

AMETAS

Good Migrations.

AMETAS White Paper Series

by Klaus Herrmann
and Michael Zapf

On Writing User Adapters

July 2000

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

www.ametas.de

Writing User Adapters

The AMETAS user adapter is a concept that cannot be found in this explicit form in other agent systems. It was designed to integrate the human user into an agent application. user adapters reside on the same layer as agents do but they exhibit properties that do not comply with agenthood. This White Paper explains user adapters and details some issues involved with writing them.

What's the Users Role in the Whole Story?

People tend to talk about single isolated agents that may communicate with other agents but normally seem to mind their own business. But really powerful agent applications are most often multi agent applications where the single agents cooperate to solve a common task. In such applications the human user must have some means to control the general direction. Thus, the user has to be integrated into the system in some well-defined way. He has to be presented with a standard user interface that receives his actions and transfers them into the agent system to instruct the individual components.

The User Adapter as a Translator

The user adapter concept was designed to be a translator which takes normal user actions like a pushed button and triggers the associated actions within its multi-agent application. In AMETAS these actions can be the start of an agent or sending a message to one or several participating Place Users. Therefore, a user adapter can be viewed as a translator between user actions and Place User actions.

Integrating the Human User

A user adapter is stationary by nature, i.e. it cannot migrate to another place. It normally displays some kind of graphical user interface (GUI) that gives a human user a means to influence the respective application. The user adapter by itself is not an agent since it does not enjoy any autonomy. However, the picture changes if we look at the combination of user adapter and user. This combination can be viewed as one AMETAS agent. The human user provides the intelligence and autonomy while the user adapter can communicate with other Place Users using messages. Actually, other Place Users perceive this combination by exchanging messages with it and do not recognize any difference to other Place Users. Viewing things in this way makes life easy since we only have to deal with Place Users.

The technical side of programming a user adapter is much easier than developing an agent. The reason is that we do not have to care about mobility and its effects. The user adapter is also developed by implementing the `invoke` method as the entry point. Initializing it with a message is not necessary since it can be manually initialized by the user or it can read configuration files from disk. It is most often started by using one of the AMETAS administrative interfaces, AMAI or AMSI. Generally, we can say that programming user adapters is much more like programming normal Java applications. The user interface, for example a GUI, can be designed in just the same way. A user adapter may enjoy much more freedom when it comes to accessing system resources.

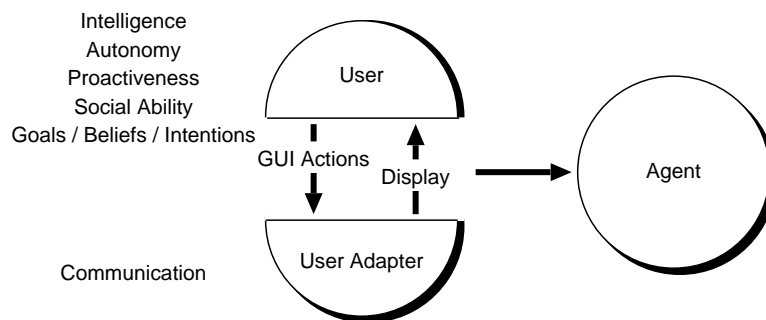


Figure 1: Combining user and user adapter to form an agent

If it is granted the necessary permissions by the author, the user and the system it may open sockets, access the file system or even start other programs.

However, one small restriction exists. At bootstrap time we do not have the possibility of reading environment variables. Such variables are a common means for specifying certain properties of an application. Since the Java VM is already running as we start a user adapter and since the possibility to access environment variables at runtime was removed from the Java API, we need a fixed directory in which to store the files needed by the Adapter. This directory is configured via the place configuration file and stays fixed for the entire life time of the place. All the files needed for bootstrapping a user adapter need to be located relative to this directory in order to be found by the Adapter. This is not really a restriction. In fact it gives us the possibility to move this root directory elsewhere without the Adapter noticing it.

Integrating Non-Agent Applications

By integrating the user into an agent application, we make this application accessible for him. But what about other, external applications that might want to use AMETAS agents? Well, there is no law that says “user adapters may only display GUIs as interfaces”. In fact, the interface could just as well be an application interface, for example listening on a socket processing some kind of network protocol from a legacy application. Or it could provide a CORBA or Java RMI server which may be called by clients to do some agent-based work. Such an Adapter would translate the requests received over this interface into agent messages just as other Adapters translate GUI input.

Imagine a CORBA server that takes requests from clients directing it to search for some information in a computer network. This search engine was implemented as a normal program opening sockets, retrieving information and filtering it according to the parameters specified by the clients. Now we decide to do this job with mobile agents in order to save bandwidth. We could write a user adapter that implements the interface of the search engine and thus looks just like the old application to the clients. They will not notice the difference. The parameters of a client’s call are now marshalled into an agent message by the user adapter. A new search agent is started and handed this message upon initialization. The agent wanders off and tries to find the desired information. When it returns, it sends back the search results to the user adapter which in turn unmarshals the results and returns them to the client. This simple example shows how flexible the user adapter concept can be applied not only to integrate the

human user but also to serve as an interface for external applications.

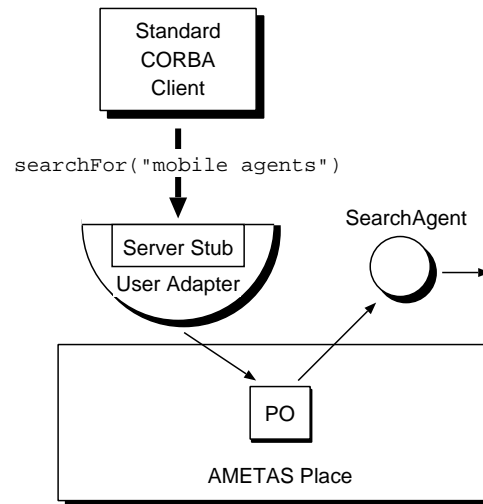


Figure 2: Integrating non-agent applications

Note that the search agents in this example demonstrate the effect of decoupling Place Users by introducing asynchronous messages. All they do is read an initialization message upon startup and go searching. When they are done, they return to the place they started from and deposit the result as a message in the Post Office. To them it does not matter who deposited the initialization message or when it was deposited. Moreover, they do not care if or when the result message is retrieved. Nor do they care by whom it is fetched. This makes them universally applicable. Whoever can produce such a initialization message is able to use the search agent, may retrieve the result later on and can display the result or hand it to other entities for further processing.