APPLYING COMPONENTWARE IN DISTRIBUTED NETWORK AND TELECOMMUNICATION SERVICES MANAGEMENT THROUGH JAVA TECHNOLOGIES

E. Yanaga, L. Nacamura Jr.

Centro Federal de Educação Tecnológica do Paraná – CEFET-PR
Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial – CPGEI
Centro de Pesquisa e Desenvolvimento em Tecnologia de Telecomunicações – CPDTT
Laboratório de Sistemas Distribuídos – LASD
Curitiba – Paraná – Brazil

ABSTRACT

The constantly evolving requirements of the telecommunication services market and their management systems demand new approaches that guarantee the scalability, robustness, and flexibility of network management systems. In addition, the new approaches must also provide cost and time reduction of the systems' development process. We believe that a *componentware* approach applied to distributed management can address all of these aroused needs. Java has been increasingly utilized in the development of management systems, and it provides some key technologies that make it feasible for the implementation of a component-based distributed management system. In this paper we will present the application of the *componentware* approach in distributed network and telecommunication services management through Java technologies such as Enterprise JavaBeans, and Java Management eXtensions.

1. INTRODUCTION

The recent changes introduced by the wireless world, the demand for the creation of new telecommunication services and the networks' size growth present new challenges to the network and systems management. With the new service-driven age now dawning [9], there are some crucial requirements that traditional management paradigms cannot satisfy.

Historically, there has been some time since people started questioning about the Simple Network Management Protocol (SNMP), the Common Management Information Protocol (CMIP), and their underlying paradigms. In the past few years, the management community started demanding strongly distributed management technology for mainly two reasons. First, strongly distributed management paradigms address some of the major shortcomings of traditional paradigms: beyond mere interoperability, they offer scalability, flexibility and robustness [3]. These three features, identified by Goldszmidt to motivate the use of his own model, Management by Delegation (MbD), can actually justify the use of any kind of strongly distributed management technology. Second, much progress was made in software engineering since CMIP and SNMP were devised, and new technologies suggested new ways of doing network and systems management [7].

One of the latest focuses in the software engineering area is Component-Based Software Development (CBSD) or *componentware*. The need for implementation of new telecommunication services currently represents a survival factor for companies in this competitive market. The emerging requirements for the development of these services and their management systems demand well established approaches that guarantee the system's robustness, dynamic extensibility, expandability, economy of the development process (through software reuse), and a fast time to market. All of these key features, needed in the next generation network management, can be addressed by the componentware approach.

Our ultimate goal is to provide a component-based distributed management architecture that can meet the constantly evolving requirements of network and services management systems. We consider that Java and its self-contained technologies have some key features that are appropriate for the implantation of the component-based architecture.

This paper is structured as following: Section 2 presents the distributed management concepts and describes its advantages. Section 3 introduces the componentware approach and describes possible manners of implementing it over manager and agent entities. Section 4 provides a description of some Java technologies suitable for the application of componentware in network management systems. Section 5 pictures our proposed architecture for component-based distributed network management system, and Section 6 ends our description with some conclusions.

2. DISTRIBUTED MANAGEMENT

The development of network management systems traditionally encompasses two groups of entities in different roles: managers and agents. Managers usually have been software systems running on powerful workstations, responsible for the processing and visualization of the network management data. Agents traditionally have been entities responsible for the collection of this data. Given the different nature of these entities, it is expected that the evolution of the paradigms behind these entities to be leveraged by different technologies.

The term "distributed management", in our view, has to be applied differently when referring to manager or agent entities. Distributed management applied to the development of management applications (managers) means that there is not only one or a couple of powerful network management stations responsible for the processing of management tasks. Rather than that, and differently from weakly distributed hierarchical paradigms [7] such as TMN, we should consider that the network itself is a pool of services provided by different servers whose location is irrelevant. Management components cooperate with one another across the network to provide distributed, scalable management functions. This dynamic and distributed architecture is accomplished through modern N-tier architectures where each layer provides a different functionality.

