A Base Component for Network-Based Service-Oriented C4ISR Systems

Sylvia Käthner and Marc Spielmann
FGAN e.V., Neuenahrer Strasse 20
D-53343 Wachtberg, Germany

{s.kaethner, spielmann}@fgan.de

ABSTRACT

Network Centric Warfare poses new challenges to C4ISR systems: future systems must provide access to a homogenous information and service space for a broad spectrum of users whose computation and communication resources vary and whose location may change frequently. We propose a design schema for network-based, service-oriented C4ISR systems and present a base component upon which such systems can be built on. From a user's point of view, the base component provides access to all services (i.e., applications, information sources, etc.) relevant for the current mission. From a technical perspective, these services are software modules which are distributed in a network and interact via well-defined interfaces. A main feature of our base component is that it is load-scalable: if the communication capacity between a user device and the network is insufficient or degrades during an operation, the base component may react to the new situation by moving computational load from the network into the user device. Under certain circumstances this re-balancing of computational load reduces the required communication capacity so that the overall system maintains a tolerable level of operability.

1.0 INTRODUCTION

Network Centric Warfare (NCW) poses new challenges to C4ISR systems. In order to enable and support NCW, future C4ISR systems need to guarantee a semantically consistent flow of information among the various levels of, and participating units in, a command-and-control hierarchy. Due to continually changing operational requirements, the systems must be highly flexible with respect to (a) the type and quality of services offered to users and (b) the computation and communication resources provided by the environment. In a nutshell, future C4ISR systems must provide access to a homogenous information and service space for a broad spectrum of users whose computation and communication means vary and whose location may change frequently [1].

From a user's perspective, accessing and operating a C4ISR system should be as simple as possible. Ideally, the system reveals only essential, mission-relevant information to the user. Any system-specific information remains hidden. In particular, the user should not be concerned with administrative tasks, like installing new software on his current access device, or locating a particular service on a specific network server. After starting up his device and logging into the network via some mandatory authentication process, the user should be able to simply access all services for which he can provide appropriate authorization. Furthermore, the services should be presented to the user via some common user interface, providing a homogenous 'look-and-feel' for all services so that the handling is consistent, simple and intuitive, and errors due to misinterpretations are reduced to an absolute minimum.

The above requirements concerning a user's perspective on a C4ISR system apply to distributed multi-user multi-task systems in the commercial world as well. They are by no means specific to military systems. There is, however, another requirement on C4ISR systems which is rather untypical for commercial systems:

Paper presented at the RTO IST Symposium on "Coalition C4ISR Architectures and Information Exchange Capabilities", held in The Hague, The Netherlands, 27-28 September 2004, and published in RTO-MP-IST-042.

A C4ISR system should be able to re-balance the computational load between a user device and the network. More precisely, the system should be able to (dynamically) move computational tasks from the network to a user device, and vice versa.

This requirement, which we call *load scalability*, is motivated by two observations:

1. Insufficient or degrading communication capacity is a notorious problem in military networks. Unlike commercial networks, military networks operate in non-cooperative environments and are subject to non-conventional interferences (e.g., jamming or actions of hostile forces). The problem is of an inherent nature for military networks; one should not assume that it can be solved simply by improving existing communication technologies or by developing new communication media.

2. Depending on the type of services requested by a user, it is often possible to reduce the communication capacity required for a tolerable level of operability by (i) moving computational tasks from the network into the user device and (ii) utilizing the resources of the user device to handle the computational tasks locally. Hence, if there is a reduction of the available communication capacity, e.g., due to jamming, the disruption might be compensated for by scaling up the computational load on the user side.

To see an example of a service where the second observation is applicable, consider a tool for computing optimal paths: depending on extrinsic hazards (hostile forces, contamination, etc.), the tool computes the optimal path from the user's current position to a certain target position. If the bandwidth between user device and network is sufficiently large (e.g., if a satellite link is available), it makes sense to run the tool on a network server and provide the user with a view on the computed results, e.g., a projection of the optimal path onto some map. In this scenario, the user device serves merely as a graphical output device. Leaving most of the computational load in the network has several advantages:

- Typically, network resources can offer much more computational power than user devices, and thus are faster and more accurate.

- Maintenance of network resources is easier and less costly. On the other hand, distributing computational load in the form of software modules increases the danger of incompatibilities.

- Often, there exist semantic dependencies between services (e.g., if the output of one service becomes input to another service). In general, such dependencies can be maintained more efficiently if services are hosted by network resources.

