Towards Interoperability between Heterogeneous Distributed Components

Sidney Cassemiro do Nascimento
Instituto Federal de Educação, Ciência e Tecnologia de Sergipe (IFS)
sidneycn@gmail.com

Felipe Oliveira Carvalho
Departamento de Computação
Universidade Federal de Sergipe (UFS)
ocfelipe@gmail.com

Tarcisio da Rocha
Departamento de Computação
Universidade Federal de Sergipe (UFS)
tarcisiorocha@gmail.com

ABSTRACT

The popularization of middleware occurred in recent years promoted the emergence of different technological models. Because of this diversity, interoperability between different models of software components becomes essential to promote the integration of heterogeneous parts. The problems involved with interoperability are treated in general by adopting middleware systems able to mediate and establish communication between different platforms. In this context, this paper proposes the InteropFrame, a framework for interoperability between different models of software components (e.g. OpenCOM and Fractal) that aims its extensibility to support other models through the development of plugins.

Keywords

Component Models, Heterogeneous Components, Middleware, Interoperability, Distributed Systems

1. INTRODUCTION

The demand for solutions in complex distributed systems is replacing the vision of homogeneous distributed systems where domain-specific applications are developed using platforms and middleware specifically designed for this domain. Independent technological solutions have been interconnected to create even richer structures, the so-called system of systems (SoS). One of the main challenges of these interconnections is the issue of interoperability: the ability of these systems to connect, to exchange data and to communicate [3, 11, 4].

One technique that has been widely used in recent years in the development of distributed systems platforms is the Component-based software engineering (CBSE) [19]. As a result of the successful use of software components, several different component models have emerged. OpenCOM [7], Fractal [5], Spring [12], EJB [10], CCM [16] and SCA [14] are some examples of these models. The heterogeneity of component models for different domains of applications is a challenge for the development of contemporaneous distributed systems. In this context, when the parts that compose these systems are developed on platforms that use different component models, some solutions have to be adopted for the interconnection. This lack of standardization limits the reusability and the composition of components.

A software component model defines a set of standards for component implementation, naming, interoperability, customization, composition, evolution and deployment [8]. A framework is an abstraction that provides a very high degree of reuse on software components. Thus, a framework for interoperability between different models of software components can be used to promote flexibility in the development and standardization of distributed applications, allowing reuse and composition of components developed in distinct models of distributed programming.

In this paper, we propose a framework to assist software developers on the task of implementing distributed applications that use distinct component models (e.g. OpenCOM and Fractal). In addition, the proposed framework has been designed to: *(i)* allow modular extensibility to support other component models, and *(ii)* allow the developer to choose the type of communication between distributed components (RMI or SOAP Web Services [22]), also allowing extensibility to other types of remote binding (e.g. RESTFul Web Services [18] and XML-RPC [23]).

The paper is structured as follows: Section 2 presents some related work; Section 3 presents the architecture of the proposed solution; Section 4 describes the scenario used for the evaluation of the proposed framework; and Sections 5 and 6 show, respectively, the discussions on the results obtained in the experiments, and the conclusions and future work.

2. RELATED WORK

The problem of interconnection between systems based on components of different models has been addressed in some academic work. These will be highlighted: **VCF** (Vienna Component Framework) [15], **SCIRun2** [24], **Interoperability of grid component models** [13] and **FraSCAti** [21].

In the first work, it was proposed a framework developed with the Java language that allows the composition of different component models using facades of abstraction for each model. However, this paper does not discuss the distributed issue, being only a integration at the local level between the

