An Integrated Network and System Management Framework based on Adapted Software Components

Martin H. Knahl
Steven M. Furnell
Network Research Group
University of Plymouth
Plymouth PL4 8AA
United Kingdom
E-Mail: martink@soc.plym.ac.uk
stevef@jack.see.plym.ac.uk

Udo Bleimann
Fachbereich Informatik
University of Applied Sciences
69425 Darmstadt
Germany
E-Mail: bleimann@fbi.fh-darmstadt.de

Holger D. Hofmann
ABB Research Centre
ABB
63115 Heidelberg
Germany
E-Mail: h.hofmann@web.de

Keywords

Integrated Network and System Management, Reusability, Componentware, Computer Networks

Abstract.

Componentware represents an evolutionary step in software development which adds a new packaging granularity to object-oriented systems: software components. Such as components in other sciences or technical domains, software components facilitate reuse even across application domains and allow the realisation of nearly any distribution scenario. In Integrated Network and Systems Management (INSM), the subjects to management are distributed. Furthermore, the heterogeneity of managed components often requires specific adaptations to the management software. The INSMware framework approach proposed throughout this paper achieves a maximum level of manageability by combining INSM with componentware. Software components by default are immutable which would even hinder minor adaptations to the management system or management semantics. To overcome this architecture-inherent limitation of componentware, we integrate the Component Adapter approach to INSMware, which allows the integration of even semantically incompatible software components.

1 Introduction

Nowadays, distribution represents a system-inherent criterion for software and hardware systems. Hardware as well as software components are to collaborate in large-scale environments such as the Internet and thus create new challenges in managing both. We develop the concept of integrated network and system management (INSM) and propose INSMware, a framework which provides component-based INSM, managing both hardware and software components [8].

Traditional management software is monolithic: stand alone applications performing all necessary functions. This paradigm is reaching its limits in terms of code reusability, extensibility, configurability and especially concerning the speed and size development. One reason for this is that traditional development approaches require the application to contain the entire functionality even if much of it is only required for very specific tasks. This paradigm also requires extensions to the design to be carried out and integrated by an application developer familiar with the system (typically the original developer). However, it is the user, not the developer, who is often the one most familiar with the application domain and aware of required extensions. The natural consequence of this would be to provide an interface that allows the user to add new functionality to the monolithic block and that defines a migration path towards compound applications.

Software components provide a different approach to the development process. Like a child with Lego™ building blocks, one can build a compound software application by using several elemental blocks – software components – rather than a monolithic entity. Once the application has been built, it has similar functionality but has the advantages of being distributable, scalable, configurable and can be modified by adding or replacing components. Configurability in this context means to use components or component-based applications in more than one application domain by setting parameters affecting the component's behaviour. Component software aims to provide an architecture to tailor individual needs easily and at low cost. The Latin proverb 'Divide et Impera' can now be changed to 'Build and Manage'.

2 Software Components and Adaptation

The concept of software components goes back to 1969 when McIlroy envisioned an industry of reusable software components and introduced the concept of formal reuse though the *software factory* concept [9]. The idea behind the concept of software components was and is to use self-contained, pre-fabricated and pre-tested units in order to build more complex units or entire applications.

As there is no agreed formal definition of a software component [13], we will first present the definition that forms the basis for our work: a software component is a piece of software with one or more well-defined interfaces that are configurable, integrable and immutable [4]. This definition highlights the immutability of software components. Though it guarantees black-box reuse [2], immutability can lead to static caller relationships between software components, a lack of system configurability, and poor support of object-oriented reuse mechanisms such as implementation inheritance [12]. Thus, the physical shape of software components highly influences their reusability.

The driving force behind the use of pre-fabricated components, be it in computer science or other domains, has always been reuse. Reuse relies a great deal on the availability of descriptions of entities to be reused.

Figure 1 shows a taxonomy of reuse potentials with respect to adaptability and modifiability.