The problem with the outlined thin-client approach is that, if the bandwidth between user device and network drops under a certain threshold, most services become unavailable because the generated views can no longer be transmitted to the user.

Now, consider a scenario where prior to an operation it is already conceivable that the available bandwidth will drop drastically in the course of the operation. For instance, consider an operation in urban terrain with civil infrastructure obstructing communication. In order to ensure that the opti-path service is available during the operation, our C4ISR system may try the following approach:

- Prior to the operation, the opti-path tool is moved to the user device together with a map of the area of operation.

- During the operation, the system sends updates on new or changing extrinsic hazards to the user device whenever communication is possible. The user runs the opti-path tool on his device using the given map and the incoming updates as input.

Observe that this smart-client approach may not yield the same results as the thin-client approach (i.e., optimal paths). If the connection to the network is disrupted or breaks down altogether, the opti-path tool may operate with outdated data and thus may compute different, possibly semi-optimal paths. Nevertheless, with smart clients the system may still be of use, rather than becoming entirely useless as it would be the case if we employ thin clients.

Recall that our notion of load scalability is symmetric: it should be possible to scale up both the load of a user device and, alternatively, the load of the network by moving computational tasks in either direction. In fact, being able to move computational load in the other direction, i.e., from the user device back into the network, is important as well. If communication improves during an operation, the system should use as many network resources as possible (and, as it is feasible), simply because services and data in the network are generally much faster, more accurate and more up-to-date. It is worth noticing that this direction of load scaling can be accomplished without any actual data transmissions. There is no need for shipping back software or data packages which have previously been moved from the network to a user device if the system keeps master copies in the network.

In this paper, we outline an architecture for network-based, service-oriented, load-scalable C4ISR systems. A system based on our architecture is service-oriented in the sense that all major building blocks of the system (i.e., applications, software tools, data pools, etc.) are modeled as global services accessible to any user that can provide appropriate authorization. The system is network-based in the sense that our services are designed as (more or less autonomous) software modules which can move freely in a network of servers, assuming that the network provides some middleware infrastructure like CORBA or Java-RMI. Load scalability is achieved by a special component for managing communication between user devices and the actual network. The component can be viewed as a kind of meta service whose main purpose is to enable users to access and use other services in the network. For that reason, we refer to the component as *access service*. After establishing a connection to the network and verifying the user's identity, the access service locates the services requested by the user and transports for each such service a front-end to the user device. A service front-end is a portion of a service intended to be run on a user device and thus can be viewed as computational load on the user's side. It consists of a graphical user interface and a set of computational tasks. The graphical user interface is displayed on the user device and governs the interaction of the user with the service. Each service may offer a variety of different front-ends; which one of the front-ends is actually presented to the user depends on various factors, e.g., the computational power of the device, the available communication capacity, and the profile of the user.

A comment on the notion of architecture as used in this paper is in order. By an architecture, we mean a collection of semi-formal descriptions of certain aspects of a system, reflecting, e.g., different views on the system or different levels of abstraction of the system. The main purpose of such descriptions is to provide the reader with an intuitive understanding of important components, processes, relations, interactions, etc. of or inside the system. We do not provide formal (i.e., rigid mathematical) descriptions. Furthermore, readers familiar with the NATO C3 System Architecture Framework (NAF, [2][3]) may notice that we do not thoroughly distinguish between the three major views identified in the NATO framework (i.e., operational, system, and technical view). This deviation from the NATO framework allows us to present our architecture in a more natural way, as we derive our system descriptions from certain requirements and refine them step-by-step during the course of the paper.

The paper is structured as follows. In Section 2, we describe our architecture for network-based, service-oriented C4ISR systems. In Section 3, we refine the architecture by presenting an access service for managing load scaling. The service is a key component of our architecture and can serve as a basis of future C4ISR systems. We conclude and briefly discuss future work in Section 4.

2.0 NETWORK-BASED SERVICE-ORIENTED SYSTEMS

Network Centric Warfare requires an unprecedented level of connectivity from C4ISR systems. In order to enable and support NCW, future systems need to meet the following connectivity requirement:

> *Any user can be connected to any other user, any piece of information, and any information processing capability available in the system (where the set of users includes sensors and actors).*

Note that this requirement does not imply that each user can actually see every piece of information (e.g., the current location of a special task force unit) or use every information processing capability (e.g., a tool for planning UAV flight paths). It only means that it is technically possible for a user to reach this level of connectivity, provided the user has appropriate authorization. Naturally, requiring this level of connectivity from an information system has a profound impact on the architecture of the system. In this section, we argue that a network-based, service-oriented architecture is a natural choice.