ARM'13, December 9-13, 2013, Beijing, China

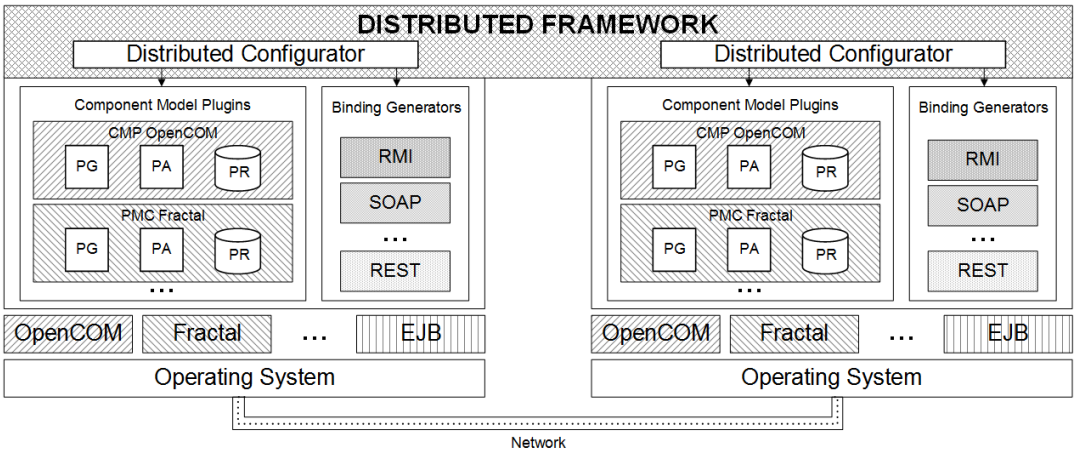


Figure 1: Architecture of the distributed framework InteropFrame.

components.

A second related work is the SCIRun2 framework [24], [17] that allows the integration of different component models into a single environment. The SCIRun2 offers a plugin architecture for component models and an approach based on bridges between them. However, the proposed model only enables interoperability between two component models: CCA (Common Component Architecture) [2] and CORBA (Common Object Request Broker Architecture) [16]. This solution supports communication among distributed components based on RMI (Remote Method Invocation) [20], [6].

In the third work, it was proposed a framework that enables interoperability between two component models for grid applications: GCM (Grid Component Model) [13] - which is based on Fractal - and CCA. The approach is based on mediators and adapters that implement the design pattern *Adapter* (*Wrapper*). However, this solution did not specify the distributed aspect, and neither it allows the connection between the components at runtime.

In the fourth work, it was proposed the FraSCAti framework. FraSCAti is a platform for hosting SCA applications and provides non-functional services such as transaction and security management. It gives runtime support for SCA with dynamic reconfiguration capabilities and runtime management features. The platform also supports: *(i)* interoperability between Java 5, Java Beans, Scala, Spring, OSGi, Fractal, BPEL and scripts based on the Java Scripting API; *(ii)* binding communication protocols such as Java RMI, SOAP, HTTP, JSON-RPC, or discovery protocols like SLP, UPNP, or integrated technologies like OSGi, JNA.

3. PROPOSED SOLUTION: INTEROPFRAME

The role of **InteropFrame** is to provide a transparent solution for interoperability between systems based on distributed components of different models, allowing the components involved in the construction of distributed applications to interact through the provided automatic mechanisms of interoperability. The approach used for this is the automatic generation of *proxies* which mediate communication between distributed heterogeneous parts.

3.1 Architecture

Figure 1 presents the detailed architecture of the distributed framework InteropFrame. Its main modules are: *(i)* **Distributed Configurator (DC)** - responsible for managing the interoperability service between distributed components, coordinating the operations of the other modules of the distributed framework; *(ii)* **Component Model Plugins (CMP)** - each plugin gives the framework the support to a different component model; and *(iii)* **Binding Generator Plugins (BG)** - each plugin is responsible for the automatic generation of source code of a different kind of binding between remote components. The CMP is composed of the following sub-modules: the **Proxy Generator (PG)** - responsible for the automatic generation of proxies that enable interoperability between heterogeneous distributed components; the **Proxy Assembler (PA)** - responsible for loading the proxies created by PG on demand and for providing the interoperability service between distributed components; and **Proxy Repository (PR)** - to manage the catalog of proxies generated by PG. The Proxy Generator can use a specific type of Binding Generator to insert into the proxy component the code that promotes the remote interconnection.

Currently, InteropFrame supports interoperability between distributed components of the OpenCOM and Fractal models, and supports the binding between remote components using the Java RMI mechanisms or SOAP Web Services. Aiming the extensibility of this framework, Component Model Plugins (CMP) and Binding Generator Plugins (BG) modules were designed to support different types of component models and bindings as completely independent plugins.

