

AMETAS

Good Migrations.

AMETAS White Paper Series

by Klaus Herrmann
and Michael Zapf

On Writing Services

July 2000

Johann Wolfgang Goethe-Universität Frankfurt/Main, Germany
Department of Computer Science

www.ametas.de

Writing Services

The inherent internal structure of any computer software known today is based on a principle that is called *client/server computing*. It may not always be obvious and it may come in very different flavors but we can always identify objects that request some service (clients) and other objects that process such requests (servers). The question who is a server and who is a client is often time-dependent since objects are able to change their role over time. If we want to employ a computer system in a useful and efficient way, this system needs some deterministic structure, i.e. we have to know where services (servers) can be found and what they can do for us. In addition, we expect that such a server will service our request without hesitation as quick as possible and give us a useful result. This is also true in mobile-agent-based systems. But does the idea of an ever-present service which passively waits for requests, services them and waits for the next request fit into our notion of agenthood? Certainly not! Therefore we developed another kind of Place User in AMETAS that lives on the same layer as user adapters and agents do but displays some fixed reliable behaviour. It is stationary and services requests. This is simply called a *service* in AMETAS.

How Does a Service Differ from Other Place Users?

As we already stated above, a service displays a very specific behaviour which could be called *wait-and-serve*. Since services are stationary and there is no proactive or autonomous behaviour noticeable, the difference to agents is quite clear. Comparing them to user adapters, the answer is not so easy. Both are stationary. Both, viewed in isolation, service requests. While a service may serve any request received within a message, a user adapter services a user's requests received via keyboard or mouse. The differences are the following:

- User adapters form a sort of stationary agent when viewed in combination with their user. No equivalent statement can be formulated for services.
- Services allow controlled access to system resources for any other Place User. User adapters allow controlled access to a multi-agent application for the user.

Services play a very vital role in AMETAS. As was stated in other White Papers of the series, an AMETAS place only provides services that are needed to keep Place Users alive. These are called *core services* and include migration, communication and events. However, most applications need some application specific services that cannot be build into the AMETAS core. Such services are an extension to a place and all extension services together with the core services form the individual environment of a place that attracts mobile agents. One reason why agents choose to migrate to a place is that it offers services that are not present on their current place.

The Structure of a Service

An AMETAS service consists of two parts: a *service manager* and one or more *service objects*. The service manager is provided by AMETAS and cannot be changed by the programmer. It represents a Place User being executed in its own thread and implementing an *invoke* method. The service manager is responsible for all the details of

invoking, maintaining and managing the actual service which is implemented in a service object. These tasks are the same for every service and thus have been encapsulated in the service manager to ease the programmers task. The task is to develop a service object which implements an interface through which it can be invoked and managed by the service manager. The programmer can specify a service Description in his service object that specifies the ways in which it may be used. This can be information on whether the service is shared or non-shared, session-based or not, how expensive the usage of the service will be and in which way the service can be mediated. Apart from that, the programmer can implement methods that initialize the service object and do the actual servicing of requests and methods that handle events received by the service. Thus, the separation of service manager and service object enables the programmer to concentrate on providing the service itself and not on dealing with the problems involved with every service provisioning.

One fundamental concept of AMETAS services is the *interaction mode*. Inspired by CORBA, we implemented four basic ways in which a client may interact with a service:

- *SHARED* services maintain a single service object that serves all client requests. services that are idempotent, i.e. services that serve a request always in the same manner no matter how many or which requests preceded it, are one example for sharable services.
- *NON-SHARED* services instantiate a new service object for every client request. The service object is removed when the request is served.
- *SHARED-SESSION* services contain a single service object on which clients can develop a session context, i.e. a long-term relationship over numerous requests. A session-based service can associate clients with a session, remember what their last actions where and react to requests accordingly.
- *NON-SHARED-SESSION* services behave in the same way as *SHARED-SESSION* services do but every client has its own service object which is maintained until the session is closed by the client.

The major task of the service manager is to instantiate, store, reactivate and remove service objects according to the interaction mode specified by their programmer.

Service Parameters

There are two different kinds of parameters for a service. *Request parameters* are sent to the service by a client on every request and specify what exactly the service should do. *Service parameters*, on the other hand, are submitted to the service object only once at startup time. They may specify the general behaviour of the services influencing every served request. These service parameters are given to the service manager upon startup and cannot be changed afterwards except if the service object explicitly provides clients with the possibility to send requests which change them. A common location where a service and its parameters can be specified for easy service startup is a file which is read by the place during bootstrap. It is also possible to start the same service with different parameter settings. While a service object is instantiated, a special method is called with the service parameters as arguments which allows the developer to initialize the service object according to these parameters.

