Virtual Private Components

An Approach for Leasing Components

Alek Opitz, Torsten Mielke¹⁾, and Hartmut Koenig

*Brandenburg University of Technology at Cottbus, Department of Computer Science,
PF 101344, D-03013 Cottbus, Germany*

¹⁾ *Torsten Mielke is now with IONA Technologies Ltd. Dublin.*

Abstract Component technology has become an important approach to set up distributed applications and systems. Components possess exactly specified or standardized interfaces and explicit context dependencies. They can be purchased at component markets and integrated into the application. The purchase of components, however, might be not the onliest solution in future. If a component requires large resources, an expensive maintenance, or a high degree of topicality it might be more favourable like in everyday life to lease a component for a certain time. This can save purchase cost and give the customer the possibility to always use the newest version of a component. The leasing of components requires a particular framework. Beside a contract which defines the conditions of the component use, a technical infrastructure is required which ensures the availability of the service and the enforcement of the contract agreements. In this paper we present a technical framework for the leasing of components called Virtual Private Components. We further present an architecture to support the leasing of components and discuss implementation issues. Finally, we present a prototype implementation.

Keywords: Component technology, Software Leasing, Virtual Private Resources, Middleware, Long-term service customer-provider relationships

1. MOTIVATION

The communication infrastructure of the Internet is constantly improving. Advanced communication technologies and protocols allow the use of services which are provided at distant places of the earth. This more and

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35584-9_19](https://doi.org/10.1007/978-0-387-35584-9_19)

more allows the construction of distributed applications and systems based on such services.

The dynamic integration of services of various providers into an application and the acceptance of possible updates require standardized interfaces for their integration and for the access to these services. The component software technology represents a promising approach to provide techniques to solve these problems. A component is a piece of software with exactly specified or standardized interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties [Szyp98]. Components will be offered and traded on open component market places. Customers can purchase needed components and import them, possibly via the Internet, into their application. Current component technology is characterized by the purchase of components, which then become property of the customer, and their direct integration into the application [Szyp98], [Mons99], [Wimm99], [Sieg00].

As in everyday life there might be situations where the leasing of components will be more favourable. This will be the case for services with large resource and maintenance requirements, and high demands on the topicality. Examples of such services are encyclopedias, information services, e. g. for stock exchange data, geographic information systems; simulations and calculations on supercomputers, high-performance and reliable web servers, dynamic security management functions and others.

To set up composite services which use remote services stable and reliable relations to the providers are required. Unlike purchased components which are directly integrated into the application leased components require mechanisms for an installation limited in time. Contracts have to be specified which define the conditions of the service provision and use. Mechanisms are required which guarantee the fulfillment of the contract. In addition, means must be provided to advertise these components on component markets as well as measures to protect consumer interests.

In this paper we present an architecture for the leasing of components called Virtual Private Components. It extends the concept of the Virtual Private Resources [Preu99b], which supports a long-term binding of services. The paper focuses on architectural aspects of the Virtual Private Components approach. In Section 2 we discuss aspects of the leasing of software. Section 3 introduces the basic principles of the Virtual Private Resources approach needed for the further discussion and summarizes the motives for a redesign of the approach. In Section 4 we describe the concept of Virtual Private Components and outline the differences and benefits compared to Virtual Private Resource approach. In Section 5 we present an architecture to support Virtual Private Components. Section 6 describes a

prototype implementation and discusses implementation issues. The final remarks give an outlook on next research steps.

2. LEASING OF SOFTWARE

The leasing of software (also called application service providing (ASP)) is a new developing sector in the software market. It provides customers similar benefits as the leasing of products in everyday life. Software leasing saves purchase cost like usual leasing and gives the customer the possibility to deploy the latest versions with the newest features each time. This is especially advantageous for small companies or organizations which cannot afford large expensive software systems. In addition, it avoids installation errors by new patches or update versions. Examples for leased software systems are office applications, websites, task management systems, enterprise software solutions and others. For the leasing of software, there are two possibilities: the temporary installation of the software received from the ASP and the invocation of a service which is provided remotely. In this paper we consider the latter case. The permanent binding of a service reveals problems concerning the availability, the security, and the billing of the service. In the following discussion we focus on the problem of the constant availability of services.

