Published as RACE Common Functional Specification L313, 1994

Service Interoperability in Teleconferences

Jörg Ott · Carsten Bormann · Ute Bormann

Technische Universität Berlin · Universität Bremen

jo@cs.tu-berlin.de · {cabo,ute}@informatik.uni-bremen.de

8 February 1994

1. Introduction

In recent years, many teleconferening systems have been developed providing a wide range of functionality such as audio and increasingly video conversation as well as functions to share and cooperatively manipulate computer-based information: telepointing tools, shared whiteboards, joint editing facilities, etc. Such teleconferencing systems mostly compose several software modules for the different functionality to achieve their total functional range. These software modules are either loosely coupled (possibly even totally independent), or they are built in a way that they fit exactly (concerning interfaces, protocols, etc.) in this (and only this!) teleconferencing system. In addition to those compound teleconferencing systems, hundreds of groupware tools have been developed that may be regarded as "stand-alone" systems because they are not designed for (other than independent) coexistence with other systems, system components, or groupware tools. As a consequence, usually no two different groupware tools (that have been *built for cooperation*) are able to interoperate (i.e. to *cooperate*). The main reason is to be seen in the lack of standards for group cooperation protocols. But even if some standards are available, interoperability is not necessarily achieved as there is no framework for (service) interoperability.

In this paper, we define a notion of services as an abstraction from the implementation of groupware tools which only reflects their functionality. We provide several interpretations of interoperability and we investigate the aspects that may lead to interoperability of services (i.e. groupware tools) if considered (or prevent it, if ignored). We provide a starting point for a framework for the abstract description of an interoperability service.

This paper is organized as follows: first, section two provides definitions for the terms "service" and "interoperability", investigates what is needed for services to be interoperable, and, finally, identifies two dimensions of service interoperability for further study within the scope of this paper. The sections three and four describe these two dimensions respectively, and provide an idea how to approach them. Section five summarizes the main points of this paper and suggests further work on this topic.

2. Services and Service Interoperability

In physical conferences several persons usually meet to solve problems. They do so by means of talking, distributing documents, making presentations, voting and the like. For modeling a (physical) conference, we call each of these human interactions an *activity*. The entire conference consists of one or more activities (which may be "ongoing" sequentially and / or in parallel). In teleconferences, one needs to provide (nearly) the same means of interaction as in physical conferences, i.e. for each *activity* found in the real world an electronic counterpart must be modeled. This leads us to the notion of *services*.

Service

The term "service" is used with various meanings so that we need to clarify our understanding of a service (and the possible relations to other usages of this term):

- In conjunction with the model of Open Systems Interconnection the term service is used with two different meanings:
  1. A service is the entire set of functionality offered by the *Distributed Abstract Machine* (*DAM*) of a layer *n* to the layer *n+1* of the OSI model by employing the service of layer *n-1* and adding further functions.

2. A service is a single function of the entire functionality provided by a layer *n* according to the definition given in 1. Such a service may be composed of one to four *service primitives* (*request, indication, response, confirmation*). Formerly, this component was called *service element* so that no ambiguities could occur; unfortunately, this a term was dropped but nevertheless we use it in this paper for the sake of clarity.

So, a service is composed of several service elements each of which offering elementary functionality mostly split into two or four service primitives.

- In terms of the PTTs a service is a certain capability offered by the public network provider at the endpoint (i.e. the socket) of a customer. Examples for such services are Telephone, Telefax, Teletex, and the like.
- In microkernel operating systems a service is the functionality offered by a *server* or an *agent* to user processes, regardless whether we are concerned with communication functionality or not (e.g. access to the hard disk).

All the above descriptions have in common that a service consists of a set of functions offered by some entity to others. In OSI terms the entity offering the service is called *service provider* the one accessing the *service user*. We use the same basic notion of a service, and, in the following, we will stick to the OSI terminology of service user and service provider.

In our understanding, a service defines the functionality of a means to interact with other participants (or, for controlling purposes, with the conferencing system), i.e. to carry out an *activity*. This means that services provide all kinds of interaction among participants in a teleconference. The (non)availability of a service is directly recognized by the user in that she may (or may not) perform a certain action, i.e. a service provides the user with a set of functionality – directly via a user interface, or indirectly via other services. Finally, services are used as an abstraction from their implementation, and we focus on implementation independent definitions as far as possible, e.g. we do not consider (or even restrict) whether services are built in software or in hardware.

The (*abstract*) *service* defines the functionality offered by a service provider to a service user. This functionality is accessible at each (of many) endpoints, via a *service access point*, *SAP*. In a specific implementation a service is provided by one or more *service (peer) entities* that communicate via a *service protocol*, and altogether form a *specific service provider (entity)*. The interface offered by a service entity at the (communication) endpoints are called *service interfaces*.

We distinguish *atomic services* and *compound services*. Atomic services cannot be subdivided into service components that are useful for the user by themselves, i.e. an atomic service forms the basic element of which the functionality for teleconferencing systems is built[1] – nevertheless, they are not be mixed up with OSI *service elements* as an atomic service may consist of several service elements. Compound services are combined of several atomic and / or other compound services and offer more complex functionality to the user. The teleconference system may be thought of being the (virtual) “top” service integrating all other services to form the conferencing system. Figure 1 depicts a possible graph of services.