Global Services. An immediate conclusion to be drawn from the connectivity requirement concerns the accessibility of the major building blocks of a system, i.e., applications, software tools, information sources, etc. Today's C4ISR systems often are conglomerates of a variety of subsystems, each tailored for a specific functional area. Not surprisingly, we frequently face the situation where a subsystem lives in its own world, i.e., has its own data model, its own data exchange format, its own user interface, etc., and thus is incompatible to the rest of the system. More to the point, on a semantic level, the major building blocks of a subsystem are inaccessible to a user outside that subsystem. This certainly contradicts our connectivity requirement. A straightforward approach to overcome this obstacle is to extract from each (sub)system as many building blocks as possible (and feasible) and model the building blocks as *global services* accessible, in principle, to any user. To protect the services against unauthorized usage, each user is equipped with a profile determining

1. precisely those services which the user is authorized to access and

2. for each such service, a level of accessibility (e.g., a group membership imposing restrictions on the user's actions, requests, views, etc. or granting certain rights and priorities).

Network-Based Systems. A direct flow of information among the various levels of, and participating units in, a command-and-control hierarchy is a particularly important aspect of NCW. An NCW-enabling C4ISR system must provide, e.g., any spontaneously formed group of users with the opportunity to directly exchange information among each other. If we view inter-user communication as a special kind of service – namely one which relays information between two or more users, the above connectivity requirement can be rephrased as follows:

> *Any user can be connected to any service.*

Another, similar requirement which we have not discussed yet follows from the fact that there may exist semantic dependencies between services, e.g., if the output of one service serves as input to another service. In order to maintain such dependencies, a system must be able to interconnect any two services so that they can exchange information. In combination, the two requirements leave us with two options for how to design an NCW-enabling C4ISR system:

1. the centralistic approach: All services are united in a central component. Each user can log into the central component and access the desired services.

2. the distributed approach: Services are designed as more or less autonomous software modules which are distributed in a network of servers. Services interconnect by means of a middleware infrastructure provided by the network (e.g., CORBA or Java-RMI), and interact via well-defined

interfaces [4]. Each user can log into the network and access the desired services through the network.

For the obvious reasons, the first approach is inadequate for military purposes (as well as for large-scale commercial systems). In the remainder of the paper, we focus on systems designed according to the second approach. We refer to such systems as *network-based systems*.

Accessing Services. In order to provide the reader with an idea of how a user may gain access to the services offered by a network-based system, we describe a procedure for logging into such a system. Suppose that a user wants to access various services via some personal access device (e.g., a PDA, notebook computer, or workstation). To this end, the user starts up some standard software module. The module can either be pre-installed on the user device or downloaded from some designated network server. (We will come back to this module in the next section, where it is called front-end manager). The module establishes a connection between the user device and a *gateway*, i.e., a special network server managing communication between the network and the outside world (see Step 1 in Figure 1). If there is more than one gateway available in the network – which will certainly be the case in a realistic scenario – the user device may first have to consult with a central directory referring the device to some free gateway.