Service mediation and binding in open service markets is mainly based on the trading concept [ISO97], [OMGS98]. The trading approach supports a short-term relation between the service customer and the provider. The service customer addresses to the trader on demand, i.e. in an ad hoc manner. The trading concept does not guarantee that the same service with defined service properties is available for a longer period of time and can be called several times. If a service customer again calls the service it cannot be guaranteed that the same service will be bound again. In addition, the service provider may withdraw a service offer at any time. The trading concept is therefore only suited for the single use of a service. In many cases, especially, when users spontaneously address to the market, this mode will meet the requirements of the users. However, for users who need certainty that a particular service is provided, e.g. for composing a new service with defined properties, this is insufficient. An approach to permanently binding services is the Virtual Private Resource concept proposed in [Preu97]. It is shortly introduced in the next session.

3. VIRTUAL PRIVATE RESOURCES

Virtual Private Resources (VPRs) are an approach for the constant binding of services with a defined Quality of Service (QoS) fixed in a contract between the service customer and provider. A VPR represents a service or a set of services that can be bound from a remote service provider for a longer period of time, e.g. weeks, months, or even years. The conditions of the VPR use are defined in the *VPR contract*. A VPR gives the customer the illusion that the service is permanently available and can be used like a private resource. This resource, however, is a virtual one. It is located in a remote domain and is mostly shared with other applications (see Figure 1). The service provider guarantees that the service is available at a defined time and assures the provision of the agreed service properties. VPRs were designed for the outsourcing of services. They can be also used to establish permanent customer-supplier relationships in electronic service markets [Preu99b] similarly to fixed supplier chains for just-in-time production.

Figure 1. Virtual Private Resources

3.1 VPR Architecture

The transparent use of VPRs requires particular measures for their provision at the provider side. Figure 2 shows an architecture for the provision of VPRs. It distinguishes two domains: the customer domain and the provider domain. *VPR customers* are organizations and enterprises that

have contracted VPRs. Applications and/or human users belonging to that organizations are called *VPR users*. The users can be supported by a *VPR manager* for adaptation purposes, e. g. if the VPR contracts has been renegotiated. VPR users and the VPR manager access the VPRs via the *VPR proxy* which represents the VPRs in the customer domain.

Figure 2. VPR Architecture

The counterpart of the proxy in the provider domain is the *VPR broker*. It mediates the requests from the *VPR proxy* to appropriate servers which implement the related services. The mediation remains hidden to the VPR users. Note that the *VPR broker* is not a trader. The binding of the server is based on resource requirements and availability. A VPR is set up by the *service manager* according to the specification received from the broker. The *service manager* can bind further subservers if required. After setting up a VPR the reference to the respective server is returned to the *VPR proxy*. It calls the server and starts the service execution. The results of the service are delivered via the proxy to the VPR user.

The presented VPR platform was implemented using ORBIX 2.3 based on CORBA 2.1. It was used to set up a traffic information system as Virtual Private Resource [Preu99a].

3.2 VPR contracts

The *VPR contract* defines the conditions for the VPR use. It specifies among others the duration of the availability of the service, the frequency of

its use, the QoS provided, and the cost of the service use. The VPR contract is the most crucial feature of the VPR concept, because it specifies the obligations of the VPR provider. The contract has to be negotiated between VPR customer and provider before binding the service. This can be agreed by representatives of both sides in conventional way or automatically if such means are available. In order to exactly specify the contract conditions the range of service properties was extended [Syrb98]. Additional properties were introduced to specify the scope of the service, the results, the execution, the duration of the provision, the required capacities, and system aspects. Each property describes a range of values limited by maximum and minimum, and threshold values to define the optimal operation range for the property.