It should be noted that the borderline between atomic and compound services is difficult to be drawn. In particular, different teleconferencing environments with different targets will inevitably get different sets of atomic services – simply because different user requirements lead to another classification of what is “useful” to user. However, some classification is needed (and this must be a global one, too, as we will see later), and, consequently, one of our work items has to consist of defining atomic and some compound services.

Examples for services (without defining whether they are atomic or compound by now) are video conversation, audio conversation, document distribution service, and joint editing.

---

[1] Note that at this point we are not concerned with aspects of system design such as modular structuring of a service, programming interfaces, etc.

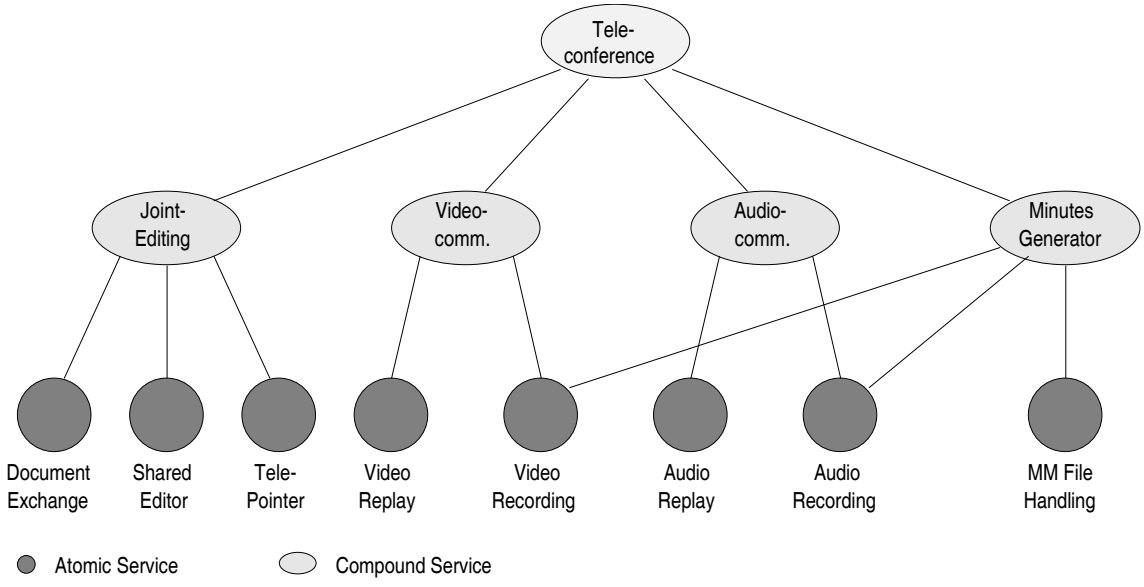

Figure 1: Atomic and Compound Services

Interoperability

We start our discussion on service interoperability by giving an operational definition of "service interoperability":

Definition:

> *Service interoperability is the capability of service entities that are parts of a conferencing system (or stand-alone systems) to establish a cooperation relationship to one or more peer entities in the context of a teleconference without the need for a prior multilateral agreement among the communicating parties outside the teleconference the cooperation process belongs to.*

Additional Requirements:

> *Service interoperability is provided regardless of the (hardware) architecture, the operating system, the available means of communication or other system specific aspects of the computer systems involved.*

This definition clarifies that we want to consider interoperability from the system's point of view in opposition to the user's or user interface's: the latter might address the capability of several services to be presented in a common user interface, to be conforming to certain style guide, and the like. As provision of a service and its presentation to the user are two independent issues – dealing with technical and human factors, respectively – we leave all the considerations of user related aspects to others and focus on the technical solution of interoperability.

Interoperability from the system's point of view must take into account the following issues (which are complementary but reflect different aspects of system design):

1. *Platform independence*. The services must be able to cooperate independent of which platform they are running on. The platform includes the hardware architecture as well as the operating system, i/o devices, etc. However, it is a prerequisite that the environment of a service must provide the functionality required for its execution. The implementation of a service has to be portable to any system that fulfills a certain set of minimum requirements (such as multitasking capabilities, provision of IPC mechanisms, network connectivity and the like).

2. *Interface definition*. If parts of services are provided as software modules / libraries (that are joined on a source or object level later on) to simplify development of (higher) compound services these modules / libraries have to conform to defined interfaces[2].
3. *Network and communication infrastructure*. A fundamental requirement is that the service is able to communicate via any network as long as the network provides a sufficient *Quality of Service* (e.g. throughput, delay, etc.). On top of different networks there may reside different communication protocol stacks (often because of the different networks!). Consequently, another (related) requirement is that services are capable of running on top of all those stacks.
4. *Information encoding*. Services are only able to interoperate if they employ the same information encoding schemes for both control and data information.
5. *Control protocol*. Besides the encoding of the information intended for the user of a service a control protocol must be available for the exchange of state information requests and the like. Interoperability can only be achieved if the services use the same protocol (with the same or a compatible functional range).
6. *Service identification*. A service is only able to interoperate with another service if it can find out about the other's existence and gets to know how to contact it.
7. *External control and coordination*. If multiple services are active at the same time in a teleconference they should be able to coordinate their behaviour (e.g. joining another conference simultaneously). This could be encouraged by a per user conference control service that controls the other services.

