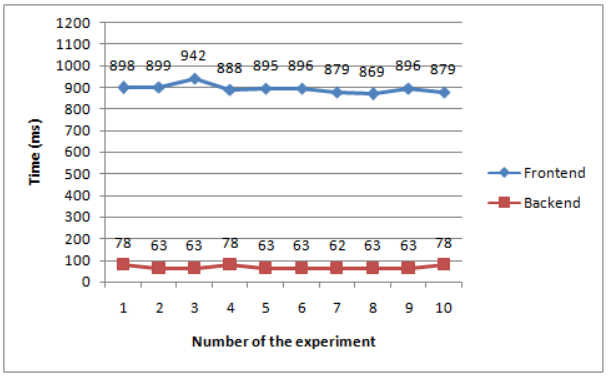

Figure 6. Time for the generation and activation of proxies using InteropFrame.

In the graph of Figure 6 is presented the measurements for the process of communication generation between two distributed remote nodes. The *Frontend* node interacts with the *Backend* to request the generation of proxies for the communication. The server side of the InteropFrame is located at the *Backend* and is responsible for setting up the SOAP Web Service server. The client side of the InteropFrame is located at the *Frontend* and is responsible for setting up the parameters for interconnection, besides triggering the server to make the connection available. The client side of the InteropFrame has more subroutines and this reflects the results obtained in Figure 6. The average time for the *Backend* to generate and activate its proxy components was 67.4 ms, while the average time in the *Frontend* was 894.1 ms.

5. Conclusions

This article presented the HighFrame, a framework in which the complexity of technical code of component models and the diversity of these, as well as the complexity of the methods for distributed remote communication and the need for interoperability between different component models were treated transparently to the developer. The proposal allows the developer to keep the concern of developing the application's business code while HighFrame generates automatically the complex technical code of development of distributed systems based on heterogeneous components.

The code annotations have been proposed to deal with the complexity of technical code for components. The graphical planner was proposed for modeling the architecture of distributed systems based on generic components, allowing the generation and communication of heterogeneous components. This allows users to make use of these resources for high-level development.

The obtained results demonstrate the feasibility of HighFrame use, since the time required for the generation of technical code, compilation and activation at runtime are considered low when compared to the manual development of these same procedures. This proposal allows to solve more broadly the challenges faced in the development of component-based distributed systems. Future works for the HighFrame will be toward assessing the usability of the graphical planner and to create templates and classes for new component models.

References

[1] G. Blair, M. Paolucci, P. Grace, and N. Georgantas. Interoperability in Complex Distributed Systems. In *11th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Connectors for Eternal Networked Software Systems*. Springer, 2011. URL `http://hal.inria.fr/inria-00629057`.

[2] S. C. do Nascimento. Um Framework extensível para interoperabilidade dinâmica entre componentes distribuídos. Master's thesis, Universidade Federal de Sergipe, 2013.

[3] N. Riba and H. Cervantes. A mda tool for the development of service-oriented component-based applications. In *ENC*, pages 149–156, 2007.

[4] R. Rouvoy and P. Merle. Leveraging component-based software engineering with fraclet. *annals of telecommunications*, 64:65–79, 2009. ISSN 0003-4347. . URL `http://dx.doi.org/10.1007/s12243-008-0072-z`.

[5] S. Sentilles, A. Pettersson, D. Nystrom, T. Nolte, P. Pettersson, and I. Crnkovic. Save-ide - a tool for design, analysis and implementation of component-based embedded systems. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 607–610, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-3453-4. . URL `http://dx.doi.org/10.1109/ICSE.2009.5070567`.