The role of agent entities in a distributed management framework is leveraged from simple passive dumb data collectors to active entities in the network management process. This idea, first presented by Goldzmidt [3], incorporates robustness, flexibility, and increased scalability to network management. With large scale distributed management, we have the spatial distribution of the system code, allowing the information processing to be nearer the data it works on – reducing the bandwidth needed to perform management tasks. Moreover, we have an added flexibility in the system, since it is now possible to distribute dynamically new pieces of code to devices as needed. Besides, if agents are now capable of executing some processing, then we might add some "intelligence" to them – allowing agents to execute some actions without responding to a manager in case of a link failure, for example.

3. THE COMPONENTWARE APPROACH

Recently there has been a renewed interest in the notion of software development through the planned integration of pre-existing software components. This is often called Component-Based Software Development, or simply componentware. The interest in componentware is based on a long history of work in modular systems, structured design, and most recently in object-oriented systems. These were aimed at encouraging large systems to be developed and maintained more easily using a "divide and conquer" approach. Componentware extends these ideas, emphasizing the design of systems in terms of pieces of functionality accessible to other pieces only through well-defined interfaces, outsourcing of the implementation of pieces of the application system, and focusing on controlled assembly of those pieces using interface-based design techniques. [6] Building software systems with reusable components brings many advantages. The development becomes more efficient, the reliability of the products is increased, and the maintenance requirement is significantly reduced. Moreover, the development time and cost can also be reduced.

The word "component" can now be considered ubiquitous, but some fundamental aspects of a component can be distinguished among various definitions. From [8], we can cite some common component's characteristics, it:

(1) has an external interface that is distinct from the component's internal implementation, and its interface is defined in a contractual manner;

(2) demands a set of operations from the environment it is deployed;

(3) provides a set of operations demanded by the environment in which it is deployed;

(4) can interact with other components in the same environment in order to form software units of arbitrary capability.

From this perspective, one of the most important features of a component is encapsulation, that allows a component to be a black-box, accessible only through its programmatic interface. Furthermore, a component can interact with its environment in four distinguished manners: it requests services (item 2); it provides services (item 3); and it interacts with other components through the generation and observation of events (item 4).

Having defined what is meant by a component, we must also present the aspects associated with the components' characteristics that lead to advantages over other approaches. First, components themselves are independent pieces of functionality providing services to other potential client components through its programmatic interface. It does not matter how a component is implemented, or whether its implementation changes on every new software release, as long as its interfaces remains unchanged. Software changes generated by requirement modifications remain restricted to the components that implement that particularly functionality.

Second, being components independent pieces of functionality, it provides favorable conditions for software reuse. And software reuse usually implies in software quality. If a component is reused, it means that it works and it has been tested. Also, componentware does not imply in software reuse *per se*. Software reuse also demands a well-defined design and an elaborated planning to be feasible.

Lastly, componentware allows that common services required by many applications to be developed as components once and reused over time. This allows the definition of a generic framework that can be used as a template for all new applications. Developers can then spend more time designing the application logic rather than spending it in common infrastructure functionality.

The application of componentware in network and telecommunication services management can address the current need for a management system capable of dealing with multiple network management protocols [5]. A distributed management framework can easily integrate legacy systems into this paradigm [4]. The integration of legacy systems is an important requirement for new paradigms, since the complete substitution of systems using CORBA, CMIP, and SNMP is simply not an option to final users, which must consider the investment already done in these systems. On the next two sections we will exploit how the network and systems management field can benefit from the key advantages provided by the componentware approach.

3.1 Componentware in the manager-side

Due to the great amount of information processing performed by a network management system, it is expected to become gradually more complex as the administrative requirements evolve. Yet, the development requirements of new telecommunication services and their management systems are in great part alike the ones present in large information technology software systems. Therefore, the software development process of a new telecommunication service and its management system can benefit from the latest advances in the software engineering area, such as the systematic utilization of frameworks, and server-side component models and architectures.

The framework concept embraces the premise that every management application has the same basic infrastructure. In its simplest form, a framework is simply a body of tried and tested code that is reused in multiple software development projects. The framework's utilization allows us to cut project costs and improve software quality all at once through software reuse. In addition, it reduces development time since developers can spend more time concentrating on the management-specific problem at hand rather than on some common basic infrastructure. A good framework also enhances the maintainability of software through API consistency, comprehensive documentation, and thorough testing.