Item 1 is an obvious requirement that is requested for most implementations and must be considered anyway when implementing a service; for abstract definitions this requirement is not applicable. There are no direct impacts on the service interoperability except that information exchange must be performed in a system independent way (a problem which is solved since the invention of the presentation layer of the OSI reference model)[3].

Item 2 directly affects the implementation and interface specification of services or service components. However, this is primarily relevant to bodies defining Application Programming Interfaces (APIs) for such services, and has mainly to be considered for a specific implementation.

Items 3 is usually addressed by a system's communication software that provides access to a (set of) network(s) and implements one or more protocol stacks. Having different networks and protocol stacks across which interoperability shall be achieved requires

- the provision of internetworking so that the service entities are basically able to address their peer(s), and
- a unified service interface with identical functionality independent from the underlying networks and protocols.

Both these aspects may be solved by an additional standardized (multipoint) communication stack that provides a well-defined set of functionality all the "higher" services can rely on. For this additional stack a (standardized) mapping to each of the underlying protocol stacks has to be provided, and potential differences in the functional range of the respective protocols have to be compensated. Provision of these

---

[2] Note, that neither item 1 nor item 2 do imply independence of programming languages. Such a goal is unnecessary (and, in fact, not achievable!) as today there is an implicit agreement on certain programming languages to be used for (system) implementations of that type, e.g. C or C++. Consequently, an interface definition for these languages is sufficient.

[3] Nevertheless, there are still problems to be solved. Different window systems, for example, provide different means for user interaction, presentation of information and the like. Consequently, service entities that interact with window systems need to know the window systems' capabilities, and differences in these capabilities may prevent interoperability, unless the same window system is made available on all platforms – which, in fact, means converging the platforms rather than being platform independent. Similar considerations may apply to other areas as well.

features is inevitable for interoperability (and thus is a requirement we assume to be already provided!). The underlying protocols and networks are beyond our control, and, consequently, are not subject to further discussion in the context of service interoperability. What we have to do here, is to ensure that our definitions for service interoperability fit on top of the communication environment that is provided.

Item 4 addresses the availability of standard encoding formats that are required for "open" information exchange. Here, standardization has developed quite differently in the various areas: for audio and / or video communication several standards have been developed (and their number is still increasing) whereas for "document-based" cooperation services such as telepointing or joint editing there are no standards – not even proprietary ones – available at all (an important and urgent problem that must be tackled by the standardization bodies). For the interchange of complete documents containing textual and graphical information a variety of formats have been defined with only some of them (e.g. ODA for documents, JPEG for still images) are international standards[4]. If no standards are available the situation is even worse because a potential description or negotiation service has no basis to operate on[5]. It is not up to us to start defining these required encodings (at least not within this study group) so that we have to wait for the standardization bodies. However, we get an action point to collect information about all the available encodings for distributed cooperation.

While item 4 is concerned with the contents encoding item 5 addresses the entire application protocol (some PDUs of which contain the information encoded as described in item 4). Again, two service entities must agree on a common protocol (operations and encoding) to be able to interoperate, and, again, only few standards (such as the ITU H-series for videotelephony) are available to negotiate the protocols and options to be used. In general, such facilities are missing, and, as a consequence, one finds manual switches or command line options to select a certain encoding instead. This is to be seen as one of the crucial points to provide service interoperability and, therefore, forms another (important) action point for our work group.

Item 6 deals with the problem of identifying a service (on a target system) within a teleconference. By now, groupware applications and conferencing systems are often governed by "hard-wired" conventions defining how to reach a certain component of a system (e.g. to ensure that the text document is sent to the text editor on the remote machine and not to the audio device), or this information is published via media like USENET NetNews or via e-mail distribution lists and has to be provided manually by the user when starting the system (component). This results in static configurations of conferencing systems in which all components are / must be "well known" and are mostly not exchangeable (this applies to the protocols and encoding formats as well). Basically, this describes the need of providing all the capabilities of a certain user's system (that is the total numbers of services available and their parameters) and / or the entire conference to services upon request, which is another major issue we have to address to allow service interoperability.

The final item 7 addresses the fact that most of the services available so far are only designed for direct interaction with their respective counterpart but are usually not capable of communicating with their (local) environment. Such a second interface is unimportant for "conventional" point-to-point services, such as file transfer or phone calls, because these services have usually provided their functionality in an isolated way, i.e. they were not depending on coordination with other services. This is different for teleconferencing systems where all the services form the compound service of a certain teleconferencing environment and often will be interdependent as far as participation in conferences is concerned. If all the activities (i.e. all the services currently in use) within a teleconference shall be controlled by the user by means of a single action (e.g. pressing the "leave" button on his user interface) rather than having to perform this action once for each service (e.g. pressing ten different "leave" buttons) there has to be a means for coordinating all the single services. Such a coordination of all conference-global actions, i.e.