3.2 Operation

To illustrate the operation of the framework a generic problem will be taken: interconnecting two components “A” and “B” with the complicating factor that they are in different nodes of a network and they are developed in different component models, “A” in OpenCOM and “B” in Fractal. In practice, “A” and “B” can be components of two subsystems (the first one developed in OpenCOM and the second in Fractal) that need to interact, thereby forming a more complex heterogeneous system.

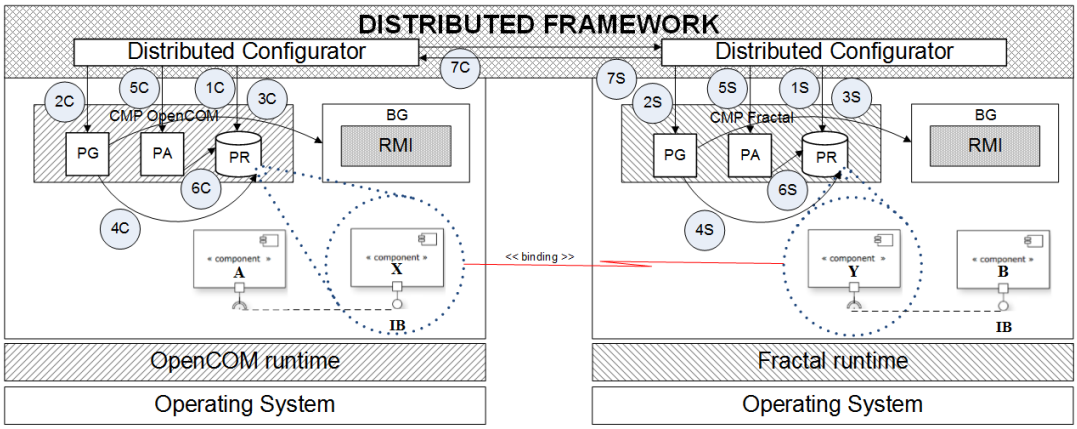


Figure 2: Diagram of the InteropFrame operation.

To promote this interconnection, the framework would receive as input from the user: the identifiers of components "A" and "B", the interface to be used in this interconnection and the type of binding to be used (RMI will be taken as example). Whereas "A" and "B" are, respectively, the client and server components, because "A" uses a service from "B", the framework would proceed with the following actions on both sides (Figure 2):

- **On the client side**

  (1C) The Distributed Configurator (DC) checks whether the client side proxy needed to promote interoperability is already in the repository, otherwise it prompts the generation of this proxy in the step 2C. If the proxy component already exists, the next step would be 5C, where this component would be used by the proxy assembler;

  (2C) The DC asks the proxy generator submodule (PG) of the OpenCOM component model to automatically generate the code of component "X" representing client side proxy;

  (3C) The PG asks the binding generator submodule (BG) based on RMI to automatically generate the code part of the component "X", responsible for remote communication;

  (4C) The PG stores the component "X" in the proxy repository (PR);

  (5C) The DC requests the proxy assembler (PA) to proceed with the initialization of the client side proxy;

  (6C) The PA obtains and initializes the client side component "X" in the OpenCOM execution environment, connecting component "A" receptacle to the provided interface of the proxy "X".

- **On the server side**

  The steps 1S to 6S performed on server side are analogous to the steps 1C to 6C on client side, whereas the generated proxy component is "Y". The server side proxy "Y" (also called skeleton) represents the component "A" on the server side that requests the services of the component "B".

Once both sides are configured, the component "A" becomes able to use the operations of the component "B" transparently. This transparency is achieved through the creation, instantiation and automatic connection of intermediate components "X" and "Y" that are responsible for propagating through network the calls and responses of the interactions between the components "A" and "B". In practice, "X" and "A" are components of the same model, so they can connect directly. Components "Y" and "B" are also of the same model. "A" requests to "X" what it would request to "B", as "X" has the same interface as "B". "X" receives the request, forwards to "Y" through RMI. "Y", in its turn, forwards the request to "B". The response of the request follows the reverse path.

