

A Model for Service-Oriented Communication Systems

Bernd Reuther, Dirk Henrici
University of Kaiserslautern, Germany
{reuther, henrici}@informatik.uni-kl.de

Abstract

Like most software products, protocols are continuously enhanced and new protocols are developed. But especially new protocols of the transport layer can not be utilized widely easily. Even if the new protocols are made available, it is still necessary to adapt many applications or the protocols are not used by the majority of applications otherwise. The current situation is that only very limited enhancements of protocols are possible without changing applications. The proposed solution is to let applications use communications services only instead of protocols. A model for service-oriented communications systems that follows the concepts of service-oriented architectures is introduced. The model enables choosing and configuring protocols autonomously with regard to environmental and temporal conditions.

1. Motivation

Protocols are continuously adapted to changing requirements and new technological constraints. Also new findings help to improve the services and properties of communication protocols. In order to benefit from new or improved protocols applications must make use of them. If a protocol is part of an application (or middleware) new protocols can be adopted by application developers. If a protocol is not part of an application but part of an underlying communication system, the adoption of new protocols requires that:

1. new protocols must be made available within the communication system, which may affect a distributed infrastructure;
2. and applications must make use of them;

In case of general desktop applications the communication infrastructure and applications are developed or maintained within independent domains, e.g. by different companies. This often leads to a negative interdependency which makes the acceptance of new protocols more difficult: as long as there are (nearly) no applications that are able to use new protocols, providers of communication infrastructures

have a low motivation to support new protocols. But while new protocols are available in few environments only, application developers have a low motivation of supporting new protocols either.

According to Schumpeter [1] an innovation consists of an invention and its acceptance. Applying this to the problem described above means that developing a new or renewed protocol is the invention and the usage of such protocols in their intended domain represents their acceptance.

New or renewed protocols are accepted easily if they can be exchanged with widely used protocols transparently for applications. Unfortunately, major enhancements of protocols or new protocols often can not be used by applications or middleware transparently, i.e. without having to modify some piece of software.

Considering only few alternative protocols the solution might be simple: support all of them! Typically this would be implemented within a middleware layer, which can be reused by several applications. But this allows supporting only already known protocols. So this approach does not support the acceptance of new protocols and thus does not support innovations.

The approach of service-oriented communication systems (SOCS) presented here aims to hide all details of the utilized transport protocols. This enables to exchange transport protocols arbitrarily. But more challenging is that SOCS also aims to optimize the configuration of protocols for individual application and environmental requirements. Transport protocol stacks are selected and configured at runtime, so that local and even temporal requirements can be taken into account. The basic approach is to negotiate services between a service user and a service provider (Figure 1). The negotiation is performed by a broker. The resulting service will then be mapped to a protocol stack and its configuration.

SOCS disburdens users of having knowledge about the underlying protocols and their configuration while still delivering an optimized service. SOCS users can choose to describe a service coarse or fine grained.

Protocols are selected at runtime enabling the usage of new or specific protocols where this is possible (e.g. within a LAN) but still use more widespread protocols otherwise.

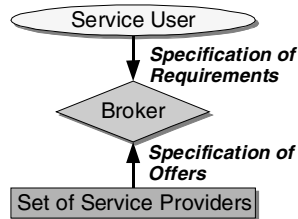


Figure 1: Brokering is based on specifications

In the next section a brief overview of related work is given. Section 3 summarized the principles of SOA which are rebuilt by SOCS. Then the model of service-oriented communication systems will be introduced. Section 5 describes how to realize the key functionality of SOCS. Finally a conclusion and outlook is given.

2. Related Work

The problem of introducing new protocols or protocols mechanisms have been addressed in several approaches, especially in the context of middleware. In the 90's some projects proposed the automatic configuration of protocols. Examples of these are the "Dynamic Network Architecture" [2], DaCaPo [3] and FuKSS [4]. Another approach for achieving a higher degree of flexibility was the Application Layer Framing (ALF) [5] used for example by RTP [6]. Further, there are protocols which are highly configurable like XTP [7] or BEEP [8] in order to fulfill various application requirements. While these approaches define flexible protocols in order to support a wide range of applications, SOCS does not define a protocol but an interface to arbitrary transport service provider (TSP).