---

[4] However, practice shows that especially for text documents the standards available today are not sufficient simply because they are not commonly accepted and often not even implemented. As a consequence, we are confronted with many different proprietary data formats that are not compatible. Fortunately, they are often convertible to a reasonable extent so that the exchange of information is possible provided one entity knows about the other one's encoding format and a converter is available.

[5] We will elaborate on this issue in section three.

actions that have an impact on all the (active) services, also improves consistency among the states (with respect to the conference) of all the services. This second interface must support some kind of conference control protocol allowing the services to gain access to global information about the conference and to be informed about changes in the conference state. Unlike items 4, 5, and 6 which are somehow related (and have to be addressed in common) this item forms a separate issue that is related to the conference management. Nevertheless, this topic has to be addressed in the context of service interoperability (not just because conference control, too, is to be regarded as a service).

Dimensions of Service Interoperability

The conclusion to be drawn from the above discussion is that – if we ignore those areas that are beyond the scope of service interoperability and those that may not be influenced – we basically get two dimensions in which service interoperability has to be provided (shown in Figure 2):

1. *Horizontal interoperability* means the interoperability among service entities (abbreviated as "S.E." in Figure 2) located on different hosts in order to provide the service to the users. According to the items listed above, at least means for naming and describing service entities available on one host and their capabilities by means of attributes have to be provided. These means must allow to identify a certain service entity (from a remote site) so that a cooperation relationship may be established.
2. *Vertical interoperability* means local (i.e. on a single host for a single user) coordination of (otherwise independent) service entities belonging to different services in order to achieve coherent reactions of all service entities as response to the behaviour of the user in a conference. This could be done e.g. by means of some dedicated entity (not depicted in Figure 2).

Of course, there may be mixtures of both interoperability dimensions such as interworking of different services (across system boundaries) which have to be dealt with appropriately, too. However, these issues may be neglected for the first step to finding (independent) solutions to achieve interoperability in the two respective dimensions.

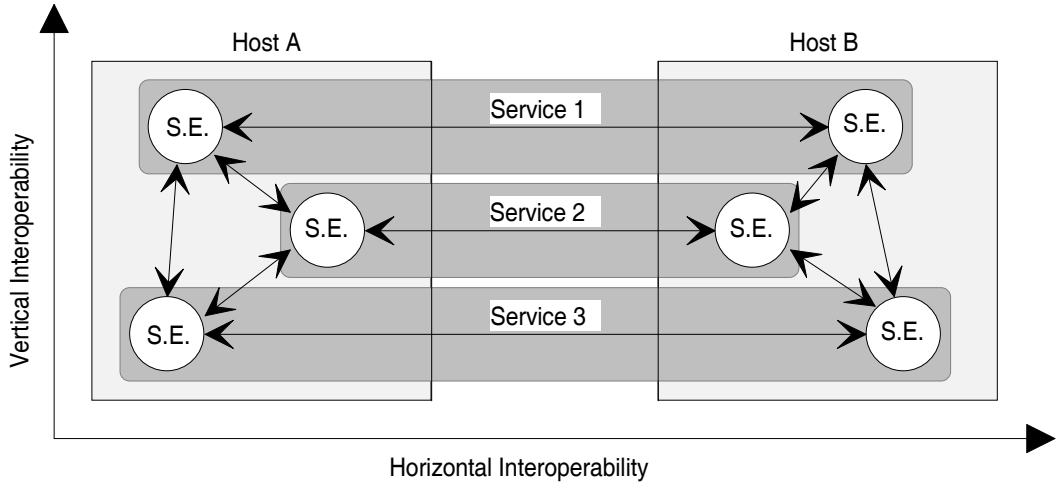

Figure 2: Dimensions of Service Interoperability

It is important to note that the integration of neither of the above items does affect existing application specific protocols so that, consequently, these need not be changed[6]. It is a very important aspect to have the service interoperability features provided entirely as an *add-on* to an existing (as well as a newly

[6] However, cooperation protocols of distributed applications usually incorporate protocol elements for controlling the cooperation. It must be considered that such protocol elements might conflict with other means of control introduced by the mechanisms provided to realize vertical and horizontal interoperability. These aspects are detailed in section four.

defined / implemented) service; a feature which is stressed by the requirement that services that have been interoperable in one environment must preserve this feature even if one of them is altered to enable interoperability to others as well. The same applies to the coordination of control.

3. Horizontal Service Interoperability

It is obvious that only two service entities that utilize the same protocols, encoding formats, etc. are capable of interworking. The main problem arising from these requirements is not that service entities do not implement more than one / all of the possible choices or even that the choices have to be reduced to a single protocol in each application area (both approaches, in fact, would be far from reality) but merely that currently there is no mechanism to find out which services are available and which protocols, formats, etc. they do implement, and how to contact the respective service entities.