Server-side component models define architectures for developing distributed business objects. They combine the accessibility and scalability of distributed object systems with the benefits of encapsulated business logic. Server-side component models are used on the middle tiers of distributed computing architectures, which manage components at runtime and make them available to remote clients. Yet, they support attribute-based programming, which allows the runtime behavior of the component to be modified when it is deployed, without having to change the programming code in the component. Due to its characteristics, server-side component models are appropriate architectures for development of distributed component-based services and management systems.

3.2 Componentware in the agent-side

The semantic richness of the information model of a management application is an indication of the expressive power of the abstractions used in this model. It measures how easy it is for designers of network and systems management applications to specify a task to be executed by a network management station or an agent. The higher the level of abstraction used to model a management application, the higher the semantic richness of the information model, and the easier it is for a human to build and design a management application. [7]

Management frameworks have traditionally offered protocol Application Programming Interfaces (APIs) such as SNMP and CMIP, constraining designers to model management applications with low-level abstractions concerning protocol primitives. This limitation has been addressed recently by the distributed objects management paradigm, and can also be addressed by the componentware approach. Components provide programmatic interfaces, making the transport primitives necessary to execute some remote task transparent to the programmer. The utilization of components grants developers with high-level abstractions of managed objects and its functionalities.

Network elements or managed objects can be easily modeled with components through the encapsulation of some data or behavior inside a component. Components allow the natural integration with legacy protocols like SNMP and CMIP, being capable of adding high-level functionality to network management and translating these to low-level protocol primitives. Common behaviors expected from agents such as polling and traps can also be easily implemented using a componentware approach. Polling maps to a service request/response (a method invocation) of components while traps match to event model. [4] The dispatching of events throughout the network can be done using a distributed observer/observable approach or through dedicated message-driven middleware.

4. COMPONENTWARE REALIZED THROUGH JAVA TECHNOLOGIES

Network and services management is an area that might involve integration of many technologies, including GUI standards, networking facilities, distributed processing, database, and object-oriented modeling. Therefore, a network and services management application might well make use of the ease of programming, extensibility, object-oriented concept, and simple description of management service provided by Java. Since the number of Java-based applications for network management is increasing, there is a corresponding increase in demand for a Java-based management framework. [6]

With the introduction of the Java 2 Enterprise Edition platform for the development of large-scale distributed applications, and more recently the Java Management eXtensions, Java has become a promising language and platform for the development of new generation network and services management systems based on a componentware approach. The next two sections will highlight two key technologies for the implantation of a component-based distributed management platform: J2EE and JMX.

4.1 J2EE: Enterprise JavaBeans and related technologies

Java 2 Enterprise Edition (J2EE) is the Java platform solution for the development of distributed enterprise applications. J2EE takes advantage of many features of the Java platform, such as portability in any platform, CORBA technology support, database access, and security model. Building on this base, J2EE defines a standard for developing multi-tier enterprise component-based applications and provides a set of services that handles many details of application behavior automatically.

The core technology of J2EE is Enterprise JavaBeans (EJB), which is a server-side component model for component transaction monitors. Component transactions monitors are robust application servers that deal automatically and transparently with facilities like concurrency, distribution, lifecycle, persistence, transactions, and security. EJB provides a standard component model for the construction of component-based applications, and offers a wide set of services to relinquish the application developer from dealing with basic software infrastructure. This allows the developer to focus on the business application logic, reducing significantly the development time and cost. In addition, components developed accordingly to the Enterprise JavaBeans specification can be reused and deployed in any EJB-compliant application server, in any platform. Figure 1 shows a layered representation of the relationship between the application server, the services provided by the Enterprise JavaBeans architecture, and the placement of the developed components.

At short-term, the utilization of Enterprise JavaBeans in the development of management and services applications can lead to a faster time to market due to the easy of programming offered by the Java platform and the infrastructure provided by EJB servers, which deal with many aspects of the applications' behavior automatically. At long-term, we can build up a development framework and a component repository to speed up even more the time to market of the new applications and increase the systems' reliability and quality, while lowering its development cost.