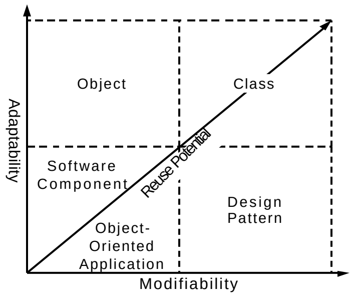

Figure 1: Taxonomy of Reuse Potentials

At one extreme we have classes, which exist in the form of modifiable source code. The reuse potential of a class is very high as one can always change the class' source code to meet the requirements of the developer. Objects, on the other hand, are adaptable, not modifiable, entities. An object exists only at runtime as an instance of a class and cannot be modified. However, an object's state and its relationships with other objects can be changed at runtime. Design Patterns [2] exist in graphical and/or textual format, and like classes, can be modified. Because a Design Pattern is dedicated to a specific use, it has very limited adaptability.

Object-oriented applications, i.e., applications that have been developed using object-oriented techniques, also have limited adaptability. Moreover, they are not modifiable since they exist in an immutable physical shape. It may come as a surprise to find software components at the lower left end of our taxonomy of reuse potentials. But a closer scrutiny reveals that software components share the same characteristics as object-oriented applications. They come in an immutable physical shape, having been implemented on distributed object-oriented architectures such as CORBA [11] and DCOM [1]. They provide, at most, limited support for the differential reuse mechanisms: inheritance, aggregation, and delegation, and are therefore not very adaptable either.

Software components are reused by composition [10], which means the grouping of a set of components to form a new component or even an application. The components are integrated on a common *composition layer* and communicate by exchanging messages. The interaction of these components is controlled by a *control logic*. Software component composition requires component compatibility. If components that are to be composed are not compatible, i.e., they are not able to collaborate because of interface or semantic reasons, the composition language can be used to realise component compatibility. We call this concept *adaptation*.

Let C_{org} be a component to be adapted that contains the implementations I_j, $)I_j$ the adapting implementation, and operator ρ an operation to combine I_j with $)I_j$. Then the adapted component C_{adapt} can be defined as:

$$C_{adapt} = C_{org} \rho\ )I \text{ with })I = \{I_1 \rho\ )I_1, ..., I_n \rho\ )I_n\}$$

The following criteria for software component reuse and adaptation mechanisms have been identified: transparency of use, black-box characteristics, configurability, reusability and architecture independence and efficiency [6]. As adaptation functionality at the level of a composition language is not reusable, it is not an appropriate mechanism for software component adaptation.

To cope with this problem, we apply the concept of Component Adapters [5]. Component Adapters are software components which represent a specific view of one or more software components to client components and which act as surrogates for these.

The incoming interfaces of a Component Adapter represent a view required by client components while its outgoing interfaces are connected to the incoming interfaces of server components to be used to realise the Component Adapter's functionality. The interface members of the Component Adapter can be mapped to one or more implementations provided by the server components. Depending on the implementation of the Component Adapter, these caller relationships can be changed at runtime or can be statically assigned at the time of development.

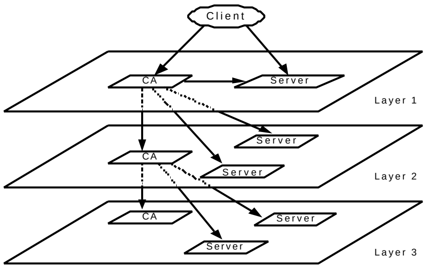

Figure 2: Component Adapter Usage Scenario

The mapping of incoming to outgoing interface members is installed by configuration and includes possible parameter and return type conversions. Though it is possible to do an automatic data conversion for elemental data types such as integer or float, the mapping of complex data types has to be configured by the user. Without any data conversion facilities only server components forming part of a subtype/supertype relation with client components could be used by the Component Adapter. Figure 2 shows a usage scenario of Component Adapters in which the approach is used recursively.