4. PROPOSED SCENARIO FOR EVALUATION

To evaluate the proposed framework, a scenario of heterogeneity was implemented, based on a composition of a Comanche Web Server [19] with a set of desired characteristics.

4.1 The Comanche Web Server

Comanche is a simple web server that offers the basic features of a web server, such as the return of static HTML pages. The seven components that compose a Comanche Web Server are described as follows: *(i)* **Receiver** - Component responsible for receiving the incoming HTTP requests, *(ii)* **Scheduler** - Component responsible for the scheduling of HTTP requests for analysis, *(iii)* **Analyzer** - Component that performs the analysis of HTTP requests, *(iv)* **Logger** - Component that records the incoming HTTP requests, *(v)* **Dispatcher** - Component responsible for the interpretation of incoming HTTP requests, *(vi)* **FileHandler** - Component that performs the request handling of a file and *(vii)* **ErrorHandler** - Component responsible for handling errors (if any exist).

The role of the *Frontend* is to receive and schedule the incoming HTTP requests, while the *Backend* is responsible for the analysis and interpretation of HTTP requests. To improve the performance of the Comanche Web Server, aiming a more homogeneous distribution of the workload in order to avoid overloading, parts of the Comanche services

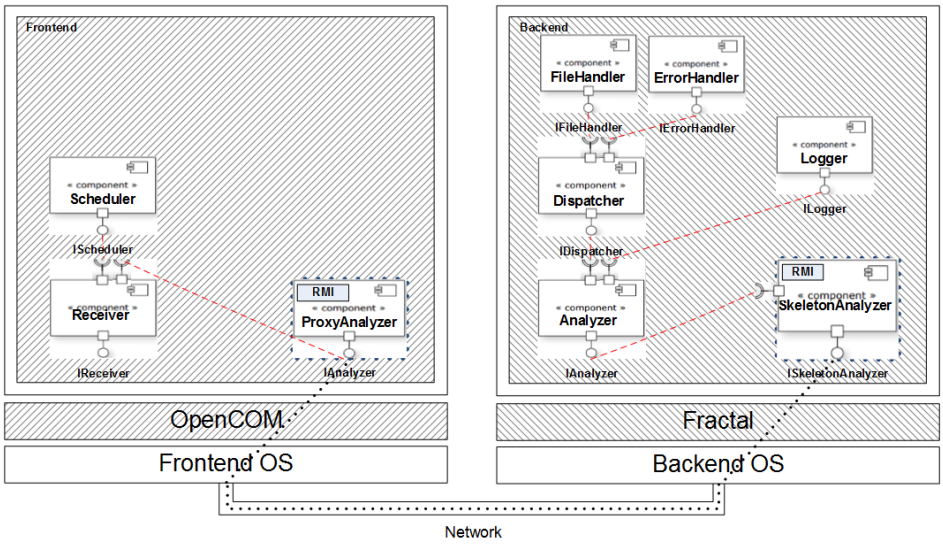


Figure 3: Comanche Web Server Architecture with distributed components.

can be distributed to other hosts. This is a scenario of load balancing use in distributed systems.

4.2 InteropFrame Use

In the experiment, a realistic scenario with two computers in the same network was used. The first computer hosted the *Frontend* and the second hosted the *Backend* of a Comanche Web Server developed with distributed components. In this scenario, the *Backend* used Comanche components available in the Fractal code base and the *Frontend* part was implemented in the OpenCOM model to simulate a heterogeneity scenario.

Figure 3 shows the *ProxyAnalyzer* and *SkeletonAnalyzer* components, which were automatically generated by InteropFrame to promote interconnection between the distributed parts of the Comanche Web Server. Whereas the *Frontend* was developed in OpenCOM and the *Backend* in Fractal, the task of the framework was to interconnect these two heterogeneous subsystems composing a single functional system.

