FLAVOR: A dynamic and open framework for the development of network measurement access and visualization tools

Ivo K. Koga[1], Leobino Sampaio[1,2], José A. S. Monteiro[1]

[1]Computing and Networking Research Group – NUPERC
Universidade Salvador (UNIFACS)
Rua Ponciano de Oliveira, 126 - Rio Vermelho
41950-275 – Salvador – BA – Brazil

[2]Informatics Center – CIn
Universidade Federal de Pernambuco (UFPE)
50732-970 – Caixa Postal 7851 – Recife – PE – Brazil

{ivo.koga, leobino, suruagy}@unifacs.br

Abstract. *This paper presents the development of a dynamic and open framework for network measurement access and visualization tools. This framework has been developed under the concept of making network measurement data access easier, providing an extensible way to develop end user tools. This is shown from a high-level and component-like point of view, explaining the nature of the components and how they are integrated into the framework. The ICE tool has been developed using the framework in order to validate this idea.*

1. Introduction

Network monitoring can provide information about the state of the network, helping network managers make decisions in order to maintain and optimize the operation of the provided services among other benefits. In this way, the monitoring activity should make network performance data available to the users, preferably in an easy way and related to the complete end to end path (i.e., including information about all the intermediary nodes).

In order to achieve this goal, some initiatives started to build measurement infrastructures that deal with the problems of network measurement data access from different network domains. Some of these efforts are the End to End performance initiative (E2Epi)[1] from Internet2, the Géant 2 Joint Research Activity 1 (JRA1)[2] and the Measurement Working Group[3] from the Brazilian National Research and Education Network (RNP). All these initiatives have provided network monitoring services that make easier the access to network monitoring data.

Beyond the availability of network monitoring services, there is also a need for flexible and easily customizable tools capable of accessing and processing the monitoring information that comes from different network services in different domains. An example of an effort that provides a dynamic and scalable system is MonALISA (Monitoring

[1]http://e2epi.internet2.edu
[2]http://www.geant2.net/server/show/nav.754
[3]http://wiki.nuperc.unifacs.br/

Agents in A Large Integrated Services Architecture)[4]. MonALISA is a scalable Dynamic Distributed Services Architecture that provides "*the ability to each service to register itself and then to be discovered and used by any other services*" [Legrand et al. 2004]. With this capability, the user can get the data from a measurement data access service and visualize them using, for example, MonALISA's GUI.

But visualizations sometimes need to be restructured or adapted for different audiences, details may be added or taken away, variables could be composed into the same visualization or translated into another. There are also some cases where users want to extend the functionalities of the current monitoring tools. This could be achieved by reusing the available network monitoring application code that serve as the basis for the visualization and analysis capabilities without having to "reinvent the wheel".

That was the major motivation for this work: the possibility of reusing the available code and make it available to the users so that it can be plugged into the user's own network monitoring application. Everything achieved by the development of a framework that provides easier possibilities of development of flexible and extensible software components.

This paper is organized as follows: Section 2 provides information about some multi-domain nework monitoring enviroments efforts, Section 3 presents some definition of components and some available technologies for their development. In Section 4 the FLAVOR framework is presented with its details and specifications. Section 5 describes the tests of the framework and how it was validated with the development of a visualization environment called ICE. Finally, we briefly discuss some future work before drawing some conclusions in section 6.

2. Multi-Domain Monitoring Efforts

Following the interest in the availability of network monitoring data, the Internet2 initially proposed a scalable and distributed system for monitoring, testing, and reporting end to end performance, called piPEs (performance initiative Performance Environment system) [Internet2 2003]. After that initiative, Internet2 and Géant2 started a joint work where they developed a document that specified a service oriented measurement infrastructure called General Framework Design (GFD)[5]. The main goal of that infrastructure was to provide the concepts needed to perform measurements between domains, making possible the exchange of measurement information through standardized services, each one executing actions that offer necessary functionalities to provide better acessibility to the network monitoring information.

The services defined by the GFD are: Measurement Point (MP), used to collect measurement data; Transformation service (TS), used to pipeline data between the other services within the framework; Measurement Archive (MA) which stores measurement data collected by the MPs or transformed by the TS; Lookup Service (LS) used to discover and publish services enabling the insertion and query of the environment services; Authentication Service (AS) which allow the authentication and authorization in the environment and provide decision attributes for what can be shown for a given resource or

[4]http://monalisa.cacr.caltech.edu/

[5]http://wiki.perfsonar.net/jra1-wiki/images/9/95/GN2-05-057v5.pdf

service; Topology Service (TopS) that provides information about the network topology. The development of a prototype called perfSONAR (Performance focused Service Oriented Network monitoring ARchitecture) [Hanemann et al. 2005] was started following these defined services.

Boote et al. emphasized that its implementation "*will follow a consistent approach that respects the multi-domain organization of the networking environment and identified user requirements*" [Boote et al. 2005] and will serve for the accomplishment of consistency tests of the GFD. This implementation aims initially at the development of the basic and simpler services of the infrastructure like the MP, MA and the LS.

Beyond the perfSONAR initiative, the Brazilian National Research and Education Network (RNP) Measurement Working Group developed and deployed an environment called piPEs-BR [Sampaio et al. 2006] which follows piPEs' principles. Currently, the Measurement WG tries to adjust the piPEs-BR with the development of an environment called piPEs-BR/GFD which uses end to end performance monitoring tools following GFD in a joint development of perfSONAR services.

