A COMPONENT FRAMEWORK FOR THE CONFIGURATION MANAGEMENT OF NETWORKS

Michael Wimmers[1], Arnulf Mester [1,2] and Heiko Krumm [2]

[1] Dr. Materna GmbH, Vosskuhle 37, D-44141 Dortmund, GERMANY
[2] Dept. of Comp. Science, Univ. of Dortmund, D-44221 Dortmund, GERMANY
Michael.Wimmers@materna.de, mester@acm.org, krumm@ls4.cs.uni-dortmund.de

Abstract: Currently the approach of component-oriented software development is in discussion. It aims to the cost-effective construction of flexible applications from megamodules. We report on an application of this approach to the configuration management of networks. A corresponding component framework was developed and encouraging experiences from application developments and their operation were gathered. We describe the framework which in particular supports scalable, easily extensible, and resource saving management applications. Moreover, an example application is outlined.

Keywords: Component framework, configuration management, management applications

1 INTRODUCTION

Modern network management systems can be classified as complex and demanding distributed applications. They control high numbers of heterogeneous network, computer, and application elements in order to keep up a growing spectrum of information processing and communication services. As enterprises rely upon the services, their quality has to comply with agreed service levels. Moreover, future demands have to be anticipated and pro-active changes shall provide for an efficient and lasting infrastructure. So, the information society technologies (IST) program of the European Commission recognizes the essential role of management systems and proposes corresponding objectives within the essential technology and infrastructure key action [5].

While changing future demands plead for open, flexible, and combinable management systems, most present systems are marked by a proprietary and monolithic

L. Kutvonen et al. (eds.), *Distributed Applications and Interoperable Systems II*

architecture. Open standards only apply to the distribution between agents and managers (cf. [7, 3]), while the main functions reside in relatively complex and inflexible manager applications.

Meanwhile first research prototypes and products, however, recognized the benefits of component-based software development (cf. [18]). Deri presented the pertinent component middleware Yasmin [4]. It supports the dynamic extension of core-applications by so-called Droplet-components. Using Yasmin, the network management application Liaison showed the flexibility and scalability of Droplet-extended management systems. It provided a comfortable Web-based management interface and proved that even multidomain management integrating CMIP, SNMP, as well as CORBA interactions can be accomplished by light and efficient applications.

Now, more recent approaches like Sun's Java Beans [15] and Microsoft's COM/DCOM [10] extended the scope of component systems. They provide rich lifecycle and middleware support enabling applications which can completely be built from a dynamically changig set of components. Assuming that appropriate components are available, the application development can concentrate on the design of the composition, on the set of components, their coupling and coordination. This task can substantially be supported by visual application-builder tools ([16], e.g. Bean Box [17]).

Recently, the Java Dynamic Management Kit (JDMK, cf. [14]) combines the component-structuring of management systems with the Java Bean component model. It defines an architectural framework for flexible multiagent management systems where management components can be pushed to dynamic autonomous agents. It provides management beans implementing core management services and supports their interconnection by infrastructure beans. Moreover, generator tools are offered. Since JDMK is very new, components dealing with special application-oriented management functions are not yet available.

Our work, like JDMK, also proposes a Java-based framework for flexible, extensible, and component-structured distributed management systems. In the absence of JDMK, we based our work directly on the Java Bean platform. We plan, however, to redirect the future developments in order to meet the JDMK architecture. Presently, there is a similarity of the architecure induced by our framework to that of JDMK supporting the mutual integration of components and subsystems.

While the present JDMK supplies only infrastructure and management components of general interest, our framework has already been applied to a specific management domain. We developed corresponding specific management components, constructed appropriate management applications, and gathered experiences from their operation. As first specific domain, we chose the field of configuration management of telecommunication networks where we could dispose of rich experiences and examples of traditional management system development. Moreover, our framework has a very important feature. The dynamic and automated configuration at runtime supports the timely and cost-effectively development. Current component models mostly only support the configuration at design time, where the elements to be managed are not accessible and therefore can't determine the detailed structure of the management application (cf. [2]).