The design of Component Adapters is based on a combination the structural Design Patterns [15] [2]: *adapter* (adaptation of object interfaces), *decorator* (addition of functionality to existing implementations), *facade* (provision of high-level interfaces to sub-systems), and the behavioural Design Pattern *mediator* (centralisation of control). This makes it possible to change communication paths between software components, to define new interfaces to existing implementations, and to centralise the required implementation for the adaptation (control logic) in one place without the necessity of modifying the concerned components.

The integration of the Component Adapter approach with a component management architecture such as discussed in [5] provides flexibility in integrating self-developed software components with pre-fabricated components and supports systems evolution. This means that even software components that were unknown at the time of the management system's development can be slightly integrated with it. The major benefit of using a management system together with Component Adapters is that software components to be managed neither have to provide specific management functionality nor specific management interfaces.

3 Component-Based Integrated Network and Systems Management

The terms network, systems and applications management stand for all precautions and actions taken to guarantee the effective and efficient use of hardware and software resources of distributed systems and their underlying communication networks [3]. Several management architectures have been proposed and standardised as a basis for integrated management (e.g., SNMP based TCP/IP management, Telecommunications Management Network by the ITU). Management platforms that integrate different management applications are available on the market (e.g., IBM Tivoli TME/10 Management Framework). However, the proposed management architectures and platforms do not provide integrated management solutions for network management of LANs and WANs and for their different requirements on systems management. They generally represent management toolboxes with differences in middleware, management protocols or incompatible management applications. Given the diverse technologies and vendor specific implementations in today’s heterogeneous networks, the management architecture itself becomes a heterogeneous mixture comprising different management applications and platforms.

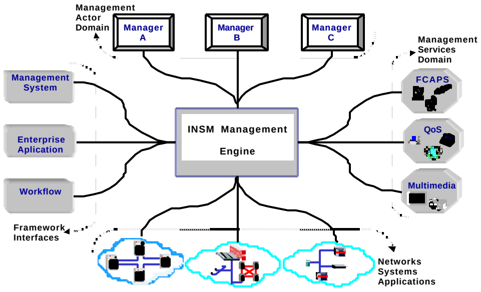

Figure 3: Management Framework

One attempt to address this problem could be to provide a single system capable of providing all Network and Systems Management services. Such a system is referred to as an Integrated Network and Management System. The problem with these is that they are very complex and therefore expensive and processor intensive. The Management Framework itself must be a distributed system with open interfaces, where the required management services are put together to provide the required management functionality (see Figure 3). Furthermore, open interfaces are required to provide interoperability to other management and enterprise applications and to expand the management framework to meet future requirements. Therefore, we propose a new component-based Management Framework [7].

The impact and leverage of distributed systems technology is prevailing not only for design and implementation of applications but also for the deployment of management systems. Thus far, management systems have typically been of two categories. Either specialised along one dimension, e.g., vertically targeting one or a few management aspects, e.g., configuration, billing or security or horizontally dedicated to management of a specific layer, e.g., network elements. Alternatively, they have resembled monolithic "main frames" based to a large extent on proprietary solutions. The presented research uses contemporary componentware technology to leverage a modular approach to the design of management systems, thus facilitating openness and extensibility on one hand and adaptability, i.e., customisation of management services, on the other.

The distribution of component-based systems mirrors the distribution of managed hardware/software components. We have already argued that a component-based approach does not inevitably guarantee reusability. We argue that this deficiency can be overcome by using Component Adapters that also allow for the integration of legacy components/applications into the management system.

As mentioned before, the conventional Management Systems cannot meet the needs of today’s rapidly changing network systems. To make the system flexible to change, we propose a new style of the network management system, in both the development and operation phases. We call it “Componentware Integrated Network and System Management”

(INSMware), as it uses the component-oriented approach for building and running the Integrated Management System [7] [8]. Our component-oriented Management System solves the problems of heterogeneous management protocols and can help reducing development costs.

The component-oriented software approach frees the developer from cumbersome coding, as the componentware based approach provides integration and customisation of Software components. This enables rapid and efficient system development, since tool software handles and validates much of the integrity of the system, but not a human.