All these efforts have the goal to provide important network monitoring data to the users, problem prevention and detection and give information that is currently unavailable or is much difficult to access like performance information between different administrative domains.

The infrastructure of the constructed environments gives a layer of services using the SOA architecture that intends to abstract the internal differences of each component, that is, abstract the interaction and availability of the provided data and opens a perspective to the creation of applications that interact with it.

3. Component-based development

The concept of building software components is not new [Crnkovic 2001]. The reuse approach to software development has been used for many years, but this alone does not lead to a consistent agreement about what software components are, probably because the term is used to denote many different things. However, there is a general agreement on what constitutes software component technology.

D'Souza outlines that components are "*A coherent package of software artifacts that can be independently developed and delivered as a unit and that can be composed, unchanged, with other components to build something larger.*" [D'Souza and Wills 1999]. Another definition that is very frequently used comes from Szyperski which defines a software component as "*... a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.*" [Szyperski et al. 2002].

So, components are closed and well defined pieces of software which have some functionalities inside, can be composed and should be developed in any programming language. It can be deployed as a black box and also have an external specification, which is independent of its internal functionalities.

Developers benefits from the flexibility, reusability, easy maintainability of those components technologies. All these can be achieved using a component-based development (CBD) which is a software development approach in which all artifacts can be

built by assembling, adapting, and connecting together existing components into a variety of configurations. Using this component-based development approach, developers can improve the development process and provide components to be used by many other projects, systems or tools benefiting all the computer network community.

There are many initiatives to provide component infrastructure technologies. Among others there are: The Object Management Group's (OMG's) Common Object Request Broker Architecture (CORBA) [Siegel 1998], Microsoft Component Object Model (COM) [Gray et al. 1998], Sun's Java-based distributed component technology (EJB[6] and JavaBeans[7] standards) and the Open Services Gateway Initiative (OSGi)[8]. All of these technologies promises processes to speed up application development.

3.1. OSGi

The Open Services Gateway Initiative (OSGi) established in 1999, is an independent, non-profit corporation working to define and promote open specifications for the delivery of managed services to networks in homes, cars, and other environments. The OSGi is composed of leading device manufacturers, software suppliers, gateway operators, and service providers.

The OSGi specifications define a standardized, component oriented, computing environment and provide a general-purpose, secure, managed Java framework that supports the deployment of extensible and downloadable service applications known as bundles [Marples and Kriens 2001]. Bundles are jar files which contain Java classes and other resources that provide functions to the end-user and, optionally, to other bundles. They contain the resources to implement zero or more services which are the exported capabilities of the bundles. They contain a manifest file that describes the JAR contents and provide information about the bundle, states dependencies on other resources, such as Java packages that must be available to the bundle before it can run, and designates a special class in the bundle to act as a bundle activator.

The OSGi framework, as depicted in Figure 1, sits on top of the Java runtime environment and allows the OSGi-compliant devices to download, install, update and remove bundles when they are no longer required. These bundles can access capabilities in the framework, the underlying virtual machine (VM), and the operating system as required using the Java Native Interface (JNI)[9] technology. The installed bundles can share resources with other bundles and also import and export Java packages under strict control of the framework which could be asserted between the parts involved using signed Java jar files.

The OSGi bundles lifecycle is depicted in Figure 2 and shows that when a bundle is installed it goes to a resolved state where the framework checks the deployment manifest for dependencies on external Java packages. Once it is resolved, the framework is free to use the Java packages this bundle provides for resolving other bundles. After that, the resolved bundle can be activated.

[6]Enterprise JavaBeans - http://java.sun.com/products/ejb/index.jsp
[7]http://java.sun.com/products/javabeans/index.jsp
[8]http://www.osgi.org/
[9]http://java.sun.com/j2se/1.4.2/docs/guide/jni/

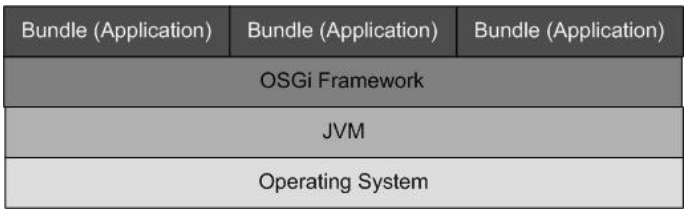

Figure 1. OSGi Service Platform Components Relationship.

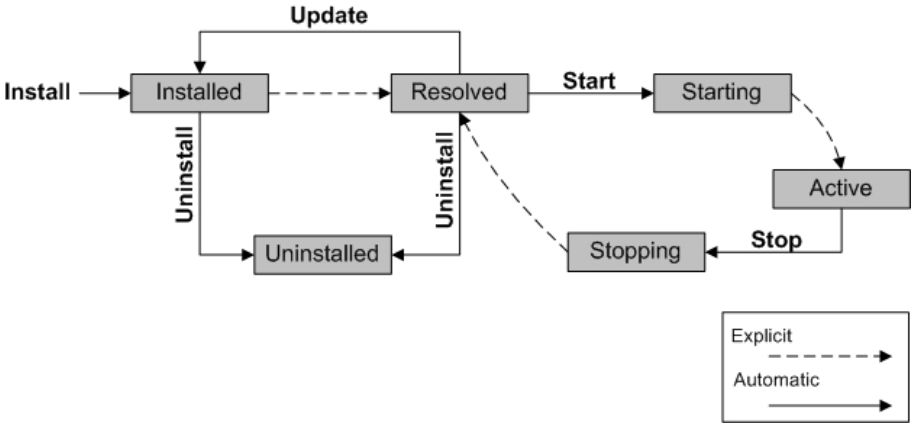

Figure 2. OSGi Bundles Lifecycle.