So far, we made very convincing experiences. We were able to develop special purpose management applications very timely and cost-effectively. The applications operate very efficiently in comparison to traditional applications which were based on large standard network management consoles. Therefore, we will extend the framework to the other domains of network management and presently are developing components for the fault management.

The paper proceeds with short introductions into component-structured software and into the domain of network configuration management. After the discussion of the principles of the framework we describe the development, operation, and experiences of relevant example applications. Concluding remarks address the directions of corresponding future work.

2 COMPONENT-STRUCTURED SOFTWARE

The approach of component-structured software envisages that future applications shall be composed from cost-effectice components which are supplied by different developers and are offered to a growing community of customers on an open market (cf. [18]). By selection, configuration, and customization of components powerful applications can be built which are tailored to the special needs of single customers. Their architecture can very flexibly reflect the user requirements and their environment. The applications are easily extensible and modifiable by dynamic changes of components and their coupling. Moreover, since applications are built by defining the communication and coordination of components, the same means can be used to integrate different applications to cooperating super-applications.

In fact, component structuring has well-known roots. In particular, components encapsulate internal details and support reuse like classical modules. Components, however, shall be selected from a multi-vendor market and their interfaces shall comfortably support dynamic application configurations. Therefore components mostly will be of a size which justifies the additional efforts of commercialization and of dynamic integration mechanisms. With respect to this, [19] uses the notion *megamodule* and consequently calls the definition of the composition *megaprogramming*. In more detail, [18] characterizes components as units of composition which provide for contractually specified interfaces and have explicit context dependencies only. They have to be independently deployable and composable by third parties.

Meanwhile a series of platforms supports components. Most prominent are Java Beans [15]. The COM/DCOM approach is well-established in PC-based environments [10]. Moreover, the CORBA initiative is extending its approach to the comprehensive support of component structures (cf. [12]). The platforms typically provide notions for the description of component types, parameter types, and interfaces. They supply rich runtime support for the coupling of components and, in particular, enable introspection, the exploration of components, their interfaces, and their properties at runtime. Additional interest is given to the comfortable construction of applications by scripting languages or visual application builder tools.

Component notion and platform support alone are not sufficient to guarantee the benefits of the component approach. Additionally, component frameworks are im-

portant. They comprise a set of rules governing the architecture of applications by defining the component types, interface mechanisms, and collaboration models to be used [8]. Frameworks moreover can supply infrastructure components and tools which actively support the construction of rule-conformant applications. Additionally, they provide appropriate collections of domain-specific components.

In connection with frameworks, component-structuring is a very effective means for the productive develoment of flexible applications. Additionally, present research aims to the systematic achievement of a whole spectrum of system-wide properties like reliability, availability, maintainability, manageability, scalability etc. (the so-called *ilities*). While some of them can directly be established by means of infrastructure components (e.g. connector components), for others, the systematic support is still an open problem [1].

The component framework presented in the sequel concentrates on the following four *ilities* which are of major interest for configuration management systems. *Scalability and extensibility* shall support management systems providing nearly exactly that functionality a specific user needs. So a secretary uses a small application in order to configure the voice mail system of a PBX, while a service technician needs a more complex application when configuring switching units. The reliability of the communication infrastructure is essential for the customers and depends strongly on the *reliability* of the management system. Finally, *small footprints* of management applications are of importance. Not only with respect to small mobile computing equipment (e.g., used by a service technician in the field) but also with respect to the general load comfortable management systems enjoin to the managed system, low resource consumptions and reduced runtime requirements are of interest.

3 CONFIGURATION MANAGEMENT

"Network management is the act of initializing, monitoring and modifying the operation of the primary (i.e. user supporting) network functions" [13]. ISO/OSI [6] identifies five management functional areas (MFAs) of network management in their "FCAPS"-model: Fault Management, Configuration Management, Accounting, Performance Management, and Security Management. As you cannot accurately manage something without having to know about it and having to be able to modify its configuration, configuration management is the important foundation of all management activities.

The main functions of configuration management are

- to identify and document functional and physical characteristics of the managed system,
- to identify, perform and document any changes to these characteristics,
- to record and report about changes and configurations.