A conferencing system may composed of many service entities (that provide the respective services) each of them used to perform a certain activity. This scenario is depicted for a point-to-point cooperation in Figure 3. For the beginning, we assume that only a single cooperation relation may be active at a time. This scenario may be extended to a multipoint configuration with multiple concurrent conferences in a straightforward way which will be dealt with afterwards (however, this extension will not affect the problem depicted in Figure 3).

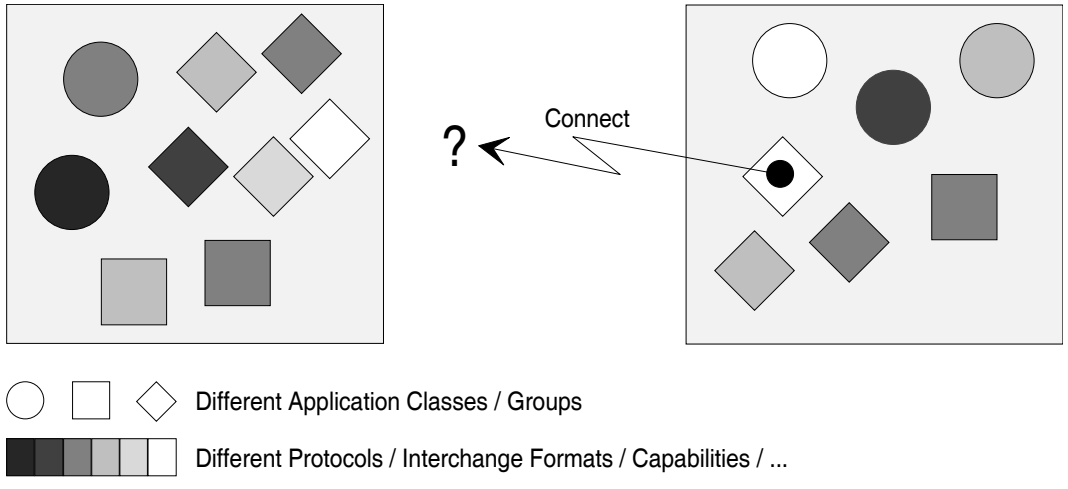

Figure 3: Issues of Horizontal Interoperability of two Conferencing Systems

Basically, service entities may behave as an active or passive component. If they are passive they simply "bind to a certain address", "register with a certain name", etc. and wait to be contacted / instantiated. This means that they require a means that allows them to provide (unknown) other service entities that potentially want to contact them with knowledge about their existence. If they are active, they try to connect – i.e. *to contact* – a peer entity on a remote host (that is waiting to be contacted). The steps the service "white rhomb" in Figure 3 has to perform – the *contact procedure* – before successfully connecting to its peer entity in a point-to-point environment are the following[7]:

1. The service must be told which host (the target host) to connect to, and it must figure out the target host's address.
2. When connected to the target host the service needs to find out whether there is a similar (this means belonging to the same service class) service available on the target host.

[7] We exclude problems such as "call collision" from this paper as these are technical problems that are not directly related to service interoperability itself.

3. If at least one remote service of this type exists, the local service must figure out if it / which of these (if any) remote services are compatible, i.e. support the same functionality, use the same protocols for the exchange of both control and data information. This may also apply to different versions of a single protocol.

4. If the service has finally determined that there exists a service it can cooperate with – i.e. an *interoperable* service – it must find out the remote service's (transport) address to be able to contact it.

To transfer this *contact procedure* to multipoint teleconferencing we make the assumption that the conference is already running, i.e. it has been set up prior to the action that we do consider now. Furthermore, we assume that our conferencing site is involved in at least two conferences each of which can be identified by its name (for simplicity). When the participants decide to use an additional service (one of many) that has not been in use before, the invocation of this new service requires a *contact procedure* compared to the one described above. The change to a multipoint environment only affects step one that has to be refined as follows:

1. The service must be told in which conference[8] it shall operate *and* to whom to connect. In this case the target of a connection may not only be a peer entity on a host as in the point-to-point environment but can also be a group of entities (identified by some kind of group name or address), e.g. a broadcast communication channel for audio information.

The following subsections show how the single steps of the *contact procedure* are provided in today's (standardized) communication environment, how improvements could be achieved by providing some means for service interoperability, and what such a service would basically consist of.

Traditional Service Identification

If one considers the recommendations belonging to the services of the OSI reference model, the above *contact procedure* is performed on a per service basis. A service contacts its counterpart by specifying its TSAP address, i.e. the host and the transport layer (application) address. During and / or immediately after connection setup the service entities enter a negotiation phase in which they either determine a common protocol (usually, only the version and options are negotiated) or find out that they do not share any common "language" and terminate their conversation. The TSAP addresses are often fixed by conventions (the name or the address) so that there is no need to dynamically locate a certain service. For mapping a service name or address to a certain entity that is willing to accept the call one often uses an intermediate agent (known as the *portmapper* in some UNIX systems or the *tsapd* in the ISODE package). This agent translates the requested service name into an address (process id, local port or whatever is used to identify the service provider entity) and passes the connection setup request on to the appropriate entity (or replies the identifier to the caller).