The activation results in the creation of an instance of the activator class referenced in the deployment manifest. The activator class implements an interface with two methods: one is called when the bundle is activated, and the other when the bundle is deactivated. These methods receive a context object that gives the bundle access to framework functionalities, such as accessing other bundles, performing bundle management operations, registering services, looking for other services, and registering itself as a listener to different types of events [Hall and Cervantes 2004].

The OSGi technology provides the standardized primitives that allow applications to be constructed from small, reusable and collaborative components. This framework manages bundles installation and update in a dynamic and scalable fashion, and manages the dependencies between bundles and services. It provides the developer with the necessary resources to take advantage of Java's platform independence and dynamic code loading capability in order to easily develop, and deploy on a large scale, services for small-memory devices [Marples and Kriens 2001].

For those reasons, the OSGi framework seems to be very useful to the implementation of the proposed Framework Layer for Access and Visualization Of network monitoring Resources (FLAVOR) where dynamicity, flexibility and resources management are needed.

4. The FLAVOR Framework

Network monitoring development teams can benefit from the FLAVOR framework by using its basic infrastructure for a fast and reliable development. FLAVOR provides ways to implement OSGi components that can be reused and assembled together in many combinations and in different contexts.

4.1. FLAVOR overview

The FLAVOR is composed of a set of Java interfaces and some developed OSGi network monitoring bundles. It allows the user develop his/her bundles implementing one of those interfaces showed in Figure 3. It also presents the FLAVOR hierarchy interfaces structure where the root interface is the *FlavorPlugin* which has the basic functionalities that a bundle has to have to be considered a FLAVOR Plugin.

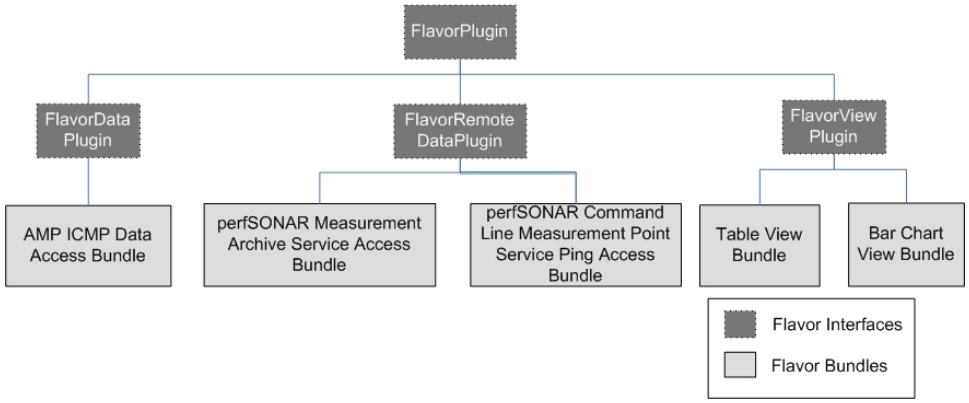

Figure 3. FLAVOR Interfaces and Bundles.

Under that interface, the *FlavorDataPlugin* is the one that provides methods that deal with local data access like the access of a local text file, a local database or any other local data source. The *FlavorRemoteDataPlugin* interface is also a child of *FlavorPlugin* and provides methods to handle remote data sources like the access of a Web Service or any other remote access.

The FlavorViewPlugin deals with the basic functionalities of a visualization plugin. It provides methods to handle the operations of visualization like opening a window inside an application. The FLAVOR framework also provides classes that implement a desktop and internal frame to do the development of the visualization bundles.

Under these interfaces mentioned before, there are the implementations of several bundles such as the AMP ICMP Data Access Bundle that provides access to ICMP measurement data from the text files of the AMP measurement tool[10] and is an implementation of the FlavorDataPlugin interface. The perfSONAR Measurement Archive Service Access Bundle that provides access to a measurement archive services from perfSONAR monitoring environment [Hanemann et al. 2005]. There is also the perfSONAR Command Line Measurement Point Service Ping Access Bundle which provides access to the ping measurement data from any Command Line Measurement Point Service[11] from perfSONAR. The latest two are implementations of the FlavorRemoteDataPlugin interface. Finally, the visualization example bundles are the Table View Bundle that provides data visualizations in a tabular view and the Bar Chart View Bundle that provides data visualizations in a bar chart view and implements the FlavorViewPlugin interface.

All these developed bundles are used on top of the FLAVOR framework which sits on top of the OSGi framework as we can see in Figure 4. Therefore, the FLAVOR

[10]http://amp.nlanr.net/

[11]http://wiki.perfsonar.net/jra1-wiki/index.php/CLMPService

framework uses the OSGi capabilities and provides useful services to the development of network monitoring applications.