Thus, the configuration management not only resembles a passive asset management, but also includes active elements like dynamic update of configurations.

The gross architectures of management systems are distributed and at least contain agents and managers. Agents are located on or colocated to the managed system, monitor it, perform management operations on it, as well as send messages on predefined situations to the manager. The manager (or nowadays a cascading manager hierarchy) receives these trap messages, queries agents, and initiates management operations on the agents [9].

For configuration management systems, several architectural design aspects are of utmost importance (besides of security and performance requirements):

- as they depend on a close interaction with management, technical and service personnel, effective user interfaces are mandatory.
- as these interaction often are made with mobile personnel, the user interfaces should be separated from other parts of the system.
- as they often include essential administrative data, a robust and safe database integration is needed.
- as they have to cope with evolving managed infrastructure, the system have to be open, extensible and flexible to support the easy and smooth handling of new management use cases (i.e. processes) and their support by appropriate system elements.
- as they manage an evolving infrastructure, they should be scalable. This includes introduction of management hierarchies and domains, which have to reflect itself in architecture and different faces of the system to different users. This also includes a scalability in underlying database performance.
- affordability should be supported by a low footprint of the system, e.g. system elements should be loaded only on need.

The configuration management of telecommunication system networks is a typical example. Digital private branch exchanges (PBX) can be used to build up corporate networks and facilitate optional services like voice mail or fax server. The configuration deals with physical and logical structures: the physical structure reflects a hierarchical contains-relationship with systems, cabinets, frames and boards. The logical structure reflects two blocks: the switching unit and the additional servers. The switching unit resembles the control and line trunk groups, which itself are build from line trunk units. The servers comprises administration and data, voice mail, call charge, text, and fax servers. PBX also can be connected to larger networks. As usual for nowadays PBX, the proprietary management is extended by an SNMP-interface which facilitates the query of system characteristics as well as the definition of SNMP alerts.

4 THE COMPONENT FRAMEWORK

According to the definition in [18], a component framework is a set of rules and interfaces that build up an infrastructure to hold software components and support their interaction. Those software components are working together to build an application

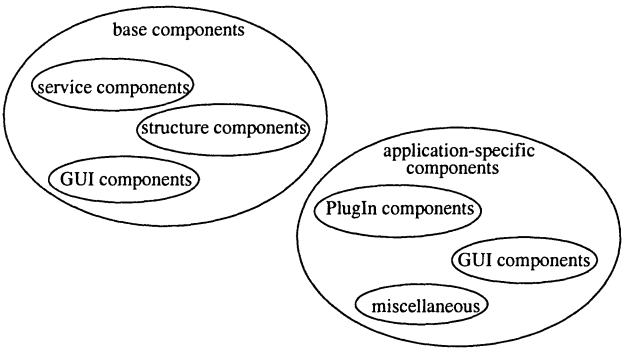

Figure 1. Component sets.

that fulfills the user requirements. The requirements of our target domain, i.e. of network management systems have been elaborated on in section 3. Moreover, the component framework should be designed specially to address scalability, extensibility, reliability, and small footprints as explained in Sect. 2.

First of all, to build a management application, tailored to meet the individual customer requirements, the framework must support the modification, removal or adding of components on all functional areas of the management system. By doing so, it is possible to influence all parts of it, rather than only a few (the GUI, for example). We classified two sets of components for a management system: base components and application specific components (see Fig. 1).

The base components implement commonly used functions for a management system like displaying network topologies or collecting events. Components that provide communication services for different management protocols belong also to this set.

Application specific components implement the logic for managing real resources like TCP/IP nodes or PBX systems. Typically, they do not build up an application on their own, but are embedded into (or reside on top of) some base components using their general services.

Managed Objects Every resource to be managed by the system is represented by a managed object. Similar to the definition in ISO/OSI [6], a managed object is an abstraction of an existing physical or logical element like a telephone, a network device or an user account. Each managed object of the system is represented by a managed object frame (MOFrame), which is a software component to encapsulate the implementation details of the management functionality for a resource. A managed object frame is a container for so called plug in components. Those PlugIns implement the required functionality for the management of the different types of resources to be managed by the system. A MOFrame with 1–n PlugIns builds a managed object.