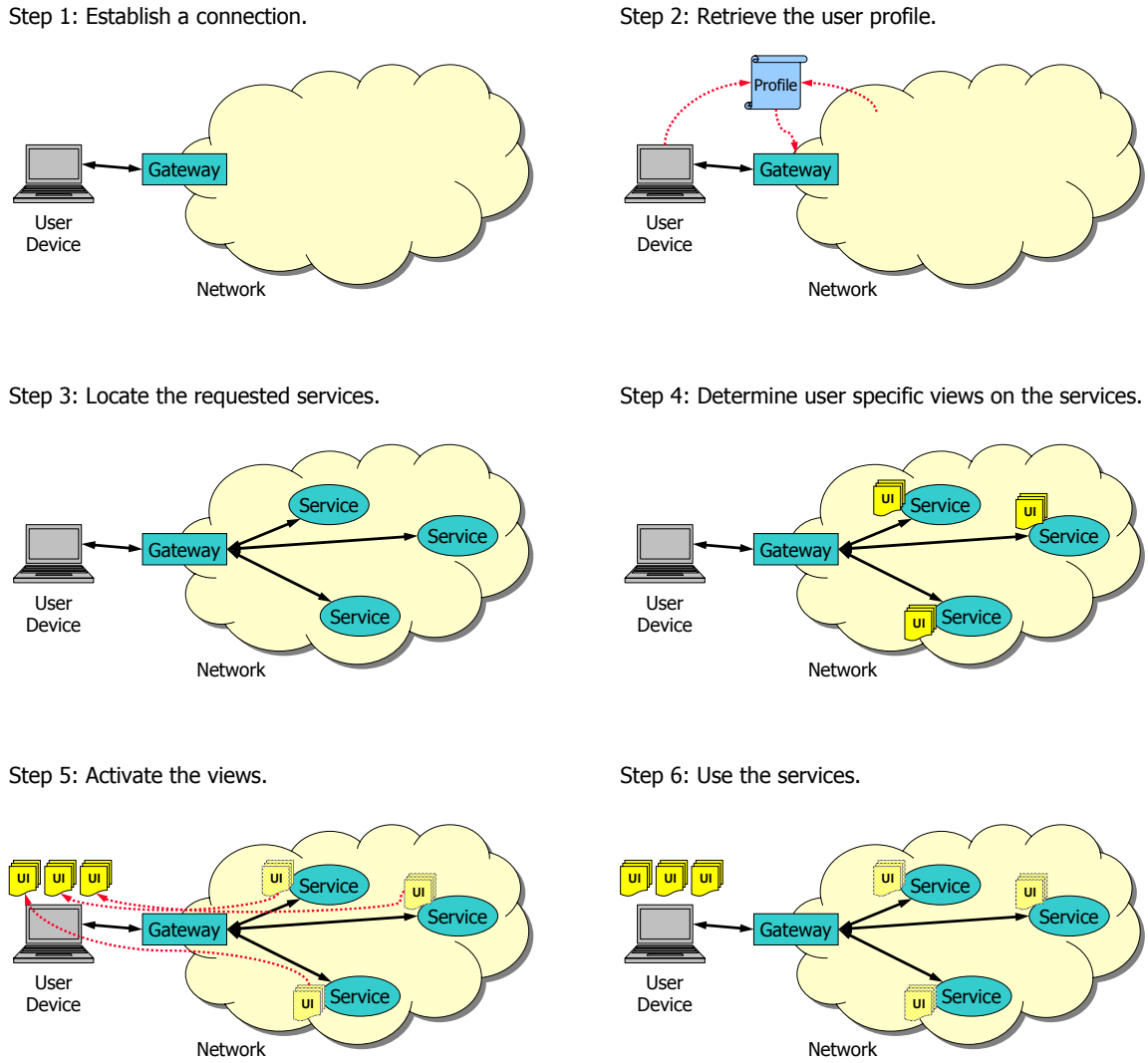


Figure 1: Accessing services in a network-based system.

Once a connection between the user device and the gateway is established, the gateway runs some authentication procedure in order to determine the user's identity, e.g., by verifying a secret password. When the user's identity is verified, the gateway retrieves from the network or from the user device or from both a user profile (see Step 2 in Figure 1 and recall our previous mentioning of user profiles in the description of global services above). Of course, a user profile may also contain non-critical information like user preferences, graphical settings, etc.

When the gateway has obtained the user profile, the gateway tries to locate the services requested by the user (see Step 3 in Figure 1). If we assume that the network provides some middleware infrastructure like CORBA or Java-RMI, then the gateway can employ the middleware's central registry to locate services in the network.

Next, the gateway sends to each localized service the user profile enriched with information about the processing capabilities of the user device (e.g., an encoding of the device's type) as well as information about the currently available communication capacity between the device and the gateway. Depending on the information contained in the enriched profile, each service determines a service user interface (UI) appropriate for the user and the currently available computation and communication resources (see Step 4 in Figure 1). We assume that the user interface provided by each service is given in a format which allows the system to (i) transport the user interface to the user device, either as a whole or in some partial form, and (ii) display and operate the user interface on the user device. There exist several techniques in the commercial world which can be employed for implementing service user interfaces satisfying the above conditions (e.g., Java Servlets, JavaServer Pages, or Java Applets). An alternative technique, providing the bases for a load-scalable architecture, will be presented in the next section.

Finally, the gateway retrieves the chosen service user interfaces from the localized services and sends them to the user device, where they are presented to the user. The requested services are now activated and accessible to the user (see Steps 5 and 6 in Figure 1).