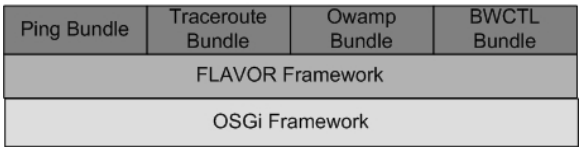

Figure 4. FLAVOR Bundles.

4.2. FLAVOR approach

After the definition of the interfaces, implementation and packaging of the network monitoring OSGi bundles, FLAVOR can be used and recognized by any tool that has an OSGi implementation support. This can be achieved when the application developer embed one of the available implementations of the OSGi framework and then integrates the FLAVOR framework inside his application. First the OSGi framework will provide the functions to manage the bundles lifecycle. The installation, search, update and removal of that bundles will be done by this OSGi framework without much development effort as it was seen in section 3.1. FLAVOR then will provide the interfaces and bundles to the developer for its own use and development.

This interaction is showed in Figure 5 where one Network Monitoring Visualization tool installs some bundles in the left side, and other tools in other environments and contexts could also benefit from this framework by using the developed bundles, which is represented in the right side of the picture. So, the developed bundles could be used by any application that makes use of the FLAVOR framework, by just embedding an OSGi framework, putting the FLAVOR Framework library in the classpath of the application and using some of the required classes like the FlavorDesktopPane which extends the JDesktopPane and provides functions to add a new window and organize the windows in the desktop or the developed monitoring access and visualization bundles.

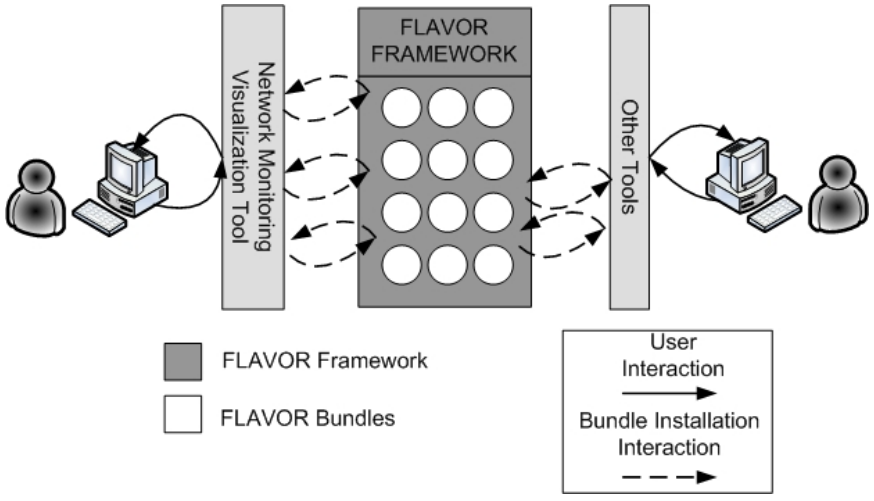

Figure 5. FLAVOR Architecture.

Another issue that has to be mentioned is that users also have the possibility to extend FLAVOR by developing their own bundles using the interfaces showed before. These

developed bundles can also be used in any tool that provides the FLAVOR framework plugins functionality. Moreover, if necessary, users can modify the framework interfaces and plug it into the framework by using its open source code, but if the interfaces are modified, it will only be visible to others by using these new interfaces inside the FLAVOR framework.

Figure 6 illustrates the steps necessary to develop and benefit from this approach. In step one, user 1 developed a bundle to access some monitoring data to his/her own use and makes it available. Another user can develop a visualization or improve them developing another bundle. If this second user makes the visualization bundle available, a third user that only wants to access the services, just have to download the bundles to his/her tool and use them without any development.

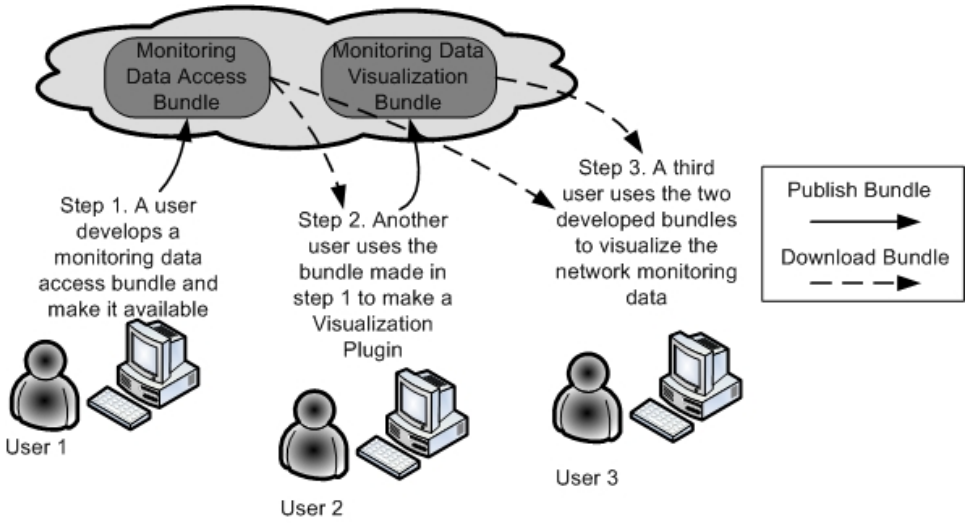

Figure 6. FLAVOR development cycle.