The developer who wants to extend the system has to develop new PlugIns and can focus his/her work on the tasks to implement resource-specific functions, rather than to deal with the component framework. The framework is hidden by the MOFrame. On the other side, from the view of the framework, the MOFrame encapsulates the PlugIns, so the framework only has to know how to handle MOFrames without a knowledge about the resource to be managed.

Fig. 2 shows the internal structure of a MOFrame. It consists of two major parts: PlugIn administration and action administration. The PlugIn administration deals with connecting plug in components and providing general services for the PlugIns, like accessing an SNMP component, for example. The action administration allows the PlugIn developer to define actions (or operations) to be executed on the PlugIn. Actions are specialized method calls, handled by the action administration. Each time a PlugIn is plugged into a MOFrame, it is analyzed by the action administration (by means of the Java Core Reflection API based on a list of agreed method signature naming patterns). If there are some actions defined on it, they will be made accessible using the MOFrame's action interface.

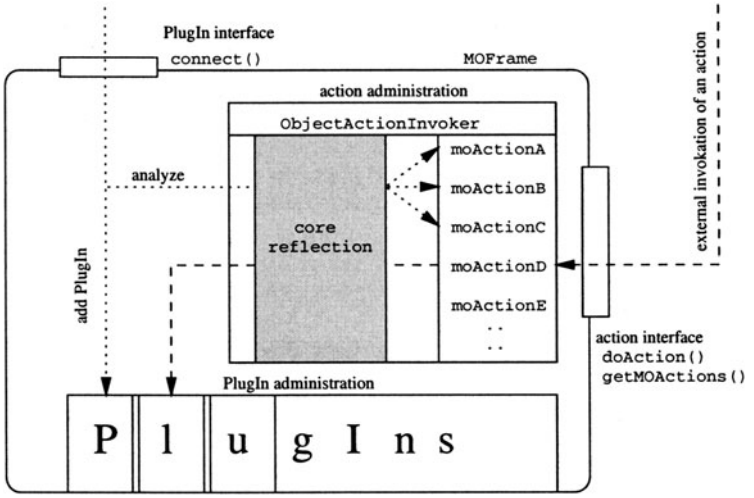

Figure 2. MOFrame's internal structure.

Services In a management system, there are many general services like performing SNMP operations or recording events. According to the requirements mentioned earlier, such services should be implemented as components. To reduce the needs of system resources, instances of those components should be shared. This means, for example, that all components of an application that need SNMP services at the same site share a single local instance of a component providing the services.

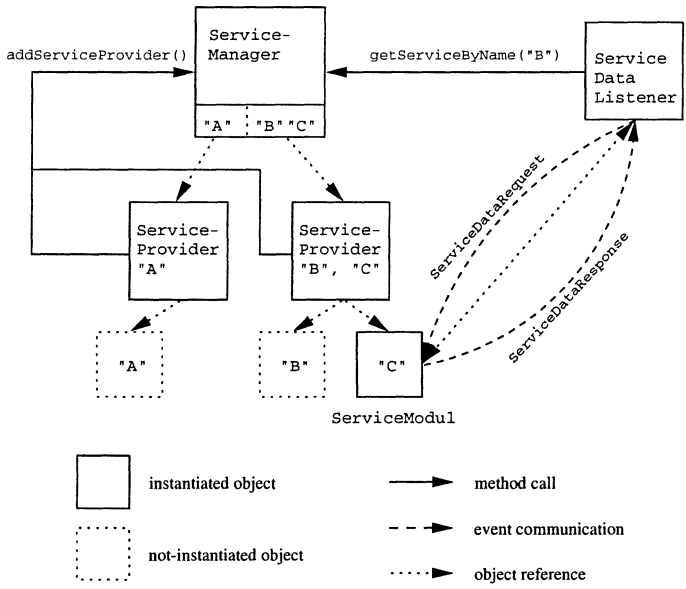

Figure 3. The Services Infrastructure.