Although the above description focuses on a single user, it easily generalizes to an arbitrary number of users (see Figure 2). Note that different users may receive different user interfaces and thus have different views on the system. Note also that, even though there are no direct communication links between users, users can nevertheless share resources and interact with each other through services. A simple example of a shared resource would be a common data pool (e.g., a sub-service of a global file storage service) which is accessible to all users. Such a data pool might be useful for exchanging non-structured data or data whose structure or type is unknown to the system.

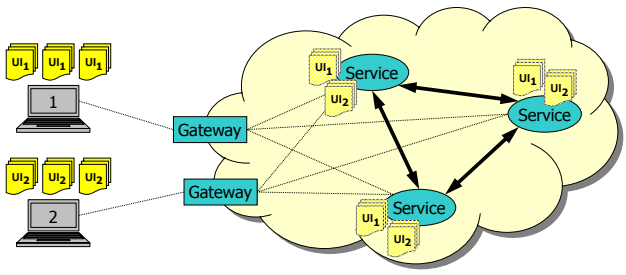


Figure 2: Multiple user scenario.

Single-User View. As was already mentioned in the introduction, from a user's perspective accessing and operating a C4ISR system should be as simple as possible. In particular, the presence of an underlying network should be transparent to the user, because under normal circumstances this information is of no concern to the user. The only components of the system that should be visible to the user are:

- the requested services in the form of user interfaces displayed on the user device, and
- a meta service which handles the login procedure, maintains the user profile, and ultimately connects the user to other services.

We call the last component *access service*, for its main purpose is to enable the user to access and use other services in the network. The access service can be viewed as the back-bone of a network-based C4ISR system whose architecture conforms to the general architecture as outlined in this section. In the next section, we refine the architecture to support load scaling.

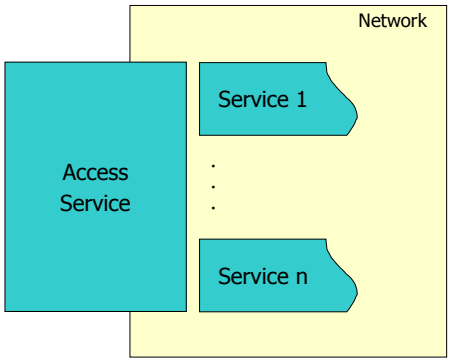


Figure 3: Single user view.

3.0 AN ACCESS SERVICE FOR LOAD SCALING

According to the load scalability requirement in the introduction, a C4ISR system should be able to move computational tasks from the network to a user device, and vice versa. Since connecting user devices with the network is a core task of the access service, load scaling particularly concerns the access service. In this section, we outline an architecture for an access service supporting load scaling.

Service Front-Ends. In order to move computational tasks from services to user devices, both services and user devices have to agree on a formal representation of computational tasks. For instance, executable code would be a straightforward choice for such a representation. Furthermore, it is necessary to render (representations of) computational tasks accessible to the access service. Now observe that both criteria, i.e., existence of a common format and accessibility to the access service, apply to service user interfaces as well. This immediately follows from our assumption in Section 2 that user interfaces are given in a format which allows the system to transport (portions of) service user interfaces to user devices. Hence, it makes sense to

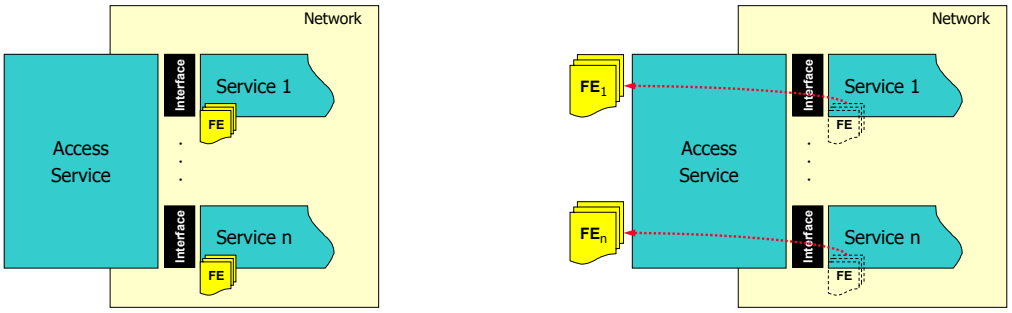
- define a common format for both computational tasks and service user interfaces, and
- equip each network service with a standardized data exchange interface which allows the access service to connect to the network service and retrieve both computational tasks and service user interfaces in the common format.

We have developed and implemented such a format together with an appropriate data exchange interface. Since a thorough presentation of the technical details would exceed the scope of this paper, we only sketch the main ideas here.