Middleware usually makes details of the TSP transparent for their applications. Thus it is possible to introduce a new TSP without having to modify applications. But middleware usually offers services that are adapted for specific communication needs, e.g. for method invocation [9][10] or message transport [11] or even specific application classes [12]. While this is appropriate for some applications, common middleware withdraws the possibility to adapt the communication to specific application requirements.

Adaptive and reflective middleware in turn explicitly enables to adapt the communication to environmental conditions and application requirements [13]. Reflective middleware is often used in conjunction with mobile systems where some applications must be aware of dramatic changes of the communication

properties [14], [15]. But the reflection mechanisms require that applications are aware of the underlying communication system.

The concept of service-oriented communication systems is to offer a service only and hide all protocol specific details from the applications and thus make applications independent of particular protocols. This idea is not new: The OSI model already states, that "...each (N)-layer provides (N+1)-entities in the (N+1)-layer with an (N)-service..." and that it should be possible to redesign a layer and its protocols without having to change the adjacent layers [16]. But this aim of the OSI model is rarely achieved by present communication systems. While the OSI model states that each layer offers a service to the next higher layer, SOCS proposes a service-oriented interface between applications and communication systems offering a transport service.

3. About Service-Orientation

SOCS aims to broker and utilize services instead of service implementations, i.e. using communication services instead of protocols. This design approach for (software) architectures is widely known as service-oriented architectures (SOA). In order to enable a SOA, two main tasks must be accomplished: a service must be found and access to the service must be provided [17]. The task of finding a service comprises to publish and discover a service. A broker is used to match service offers with service requests (Figure 2).

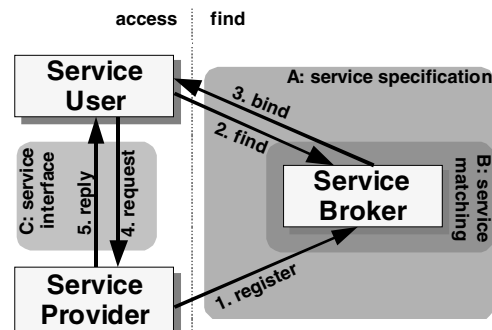


Figure 2: Model of service-oriented architectures

Step one and two require a common form of service specification. Step three corresponds to the task of the broker, i.e. to find matches between service offers and requests. Step four and five represent the usage of a service. In a SOA this requires an interface that hides all implementation specific details to the service user. A technology that supports the building of SOA must support these five steps of the SOA model which requires providing mechanisms for the following three key functionalities: the specification of services (A), the matching of services (B), and a service-oriented

interface between service users and providers (C) (Figure 2).

4. A Model for SOCS

Enabling applications to use communication services instead of transport service providers (TSP)¹ requires shifting the decision for a particular TSP from the application to an external broker. The broker will select and configure an appropriate TSP based on user requirements and available service offers. In general the “quality” of decisions depends on the available information; correspondingly the decision for a TSP is based on the offer and request specifications (Figure 1). For a „good“ decision it is necessary to provide as complete and as precise service descriptions as possible. Obviously, the information about the services also has to be correct. Thus it is essential to take into account all available information sources and to be able to specify this information.

4.1. Requirements

The specification of requirements is provided by the service user. The direct user of a transport service provider is an application (or process) running on the same host. A person who is using an application is an indirect user of the service. An application should specify all requirements that are necessary for that application to work properly, e.g. *reliable* transport of data. Whereas a person can may specify requirements based on personal favors, .g. according to *costs* or *quality*.

4.2. Offers

In order to provide a complete and precise specification of the offered communication services, the dependencies of the actually provided communication services need to be taken into account. First of all, a communication service is provided by a stack of protocols. Some protocols may be configured by options, which change the behavior of the protocols and thus change the provided service.