The component framework supports instance sharing by an infrastructure for dynamic registration, de-registration and request of service-providing components. Our solution is oriented at the CORBA notion of service trading [11], but provides a restricted, resource saving, and efficient component-based implementation (cf. also [2]).

This is achieved by four different types of components:

- ServiceManager: The service manager's task is the administration of all services in a system/application. It provides mechanisms to register, de-register and request services. It also maintains a directory of services that can be queried by other components. Only one instance of the ServiceManager can exist per application (singleton).

- ServiceProvider: Components of this type offer 1-n services to the system. They register their services at the ServiceManager. Each time a specific service is requested, the ServiceManager delegates the request to the responsible ServiceProvider.

- ServiceModule: The ServiceModule implements a specific service. ServiceModules are managed by ServiceProviders. If a service is requested, the requesting component will be automatically connected to a matching ServiceModule by its corresponding ServiceProvider.

- ServiceDataListener: Every component that wants to use a service has to implement this interface. It defines the required functionality for connecting to a ServiceModule.

Figure 3 shows the interaction of the participating components.

An important feature of the framework is the mechanism to connect ServiceModules and ServiceDataListeners automatically at run- or designtime. Automatic connections at designtime are useful in visual application builder tools. The application designer just "drops" a component into the design, and it automatically "snaps" into the right position, that means, connects itself with the required ServiceModules. Automatic connections at runtime are useful in situations where a connection is required only for a limited period of time when a component is only temporarily used.

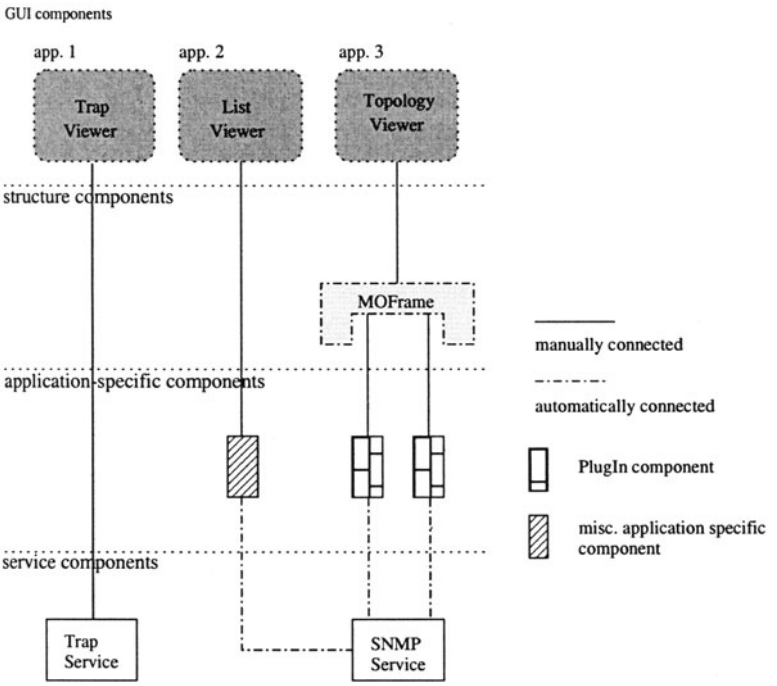

Figure 4. Component Layers.

All ServiceModules are designed to be used standalone (without ServiceManager and ServiceProvider) in small applications, or as part of the services infrastructure in larger applications. This improves the flexibility of the application design. Figure 4 shows three examples of applications with different complexity. The figure shows also the four functional layers of the system.The simplest application (left side) only uses components of two layers: A GUI component and a ServiceModule. The two components are connected manually by the user. The next application (in the mid-

dle) is more complex. It uses an application specific component which is, in this case, a "SnapIn-component". It automatically connects to the ServiceModule "SNMP Service" by using the ServiceManager, which is not shown in the picture. The last application (right side) is the most complex. It contains a managed object, build up by a MOFrame and two PlugIns. The GUI component, the MOFrame and the PlugIns are connected manually by the user, but the PlugIns connect to the SNMP Service automatically.