We combine computational tasks and service user interfaces into what we call *service front-ends*. Intuitively, a service front-end is a portion of a service which can be separated from (the rest of) the service, transported to a user device, and run on the user device. The front-end can communicate with the portion of the service remaining in the network and may allocate resources of the hosting user device (e.g., memory space and computation time). In this respect, a service front-end can be viewed as load intended to be moved to and managed by user devices.

Conceptually, a service front-end consists of a graphical user interface and a set of computational tasks. In appearance and usage, the graphical user interface can resemble the user interface of any common software tool (e.g., Microsoft Word). On a technical level, however, the graphical user interface provided by a service front-end is composed of generic building blocks, called *abstract frames*. Such frames determine the visual appearance and (most of) the functional aspects of the graphical user interface. They may also invoke some of the computational tasks associated with the service front-end. Another important aspect of our abstract frames is that the collection of abstract frames, from which a service front-end is composed of, induces a natural partition of the service front-end into functional-logical chunks. These chunks can be transmitted to a user device separately and on demand: whenever a user needs more pieces of the service front-end, the access service can retrieve the corresponding abstract frames from the portion of the service remaining in the network. This mechanism can help reducing the required communication capacity between the user device and the network. Moreover, chunks can be updated during run-time so that the overall system becomes more flexible and easier to maintain. It is worth noticing that a service may have several different front-ends, each specially tailored for a certain group of users, a specific class of user devices, and a particular type of communication means. In this respect, a service front-end can be regarded as a formal representation of a user's view on the service.

Service Access Interface. Service front-ends are obtained from services via a standardized data exchange interface. To be more precise, each service is equipped with (an identical copy of) an interface which allows the access service to interact with the service. Any communication between the service and the access service (and thus between service and user) is routed through this interface, which we call *service access interface*. As an example, consider Figure 4. It is a refinement of Figure 3 and displays the situation right after Step 4 in Figure 1, now from the simplified viewpoint of a single user. That is, the access service has already located the services requested by the user (Services 1, …, n) and has sent to each of these services an enriched user profile. Furthermore, each such service has analyzed the enriched profile and has selected a front-end (FE) appropriate for the user and the currently available computation and communication resources. In the next step (see Figure 5), the access service retrieves the chosen front-ends and presents the graphical user interfaces included in those front-ends to the user (which corresponds to Step 5 in Figure 1). Any data exchange between the access service and the services described so far is routed through the service access interfaces (see the black boxes in Figures 4 and 5).



Figures 4 and 5: Transmission of front-ends via the service access interface.

Architecture. We are now in the position to describe the architecture of our access service. The main components of the service are given by two software modules, called *front-end manager* and *service manager*. A front-end manager is a kind of universal run-time environment for service front-ends. It is installed on each user device, e.g., before shipping the device to a user, or by means of some installation wizard which downloads the most recent version from a designated network site. A service manager can be thought of as a mediator between a front-end manager on the one hand and a set of network services on the other hand. A service manager is created by a gateway whenever a front-end manager tries to establish a connection to that gateway (recall Step 1 in Figure 1). If the gateway accepts the connection, it redirects the connection to the newly created service manager. Subsequently, the front-end manager directly communicates with the new service manager, bypassing the gateway.

As an example, consider Figure 6. It shows a front-end manager running on a user device and a service manager running on a gateway. The figure is a refinement of the access service in Figure 3 and depicts the situation right after a connection between the user device and the gateway has been established (recall Step 1 in Figure 1). Again, we here focus on the single-user scenario. In particular, Figure 6 does not show other service managers which may run on the gateway at the same time, handling the communication with different user devices.

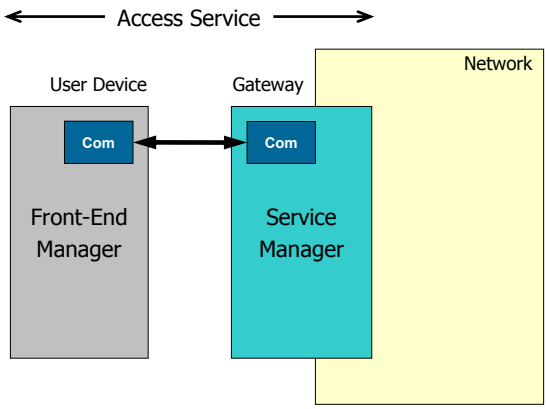


Figure 6: Access service architecture.