Further a communication service depends on its environment. The environment consists of platforms, on which the protocols are implemented, and of information about the service user. Platforms for protocols are end systems and the network infrastructure (see Figure 3). Platforms affect protocol processing, while service users affect behavior of protocols respectively. For example:

- the network affects a service, if a firewall prevents the usage of some protocols;

- the end systems affect a service, in case rare CPU time slows down the processing of protocols;
- the application affects a service, if the application's traffic characteristic is unknown (thus it can not be determined a priori how much resources will be needed);
- the user affects a service, if he does not provide a required password for authentication;
- Since the environment may significantly affect the provided communication service, the SOCS model also takes into account environmental conditions as far as they are known. Thus a communication service depends on:
 1. the protocol stack;
 2. the configuration of the protocols;
 3. the environment where the protocols are used;

Specifications of offered communication services should enable to describe these dependencies completely and precisely.

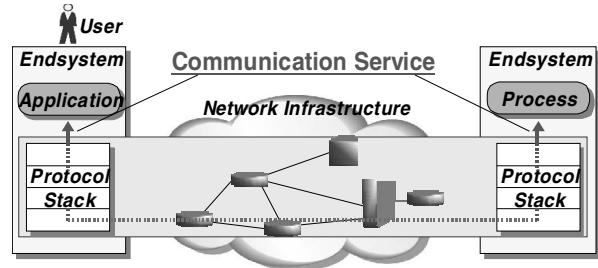


Figure 3: Implementation of communication services

4.3. The Model

Summed up, a model for SOCS must include the three key functionalities for SOA: the service specification, the service matching, and the service interface. In order to enable complete and precise specifications, the model must take into account that information about offer and requirements specifications are provided by different sources.

Figure 4 illustrates the suggested model for service-oriented communication systems. The model is composed of three basic elements, which correspond to the entities of the SOA model: the *Service User*, the *Service Broker*, and the *Service Providers*. The latter two build the service-oriented communication system. The service user is made up by an *Application* and an *Application User*.

The user accesses communication services via a common API, which is outlined in section 5.3. The SOCS model supports different and even new protocols stacks. To enable the integration of various protocol stacks an *Adapter* is responsible for mapping a general Service Provider Interface (*SPI*) to protocol specific parameters and controls. The *Adapter* controls

¹ a protocol stack offering a transport (i.e. Layer 4) service.

a protocol stack and enables its usage, e.g. mapping a host name to an IP address or performing the authentication of a user with a RADIUS server.

The data flow itself is not affected by the SOCS model, so that an efficient transport of data between applications and protocol stacks is possible.

Specifications of offered communication services are not part of the protocol stacks but are provided separately, e.g. by a local database or configuration files. The *PS Offer* describes the service of a protocol stack. This includes the description of protocol options and their influence on the delivered service. The specification of communication services is static, but services may depend on dynamic information about the hosts (*Host status*) or the network (*Net Status*). Such dynamic information could be provided by monitoring tools. For example, a static description of a service can specify an expected *delay* depending on the current network load.

The task of the *Broker* is to find matches between a given requirement specification and the offer specifications. Only offers that fulfill all necessary requirements will be regarded as a match. Then all matches will be rated on the basis of desired requirements in order to determine the best match. The

resulting offer will be replied to the API, which in turn will contact the adapter of the selected protocol stack.

5. Key Functionality of SOCS

This section provides a brief overview of the SOCS API and presents a flexible and scalable specification form for transport communication services.

5.1. Accessing Communication Services

A service-oriented communication interface must hide protocol specific details to the users. But common interfaces like the BSD socket interface [18] require protocol specific parameters at least when optimizing the provided service. For example when using the TCP_NODELAY option. Furthermore, applications have to use different sets of functions for connection oriented and connection less services. Thus a new API has been defined and implemented to be used for SOCS. The API defines basic functionality for flow handling, sending and receiving data and an interface to the broker. Names are used instead of addresses, but port number are still allowed for compatibility reasons (see [19] for more details).