These monitoring data bundle can be developed by just extending the RemoteServiceDataSource abstract class from the FLAVOR framework which has some standard methods already implemented (like the start and stop methods from the OSGi activator interface) or directly implementing the FlavorDataPlugin or FlavorRemoteDataPlugin interfaces. The following example of code shows the class declaration of the perfSONAR Command Line Measurement Point Service Ping Access Bundle implementation that extends the RemoteServiceDataSource abstract class which implements the FlavorRemoteDataPlugin:

```
public class CLMPPingBundle extends
br.unifacs.nuperc.flavor.api.data.impl.remote.RemoteServiceDataSource
```

This bundle has to be package in a jar file with at least this class and a manifest file that contains metadata needed by the OSGi framework for using the bundle. It describes the implementation and provides information about dependencies and the Activator class. The manifest file of the CLMPPingBundle is showed below:

```
Bundle-Activator: br.unifacs.nuperc.ice.osgi.perfsonar.clmp.ping.CLMPPingBundle
Export-Package: br.unifacs.nuperc.ice.osgi.perfsonar.clmp.ping
Bundle-Classpath: .
Import-Package: org.osgi.framework, br.unifacs.nuperc.flavor.api.data.impl,
br.unifacs.nuperc.flavor.api, javax.swing,org.jdom, org.jdom.input, org.jdom.output,
```

```
javax.xml.rpc, org.apache.axis.client,javax.xml.namespace, javax.xml.parsers,
org.xml.sax, org.apache.axis.message,
org.w3c.dom,br.unifacs.nuperc.ice.osgi.perfsonar.clmp.ping
Bundle-Name: CL-MP Ping Data Bundle
Bundle-Description: A bundle that registers a perfSONAR Command Line Measurement Point
Service Ping Access
Bundle-Version: 1.0.0
```

The Bundle-Activator item points to the implementation of the Activator which in this case is the CLMPPingBundle class. The Export-Package defines the packages to be exported. The Bundle-Classpath defines the classpath of the bundle, and the Import-Package defines the names of any package that the developer wants to import from the runtime environment. The other items are just for human consumption and do not affect the OSGi framework, describing the bundle just for user information.

After implementing this class and packaging it with the described manifest file in a jar file, this bundle can be installed in any tool that is "FLAVOR ready" providing the services implemented and exported by this bundle.

5. Validating the FLAVOR Framework

The users of the piPEs-BR environment initially used some utilities to make its visualization. It was adopted RRD-Graphs, Gnuplot, PHP and Perl scripts to generate charts of the data retrieved from the measurement tools. After that, it was used the MonALISA framework, that accessed a centralized measurement database through Web Services to make the visualization of determined measured data using the MonALISA GUI. However, all these tools were not integrated. In some cases to make the visualization of different measurement data it was also necessary to execute different visualization applications and it wasn't possible to develop customized and flexible visualizations for the users.

In order to solve this problem, the ICE (Internet Computer network Eye) environment[12] has been developed since 2005. It had the goal to integrate several visualizations into the user's environment and access the features that the piPEs-BR network monitoring environment provided. It has been developed using Java, JNI, Apache Axis[13] SOAP implementation, a chart library called JFreeChart[14] and allows the users to query in an integrated way the network monitoring services from piPEs-BR.

During the ICE development it was noted the need for a more flexible and reusable development approach. The code of the application could not be reused inside the application in other contexts or in other tools. For that reason the FLAVOR framework has been designed and used in the development of the newer ICE functionalities in order to validate the framework and also to provide the components technology to the ICE environment.

5.1. Rebuilding ICE

To demonstrate how ICE was restructured to incorporate the FLAVOR functionalities, the diagram in Figure 7 shows the steps needed to reach this goal. In step 1 ICE embedded the OSGi framework implementation called Felix[15]. In step 2 the FLAVOR framework

[12]http://wiki.gt-med.ufsc.br/ice/
[13]http://ws.apache.org/axis/
[14]http://www.jfree.org/jfreechart/index.php
[15]http://cwiki.apache.org/FELIX/index.html

library was put in the ICE classpath to provide a way to use the FLAVOR classes. In step 3 ICE used some available developed classes from FLAVOR like the desktop and plugin manager that makes easier the user interaction with the environment. Finally, in step 4 ICE installed some FLAVOR bundles for its use.