Communication Component. Both front-end manager and service manager are equipped with a *communication component* (abbreviated Com in Figure 6). Any communication between the two managers is routed through their Com components. There are no further means of communication. The main task of a Com component is to provide a *message exchange service* to its owning manager:

1. The manager can ask its Com component to send a (formatted) message to the manager's peer manager.

2. The Com component is constantly listening for incoming messages sent by the manager's peer manager. If a message arrives, the Com component notifies its owner and hands over the received message.

Notice that we do not require the message exchange service to be connection-oriented (like TCP/IP). In principle, the service can be connection-less (like UDP/IP). In fact, the access service may offer users to choose among different types and qualities of message exchange services, depending on the currently available communication medium (e.g., satellite link, radio link, or mobile phone). It is only important that the Com components of two communicating managers agree on the same service.

Service Manager. During its time of existence, a service manager distinguishes two phases. The first phase, called initial phase, essentially concerns user authentication, service localization and service activation (recall Steps 2-5 in Figure 1). The main tasks of a service manager during this phase are:

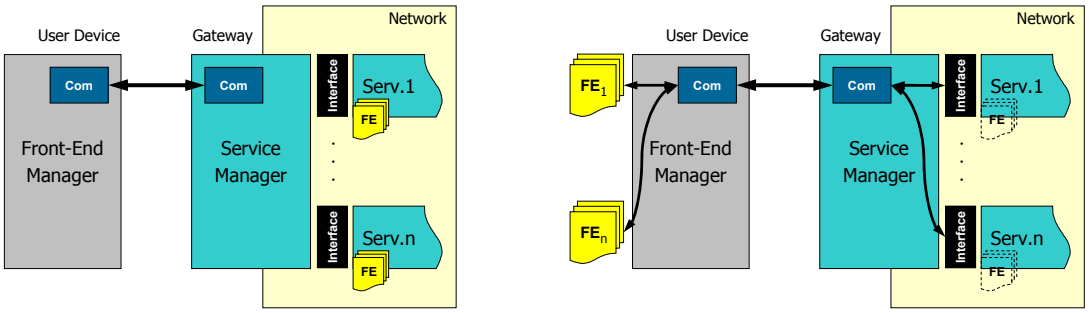
1. retrieve a user profile (e.g., from some central user management service in the network)

2. locate the services requested by the user (e.g., by contacting some central service registry in the network)

3. establish a connection to each localized service via the service access interface (which we assume is implemented by each service), and

4. obtain from each contacted service a service front-end (either as a whole or in the form of an initial chunk) and forward the obtained front-ends to the front-end manager.

The last two steps are visualized in Figures 7 and 8. Observe that the two figures are refinements of Figures 4 and 5, respectively.



Figures 7 and 8: Transmission of front-ends via the service and the front-end manager.

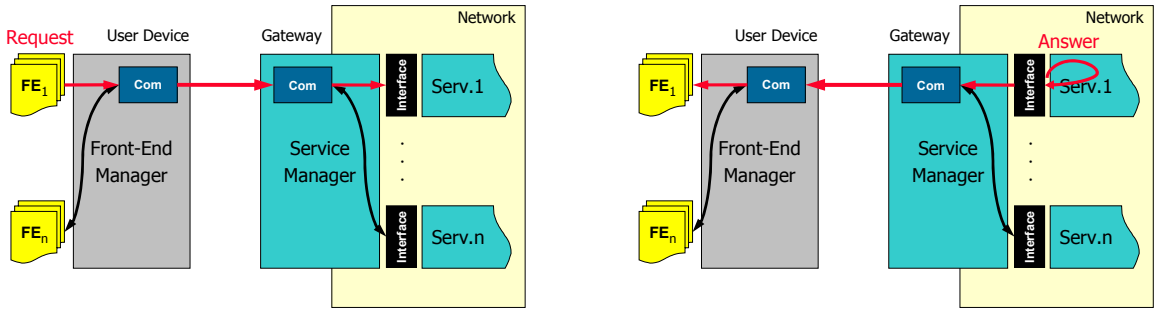
Once the service front-ends have been forwarded to the front-end manager, the services are considered activated (from the point of view of the service manager). The service manager now switches to the second phase, called operation phase. In this phase, the service manager acts as a relay station between the front-end manager and the activated services. Its main tasks in this phase are:

1. forward front-end requests to services: A *front-end request* is an inquiry initiated by a service front-end and addressed to the *back-end* of the corresponding service, i.e., the portion of the service remaining in the network. For instance, a front-end may ask its back-end to verify a password provided by the user. The service manager receives front-end requests from the front-end manager in the form of formatted messages. When such a message arrives, the service manager analyzes the message, identifies the addressed service, recovers the original request from the message, and transmits the request to (the back-end of) the service via the service's access interface (see Figure 9).