The contract conditions are enforced by the VPR provider for the whole period of the contract validity. Two different kinds of measures are applied: reactive and post-use measures. Reactive measures are taken by the node managers which observe the operation of the servers. Examples are the assignment of other priorities to VPR invocations, the activation of standby resources, and the migration to other servers. Post-use reactions are based on monitoring the execution process. The observers constantly measure the values of the dynamic service properties and return usage profiles as feedback to the VPR broker. These profiles are used for new VPR invocations to select the servers. Detailed descriptions of the VPR mechanisms and the related procedures are given in [Syrb98], [Preu99a].

The concept of Virtual Private Resources mainly aims at the provision of complex services rather than smaller partial services which can be combined with other services. The approach requires a pretty tight relationship between the customer and the provider which makes a change of the provider difficult. This concerns less the conditions of the VPR contract which can be negotiated flexibly, but more the technical aspects of the customer and provider relation. The application has to support the mechanisms for binding and calling the VPR. Currently these mechanisms must be directly coded into the application and linked together with class libraries of the VPR provider. The change of the provider requires an experienced system engineer to handle the technical details at both sides. Similar problems appear when the interface of the service changes. Therefore a redesign of the concept has been considered appropriate.

4. VIRTUAL PRIVATE COMPONENTS

4.1 Basic Characteristics

Virtual Private Components (VPCs) aim at the leasing of components which are provided at a remote location in the Internet. A VPC represents a partial service needed to set up more complex distributed applications or services. It can be bound for a certain time and combined with other bought or leased components [Miel00]. The leasing period is supposed to be a longer period of time, e.g. several months or years. During this time the provided service is supposed to be always available. It can be activated with a constant QoS as often as needed according to the agreements of the *VPC contract*. Unlike the VPR concepts the VPC approach allows that an application can simultaneously bind components from different VPC providers. It also can change a VPC provider if required.

The upper part of Figure 3 gives an example of a possible VPC application: a credit card checker to verify the trustworthiness of the credit card holder. When the holder wants to pay with his/her card the bank proves whether the client still has credit and whether the payment corresponds to his/her usual card deployment. For the latter, client profiles are used which indicate the average sum of payments, preferred goods, countries and frequency of the card deployment, and other criteria. Instead of developing its own verification software a bank can use a service offered by a VPC provider which is specialized in such checks. The bank calls the service within its payment handling procedure. It is thus the *VPC customer* and the checking company the *VPC provider*. The credit card holder and the shop assistant are not affected by this. They do not even know that a VPC is used. Further examples of possible VPC applications are traffic information systems which inform drivers about the actual states of the streets when he/she enters a new region to help them to navigate through this region, web servers for small businesses which provide the whole hardware and security infrastructure, virtual printer services to handle print orders of different clients, and many others [Opit00].

What are the benefits of using VPCs? A VPC gives the customer the opportunity to benefit from varying offers of the VPC providers. So a bank can change, for instance, the credit card checker service when an improved verification algorithm appears on the market. Analogously to the selection of the cheapest provider at the telephone market, the VPC customer may select the service which best meets its requirements. Selection criteria may be the service quality, security levels, availability, topicality and of course the prize.

Figure 3. Credit card (CC) checking service

4.2 Requirements to a VPC architecture

An architecture to support Virtual Private Components has to meet different requirements related to functionality, performance, and security.

Functionality

The VPC architecture should be flexible and independent of different providers as well as of the middleware platforms on which it will be implemented. It should meet in particular the following demands:

- *Change of service providers*

The application should be able to select other service providers which provide better contract conditions. The technical efforts to change the service provider should be minimized. The customer should not require a specialized technical knowledge. Changes in the source code and recompilation should be avoided.

- *Replacement of leased and imported components*

It should be possible at any time to replace leased components by imported ones and vice versa to allow the client to buy a component when this becomes more advantageous than leasing (or vice versa).