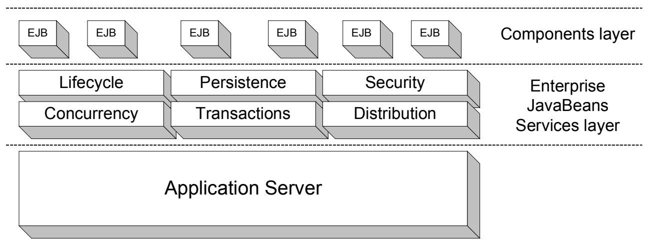

Figure 1 – Simplified representation of the Enterprise JavaBeans architecture

Some other J2EE technologies that can be relevant for the development of network and services management solutions are:

- JNDI (Java Naming and Directory Interface). JNDI provides a standard interface for different naming and directory services. This allows seamless access to directory objects through multiple naming facilities;
- JMS (Java Messaging Services). JMS is a standard interface for accessing message-oriented middleware;
- Servlets & JSP (Java Server Pages). Servlets & JSP are the key technologies of the J2EE presentation layer. They are a powerful solution for the representation of dynamic content over request/response protocols such as HTTP. Moreover, it can easily handle the dynamic generation of XML content for the distribution over heterogeneous environments.

4.2 Java Management eXtensions (JMX)

The Java Management eXtensions (JMX) define the architecture, the design patterns, the APIs and the services for network and systems management in the Java platform. JMX provides developers of Java technology-based applications across all industries with the means to instrument Java platform code, create smart agents and managers in the Java programming language, implement distributed management middleware, and smoothly integrate these solutions into existing management systems. [9] The JMX architecture is divided into three levels:

- Instrumentation level – acts at the level of managed objects, giving manageability to any defined component;
- Agent level – provides JMX management agents, which are containers that provide core management services that can be dynamically extended by adding JMX resources;
- Manager level – provides management components that can operate as a manager for distribution and consolidation of management services.

In addition, the JMX specification also provides interfaces to widespread protocols such as CMIP, CIM/WBEM, and SNMP.

JMX allows the management of managed objects as components (managed beans or MBeans), which acts as wrappers for applications, components, or resources in a distributed network. This functionality is provided by a MBean server, which serves as a registry for all MBeans, exposing interfaces for manipulating them. Furthermore, JMX contains the m-let service that allows dynamic loading of MBeans over the network. [2]

Besides different protocols and technologies integration in the Java platform, JMX also offers the ability do dynamically download code in one agent. If a new service and its management application are developed, then we can simply download a dynamic MBean in the agent at run-time, and that agent will be ready for the utilization and management of that service. This is an important feature of the JMX technology, since the static upgrade of the software agents would be very costly or even almost impossible. Moreover, since the dynamic load of code has some potential security threats, the security mechanism provided by the Java platform makes JMX a reliable and safe technology for the run-time upgrading of software agents.

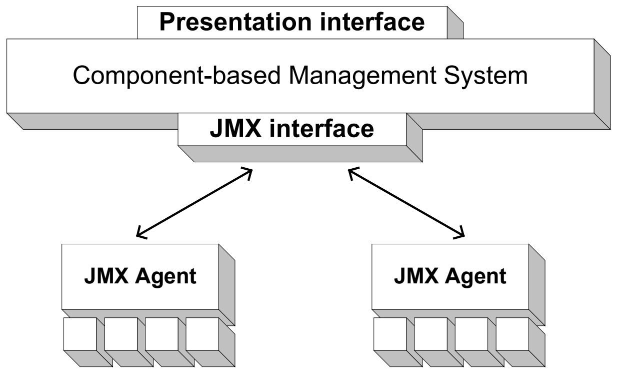

Figure 2 – Proposed component-based architecture

5. PROPOSED ARCHITECTURE

Our effort in the construction of a component-based distributed network management architecture, motivated by the ideas presented before, is applied in two directions: the application of J2EE technologies at the manager level, and the utilization of JMX at the instrumentation of managed objects and implementation of agents. Figure 2 depicts the proposed architecture.