2. forward replies to the front-end manager: When a service receives a front-end request via its access interface, the service may reply by transmitting the requested data to the service manager, again via the service's access interface. The service manager regularly checks all access interfaces for incoming transmissions. When it receives a transmission containing an reply to a preceding request, it converts the transmission into a formatted message and sends the message to the front-end manager (see Figure 10).

3. forward service interrupts to the front-end manager: A *service interrupt* is a notification sent by a service to one of its front-ends (running on some user device). For instance, a service may notify a front-end that for maintenance purposes the service will not be available for a certain period of time, and that the user is to be informed of this fact via the front-end's graphical user interface. The service manager handles service interrupts analogous to replies to front-end requests.

4. forward interrupt acknowledgments to services: Upon receiving a service interrupt, a front-end may acknowledge the interrupt by asking the front-end manager to send a corresponding message to the service manager. The service manager handles these messages similar to front-end requests.



Figures 9 and 10: Handling of front-end requests.

Front-End Manager. Like a service manager, a front-end manager distinguishes between an initial phase and an operation phase. During the initial phase, its main tasks are:

1. request authentication from the user and send the obtained information to the service manager, and

2. receive service front-ends from the service manager and display the included graphical user interfaces (recall Figure 8).

Once the graphical user interfaces of all activated services are displayed, the front-end manager switches to the operation phase. Its mains tasks during this phase are:

1. handle front-end events: There are several types of event which may occur in a front-end. For example, an event may occur in the graphical user interface due to a user interaction (like pressing a button). Another type of event is the arrival of a service interrupt. When such front-end events occur, the front-end manager consults with the front-end in order to decide how to react on the events, and then performs the appropriate actions.
2. forward front-end requests to the service manager: When a front-end event occurs, the front-end may decide to send a request to the service back-end. The front-end manager converts these requests into formatted messages and sends them to the service-manager (recall Figure 9).
3. forward replies to front-ends: The front-end manager receives from the service manager replies to preceding front-end requests in the form of formatted messages. When such a message arrives, the front-end manager analyzes the message, identifies the addressed front-end, recovers the encoded reply from the message, and hands the reply over to the front-end (recall Figure 10).
4. forward service interrupts to front-ends: Similar to Task 3 above.
5. forward interrupt acknowledgments to the service manager: Similar to Task 2 above.

Prototype. We have validated the proposed architecture by means of a prototype implementation. All components of the prototype are implemented in Java and thus are easily portable to any common operating system. In particular, our graphical user interfaces are based on Java Swing. As middleware infrastructure we use Java-RMI. Hence, our service access interfaces are given in the form of (quite simple) RMI interfaces.

4.0 CONCLUSIONS

We have outlined an architecture for network-based, service-oriented C4ISR systems. Our architecture is specially tailored to satisfy two (non-standard) requirements implicitly expressed in the NCW doctrine. Firstly, NCW-enabling systems need to ensure a level of connectivity unprecedented in existing systems. Secondly, NCW-enabling systems need to provide a flexible connection between user devices and network (services) such that (a) communication bottlenecks can be compensated for and (b) the broad spectrum of different user devices and communication means necessarily present in these systems can be handled in a uniform way. We have developed such a flexible connection in the form of an access service for transporting and managing portions of network services. If the communication capacity between a user device and the network is insufficient or likely to degrade during an operation, our access service can be used to re-balance the computational load between the device and the network in an attempt to maintain a tolerable level of operability. It is worth noticing that in the present architecture this re-balancing (load scaling) is not performed dynamically, i.e., during an operation. To obtain a truly adaptive system, which is capable of dynamic and possibly (semi-)automatic load scaling, our architecture needs to be enriched with appropriate monitoring and analysis components.

5.0 REFERENCES

[1] Alberts, D.S. & Hayes, R.E. (2003): Power to the Edge. Washington, DC: CCRP.

[2] NATO C3 Board. *NATO C3 System Architecture Framework (NAF)*. 20 October 2003.

[3] ISSC NATO Open Systems Working Group. http://nc3ta.nc3a.nato.int. December 2003.

[4] Wunder, M. (2002): Zukunftsweisende Architektur für FüInfoSys. IT-Report, 2 (2002), 42-46.