4 INSMware

Limitations and restrictions of existing Network and System Management frameworks such as distribution of the management services and adaptation and integration of new management services can be overcome by providing a component-based approach [7] [8].

INSMware is a componentware-based framework for Integrated Network and System Management. We provide a component-based development approach meeting requirements for integrated management services. There are two versions of INSMware: one implementation is using CORBA [11], the other is based on DCOM [1]. This allowed us to study both middleware architectures in detail.

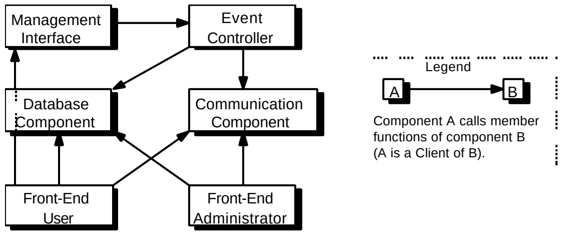

Figure 4: INSMware Components and Connectivity

The design of the individual INSMware components (see Figure 4) is based on a domain specification that subdivides the entire application domain into subdomains. First, the data processing system requires a connection to a data source. This is realised by the Management Interface component that exists in several different forms, similar to the device drivers of an operating system. It is configurable as required for different data sources.

The Management Interface component interprets the received data. It is filtered and analysed, and the component notifies the event controller when particular pre-defined exceptional states occur. Data storage is accomplished by a call to the database component and user notification is effected via communication components. It should be emphasised that all information about the users that need to be notified, e.g., access to user, user's role regarding the monitored processes, are stored within the system. The communication component itself consists of a set of several sub-components that again implement sub-domains, for example, faxes, voice mails, e-mails. The user can visualise system states using the front-end user component and can maintain the system by using the front-end administrator component.

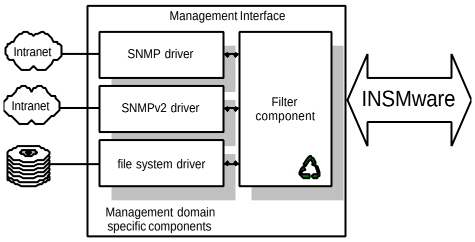

Figure 5: Management Interface Component

In our approach, a new management protocol can be installed by merely adding a new management domain specific component that represents the protocol into the Management Interface. This means that the protocol-handling part of the Management System is usually encapsulated to one component. This strengthens the adaptability of the component-oriented NMS to new management protocols, as the developer only has to develop the communication component specific to the new protocol. Therefore, we use a driver concept for the Management Interface that consists of the specific management domain and generic Filter component as shown in Figure 5.

4.1 Integration of Security Component

In any application domain, all data in the INSMware system is stored using the database component. To prevent unauthorised access to security-relevant data, clients must encrypt data before being transmitted to a database component that then decrypts it. Similarly, the database component must encrypt the data to be transmitted while client components have to decrypt the data. It was decided to add this functionality to the INSMware system, which means that four existing components, namely, the database component, the event controller component, the front-end user component, and the front-end administrator component have to be modified and re-built. These changes are not required if the additional functionality is integrated into the INSMware system in the form of Component Adapters via surrogate substitution. We demonstrate this using the DCOM version of INSMware.

Two approaches are possible:

i. realisation of the required security functionality inside a Component Adapter
ii. realisation of the required security functionality by a separate software component that can be used by a Component Adapter.

Approach (i) implies that the security functionality has to be realised in the programming language in which the Component Adapter is implemented. This restricts the reusability of the security functionality to one particular programming language. Approach (ii) allows to possibly reuse the provided security functionality not only by the Component Adapter, but also by several other software components. Coming from these two scenarios, we chose to realise approach (ii) because of reusability issues.

When realising approach (ii) either a security component can be developed from scratch or a software component can be purchased on the market that meets the requirements. We decided to develop our own security component to be able to reuse it in our various management domains. One demand for the realisation of a security component was the integrability with the development tools used. As these tools were ActiveX-capable, i.e., Microsoft ActiveX components could be integrated with these, we decided to implement the security component as an ActiveX component. ActiveX is a part of the Microsoft DCOM architecture that allows the realisation of scripting-capable software components.