To enable the interconnection between the OpenCOM component *Receiver* located in the *Frontend* and the Fractal component *Analyzer* located in the *Backend*, it was only needed to inform InteropFrame the identifiers of these two components, the interface used in the interconnection and the type of *binding*. After that, InteropFrame automatically generated the *proxies* that promote the transparent connection between these remote components: **ProxyAnalyzer** - the OpenCOM component that represents locally the remote component *Analyzer* and **SkeletonAnalyzer** - the Fractal component that defines a receptacle for requesting services from the *Analyzer* component.

In this scenario, the components for the Comanche Web Server are located in a local network. For this reason, the binding generator which makes use of Java RMI mechanisms for remote communication between the heterogeneous distributed components was used. Considering the same example for different networks, it was used a similar approach with the adoption of the binding generator which uses SOAP Web Services.

In this example, with the InteropFrame use, the binding between the OpenCOM component *Receiver* and the Fractal component *Analyzer* occurred in a transparent and dynamic form.

5. EXPERIMENTAL RESULTS

In this section, it will be presented the results of the evaluation of the framework in the proposed distributed scenario, in aspects of performance, usability and extensibility.

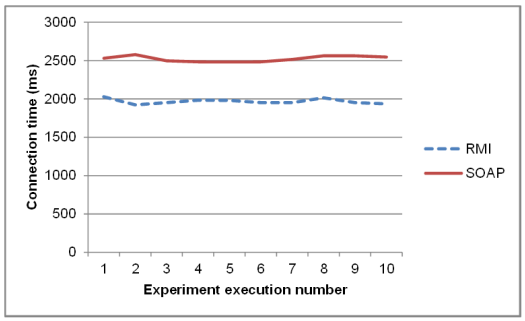
5.1 Performance Evaluation

To carry out the evaluation of performance, it was measured systematically the time that the framework took: *(i)* to connect a component to another remote one and *(ii)* to establish communication between the components, varying the component models between OpenCOM and Fractal, as well as the variation of the binding between Java RMI and SOAP Web services among these two component models.

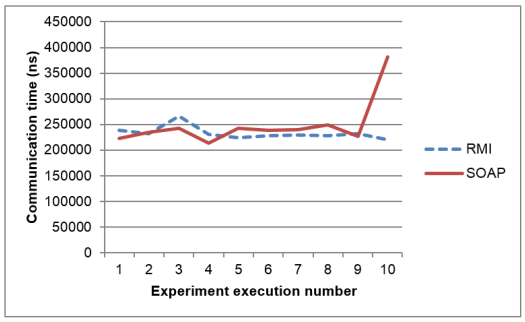
To test the influence of the bindings on performance, two experiments were implemented for each evaluated scenario, varying the intermediate mechanism of remote communication among the distributed heterogeneous components between Java RMI and SOAP Web Services.

In the experiments, for each parameter variation in the performance evaluation, each set of tests was run 11 times to allow an analysis of the execution with a more accurate average time. It was observed that the first run presented an *outlier*, so this measured time could be discarded, as the system was not stable yet. Thus, this first value was excluded from all samples. The time was measured using two static methods of the *java.lang.System* class: *currentTimeMillis* and *nanoTime*.

To perform the performance evaluation of the framework, two computers were used: *(i) Intel (R) Core (TM) i5 CPU M450 2.40GHz* with 4GB of RAM, with the operating system *Microsoft Windows 7 Professional 64-Bit Service Pack 1* installed and *(ii) Intel (R) Core (TM) i5 CPU M2450 2.50GHz* with 6GB of RAM, with the operating system *Microsoft Windows 8 Pro 64-bit* installed. To connect the



(a) Worktime results.



(b) Workload results.

Figure 4: Test results between OpenCOM and Fractal.