- *Simultaneous use of different service providers*

Analogously it should be allowed to simultaneously deploy VPCs of different providers. The architecture should not confine the selection of the VPC providers. Each leasing decision should be independent of the others.

- *Platform independence of the components*

VPCs should be independent of a certain computing platform (i.e. independent of a certain hardware, operating system, or middleware) to facilitate their installation in the customer domain.

Performance

A VPC has to reduce the runtime overhead caused by the VPC mechanisms in the customer domain and in the interaction with the VPC provider. The functionality at the customer should be as lean as possible. For the interaction with the VPC provider, an efficient protocol is required. The frequency of interactions can be reduced by caching.

Security

Security is another important aspect of the VPC approach. It concerns the customer domain and the interaction with the VPC provider. The VPC architecture has to ensure that the VPCs do not endanger the customer domain. They should not provoke system breakdowns or influence the functionality of applications running in this domain. A secure communication between VPC customer and provider is complicated to realize, because this communication crosses domain borders. It has to pass the firewalls which protect the intranets borders. Furthermore, the communication has to include known security such as authentication, authorization, protection against data changes, data encryption, and auditing.

Further features

Additional features are required to facilitate the leasing process:

- *VPC exchange by plug&play*

The exchange of the service provider as well as of components should be allowed at runtime without interrupting or disturbing the application. The VPC platform should support plug&play mechanisms for a dynamic replacement of the service provider or other components. Plug&play techniques should be also used for automatic maintenance and updating of the bound services [Aage99], [Raza99].

- *Component market places*

In order to give the customer the opportunity to select among various service providers places are required where the customer can inform about the actual offers and make their choice. Service mediating entities are required to manage the available service offers and to support the customer in its selection process including contract negotiation. The implementation of such a market place does not belong to the VPC approach in closer sense. It is an infrastructure feature to support the concept.

In this paper we focus on architecture aspects. We discuss the functionality needed to install leased components in an application.

5. A VPC ARCHITECTURE

In the section we present a possible architecture to support Virtual Private Components. The proposed architecture is generic. It is independent of a certain middleware platform. Aspects of the implementation on existing middleware platforms are considered in the next section.

5.1 Location of the VPC proxy

An important aspect in the design of the VPC architecture is the location of the VPC proxies. In the VPR approach the proxy was placed in the customer domain (see Figure 2) to mainly hide the provider activities and to check the access rights of the VPR clients. This solution, however, is not obligatory, because most of the functions can also be accomplished using a provider based proxy [Opit00]. It makes no difference for middleware platforms like CORBA whether the client application accesses to an object at the customer or the provider side. For the VPC architecture, nevertheless a customer based solution has been chosen (see Figure 3) for the following reasons:

- Customer based proxies are more flexible in the communication with the VPC provider. They better support the selection of the most appropriate communication paradigm, e. g. to fulfill different security requirements.
- They make the communication with the VPC provider more efficient. An intelligent caching can shorten the response time by reducing the number of method invocations. Appropriate data compression methods can further increase the speed of the interaction.
- The VPC proxies can provide service specific graphical user interfaces in addition to their management interfaces.

All in all the customer based location of the VPC proxies gives more flexibility related to speed, security and operation. It also better reflects the idea of a virtual and private service [Opit00].

A VPC consists thus of two parts: the VPC proxy and its implementation at the provider side. The communication between the VPC proxy and the provider is determined by the provider. A VPC can provide several VPC interfaces to the VPC customer. These interfaces must be specified in the VPC contract and can be used from the customer for different applications (*VPC clients*). A client application can simultaneously access to several VPCs which can be provided by different providers. To unify the access to the VPCs we use like in other component concepts [Mons99], [Sieg00] a provider independent framework. This function has been assigned to the VPC manager (see **Error! Reference source not found.4**) which played a minor role in the previous VPR approach. The VPC manager handles the integration of the VPC proxies in the customer domain. It provides functions to install and to remove VPCs and to make needed assignments. The enforcement of the VPC contracts and the billing of the VPC use remain at the provider side as described in Section 3. We can apply here the same mechanisms as used for VPRs (see [Syrb98]).