Figure 6 shows the integration of a security component into INSMware. A client component accesses a Component Adapter that encrypts the data and sends it to a Component Adapter located on the same host as the database component. The second Component Adapter decrypts the data and sends it to the database component. After processing the client request, the database component transmits the results to the local Component Adapter that encrypts the data and sends it to the client-located Component Adapter. Finally, the results are decrypted and transmitted to the client component.

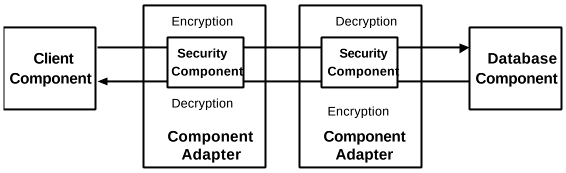

Figure 6: Component Adapter Integration

Both Component Adapters are located locally to the adapted software components. This location constraint is obvious as data transmission between the adapted components and their associated Component Adapter is still insecure. This does not represent a problem to local inter-component communication while this is not acceptable for remote communication.

The integration of Component Adapters with the INSMware system implies that every client of the database component, i.e., the event controller component, the front-end user component, and the front-end administrator component, accesses a client-located Component Adapter instead of directly accessing the database component. This integration is transparent to all client components and can be dropped if necessary.

Implementing a separate security component optimises the reusability of implemented algorithms, but its integration to a system may adversely affect the system's performance since the number of inter-component communications necessarily raises.

The Component Adapters must implement the interface of the software component to be adapted and expose this interface to client components. In the case of the INSMware system, this is the interface of the database component.

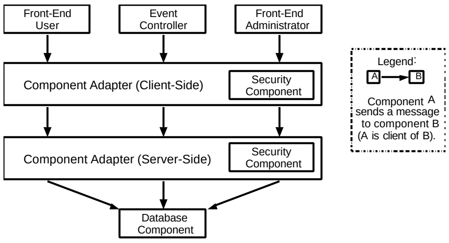
Front-End User
Event Controller
Front-End Administrator
Component Adapter (Client-Side)
Security Component
Component Adapter (Server-Side)
Security Component
Database Component
Legend:
A
B
Component A sends a message to component B (A is client of B).

Figure 7: INSMware Component Adapter Integration

Two Component Adapters, one for the client and one for the server side, are required. On the client side, data has to be encrypted before being sent to the server-located Component Adapter and data has to be decrypted after results being received from the server-side Component Adapter. The reverse behaviour is required on the server side.

The Component Adapters are integrated into the INSMware system by surrogate substitution and thus are transparent to client components. Each component of the INSMware system that accesses the database component does so through a client-side Component Adapter that communicated with a server-side Component Adapter. This is shown in Figure 7.

4.2 Management of SW Components

Component monitoring provides data about the run-time state of a set of software components. Data such as server host utilisation and memory usage of software components can be monitored. Without the use of Component Adapters, software components would have to be especially designed and implemented for the use within the management framework. This would far limit the components that can be used within our management environment. With Component Adapters, software components, with or without individual monitoring facilities, can be integrated and managed.

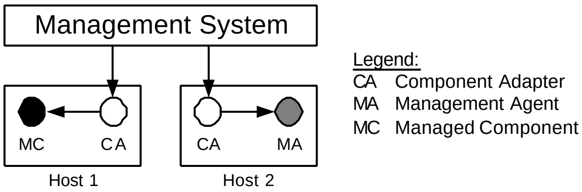

Figure 8: Component Adapters - Component Monitoring

Figure 8 shows two types of software component monitoring. In Host 1, a Component Adapter directly monitors a managed software component (MC). This approach can be applied if a MC supports monitoring functionality or if the Component Adapter gathers information about a component by making calls to its operational interface. In Host 2, the Component Adapter monitors a management agent (MA) that gathers information about a software component's environment, for example, the number of running processes or software component instances on a particular host, the operational status of hardware components (e.g., on/off/stand-by).