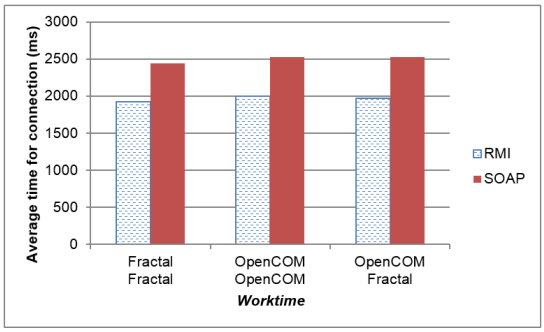


Figure 5: Comparison of the worktime means

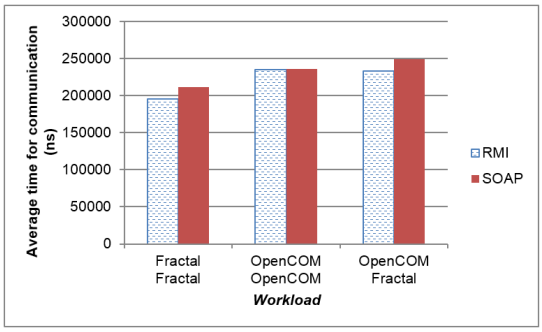


Figure 6: Comparison of the workload means

client and server machines it was used a cross cable, so the network traffic between the machines was isolated from all other traffic to avoid interference in the results. Furthermore, as far as possible, other processes/applications that were competing for network resources or running in parallel to the tests were disabled on the machines, except those necessary for the correct operation of the operating system. In order to perform the tests, all the source code was compiled and run from the *Eclipse Juno IDE*, using the virtual machine *Java (TM) SE Runtime Environment (build 1.7.0_11-b21)*.

To test the performance, three tests were made: *(i)* the first performance test used a scenario with homogeneous components in the frontend and backend using the Fractal component model, *(ii)* the second performance test also used a homogeneous scenario with all components implemented in the OpenCOM component model and *(iii)* for the third test performance it was used a heterogeneous scenario with OpenCOM components in the frontend and Fractal components in the backend. The experimental results of the third test are presented in the graphs of Figure 4, which show the same patterns observed in the experiments for each evaluated scenario.

In Figure 4(a) the results obtained in the performance evaluation of the connection between two distributed components are presented. In this graph, the X axis indicates the experiment number performed and the Y axis indicates the time required for the connection between the remote components. This experiment includes the total time for generating, compiling and loading the *proxies*. Figure 4(b) shows the results obtained in the performance evaluation of communication between remote components. In this graph, the X axis represents the number of test runs and the Y axis represents the time required for the communication.

A comparison between the average times for the connection between the remote components related to the three previous experiments are presented in the graph of Figure 5. There is an overall better performance of the experiments using the binding with RMI than with SOAP. This occurs because this process involves the instantiation and activation of proxies, which is a more expensive process using SOAP Web Services than Java RMI. Another aspect that can be observed is that the mean values do not differ greatly from one worktime to another. The graph of Figure 6 shows a comparison between the average times for communication between remote components related to the three previous experiments. A better performance when all components are implemented in the Fractal model can be observed. The performance of the communication between components using the remote binding with RMI was always better than with SOAP - this phenomena was already reported in [9].

5.2 Usability Evaluation

The framework is lightweight and its installation is quick, simple and does not require many steps, requiring only downloading and installing.

InteropFrame is available through file *InteropFrame.zip*, which contains a file *IFrame.jar* besides the required libraries in the *lib* folder for its operation. InteropFrame is available for download with its manual of installation and use in `http://computacao.ufs.br/pagina/interopframe-9682.html`.

Installing InteropFrame is simple. To use the framework is necessary to extract the contents of *InteropFrame.zip* in the root folder of the system developed in some supported component model (OpenCOM or Fractal). To configure InteropFrame for the use, the files from the *lib* folder and

the file *IFrame.jar* must be added to the *Build Path* of the project.

To evaluate the ease of use of the distributed framework and the developer effort in coding for interconnecting distributed heterogeneous components in the proposed scenario, it was examined the Source Codes 1 and 2 that demonstrate the operation of InteropFrame.

Source Code 1: Execution File *TestServerFractal.java* at server side

```
1  package comanche.fractal;
2  public class TestServerFractal {
3    public static void main(String[] args) {
4      //Fractal Server Runtime
5      ...
6      //InteropFrame Server Side Runtime
7      new DefaultServerRMI();
8      new InteropFrame(analyzerComponent);
9    }
10 }
```

The effort expended in terms of the number of source instructions (correct code lines), without errors, to integrate the generated *proxy* and *skeleton* to their respective peers in the proposed scenario (Comanche Web Server with distributed components) is small, since 6 lines are spent for the execution of the implemented solution using the distributed framework (considering lines 7 and 8 at server side, and lines 7, 8, 9 and 10 at client side).