Designing a Service

The first fundamental decision of every service developer is the interaction mode. He has to analyze the nature of his service and specify the mode. Then he should develop the service protocol. Remember that services, like every other Place User, exchanges messages with their clients. The protocol should state which messages can be served and which messages will be send in response to which requests. As this protocol is developed, the programmer normally discovers which general service parameters are needed and starts to design the initialization process implementing the service object's `initService` method. When the protocol development is finished, the `startService` method is implemented. This method interprets requests and triggers the servicing of requests. AMETAS does not define the way in which the service is provided. It is up to the programmer to write the needed classes that may access the file system, communicate over sockets, query databases etc. Everything is allowed if the service is given the required permissions by the author, the user and the system.

Security restrictions may also be enforced related to specific clients. By specifying the *Required Privileges* for the service the author decides what permissions clients need to access the service. If a client does not have the required permissions, its request is not forwarded to the service object. The service manager enforces these restrictions, deciding which requests to forward and which requests to ignore¹.

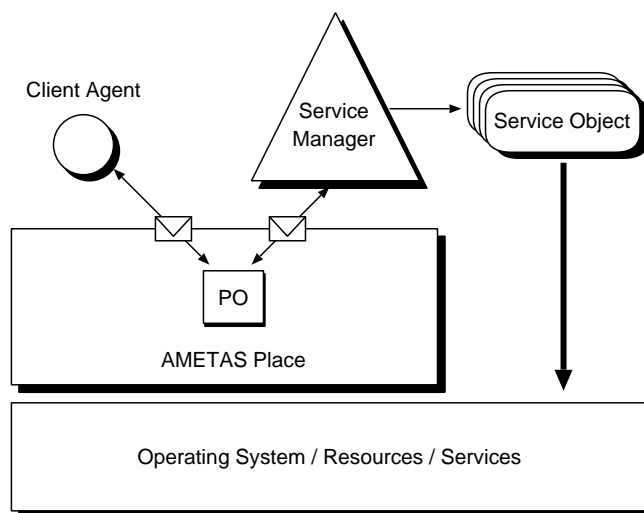


Figure 1: Client/server interaction in AMETAS

Let's give an example that might make things clearer. Suppose that a programmer faces the task to develop a network management application which enables agents to move between places and measure system parameters. The measurement of these parameters involves accessing a local SNMP entity via datagram sockets. This is something that an AMETAS agent will never be allowed to do on its own due to the security restrictions in AMETAS. Therefore, we need a service that can do this job for us. Clients should be able to request this service to send a SNMP message to the SNMP system and return the SNMP response. The service programmer will design a simple

¹Read the White Paper on security in AMETAS for more information on this topic.

protocol allowing this interaction and when a request arrives, the service will marshal the request's parameters into a SNMP message and send this message via UDP to the SNMP system on the local host. When the SNMP response arrives, the service will transform it into an AMETAS message and send it to the client. service parameters could be the SNMP sockets that clients are allowed to query or whether clients are allowed to manipulate system setting through SNMP set requests.

This example shows how AMETAS services may be used to allow agents limited access to system resources. The service controls which actions agents can execute on the resource. A client of the SNMP service is not given the right to do anything it likes on the SNMP system. Instead, by specifying the service parameters the person installing the service can restrict the clients' rights. Only a very limited part of the SNMP system needs to be made accessible for clients to prevent evil agents from tampering with the system and damaging it.

The SNMP service also gives an example for a service using sockets. Using sockets usually involves creating threads that listen for responses and data structures in which to store pending client requests. The programmer is not restricted in how he designs the service. Very complex and powerful structures can be built.

Using a Service

At message level there is nothing special about service usage. A client sends a request message compliant with the service's interface and will eventually receive a response message. It might implicitly rely on the interaction mode of the service but does not have to care about it explicitly. A difference to communication with agents or user adapters arises if we take a view on higher levels of abstraction. Since services are stationary and do nothing else than servicing requests, a client can to some extent rely on getting a response rather quickly. Agents on the other hand might not send a reply in a timely fashion. Therefore, we can apply abstractions for service usage that are more oriented towards normal programming paradigms and make life easier for client programmers. An example for this can be found in the AMETAS White Paper on communication where *Service Proxies* are introduced. These Service Proxies can be used by a client to hide the construction and sending of a request message, the blocking until a response message arrives and the enforcement of timeouts. Such actions are encapsulated in methods which results in a convenient method based service interface.

Before a client can send a request to a service, first it has to get the service's address. Every AMETAS place has a mediator which takes a description of a Place User and returns its Place User ID (its address) if this Place User is known to the mediator. This mediation process can be used to contact any Place User, not only services².

²Read more about mediation in AMETAS in the White Paper on types.