Figure 4. Integration of VPCs into the customer domain

The differences between the VPR and the VPC concept thus mainly concern the customer domain. The mechanisms at the provider side remain the same. Therefore we only consider the customer domain in the sequel.

5.2 VPC manager

The VPC manager comprises all procedures and interfaces to integrate the VPCs into the customer domain and to make the respective services available to the client application according to the requirements formulated in Section 4. Figure 5 shows the structure of the VPC manager. It consists of runtime environment to take in the VPC proxies, several management modules and 3 interfaces. These components are described next.

The **VPC runtime environment** is the container for the VPCs proxies. It represents the execution environment of the VPC proxies and provides the

access to various management functions to support the integration of the VPCs into the client application.

Figure 5. Structure of the VPC Manager

The **mapping module** has the task to map service IDs onto the corresponding VPCs or VPC interfaces, respectively (see Figure 6). Service IDs are used to facilitate the change of the VPC provider. A client application does not directly refer to a VPC. It uses a service ID. When another provider is selected a new VPC proxy has to be installed and the service ID is set to it via the configuration interface. The old proxies can remain in the VPC manager for the use in other applications. The mapping between service ID and VPC proxy is not always one-to-one, because several service IDs can refer to the same VPC proxy.

The **VPC registration** specifies the VPCs connected to the VPC manager. For each VPC, it contains an IDL description of the supported VPC interfaces. The registration is used for the mapping of the service IDs onto the VPCs.

The **access management** verifies the access rights of the client application to a VPC. The access rights are defined in the VPC contract. It is useful to connect the access rights with the service IDs. The inclusion of the access management into the VPC manager supports the easy change of the VPC provider, because the access rights must not be reset when installing a new VPC.

The **implementation repository** contains the VPC proxy implementations. When a VPC is called, the respective implementation is activated.

Figure 6. Mapping of the service ID onto VPCs

The **call interface** is the access point to the VPCs for the client application. The call is forwarded via the VPC proxy to the VPC provider.

The **configuration interface** serves for the installation and removal of VPCs. The interface can be used by the application designer to install VPC proxies, to configure the VPCs (if required), to assign service IDs to VPCs interfaces, and to determine access rights. The change of a VPC provider is also carried out via this interface.

The **communication module** is responsible for the interaction with the VPC providers. It can run different communication protocols. Depending on the applied security strategy several security levels can be applied to the communication with the VCP provider. The separation of the communication module makes the VPC proxies independent of the deployed communication protocol. So the VPC proxy only has to implement the access to the communication module.

6. PROTOTYPE IMPLEMENTATION

The VPC manager has been implemented as a prototype. For this implementation, we compared different middleware platforms to examine their suitability [Opit00]. We considered COM+, Java, Enterprise Java Beans (EJB) and CORBA. The limited space of the paper does not allow to describe this discussion so that we can only summarize the results of the comparison here. A COM+ based implementation would support the approach in many ways, but it is confined to Microsoft operating systems, in particular to Windows 2000. A pure Java implementation would require to implement the whole infrastructure. Enterprise Java Beans give the best support. An EJB container would also support the parallel handling of VPC calls. A CORBA implementation has the shortage that the components are not platform independent and would have to exist in several variants (see Figure 7A). We decided for a CORBA implementation with the use of a Java ORB. This implementation combines the benefits of CORBA with the

platform independence of Java (see Figure 7B). The selection was lastly also determined by the availability of licenses in our group. The implementation uses ORBIX 2.3 and JDK 1.3 including the contained Java ORB of Sun Microsystems.