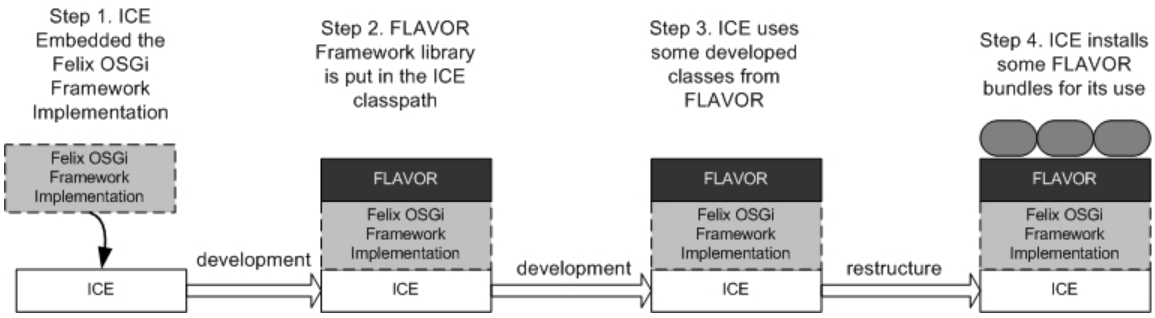

Figure 7. ICE Rebuilded.

With this new approach, the interaction with ICE was changed and is showed in Figure 8. First, in Step 1 the ICE environment is initialized where all the variables and the OSGi framework implementation are started. In Step 2 the user gets some available bundles and requests the installation of those bundles into ICE. Finally, in Step 3, the user just interacts with the installed bundles requesting and visualizing the measurement data. The next time ICE initializes, the OSGi framework looks for the installed bundles and initializes all of them without the need of any human interaction.

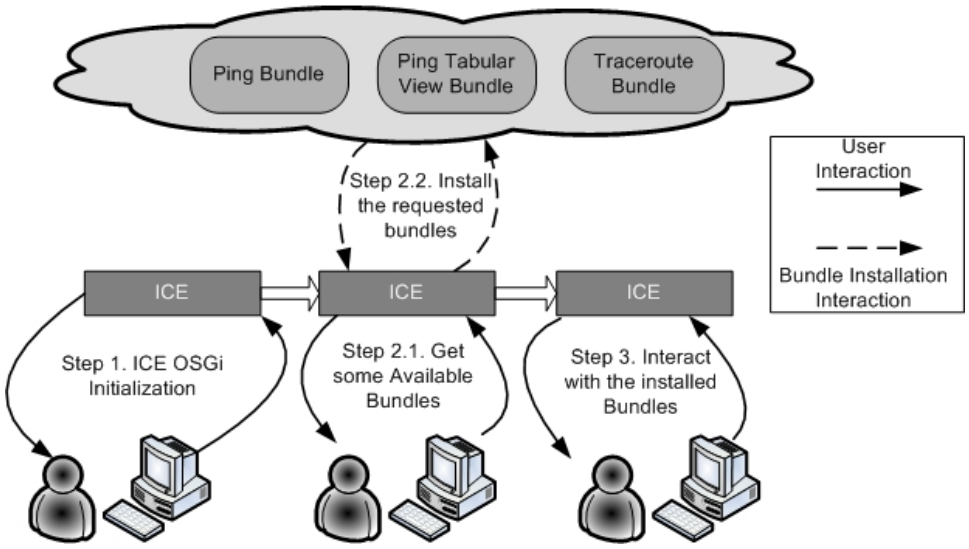

Figure 8. How ICE can be used using FLAVOR Framework.

Now all the action to request and visualize the measurement data are done with the installed bundles inside the framework. Figure 9 shows the ICE architecture where it's possible to install or remove visualization and data bundles to access services or data from different sources (remote network measurement data sources or local data sources). With this, the user can assemble the bundles in the ICE environment and use them when required. This approach defines possibilities to use one data access bundle for many

data visualization bundles or many data access bundles can be accessed by just one data visualization bundle which can compose this data inside one visualization screen.

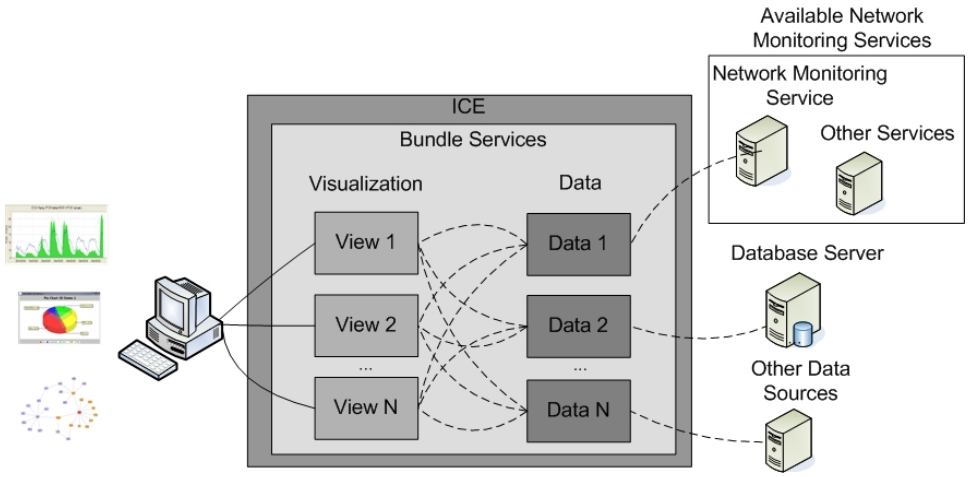

Figure 9. ICE architecture.

5.2. ICE usage scenario

Initially the ICE tool were used to access RNP's backbone measurement infrastructure which is shown in Figure 10. This infrastructure has four measurement points (MPs) deployed at the cities of São Paulo, Rio de Janeiro, Florianópolis and Salvador where initially the perfSONAR CLMP Measurement Point services and the AMP measurement tools were deployed. The metrics available by these installed measurement services and tools were: one-way delay, round-trip delay and traceroute.