Source Code 2: Execution File *TestClientOpenCOM.java* at client side

```
1  package comanche.opencom;
2  public class TestClientOpenCOM {
3    public static void main(String[] args) {
4      //OpenCOM Client Runtime
5      ...
6      //InteropFrame Client Side Runtime
7      IFrame iFrame = new InteropFrame();
8      iFrame.setParameters("comanche.opencom.
           Receiver", "comanche.opencom.IAnalyzer"
           , "OpenCOM", "comanche.fractal.Analyzer
           ", "comanche.fractal.IAnalyzer", "
           Fractal");
9      iFrame.setParametersBinding("RMI");
10     iFrame.remoteBinding(opencomRuntime);
11     //Execution of Client Components
12     ...
13   }
14 }
```

It could be observed that the process of automatic generation of *proxies* to interconnect the OpenCOM component *Receiver* to the Fractal component *Analyzer* in the proposed evaluation scenario was simplified through the automatic mechanisms for interoperability provided by the proposed framework. This level of automation allows developers to reduce coding time and effort to promote the interconnection of the heterogeneous distributed components.

5.3 Extensibility Evaluation

InteropFrame allows the extension to new component models through the development of new plugins. The developer that needs to extend InteropFrame needs to know in details the operation of the desired component model, besides following some standards established in the creation of the InteropFrame framework, like the tool to generate code from templates (*Apache Velocity* [1]) and other technologies used.

Support for extensibility is done natively in the framework architecture, enabling new component models (e.g. EJB, OSGi) and new types of bindings (e.g. RESTful Web Services , XML-RPC) to be supported by it through the development of new plugins. It is possible to observe that the extension or modification of any InteropFrame functionality is a task that requires knowledge of the implementation architecture and technologies used.

6. CONCLUSION

Despite the evolution of *middleware* technologies with the adoption of component models, it appears that the area of distributed systems presents several barriers for the interoperability between them, and the heterogeneity of *middleware* platforms is a challenge for distributed programming based on components from different models.

Developing software systems for distributed environments that combine different component models is a difficult task. The problem of interoperability between heterogeneous platforms is very complex because it involves dealing with issues related to the differences in distribution models (e.g. a platform can be Object Oriented and another can be a Message Oriented one), session protocols (e.g. a Request-Reply protocol or a General Inter-ORB Protocol), diversity of services, among others. In this scenario, the main contribution of this paper is the proposal of the InteropFrame framework, which has as differential its plugin architecture for component models and an approach based on the generation of proxies to promote interoperability between different models of distributed components. Also, InteropFrame's extensible architecture gives support to other component models and other types of bindings through the development of plugins.

The implemented distributed framework may serve as a basis for further and broader research on interoperability between component models, as well as to develop other mechanisms for interoperability between different platforms.

For future work, some opportunities can be identified, including: *(i)* the extension of the InteropFrame to support components developed in different languages (the current solution is for Java components), *(ii)* the development of plugins for other component models (such as EJB and OSGi) and plugins for other types of bindings (such as RESTFul Web Services and XML-RPC), *(iii)* the migration of the InteropFrame platform to OSGi and the evaluation of possible performance loss of the framework caused by the use of OSGi, and *(iv)* to design and implement a version of InteropFrame for devices with limited processing power, limited computing resources and equipped with wireless communication, such as mobile phones.

7. ACKNOWLEDGEMENTS

The authors thank CAPES and COPES/UFS for the financial support to the development of this research.

8. REFERENCES

[1] Apache. Velocity Template Engine (Velocity).

[2] R. Armstrong, G. Kumfert, L. C. McInnes, S. Parker, B. Allan, M. Sottile, T. Epperly, and T. Dahlgren. The CCA component model for high-performance scientific computing. *Concurrency and Computation: Practice and Experience*, 18(2):215–229, 2006.