The three examples show the advantage of optionally automatic connections. The user (application builder) can use each component standalone and establish all the connections manually. Then he has full control over the design process. On the other

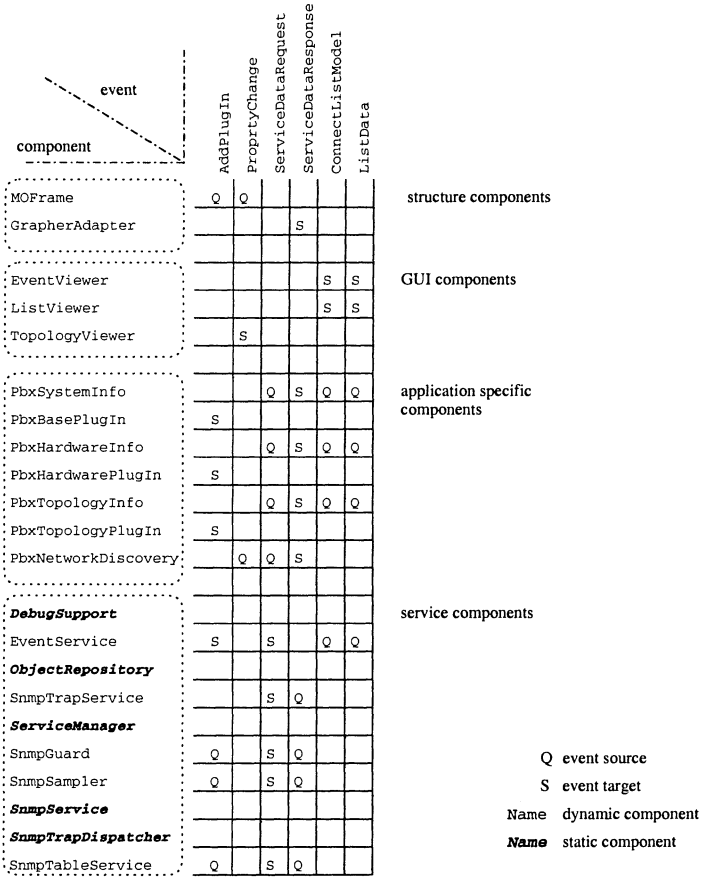

Figure 5. Event Sources and Targets.

hand, if he/she uses components which require some specific services to work, they connect automatically without further user interaction.

All component interaction is performed by sending and receiving events, according to the JavaBeans specification. The framework defines a few event types and sources and targets for events. Fig. 5 shows all components of the system, and all system specific events a component sends or receives. There are some components in the table printed boldface. These components are static and they are accessed by an API. They do not communicate by events, because they are only used internally. The user does not connect them with any component. In the described form, the framework was developed with an effort of one person year. Based on the existing framework, applications like the following example could be developed within two days.

5 EXAMPLE

This chapter briefly describes an example application, build entirely out of software components of this study. The builder tool used to create the application in this example is the BeanBox of the Beans Development Kit (BDK), version 1.0 from March 1998.

The application should have the following features:

- Automatically discover a telecommunication network consisting of PBX's of the same type (in this example, a family of PBX's from a major german vendor)
- Display the discovered network graphically
- Allow the user to access textual information about the actual hardware- and network-configuration of each PBX

We developed application specific software components for the management of the PBX system in this example. As mentioned in the last section, each resource to be managed by the system has to be represented by a managed object, which is implemented by a MOFrame and one or more plug in components. So the resource-specific management functions for a PBX of this type are implemented by three PlugIns:

- PbxSystemPlugIn: This PlugIn deals with the general identification and configuration data for a PBX. Every MOFrame must contain exactly one PbxSystemPlugIn to assign it to a real PBX.
- PbxHardwarePlugIn: This PlugIn acquires information about the hardware configuration of a PBX.
- PbxTopologyPlugIn: This PlugIn handles the network configuration of a PBX (gathers information about configured Trunks and TrunkGroups, for example).

