

# **AMETAS**

*Good Migrations.*

## ***AMETAS White Paper Series***

by Klaus Herrmann  
and Michael Zapf

# **On Types**

July 2000

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

[www.ametas.de](http://www.ametas.de)

## I Need an Agent That...

In classical programming paradigms, types are used to group entities that share some common structure and behavior. In other words, when you have an entity of a certain given type, you also know how to treat this entity even though you might work with it the first time. You also know how to treat any other *instance* of this type.

In agent systems, the notion of types is quite new. Agent applications that are designed to only work with well-known components normally do not need types. But in an open-world scenario you should expect various kinds of agents, coming from different authors, being equipped with different capabilities. Types can help to determine if the agent capabilities are compliant to some basic requirements.

## What Is Mediation?

Every Place User in AMETAS has its own unique identifier called Place User ID. Using this ID you can send a message to this Place User and be certain that only this agent receives the message. As long as you design your application as being *closed*, the communication will work flawlessly. But from that very moment on when you intend to design real agent societies that come from different authors and senders in order to collaborate, you will most certainly encounter a big problem: How do I get to know the ID of the receiver?

Normally, it is the duty of a repository to allow a mapping between agents and their respective IDs, but as you cannot present any reference of an agent<sup>1</sup>, what can you say about the agent in order to utilize such a repository? How can you *describe* it? The process of finding a special object by presenting a suitable description is called *mediation*.

Mediation can be used for two main purposes:

1. You need the ID of a running Place User in order to send it a message.
2. You need to start a Place User that has certain capabilities.

The first case is called *instance mediation*, the second one is called *type mediation*. Whenever you want to get in contact with an existing Place User, you need to use instance mediation. When you are in need of a suggestion which Place User to start for a given problem, type mediation applies.

## The Traditional Ways of Mediation

Let us start right off with the way of mediation that is supported in the versions of AMETAS as of June 2000. Later we will give you some brief introductions how the mediation will look like in the oncoming AMETAS version.

### Multicast

The simplest version of mediation is not really a mediation but a *selective broadcast* or *multicast*. That means in order to have a message delivered to a specific Place User, you send it to all currently running Place Users that belong to a special group. The Place User ID contains two fields called *Name* and *Group* (see Figure 1). While names

---

<sup>1</sup>No Place User may get any reference to another Place User.

should be associated with one special agent class (or collection of classes belonging to this agent), the group indicator is used to define groups of Place Users. These fields may be set by the Place User without any restriction, enabling it to join any existing group or defining new ones.

|            |           |      |       |
|------------|-----------|------|-------|
| IP Address | Timestamp | Name | Group |
|------------|-----------|------|-------|

Figure 1: The *Place User ID*

ID masks are similarly defined as IDs but allow null entries which represent wild-cards. In order to achieve a multicast, one simply needs to define an ID mask that is matched by the IDs of all Place Users in the group.

Actually, there are quite some drawbacks of this simple strategy. Several points should be stressed:

- How can I decide which of several matching agents is the right one?
- Other agents could use the same name or group in their ID. (It is just the first two fields that differ and make this ID unique.) How can I prevent other agents from getting a copy of the message when I do not want them to get one?
- How do I know how many agents actually got the message? As the sender, I do not get any information if and by whom the message was received.
- If the actual receivers are unknown: Which name and group should be used?

Multicast is no mediation. However, it is the fastest method to distribute messages among a group of Place Users without having to know the precise ID.

### Service Mediation

A better way to find a special instance is by using a mediator. AMETAS offers an implementation called *Trivial Service Mediator*. This mediator uses a table of currently running services to answer incoming requests by clients. The implementation is simple but, on the other hand, very fast:

- It can only mediate services. It does not handle requests for looking up agents or user adapters.
- It uses a string to find the service in the table.
- It can deliver a list of currently running Place Users.
- It only mediates running services. You cannot ask for a service that has not been started yet.

Strings to be used to look up those services normally have the format

*ServiceName#ParamSetName*

The name of the service is determined by the class name used for the main class of the service. The name of the parameter set denotes the set of parameters which have been passed to the service at the moment of startup. It is possible to start services in

```

public void useMovieService() {
    // Ask for a service called "MovieService"
    AMETASMediationResult[] amrs =
        m_Driver.request(new AMETASMediationRequest("MovieService", true));
    if (amrs != null) {
        // If a suitable service was found, get the ID
        AMETASPlaceUserID idService = amrs[0].getPlaceUserID();
        // Start the communication with the service
    }
    else {
        m_Driver.output("I did not find the movie service. Bye!");
        return;
    }
}
}

```

Figure 2: Using the mediator.