[3] G. Blair, M. Paolucci, P. Grace, and N. Georgantas. Interoperability in Complex Distributed Systems. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems*, volume 6659 of *Lecture Notes in Computer Science*, pages 1–26. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-21455-4_1.

[4] Y.-D. Bromberg, P. Grace, L. Rï£¡veillï£¡re, and G. S. Blair. Bridging the interoperability gap: Overcoming combined application and middleware heterogeneity. In F. Kon and A.-M. Kermarrec, editors, *Middleware 2011*, volume 7049 of *Lecture Notes in Computer Science*, pages 390–409. Springer Berlin Heidelberg, 2011.

[5] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The FRACTAL component model and its support in Java. *Software: Practice and Experience*, 36(11-12):1257–1284, 2006.

[6] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair. *DISTRIBUTED SYSTEMS: Concepts and Design.* Addison-Wesley, Boston, 5 ed. edition, 2011.

[7] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan. A generic component model for building systems software. *ACM Trans. Comput. Syst.*, 26(1):1:1–1:42, Mar. 2008.

[8] I. Crnkovic, J. Stafford, and C. Szyperski. Software Components beyond Programming: From Routines to Services. *IEEE Software*, 28:22–26, 2011.

[9] C. Demarey, G. Harbonnier, R. Rouvoy, and P. Merle. Benchmarking the round-trip latency of various java-based middleware platforms. *Stud. Inform. Univ.*, 4(1):7–24, 2005.

[10] L. DeMichiel and M. Keith. JSR 220: Enterprise JavaBeans. Sun Microsystems, Santa Clara, 3.0 edition, 2007.

[11] P. Inverardi, V. Issarny, and R. Spalazzese. A theory of mediators for eternal connectors. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6416 of *Lecture Notes in Computer Science*, pages 236–250. Springer Berlin Heidelberg, 2010.

[12] R. Johnson, J. Hoeller, A. Arendsen, C. Sampaleanu, R. Harrop, T. Risberg, D. Davison, D. Kopylenko, M. Pollack, T. Templier, E. Vervaet, P. Tung, B. Hale, A. Colyer, J. Lewis, C. Leau, M. Fisher, S. Brannen, R. Laddad, and A. Poutsma. The Spring Framework - Reference Documentation, 2.5.6 edition. 2008.

[13] M. Malawski, M. Bubak, F. Baude, D. Caromel, L. Henrio, and M. Morel. Interoperability of Grid Component Models: GCM and CCA case study. In T. Priol and M. Vanneschi, editors, *Towards Next Generation Grids*, pages 95–105. Springer US, 2007.

[14] OASIS. Service component architecture (SCA) assembly model specification. OASIS Open, 1.1 edition, 2011.

[15] J. Oberleitner, T. Gschwind, and M. Jazayeri. The Vienna Component Framework enabling composition across component models. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 25–35, Washington, DC, USA, 2003. IEEE Computer Society.

[16] OMG. OMG CORBA component model (CCM) specification. OMG, Needham, 3.0 edition, 2002.

[17] S. Parker, K. Zhang, K. Damevski, and C. Johnson. Integrating Component-Based Scientific Computing Software. In P. R. M.A. Heroux and H. Simon, editors, *Parallel Processing for Scientific Computing*, pages 271–288. SIAM Press, 2006.

[18] L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach, 4rd Edition.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[19] R. Rouvoy and P. Merle. Leveraging component-based software engineering with Fraclet. *Annals of Telecommunications*, 64:65–79, 2009. 10.1007/s12243-008-0072-z.

[20] B. Ruixian. Distributed computing via rmi and corba. 2000.

[21] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, and J.-B. Stefani. A component-based middleware platform for reconfigurable service-oriented architectures. *Software: Practice and Experience*, 42(5):559–583, 2012.

[22] W3C. SOAP Specification, 2012.

[23] D. WINER. XML-RPC Specification (2003), 1999.

[24] K. Zhang, K. Damevski, and S. G. Parker. SCIRun2: A CCA framework for high performance computing. In *In Proceedings of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS*, pages 72–79. IEEE Press, 2004.