It is important to note that the above scheme – which works well for given (restricted) environments – depends on static names or addresses assigned to certain services that cannot be extended dynamically unless new standards are provided that define additional services[9]. Furthermore, the capabilities of a service provider entity are not known unless it is contacted, and then one needs a standardized negotiation protocol (on a per service basis). Consequently, it is in general not possible to choose one of several service providers – all of which are offering the same basic service – by its capabilities, e.g. the communication protocol used, the set of operations provided, and the like. This drawback has not been noticed so far to be severe because the communication among two sites was limited to a few (very basic)

---

[8] We have to consider services that allow interoperation among several conferences (in accordance with the conference management policy, of course) as well. In such a case the *contact procedure* has to be performed once for each conference. Further details are beyond the scope of our work.

[9] Of course, standardization still remains the single really acceptable way to introduce new applications and protocols. However, it would be useful to have services also identified by means of a standardized description of their features. Such an approach allows having several service entities with different capabilities (even including private extensions) for the same service that may still be distinguished and eliminates name or address clashes among "private" services that have been developed independently.

services, such as electronic mail (or MHS), remote login programs, time protocols and so on, each of which being a *closed group for itself without the need for interoperation with other services*. With the advent of teleconferencing systems where several services *must* cooperate to meet the users' requirements more flexible means to achieve interoperability are to be found. But in this case the services are different: the old ones need not change as they will still be able to do fine with the above scheme.

Service Identification in Teleconferencing Systems

Teleconferencing systems that are composed of many service entities require a different approach that allows a more flexible and dynamic handling of services. We assume a dedicated *naming entity* per conferencing system that registers all the available services as a conventional name server does but additionally also stores all the services' capabilities that could be of interest for another service trying to contact one of them. The local communication among the several service entities and the *naming entity* must be separated from the basic functionality provided by the service itself (an extreme would be that a service does not register itself but is registered manually so that really no changes are required to this specific service implementation). Such an approach has to be taken for the following reasons

- the variety of services that may be thought of makes static assignments of service addresses very impractical / unmanageable, if not impossible[10],
- per service negotiations after connection setup are expensive because a calling service must possibly contact (one by one) each of the remote services until it finds a compatible peer entity (or notices that there is no peer entity available),
- in order to contact each of the services of a target system a service needs to know about all of them,
- it is possible that several services exist (without knowing of each other) that provide exactly the same functionality and several connection setups intended for the same service entity may reach different ones,
- the connection setup as well as negotiations might lead to confusions (within the respective service implementations) if the protocols are not compatible,
- having per service negotiation protocols would require potentially all the service specific protocols to be refined to incorporate the new elements, and
- in a teleconferencing environment there are – besides the (active) services – further (passive) *resources* available which are to be described, too, but are – due to their passive nature – not capable of executing negotiation protocols; examples are communication channels, tokens, and the like. Furthermore, these (limited) resources must be shared[11] by all the services so that a coordination by some (system) global entity is required.

As stated above such a *naming service* – we refer to it as the *Dictionary* because it is used to translate known information into the (so far unknown) information needed – has to store all attributes of any service (and any resource) on a system which are considered to be relevant if the service / resource is to be located from a remote system. This information is (partially) provided during a registry operation (and may be modified, completed, or updated later) either by the service itself, by other system components (that know about the service's existence and its capabilities) or the human user (step 1 in

[10] In ASN.1 the type OBJECT IDENTIFIER is defined that uses a hierarchical naming concept to avoid such naming problems. However, as the names are made of sequences of integers no semantics, capabilities or other information may be intuitively derived from this "name" unless an X.500 directory service is employed to resolve the OBJECT IDENTIFIER and retrieve this object's attributes.

[11] In this context, "shared" has two meanings: first, it must be ensured that no two (independent) services accidentally use the same resources – i.e. the pool of resources must be shared among the several services – and, second, if two services want to use the same resource (e.g. a token to synchronize themselves) they must be able to express the wish (and gain access to the same resource) – i.e. a number of services must be able to share a single resource.

Figure 4). The *Dictionary* has to provide flexible means to retrieve a certain service / resource including (restricted) attribute-based retrieval as well as retrieval by a (hierarchical) name. The *contact procedure* within a teleconference then basically works as depicted in Figure 4:

A service contacts the (conference global or target system specific) *Dictionary* and provides it with either a predefined name or a set of capabilities (or both) of a service it wants to contact (step 2). The (distributed) *Dictionary* searches its database for services matching the specified parameters and returns one or more entries to the caller. The attributes returned per service that has been found include especially its name and address(es). Based upon these information the caller may decide whom (if anyone) to contact and *how to do so*. The calling service then executes its usual connection setup (possibly including further service specific negotiation) but at this point the caller may be sure that the remote service it connects to is compatible, so that the connection setup usually is expected to succeed (step 3). The procedure to find a passive resource is basically the same. However, the registration of a resource is performed by some service (e.g. the one that brings it into being), and after a certain resource has been found it is simply accessed (or used) rather then connected to.