5.2. Specification of Communication Services

The following requirements for the specification of services are regarded to be important for SOCS:

- The specification form should not rely on a fixed semantic, since this may not be suitable for the description of new services. A more flexible specification form would also enable later adaptations to yet unknown practical needs.
- Requirement specifications should allow fine granular and precise descriptions of the desired service as well as simple and coarse grained descriptions where appropriate.
- The specifications must enable a service broker to decide which service is the most suitable for an application.

Sets of Properties

We propose to specify communication services by sets of properties. Thereby the properties must be related to observable behavior for the service user and must not be related to protocol mechanisms.

The specification of requirements distinguishes between necessary and desirable requirements. Thereby applications rely on the fulfillment of the necessary requirements. The specifications of offers distinguish between properties that are inherently given by the definition of the protocols and qualitative properties which may depend on the environment of protocols. For the specification of communication services we therefore propose to differentiate between

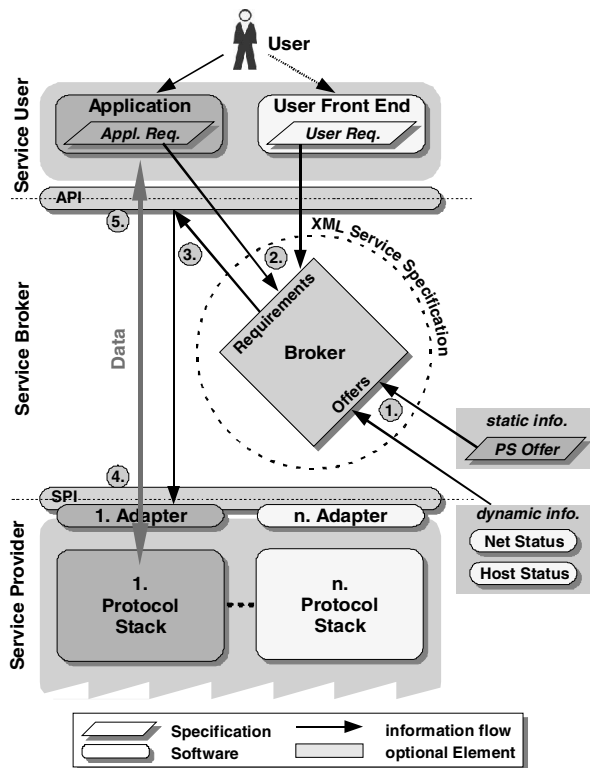


Figure 4: Model for SOCS

inherent properties and qualitative properties (see also [20]):

Inherent properties are guaranteed and do not contain any rating of a service. It must be clearly determinable whether a service has a specific inherent property or not. Examples of inherent properties are: *reliable transport* or *confidential transmission*. In a requirement specification inherent properties represent the necessary properties.

Qualitative properties are not guaranteed and explicitly rate a communication service. Examples of qualitative properties are: *loss rate* or *degree of confidentiality*. In a requirement specification, qualitative properties describe desired service properties.

The sets of possible inherent and qualitative properties are disjoint, even though both types of properties can be related to each other. For example: *reliable transport* implies an optimal *loss rate* and the *quality of security* only makes sense if there actually is a *confidential transmission*.

Each communication service CS_i is specified by a set of inherent and qualitative properties:

$$CS_i = \langle Pi_i, Pq_i \rangle \text{ with} \quad (3)$$

$$Pi_i = \{ pi_{i1}, \dots, pi_{in} \}; Pq_i = \{ pq_{i1}, \dots, pq_{im} \}$$

In this specification, Pi_{ij} denotes an inherent property and Pq_{ik} a qualitative property. With this definition, two services CS_i and CS_j are considered to be identical if:

$$CS_i = \{ Pi_i, Pq_i \} = \{ Pi_j, Pq_j \} = CS_j$$

CS_i and CS_j are considered to be equivalent, if:

$Pi_i = Pi_j$ i.e. CS_i and CS_j are exchangeable, because both service fulfill the same necessary requirements.