For the family of PBXs in this example, an SNMP proxy agent exists which delivers information about all connected systems, so all information can be acquired using SNMP. Because all PlugIns of all managed objects need access to SNMP, we use the components ServiceManager and SnmpService to let the PlugIns connect themselves automatically to the required SNMP ServiceModules.

The discovery of the telecommunication network is done by the component PbxNetworkDiscovery, which is an example for an application specific component other than a PlugIn. PbxNetworkDiscovery queries an SNMP proxy agent and generates a MOFrame with the desired PlugIns automatically for each PBX found. This component also accesses the SNMP services via the ServiceManager. All generated managed objects are stored at another software component, the ObjectRepository.

Last, we need some GUI components to create the interface. The component TopologyViewer can be used to show the network elements discovered by PbxNetworkDiscovery. We also connect a button to this component to trigger the discovery process. Fig. 6 shows all components of the design and how they have to be connected by the user (application designer). Each arrow represents a connection between an event source and an event target. The name of the event and the target's method to be invoked are shown next to each arrow.

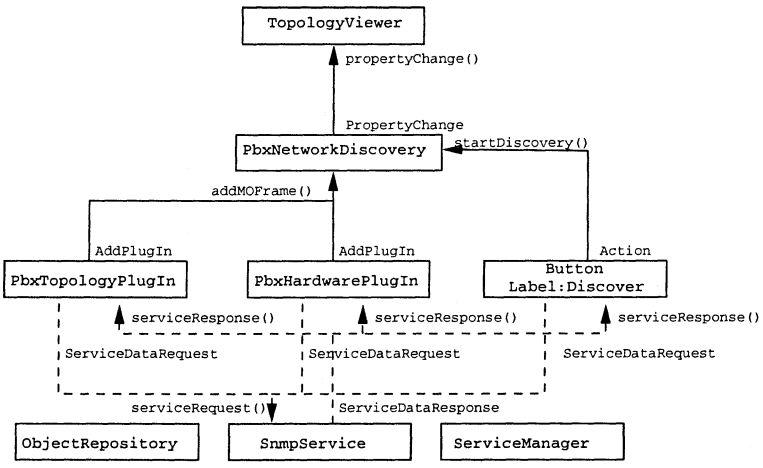

Figure 6. Components and Connections of the Example Application.

There are many components and connections that are created at runtime that are not displayed in the figure. During the discovery process, for each PBX and each trunk group (connection between two PBX systems) a MOFrame and some PlugIns are generated and connected to the TopologyViewer. Additionally, each time the user invokes some actions on a managed object to get some information about a PBX, a mini-application is created "on the fly" to acquire and display the data on the screen. This also results in dynamically created and connected software components. In detail, the following components are created at runtime: MOFrame (N), PbxSystemPlugIn (N), PbxHardwarePlugIn (N), PbxTopologyPlugIn (N), ListViewer (X), PbxSystemInfo (X), PbxHardwareInfo (X), PbxTopologyInfo (X), where N stands for the number of the discovered network elements, and X depends on the user's behavior, because

the components ListViewer and PBX*Info are created and connected to display some information on demand.

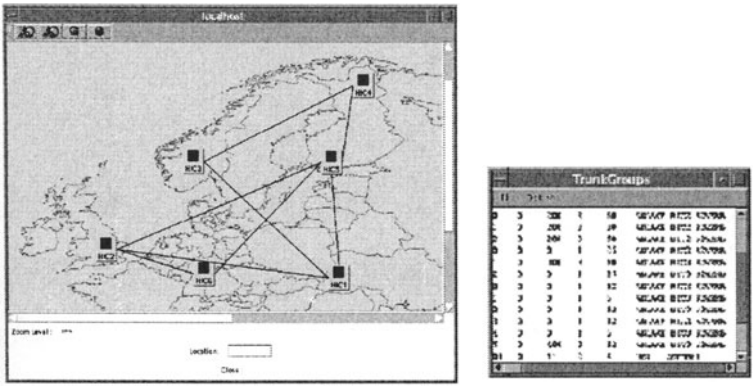

Figure 7. The TopologyViewer.