With those metrics available, ICE had access to them by requesting and providing visualizations to the services and tools in static frames to do the requesting of measurement data and some charts to view the measurement data. After the development of FLAVOR, some bundles were developed and installed in ICE to handle those available metrics. These bundles provided flexibility in the development of new visualizations, as we could develop many views using the same installed data access bundle and other services that needed the access of the measured data could also install and use the developed data access bundles.

Figure 11 shows the use of FLAVOR by showing in the left side the window of the Tabular View Ping Plugin which is a visualization plugin that can make the visualization of the measurement data in a table view. The window on the right side is the Plugin Manager which is a class of the FLAVOR framework and controls the installed plugins in the ICE environment by providing the needed interaction with the OSGi framework functionalities. This Plugin manager is showing the installed bundles: Tabular View Ping Plugin, a visualization bundle and the Ping Plugin which is a data access bundle.

The integration with FLAVOR and the restructuring of ICE provided the flexibility needed to implement bundles that could be reused by any application and showed easy ways to develop other new functionalities inside this approach of reusing the developed code from bundles already implemented. The developed code from ICE has been restructured to follow this new approach and specification and other tools and applications could do the same to benefit from the FLAVOR framework functionalities.

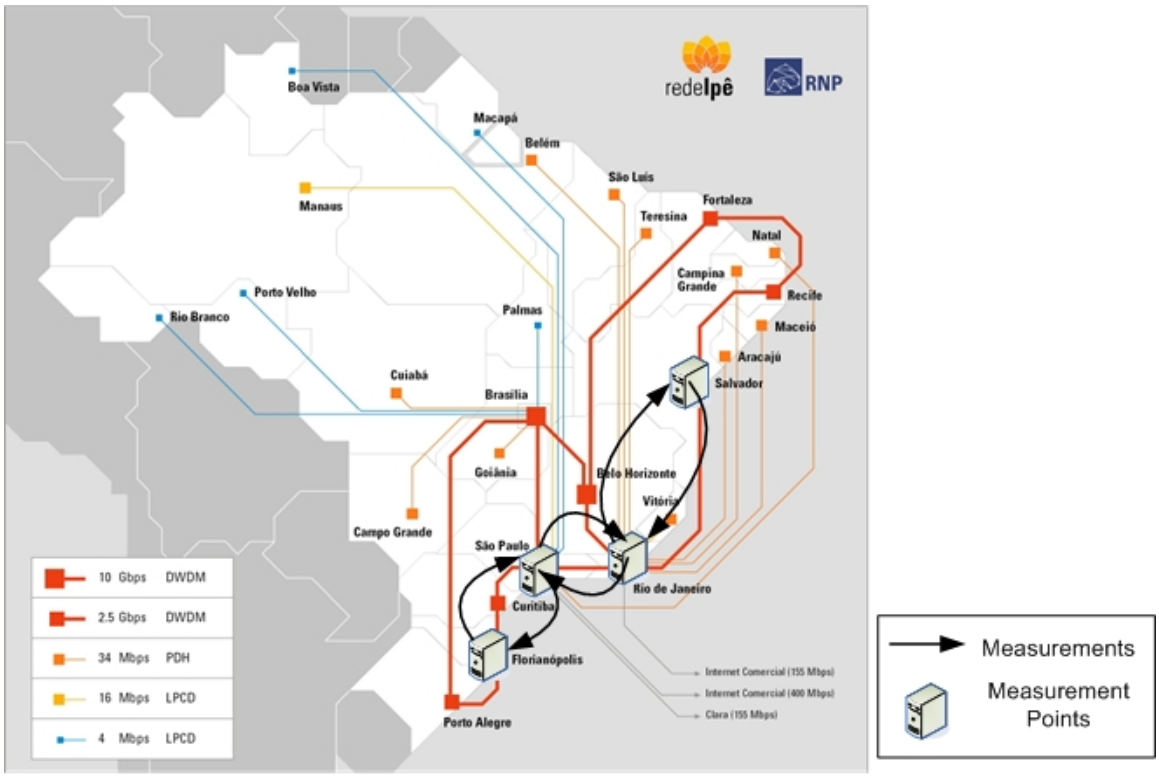

Figure 10. ICE scenario user case.

6. Conclusion and Future Work

This paper has explored the development of a framework that uses OSGi and some Java Interfaces to provide basic functionalities in the development of flexible and reusable network monitoring bundles. With the use of FLAVOR, the network monitoring tool developers can use the components approach to provide flexibility, adaptability, reuse among other benefits it provides.

The application users (i.e., the users interested in data, analysis and visualization of the network measurement data) could benefit from it in the moment that the user can choose their favorite components to use inside their application. The developed bundles can be assembled and composed into their applications using the benefits of the OSGi to deploy components dynamically and, consequently, configuring their systems dynamically.

Future efforts will concentrate on the spread of the development of network monitoring bundles, providing reusable components to the entire network monitoring community. There is also an intention of using this approach in other multi-domain network environments and in other application domains like network management. This could benefit many other users and prove the benefits of its use.