A service CS_j is considered to be a refinement of a service CS_i if:

$Pi_i \subset Pi_j$ i.e. CS_j can always replace CS_i but not vice versa.

With these definitions a service broker is able to compare requirements and offers basically by comparing their sets of properties. Comparing the inherent properties determines if an offered service can be used by an application, whereas the qualitative properties are used to find the most suitable service. Examples for properties of TSP are described in [21].

Specifying services by sets of properties implicitly allows adjusting the level of detail, simply by specifying a different number of properties. A requirement specification should contain only as many

properties as necessary. But an offer specification should obviously always be as precise as possible.

Specification Framework

In order to achieve a high degree of flexibility, only the syntax of a specification is defined. The semantic is referenced by a URI instead. Each property is named and identified by a URI and may be described further by quantitative values. A unique syntax is used for inherent and qualitative properties, respectively. This enables the broker to compare services without having to interpret the semantic. At runtime the service broker handles a URI like a keyword. Using a fixed syntax and references to external semantic definitions is more like a specification framework than a specification language. But the resulting flexibility enables to add and change types of properties without having to change the application or the service broker.

Quantification

Some inherent properties and all qualitative properties need a specification of quantitative values additionally to their URI. Even though the meaning of the values depends on the semantic of the specific property, the syntax for the representation of values can be defined.

Each inherent property pi is specified by: $pi = \langle URI, lb, ub \rangle$ requiring or guaranteeing that $\exists i$ with $i \in [lb, ub]$. Using an interval simplifies the definition of boundaries. For example, a requirement specification may contain an inherent property:

$pi_R = \langle http://domain/BinMsg, 500, - \rangle$ meaning that the lower bound for the binary message size must be 500 bytes. An offer specification may define:

$pi_O = \langle http://domain/BinMsg, -, 1400 \rangle$ meaning that the upper bound for the binary message size is 1400 bytes. Then pi_R matches pi_O because the URI is identical and the intervals overlap.

Ideally qualitative properties should also be specified by precisely measurable values. But sometimes this is not possible, for example the precise *average delay* of a connection in an IP network is in general hard to determine. Usually application developers have to select transport protocols without knowing precise qualitative properties of all environments where their application will be used. Thus the protocols are selected by commonly assumed differences between protocols, e.g. one usually assumes that the delay of UDP/IP is lower than that of TCP/IP. Relations like protocol A is better than B (according to a specific property) are often easier to determine, but are obviously less precise than using measurable ratings.

The quantification for qualitative properties in SOCS enables to use both representations and even to mix them:

1. All offered TSP_i are rated according to a single qualitative property P_q by an appropriate function $Q_{Pq}(TSP_i) \rightarrow \mathbb{R}$ where increasing values denotes increasing quality.
2. A linear mapping R:
 $R[\min(Q_{Pq}(TSP_i)), \max(Q_{Pq}(TSP_i))] \rightarrow [0, 1]$
 specifies the quality of a TSP according to P_q by a value $q \in [0, 1]$ with 0 ~ “worst quality” and 1 ~ “best quality” available (example Figure 5)
3. The steps 1 and 2 are performed for each qualitative property.

A qualitative property *pq* of a requirement or offer is then specified by:

$pq = \langle URI, qr, q0, q1 \rangle$ with $qr \in [0, 1] \wedge q0 \leq q1 \in \mathbb{R}$
 whereby *q0* and *q1* are optional parameters which represent precise and measurable values for the quality levels 0 and 1 respectively. In a requirement *qr* specifies a weight for a property and in an offer it specifies a quality level. In both cases *qr* is independent of the measuring unit used by Q_{Pq} . If *q0* and *q1* are specified in an offer and a requirement then *qr* of an offer can be adjusted according to *q0* and *q1* of the requirement. Meaning to use the same definition of “worst” and “best” quality. Thus using the optional parameter *q0* and *q1* enhance the preciseness of a quality description.