AMETAS using different parameter sets but same implementations. An example is shown in Figure 2.

As you can see, there is no message sent out in hope of some suitable object receiving it. Instead, the mediator is queried using the driver method `request` which returns an array of results. Assuming that the first hit is the right one, the code retrieves the corresponding ID and may now directly send messages to this Place User.

The agent system automatically creates the entries for the service table that is used during the query processing. The respective parameter sets and corresponding entries of the format *ServiceName#ParamSetName* must be defined in a special configuration file that is read during place startup. Using special methods, entries can also be added or deleted at run-time.

## Mediation — The Next Generation

Mediation as described above can only be used with services, and, more important, does not solve the problem with unknown names. If your agent actually does not know how the service is called, it will not be able to successfully formulate a request for the mediator. Therefore, a more powerful mediation system was designed which will be integrated in future AMETAS distributions. To give you an idea what you may expect to get, we describe the new system principles and how to use it.

### Types

The new mediation system is based on a *type system for mobile agents* (which will, of course, be applied to all Place Users). By *type* we understand a set of objects that abide to a special logical predicate. For example, the type *integer* consists of objects each one of which may represent one whole number in the range from  $-2^{31}$  to  $2^{31} - 1$  and allows certain operation with other objects of this type, namely addition, multiplication and so on. With Place Users, the situation is similar: Each Place User is written to perform some specific task, to serve some given purpose. The goal is to find a suitable formalization of this purpose.

As the message-based asynchronous communication is the only way of communication in AMETAS all Place Users may be characterized by the set of messages they accept from and those they send to other Place Users. We designed a type description language that allows to formulate the set of message types that are used during message

exchange, the sequence of messages, and also the meaning of messages and the possible area of application of the Place Users. The semantic informations are represented using a special knowledge representation technique called *conceptual graphs*.

### An Example

Figure 3 gives you a simple example to illustrate what such a type description could look like.

```
messages {
  in {
    I1: { AMETAS.data.ALong };
  }
  out {
    O1: { java.lang.String };
  }
}
states {
  S1 = I1 > S2:O1;
}
annotations {
  self=CG: {
    [Agent]->(has)-> [Name:{Test1}]
  };
}
```

Figure 3: Simple agent type description

As you can see, the agent is declared to accept a message with one item of class *AMETAS.data.ALong* and to emit a message with a string inside. The states part declares that the agent is in the state *S1* at the beginning and changes to state *S2* on receiving the message labeled *I1*. The state change is accompanied by the emission of a message labeled *O1*. The annotations section can contain annotations of messages, message items, or states. The *self* annotation describes the agent itself. This object is described as being an *agent* that has the name *Test1*.

### Mediation by Types

In order to find an agent with of the type described in Figure 3, the requestor must formulate a suitable request. The request need not match exactly, it can also be more general than the description that has to match. For example, it is valid to accept more types of messages than expected; vice versa, it is also valid to send fewer types of messages than expected. However, when a state transition graph is given, every sequence of interactions that is expected (accepting a message and sending another one) must be represented in the agent that is presented as a search result.

Concerning the semantic information, the conceptual graphs themselves support the notion of generalization. So the request to find “an agent” is a generalization of finding “an agent called *Test1*”. Actually, the user will normally hardly know more than the intended usage contained in the self annotation so that he cannot formulate a more precise request than to simply use a self annotation. The messages that are accepted and sent can then be retrieved from the type description instance contained in the mediation result.

In practise, a request is formulated similarly to the code in Figure 2; however, it is required to get a type description instance at first. This can be achieved by passing a string containing the description to the mediator which creates a run-time representation as an instance of the class *AMETASType*. This instance must be used to create a *MediationRequest* and will then be passed to the mediator by the driver method *request*. As a result, the mediator will again return an array of possible matches.

Unlike the traditional mechanism, the type system may also be used for inactive Place Users. The PUs are stored in container files that are equipped with the corresponding type description. This description can be registered in the repository to allow requestors to search for a convenient Place User to utilize for some task.

### **Availability**

The type system as described above will be available in Q3 2000.