Figure 7. Implementation variants for the VPC manager using CORBA

The structure of the prototype implementation of the VPC manager is depicted in Figure 8. The white fields indicate the implemented classes which are shortly described below. The class *Communicator* has not been implemented yet. Next we give some details about the class implementations. The subtitles refer to the logical structure of the VPC manager of Section 5.2.

Call Interface: The class *CallInterface* connects the client application with the called VPC. It addresses to the runtime environment to receive the reference of a matching VPC. The *CallInterface* returns this reference to the client application which uses it for further calls of the VPC. The returned reference is a reference to a CORBA object. It is registered in the CORBA name service. The use of CORBA in class *CallInterface* makes the implementation of the client application independent of that of the VPC manager. This also concerns the selection of the programming language for the client application which can be freely selected.

Figure 8. Structure of the prototype implementation of the VPC manager

Configuration interface: This interface consists of several GUI classes to allow the access to the different management modules. Via different windows the administrator can input all management data. It also provides a window for the instance manager to supervise the installed VPCs.

VPC Runtime Environment: The environment represents the kernel of the implementation. It contains the instance manager and the VPC proxies.

The **instance manager** implemented by the class **InstanceManager** administers the VPC instances. The class can be implemented differently using sophisticated management mechanisms. In the current implementation a simple strategy has been applied which creates a new instance for each invocation. When a VPC is called via the **CallInterface** it addresses to the **MappingModule** to map the service ID (see Section 5.2) on the matching VPC. After that the **VPCImplementationRepository** is called to determine from which VPC provider a VPC is supposed to be instantiated. The **InstanceManager** then creates the new instance. It also releases this link later on. To create a new instance the Java starting class of the VPC has to be instantiated first. This object creates a CORBA object for the VPC which is then used in the client application. To avoid name conflicts and to support runtime searches an own class loader is deployed for each VPC.

The **VPC proxies** are installed in the VPC Runtime Environment. The environment represents itself towards the VPC proxies as a set of interfaces via which they can access the different management modules and the communication module. If these interfaces are standardized, platform

independent VPCs can be defined. Note that it is not necessary to standardize the whole VPC manager, because the applications can principally use different managers, but the interface must be the same. In our prototype implementation the VPCs have free access to the Java ORB which is, however, not completely standardized. Therefore the VPCs are not allowed to use ORB specific extensions. Beside the CORBA interfaces the VPCs are only allowed to access to the Communicator.

Mapping Module: The class `MappingModule` implements the mapping of the service IDs onto a concrete VPC interfaces stored in the `VPCInterfaceRepository`.

VPC Interface Repository: The class `VPCInterfaceRepository` registers the VPC interfaces. For each interface, an IDL description has to be specified. The `VPCInterfaceRepository` generates the respective Java interfaces using the IDL compilers of the ORB. The registrations are persistent.

Access Manager: The class `AccessManager` checks the access rights to the VPCs for the given service IDs. It has not been implemented so far. The access rights are defined in the VPC contract. They relate to the conditions agreed between the VPC provider and customer and do not concern the principle work of the VPC manager.

VPC Implementation Repository: The class `VPCImplementationRepository` contains the implementation of the installed VPC proxies. It works similarly to the VPC interface repository. To install a VPC it is necessary to store the IDL descriptions and a Jar file with the Java classes. The IDL descriptions are used to generate the Java classes for the respective stubs and skeletons which fit to the used ORB. The Jar file contains the logics of the VPC proxy. One class has to be marked as starting class for the VPC manager to generate a VPC instance. Furthermore, the interfaces supported by the VPC have to be specified as well as a name has to assigned to the VPC. After that the VPC can be installed. All required steps are executed by the instance of the class `VPCRepository` in connection with the IDL compiler of the ORBs.

There are two possibilities to initialise the VPC at the provider. This can be either done when the VPC proxy is installed in the VPC manager or when the VPC is called for the first time.

Communicator: The `Communicator` implements the communication between VPC proxy and the VPC provider using the IIOP. Currently this is not contained in our implementation. The interaction of the VPC proxies with the providers is simulated.