Because this framework specifies the syntax for specifications only, the methods of rating qualitative properties can be changed without changing the broker. Thus the rating of a qualitative property may be based on personal estimations and may be replaced by sophisticated benchmarks, when these become available.

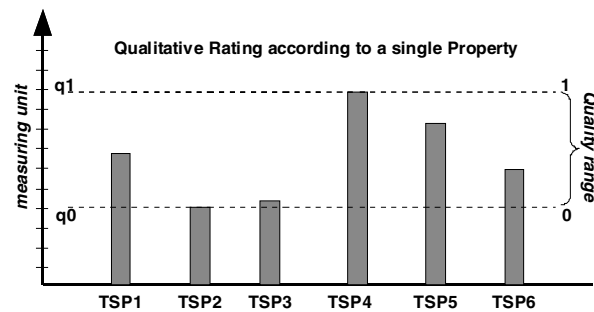


Figure 5: Quality rating example

Comparing requirements and offers

Comparing the inherent properties of a requirement and an offer determines whether the offer is suitable or not. If the requirements specification is equivalent to

or a refinement of the offer then the offer matches the requirement.

Qualitative properties are used to find an optimal service. Thus it is not sufficient to compare single properties with each other. Each qualitative property defines a qualitative rating *qr*. The *qr* of an offer specification defines the qualitative rating of a TSP according to other TSP, but the *qr* of an offer specification is used to rank the importance of a property according to other properties. This is expressed by the metric:

$$QM(Pq_R, Pq_O) = \sum_{\forall_j} qr_{R_j} qr_{O_j} \text{ with } \\ URI(pq_{R_j}) \text{ equals } URI(pq_{O_j})$$

Note that QM considers only those qualitative properties which are defined in the requirement and the offer specification. Assuming a small default value for qr_{R_j} whenever an offer exists without a corresponding requirement, then a service will also be optimized according to properties that have not been requested explicitly, using a minimal priority.

The presented framework enables to broker communication services while still being highly flexible according to the specification of services. It is possible to adjust the preciseness of specifications by the number of properties considered and by using abstract or precise quality descriptions. Furthermore it is possible to add new types of properties without having to modify applications or the broker.

5.3. Specification of Transport Service Providers

The previous section has described how to specify a single communication service. But the service of a TSP depends on its configuration and the environment of the TSP. So a TSP may offer many different services which may differ significantly. Listing all services that may be provided by TSP is not feasible. Our approach is to specify a basic service which represents the service of a protocol stack in its default configuration and a default environment. Additionally options for modifying the basic service are specified, which are called a service option. It is taken into account that some service options may exclude other options.

Further the effects of the environment on the offered communication service must be specified. Since the environment does not change the protocols² only the qualitative properties of a service are affected. Additionally, the environment may prevent the usage

2. At least all modifications of the exchanged data units within the network (e.g. within DiffServ or MPLS domains) must be transparent for the applications, this implies that inherent properties of a service are not eliminated.

of protocols or protocol options. For example, some older routers drop TCP packets when ECN is used; in this case the environment denies the use of the ECN option. Or a user may have to authenticate him before a certain service is available. So the environment may affect an offered service by two means: the quality of the services may be changed or the provision of a service may be prevented.