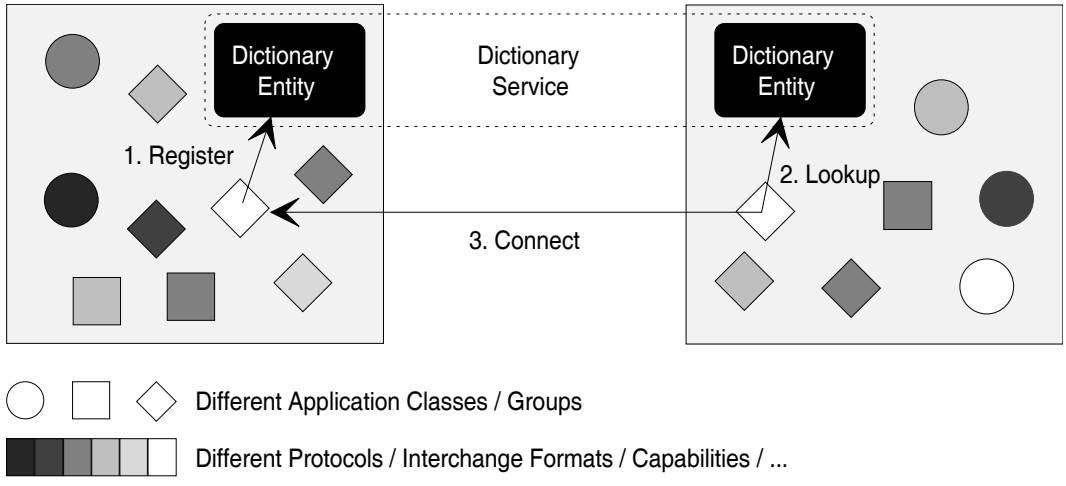

Figure 4: Contact Procedure using the Dictionary

The functionality to be provided by the *Dictionary Service* is different from both conventional naming services as well as the ISO OSI Directory Service (X.500):

- Unlike conventional naming services, we have to use a combination of hierarchical and attribute based naming, and we have to define standard service classes as well as attribute types and values that must be considered by services developed for the teleconferencing environment.
- Unlike X.500, we are primarily concerned with (highly) dynamic information objects and frequent changes to their attributes imposing stronger demands on consistency of information (affecting possible replication techniques) and on performance for access and retrieval.

Nevertheless, X.500 and other naming services should be investigated to get hints for the design of the *Dictionary*. The steps that must be taken to develop a *Dictionary* as one part of service interoperability are

- to define how the *Dictionary* fits into the conference cooperation architecture (explicitly clarifying its relations to a conference management system) – this step includes the development of an architecture (distributed or centralized) for the *Dictionary* –,
- to specify the *Dictionary Service*, i.e. the set of functions offered by the *Dictionary* for registration, lookup, etc.,

- to specify a *Dictionary Protocol* as well as the underlying service for communication between the *Dictionary* and the service entities, and
- to define service and resource classes as well as (generic and class specific) attributes – which is very important as a standardized set of classes, attribute types and values is a necessary condition to make independent services understand each other's service descriptions.

Vertical Service Interoperability

For a user's teleconferencing system that is composed of many service entities an important demand is that all these service entities may be controlled from a single point. This is motivated by the following facts:

- The conferencing system of a user may / should be viewed as a single entity within a conference acting on behalf of a user (as the user himself is a "single entity" when being involved in a physical conference). Regardless of how many components are required to achieve the conferencing functionality each of them represents the user in a conference just providing different abilities. Having only a single control panel for the entire conferencing system (rather than one for each service entity) allows to preserve consistency among all of the service entities active in a conference when the user performs an action altering her status in the conference[12] .
- If the services are not controlled by some dedicated entity that also informs them about the conference and the user's actions the service entities have to store and maintain this status information themselves. Then the same functionality has to be replicated for all services making the service functionality as well as the service protocols more complex and increasing the probability of inconsistencies among the status information held by the several services – this, finally, may introduce the "Who-is-right?" problem if several services with different understandings of the conference status try to interoperate).
- Coordination of service control provides a hook for integrating common security mechanisms (such as authentication) into a teleconferencing system, again, because functionality and protocols need not be replicated.
- Regardless of internal aspects of the conferencing system a single point of control simplifies the usage of the system and helps to avoid user errors.

Functional Model

The single point of control is the so-called conference management system (CMS) that is intended to represent the user in the teleconference as far as his presence in a conference is concerned. His behaviour in a conference, of course, affects all of his services but neither of the other users'. Consequently, information is only conveyed locally to the service entities acting on behalf of this user so that only his *abilities* are influenced by his behaviour. Furthermore, the actions of other participants concerning their respective status in the conference – such as joining or leaving – may induce local messages as well (indirectly via the local CMS). The coordination of service control is to be performed by means of a *Conference Information and Request Protocol (CIRP)* that basically consists of

- *information service elements (ISEs)* by which the CMS informs all the service entities about actions of the user that affect her status concerning the conference as a whole (i.e. her *presence*) or actions other users have performed, and
- *request service elements (RSEs)* by which the service entities may query the CMS for certain information (not automatically conveyed with the ISEs).