At the manager level, the architecture includes currently a minimal framework that will provide some basic functionality to newly developed management applications. In addition, the architecture has some components that we intend to be reused in other management systems developed within this architecture. Figure 3 shows the schematic representation of the component-based management system (manager entity). It is important to notice that although the manager in the picture is represented in a monolithic way, the services, the framework's pieces and the components can and much probably will be deployed in a distributed way across a network.

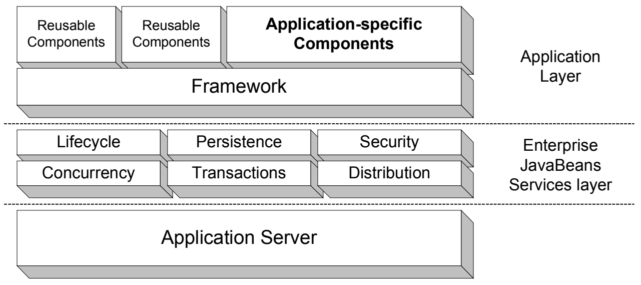

Figure 3 – The component-based management system

6. CONCLUSIONS

We have presented in this paper our proposal and ideas behind it for a component-based distributed network and services management system. The componentware approach is applied differently in manager and agent entities, and as so is also implemented in our architecture by different technologies: respectively J2EE and JMX. Since the development requirements of management systems are much alike the ones present in information technology systems, the extensive amount of work already provided by the software engineering area in this subject can and should be applied in large-scale network management systems. The componentware approach also seems to fit naturally in the design of agent entities due to the characteristics encapsulation and dynamistic of the approach and its implementation technology.

Concomitantly to the development of this architecture, we are also developing a pluggable component that will allow the network management systems built accordingly to our architecture to represent management interfaces in heterogeneous display devices. It will be done through the specification of a XML (extensible Markup Language) representation information model that can be automatically converted in markup presentation languages like HTML and WML through the application of XSLT (eXtensible Stylesheet Transformations).

7. ACKNOWLEDGMENTS

Edson Yanaga is a Master Student at CPGEI – CEFET-PR sponsored by CAPES.

We would like to thank our colleagues Anderson Massashi Shirata, and Fábio Henrique Flesch for the support and aid in the development of this research. Special thanks extended to our supervisor, Luiz Nacamura Júnior, for the precious advisory.

8. REFERENCES

[1] Brown A., Barn B. "Enterprise-Scale CBD: Building Complex Computer Systems from Components". *Proceedings of the 9th International Workshop on Software Technology and Engineering Practice*. Pittsburgh, USA. 30 August – 2 September, 1999.

[2] Fleury M., Lindfors J. "Enabling Component Architecture with JMX". *O'Reilly Network*. http://www.onjava.com/lpt/a/578. February, 2001.

[3] Goldzmit G. "Distributed Management by Delegation". *Ph.D. thesis, Columbia University, New York, NY, USA*. December, 1995.

[4] Jeong C., Shahsavari M.M. "Component-based Distributed Network Management". *Proceedings of the IEEE Southeastcon 2000*. Pages: 460-466.

[5] Kawabata T., Yoda I., Maeomichi H., Tago M., Yata K. "Component-oriented Network Management System Development". *Network Operations and Management Symposium, 2000.* Pages: 395-408.

[6] Lee J.O. "Enabling Network Management Using Java Technologies". *IEEE Communications Magazine.* Volume 38, Issue 1. January, 2000. Pages: 116-123.

[7] Martin-Flatin J.P., Znaty S., Hubaux J. "A Survey of Distributed Network and Systems Management Paradigms". *JNSM, Special Issue on Enterprise Network and Systems Management.* December, 1997.

[8] Page-Jones M. "Fundamentals of Object-Oriented Design in UML". ISBN 0-201-69946-X *Dorset-House Publishing, New York, USA*. 2000.

[9] Sun Microsystems, Inc. "Java^{TM} Management Extensions White Paper – Dynamic Management for the Service Age". Revision 01. June, 1999.