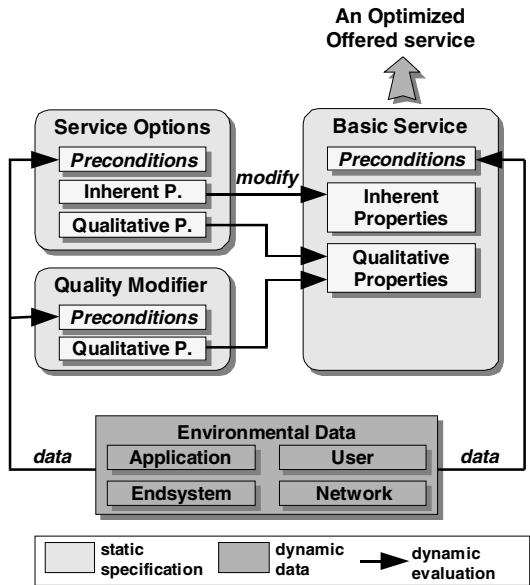


Figure 6: Evaluation of a Service Specification

The influence of the environment on the quality of a service is described by *quality modifiers* which specify a modification of a service similar to service options. Whether such a quality modifier must be applied is dependent on current environmental conditions. Preconditions enable to check whether specific environmental conditions are available. In the same way preconditions are used to specify whether basic services or service options are usable or not. Figure 6 illustrates the resulting dependencies of an offer specification for a TSP.

The preconditions are boolean expressions which utilize environmental data. For example:

- A TSP may require the environmental data *login* and *password* in order to allow an application to use the TSP.
- An option for resource reservations could be used if the application specifies its traffic characteristic. The resource reservation option will change the properties of the basic service.
- Status information about the current network load may be used to adjust the qualitative properties *Delay* and *Lossrate* of a TSP.

The availability of environmental data may differ in dependence of the used application, system platform or network environment. In order to enable a proper usage of preconditions it must be avoided that missing environmental data are considered as a failure. Simple functions like “*if exists*” in addition to boolean operators enable robust preconditions.

5.4. Brokering of Communication Services

Brokering of communication services means to select a TSP and its configuration for a given requirement with respect to known environmental conditions. In general, a broker performs the following steps in order to find the most appropriate TSP:

1. The broker joins the application and user requirements to one requirement specification.
2. Find all TSP offering a matching service.
3. Optimize the configuration of all remaining TSP according to the user requirements.
4. Build a priority list of all services that fulfills at least all inherent requirements.

In our implementation of the broker steps 2 and 3 are combined for performance reasons: The broker assigns a rating $\langle R_I, R_Q \rangle$ to each offered TSP. R_I is the number of inherent properties required and R_Q is the qualitative rating according to the metric QM. The broker initializes the rating by the specification of the basic service and performs the following steps:

- If a Service Option does not exclude another one and the option increases $\langle R_I, R_Q \rangle$ then the option will be used for this TSP. Increasing R_I is favored for increasing R_Q if there is a conflict.
- Service Options which exclude other options are considered at last. If two or more options are able to increase $\langle R_I, R_Q \rangle$ a recursive search for the optimal configuration is required.

After these steps a priority list of the most suitable service is available. An application which actively establishes a communication relation would use the first (i.e. the best) service of the priority list. If it turns out that this TSP can not be used the next service of the list be utilized. Note that this means that there are information missing about the environment. When opening a passive communication relation then all suitable TSP should be used. If too many TSP are suitable one might limit the number of TSP.

We suggest marking a default TSP that should always be used when opening a communication relation passively, in order to guarantee that widely used TSP are supported if appropriate. Such marked TSP could also be used during an active open, in order to reduce the number of trials, when the most appropriate TSP has already failed.

An outcome of the service brokering is also a set of Service Options for each TSP which represents the optimal configuration of the corresponding TSP. The mapping of Service Options to actual protocol parameters is the task of the Adapters (see Figure 4)

6. Conclusion and Outlook

The goal of SOCS is to support innovations of new protocols by making the usage of new or renewed protocols transparent. Therefore two main functions are required: applications must become independent of protocols, while it should still be possible to optimize a TSP according to user and applications needs and the given environment. The latter demands a specification form for TSP which is flexible and extendable and is also not related to protocol mechanisms. Service specification of offerings can reference to external data of dynamic information about the environment. The definition of Service Options enables the broker to determine an optimized configuration of a selected TSP.