7. FINAL REMARKS

In this paper we have presented with the Virtual Private Components an approach for the leasing of components. It extends the current component paradigm which is based on the purchase of components to leasing. Like in real life the leasing of components will give the user more flexibility and freedom. Component leasing is useful, if the purchase of components is (too) expensive or if topicality demands are very high. The Virtual Private Component approach extends the Virtual Private Resources concept. It enhances the applicability of the concept and makes its application more customer friendly and flexible. The strong dependencies between provider and customer were removed. It is much more easier to install a new VPC now. This allows the customer to adjust to changing market situations and to select other services when more favourable offers are available. The service provider, on the other hand, can make its relation to the service users more efficient to maintain and update the service.

We proposed a possible architecture to support the use of Virtual Private Components and described a prototype implementation of the VPC manager. Our next research steps aim at the completion of the VPC platform. Current research is focusing on the development of a security concept for VPCs. Future research will deal with refinements of the contract enforcement mechanisms, the integration of plug & play techniques, and resource accounting.

REFERENCES

- [1] [Aage99] Aagesen, A.; Helvik, B.: Towards Plug&Play Architecture For Telecommunications. In Kutvonen, L.; König, H.; Tienari, M. (eds.): Distributed Applications and Interoperable Systems II. Kluwer Academic Publishers, 1999
- [2] [ISO97] ISO, IS 13235-1, ODP Trading Function (1997)
- [3] [Miel00] Mielke, T.: Deployment and Evaluation of Virtual Private Resources. Diploma thesis, BTU Cottbus, Department of Computer Science, 2000 (in German)
- [4] [Mons99] Richard Monson-Haefel, R.: Enterprise JavaBeans™. O'Reilly & Associates, 1999
- [5] [OMGS98] OMG, „CORBAServices: Common Object Services Specification“, <ftp://ftp.omg.org/pub/docs/formal/98-12-09.pdf> (1998)
- [6] [Opit00] Opitz, A.: Design and Implementation of a VPC kernel. Technical Report 10/00, BTU Cottbus, Department of Computer Science, 2000 (in German)

- [7] [Preu97] Preuß, T.; Syrbe, J.-H.; Koenig, H.: *Virtual Private Resources - An Approach for Long-term Binding of Services*. In Proc. IEEE EDOC, Brisbane, IEEE CS Press, 1997, pp. 216 - 226.
- [8] [Preu99a] Preuß, T.: *Long-term Service Supplier Chains for Electronic Markets*. Ph.D thesis, BTU Cottbus, Department of Computer Science, Shaker, Aachen, 1999 (in German)
- [9] [Preu99b] Preuß, T.; Syrbe, J.-H.; König, H.: *Permanent Customer-Provider Relationships for Electronic Service Markets*. In Steinmetz, R. (eds.): *Kommunikation in Verteilten Systemen*. Springer, 1999, pp. 326 - 337.
- [10] [Raza99] Raza, S. K.; Bieszczad, A.: *Network Configuration with Plug and Play Components*. In *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management*, 1999.
- [11] [Sieg00] Siegel, J.: *CORBA 3 – Fundamentals and Programming*. Object Management Group, John Wiley & Sons, Inc., 2000
- [12] [Syrb98] Syrbe, J.-H.; Preuß, Th.: *An Enterprise-level Analysis of Customer-Provider Relationships & Implications for QoS Enforcement*. In Proc. IEEE EDOC, San Diego, IEEE CS Press, 1998.
- [13] [Szyp98] Szyperski, C.: *Component Software – Beyond Object-Oriented Programming*. Addison-Wesley, 1998
- [14] [Wimm99] Wimmers, M.; Mester, A.; Krumm, H.: *A Component Framework for the Configuration Management of Networks*. In Kutvonen, L.; König, H.; Tienari, M. (eds.): *Distributed Applications and Interoperable Systems II*. Kluwer Academic Publishers, 1999