With this dissemination of bundle development and with our experience building the FLAVOR framework and the ICE network visualization environment, network monitoring tools development will need a network monitoring bundle repository that provides advanced features to this development such as publish and search of bundles. This common place to find the necessary bundles without having too much effort to search for

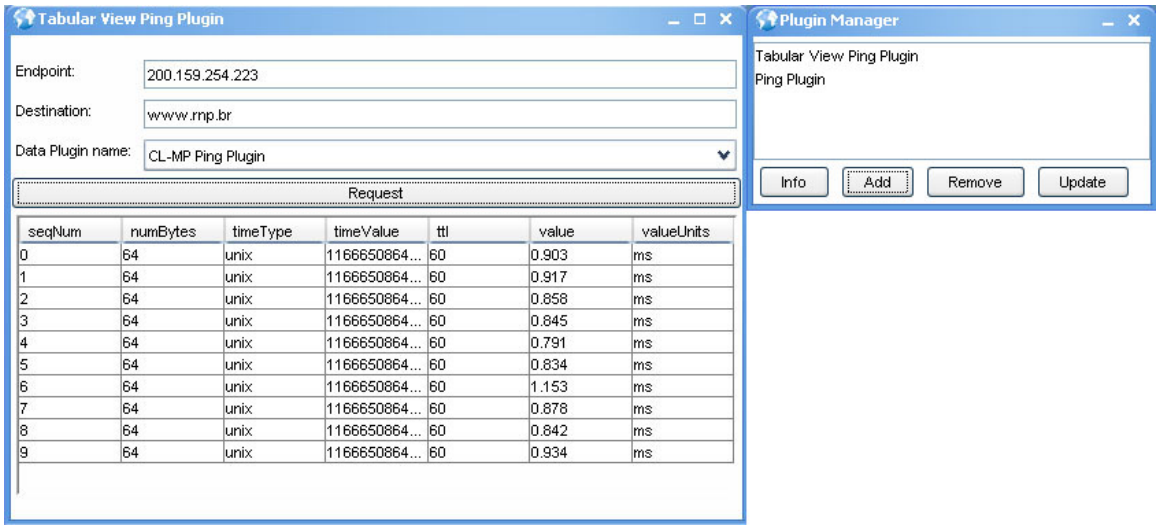

Figure 11. An Example using the CLMP Tabular View Ping Plugin and the Plugin Manager.

them can minimize the time to publish and to discover, making life easier to the network monitoring tool development teams.

Acknowledgments

The authors would like to thank The Brazilian National Research and Education Network (RNP), The Bahia's State Research Foundation (FAPESB) and The Brazilian National Council for Scientific and Technological Development (CNPq) for all its support during the development of this work.

References

Boote, J. W., Boyd, E. L., Durand, J., Hanemann, A., Kudarimoti, L., Lapacz, R., Simar, N., and Trocha, S. (2005). Towards multi-domain monitoring for the european research networks. *TERENA Networking Conference*.

Crnkovic, I. (2001). Component-based software engineering - new challenges in software development. *Software Focus*.

D'Souza, D. F. and Wills, A. C. (1999). *Objects, Components,and Frameworks with UML - The Catalysis Approach*. Addison Wesley.

Gray, D. N., Hotchkiss, J., LaForge, S., Shalit, A., and Weinberg, T. (1998). Modern languages and microsoft's component object model. *Commun. ACM*, 41(5):55–65.

Hall, R. S. and Cervantes, H. (2004). Challenges in building service-oriented applications for osgi. *IEEE Communications Magazine*, 42(5):6.

Hanemann, A., Boote, J., Boyd, E., Durand, J., Kudarimoti, L., Lapacz, R., Swany, M., Trocha, S., and Zurawski, J. (2005). Perfsonar: A service oriented architecture for multi-domain network monitoring. In *Proceedings of the Third International Conference on Service Oriented Computing (ICSOC 2005)*, pages 241–254. ACM Sigsoft and Sigweb.

Internet2 (2003). E2EpiPEs: End-to-End Performance Initiative Performance Environment System Architecture. `http://e2epi.internet2.edu/E2EpiPEs/e2epipe11.pdf`.

Legrand, I. C., Newman, H. B., Voicu, R., Cirstoiu, C., Grigoras, C., Toarta, M., and Dobre, C. (2004). Monalisa: An agent based, dynamic service system to monitor, control and optimize grid based applications. *Computing in High Energy and Nuclear Physics (CHEP) conference*.

Marples, D. and Kriens, P. (2001). The open services gateway initiative: An introductory overview. *IEEE Communications Magazine*, 39(12):5.

Sampaio, L., Koga, I., Monteiro, H., Koga, I., Rhoden, G., Vetter, F., Nunes, G., Melo, E., and Monteiro, J. A. S. (2006). pipes-br: Uma arquitetura para a medição de desempenho em redes ip. In *XXIV Simpósio Brasileiro de Redes de Computadores*, Curitiba, PR. Anais do SBRC 2006.

Siegel, J. (1998). Omg overview: Corba and the oma in enterprise computing. *Commun. ACM*, 41(10):37–43.

Szyperski, C., Gruntz, D., and Murer, S. (2002). *Component Software: Beyond Object-Oriented Programming*. Addison Wesley.