For highly specialized applications it should not be expected that an automatically chosen and configured protocol stack is as efficient as protocols that are chosen and configured by experts. But for common applications that are not designed for a single well-known environment, the SOCS model improves the current practice of using always the same protocols with the same configurations.

In the future the specification of qualitative properties should be based on benchmarks to improve their accuracy.

References

- [1] J.A. Schumpeter, *The Theory of Economic Development Cambridge, MA. Harvard University Press* 1934
- [2] S. W. O'Malley, L. L. Peterson., "A Dynamic Network Architecture", *ACM Transactions on Computer Systems*, Vol. 10, No. 2, May 1992, pp. 110-143
- [3] M. Zitterbart, B. Stiller, A. Tantawy, "A model for flexible high performance communication subsystem", *IEEE Journal on Selected Areas in Communications*, 1993, pp. 507-518
- [4] T. Plogemann, B. Plattner, M. Vogt, T. Walter, "Modules as Building Blocks for Protocol Configuration", *Proceedings International Conference on Network Protocols (ICNP'93)*, San Francisco, USA, October 1993, pp. 106-115
- [5] D.D. Clark, D.L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols", *Proc. ACM SIGCOMM 1990, Computer Communication Review*, 20(4):200-208, September 1990.
- [6] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", *IETF RFC 3550*, July 2003
- [7] T. Strayer, ed., "Xpress Transport Protocol Specification", Revision 4.0b; *XTP Forum*, 1998
- [8] M. Rose., "The Blocks Extensible Exchange Protocol Core", *IETF RFC 3080*, March 2001.
- [9] Object Management Group: "The Common Object Request Broker: Architecture and Specification" 2000. URL: <http://www.omg.org>
- [10] Sun Developer Network, "Java Remote Method Invocation (Java RMI)", URL: <http://java.sun.com/products/jdk/rmi/>
- [11] G. Banavar, T. Chandra, R. Strom, D. Sturman, "A case for message oriented middleware" *Distributed Computing 13th International Symposium*, September 1999, LNCS Vol. 1693, pages 1-18.
- [12] B. Reuther, A. Ebert, P. Müller, H. Hagen, "Interactive Poster: A powerful communication framework for distributed virtual reality", *IEEE Visualization 2002*, October 2002.
- [13] L. Capra, W. Emmerich, C. Mascolo, "Reflective Middleware Solutions for Context-Aware Applications", *Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, 2001, LNCS, Vol. 2192, pp 126-133
- [14] T. Fitzpatrick, J. J. Gallop, G. S. Blair, et al. "Design and Application of TOAST: An Adaptive Distributed Multimedia Middleware Platform", *Proceedings of the 8th International Workshop on Interactive Distributed Multimedia Systems*, 2001, LNCS, Vol. 2158, pp 111-123
- [15] F. Kon, F. Costa, G. Blair, R. H. Campbell, "The case for reflective middleware," *Communications of the ACM*, vol. 45, pp. 33--38, June 2002.
- [16] ISO/IEC 7498-1, ITU-Rec. X.200, "Information Processing Systems – OSI Reference Model – The Basic Model", *International Organization for Standardization*, 1984
- [17] World Wide Web Consortium, "Web Services Glossary", URL: <http://www.w3.org/TR/ws-gloss/>
- [18] W. R. Stevens, *UNIX Network Programming*, Volume 1, Second Edition: Networking APIs: Sockets and XTI, Prentice Hall, 1998.
- [19] D. Henrici, B. Reuther, "Service-oriented Protocol Interfaces and Dynamic Intermediation of Communication Services", *2nd IASTED International Conference on Communications, Internet and Information Technology*; CIIT 2003; Arizona; USA
- [20] B. Reuther, D. Henrici, M. Hillenbrand, "DANCE: Dynamic Application Oriented Network Services", *30th Euromicro*, August 31st - September 3rd, 2004
- [21] D. Henrici, "A Universal Scheme for the Classification of Network Services", *Diplomarbeit, University of Kaiserslautern*, 2002