This means that the information exchange between service entities and the CMS is bi-directional. Examples for ISEs could be "conference-leave" or "new-user-in-conference", typical RSEs could be

[12] Such actions might be joining or leaving a conference, changing to another conference, and the like.

"get-conference-user-list" or "get-conference-name". A detailed list of required service elements and their parameters is subject to research within the service interoperability definitions.

Figure 5 depicts a sample configuration of two teleconferencing systems with a conference user interface (C-UI) as a single point of control, the CMS, and two service entities respectively. The service entities – (groupware) applications – communicate via their (unchanged!) service protocols (SP-1 and SP-2) and are informed by the CMS about status changes in the conference via CIRP. The two CMS' exchange conference information via some kind of conference control protocol (CCP) so that they are informed about status changes on the other system.

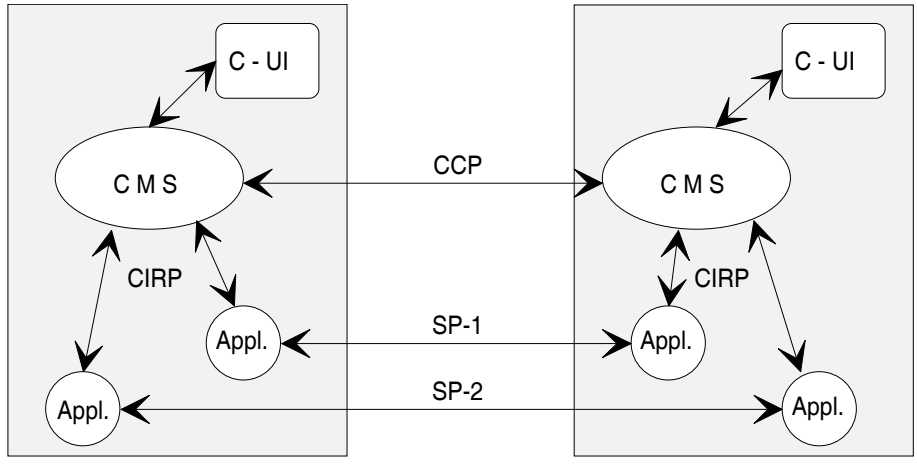

Figure 5: Coordination of Service Control via CIRP

Integration into a Service Entity

On one hand, CIRP automatically notifies all service entities of a user about status changes in the teleconference, on the other CIRP offers certain mechanisms to the services they may use to query information from the CMS – possibly, this approach is to be extended to allow services to specify requests directed to the CMS. The coordination functionality itself that is provided by CIRP is totally independent from the remaining service functionality of a service and its implementation – in particular, it does not affect a service specific user interface[13] – as well as from the *Dictionary Services* described above. However, if a service is to provide the CIRP interface it must also implement appropriate reactions to the single *events* signaled via CIRP. These reactions are independent for each event. In some cases, such actions might be simply ignoring the event, in others, events might cause service entities to instantiate new entities, to terminate, to perform other (internal) actions, or to query further information from the CMS. Service entities that do not implement the CIRP interface or the ignore all the events are not affected by the introduction of the coordination of control at all. This means that existing services of teleconferencing systems may be upgraded one by one without affecting the functionality of the entire system.

As CIRP is intended to provide control information about conferences to all service entities of a user with the aim of centralizing the conference control, a conflict arises when service protocols employed within a conference themselves provide means for the exchange of conference control information for the respective services. In this case two independent control protocols are employed to exchange *the same type of information* (namely the conference status, requests to change it, and the like) but *not necessarily the same information values*, i.e. the content, (e.g. due to inconsistencies). In Figure 5 the same status information may be conveyed via *SP-1* as well as via *CCP* and *CIRP*. Care must be taken of

[13] However, if a user interface displays state information about the conference (and this will usually be the case) the display must be updated according to the information received via CIRP from the CMS.

this problem to avoid inconsistencies of status information across different services and the – following – unpredictable behaviour of the respective system components. A possible solution to this problem could be to extend the RSEs described above by means to issue conference control commands from service entities that are to be executed by the CMS. This approach is mandatory if the single service entities have their own user interfaces and (partial) control of the conference shall be possible from these interfaces, too, or cannot be prohibited.

Summary

In this paper a definition for service interoperability in teleconferences has been given, and many aspects of interoperability have been investigated. Some of these aspects have been identified to be essential within the scope of our work, and these aspects have been characterized by *horizontal* and *vertical* interoperability. Two schemes have been outlined to approach the respective issues.

The *Dictionary* approach allows locating services (by name or by capabilities) in a teleconference. Thus it provides one of the basic means to achieve service interoperability, namely the ability to find out about the existence of compatible services / resources in arbitrary complex environment with any number of different services, and to obtain all the information necessary to contact it.

The *Conference Information and Request Protocol* defines a means that allows coordinating the control of all services belonging to a user via the CMS (as the single point of control) and provides the services – automatically or on request – with consistent information about the conference status. This leads to the set of services comprising the teleconferencing system of a user to form an integrated whole.

Further study is required to detail both these constituents of a virtual *interoperability service* that incorporates both *horizontal* and *vertical* interoperability, as well as to look for further components that might be needed to complete a framework for service interoperability.