Using the Component Adapter approach, monitoring software components can be easily integrated with new or existing management systems. If, for example, Component Adapters implement a Simple Network Management Protocol (SNMP) interface [14], they can be integrated with any SNMP-compliant management system.

Along with the distribution of management components comes a distribution of management knowledge. In particular scenarios, it may be required that elemental management tasks are performed by local management components while complex tasks that might require user interaction may be co-ordinated by a central authority.

5 Conclusions and Outlook

Our work with INSMware has shown that a distribution of management software and thus of management knowledge can help in mastering the inherent complexity of distributed hardware/software systems. The realisation of management systems as componentware, i.e., software entirely composed of immutable pieces of software called "software components", can significantly support software reuse in this domain. In a scenario where normally the management software would have to be adpated to changing requirements, we propose the use of the Component Adapter concept. The latter helps to integrate and configure even incompatible software components and supports a common denominator on the software level.

For particular application domains, pre-fabricated Component Adapters may exist providing specific interfaces to managed components or to management systems. Other scenarios might require custom adapters whose development could be supported by libraries or code skeletons providing basic functionality.

6 References

[1] N. Brown, C. Kindel. *Distributed Component Object Model Protocol - DCOM/1.0*. Microsoft Corporation, Network Working Group, 1996.

[2] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns — Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[3] H. Hegering, S. Abeck, B. Neumair. *Integrated Management of Networked Systems.* Morgan Kaufmann, 1999

[4] H.D. Hofmann. *Componentware — Integration of Software Components in Distributed Computing Environments*. M.Sc. Thesis, Cork Institute of Technology/Ireland, 1997.

[5] H.D. Hofmann, J. Stynes. *Implementation Reuse and Inheritance in Distributed Component Systems*. Proceedings of Twenty-Second Annual International Computer Software and Applications Conference (COMPSAC'98), Vienna/Austria, 1998.

[6] H.D. Hofmann, J. Stynes, G. Turetschek. Software Reuse by Adaptation. *Proceedings of the Second International Network Conference (INC 2000)*. University of Plymouth: Plymouth, UK.

[7] M. H. Knahl, H.D. Hofmann, A. D. Phippen. A Distributed Component Framework for Integrated Network and System Management. Information Management and Computer Security, 7(5), pp. 254-260, MCB University Press, Bradford/UK, 1999.

[8] M. H. Knahl, U. Bleimann, H.D. Hofmann, S. M. Furnell. 2000. An Integrated Management Architecture for Heterogeneous Networks: INSMware. In: Jajszyczyk, Andrzej (Editor), Proceedings of the IEEE Workshop on IP-oriented Operations and Management (IPOM '2000). September 2000. pp. 111-118.

[9] M.D. McIlroy. *Mass-produced software components*. In J.M. Buxton, P, Naur, and B. Randell (editors), *Software Engineering Concepts and Techniques*, pp. 88-98, 1968 NATO Conference on Software Engineering, 1976.

[10] O. Nierstrasz, S. Gibbs, D. Tsichritzis. *Component-Oriented Software Development*. Communications of the ACM, 35(9), pp. 160-165, 1992.

[11] OMG. *The Common Object Request Broker: Architecture and Specification*, Revision 2.2. OMG Document 98-07-01, Object Management Group, Inc., 1998.

[12] M. Sakkinen. *Inheritance and Other Main Principles of C++ and Other Object-oriented Languages*. PhD thesis, University of Jyvaeskylae/Finland, 1992.

[13] J. Sametinger. *Software Engineering with Reusable Components*. Springer, 1997.

[14] Stallings, William. SNMP and SNMPv2: The Infrastructure for Network Management. IEEE Communications: Management of Heterogeneous Networks, 36(3), 1998.

[15] W. Zimmer. *Relationships between Design Patterns*. Proceedings of PLoP '94 - Pattern Languages of Programs, Addison-Wesley, 1995.