Figure 7 shows the TopologyViewer after the discovery of a network consisting of six PBX systems and a window showing some information about trunk groups. This window consists of the software components ListViewer and PbxTopologyInfo and was generated "on the fly" by the managed object of the corresponding PBX.

6 CONCLUDING REMARKS

We developed a component framework for the configuration management of networks and applied it in order to design, construct, and operate a series of special-purpose management applications. We experienced, that component-orientation is generally well-suited for the development of scalable, easily extensible and resource saving management applications. The flexibility, however, depends on the availability of rich component collections which not yet exist. Therefore, up to now, traditional management platforms support a broader spectrum of management functions. The framework itself is also extensible. Presently, fault management components were integrated while no extensions of the existing component interaction mechanisms and infrastructure components were necessary. Thus, we plan the integration of further component collections in order to enhance range and flexibility of the approach.

References

[1] AGHA, G., Compositional Development from Reusable Components Requires Connectors for Managing Both Protocols and Resources. *Workshop on Compositional Software Architectures*. Monterey, California, January 1998

[2] BEN-SHAUL, I., GISH, J.W., ROBINSON, W., An Integrated Network Component Architecture. *IEEE Software*, Sep/Oct 1998.

[3] CASE, J.D., FEDOR, M., SCHOFFSTALL, M.L., DAVIN, C., *Simple Network Management Protocol (SNMP)*. May 1990. (Status as of this writing: Standard)

[4] DERI, L., *A Component-based Architecture for Open, Independently Extensible Distributed Systems*. PhD Thesis. University of Berne, Switzerland, Jun 1997.

[5] EUROPEAN COMMISSION, *1999 IST Workprogramme*, Draft, Brusselles, Sep 1998.

[6] ISO. *Information Processing Systems - Open Systems Interconnection - Systems Management Overview*. ISO 10040. Genf, 1992.

[7] ISO/IEC, ITU. *Information Technology – OSI, Common Management Information Protocol (CMIP) – Part 1: Specification*. ISO/IEC 9596-1, ITU Recommendation X.711, 1991.

[8] JOHNSON, R.E., *Frameworks = (Components + Patterns)*. CACM 40(10)39-42. Oct 1997.

[9] KAHANI, M., BEADLE, H.W.P., Decentralized Approaches for Network Management. *ACM Computer Communications Review*. Jul 1997.

[10] MICROSOFT. *The Microsoft COM Technologies*. http://www.microsoft.com/com/comPapers.asp, 1998.

[11] OBJECT MANAGEMENT GROUP. *CORBA Trader Specification*. Document orbos-96-05-06, May 1997.

[12] OBJECT MANAGEMENT GROUP. *CORBA Component Model Request for Proposals*. June 1997 (Status: revised submissions received Nov, 1998).

[13] PRAS, A., *Network Management Architectures*. Centre for Telematics and Information Technology, D-thesis series No. 95-02, Enschede/NL, 1995

[14] SUN MICROSYSTEMS. *Java Dynamic Management Kit*. http:// www.sun.com/software/java-dynamic/, 1998.

[15] SUN MICROSYSTEMS. *Java Beans Specification*. http://java.sun.com/beans/docs/spec.html, 1998

[16] SUN MICROSYSTEMS. *Visual Application-Builder Tools Overview*. http://java.sun.com/beans/tools.html, 1998

[17] SUN MICROSYSTEMS. *Java Beans Development Kit (BDK)*. http://java.sun.com/beans/software/index.html, Jul 1998.

[18] SZYPERSKI, C., *Component Software*. Addison Wesley Longman, 1998.

[19] WIEDERHOLD, C., WEGNER, P., CERI S., Toward Megaprogramming. *CACM* 35(11)89-99, 1992

Biographies

Michael Wimmers is software and systems engineer in the network- and systems management department, as well as in the Authorized Java Center of Dr. Materna GmbH, an European 500-person IT-consulting, integration and software development company.

Arnulf Mester is research associate and teaching assistent with the department of computer science of Dortmund University and scientific consultant with Dr. Materna GmbH.

Heiko Krumm is full professor for distributed systems and computer networks in the department of computer science of Dortmund University.