ESPRIT Project No. 25 338

Work package H

User Access

Design & Interfaces

ID:	DH3 & DH4	Date:	13.3.98
Author(s):	Elcha Triep, Michael Breu Fast e. V.	Status:	Version 1.0
Reviewer(s):	Douglas Donalson	Distribution:	Internal

Change History

Document Code	Change Description	Author	Date
Design & Interfaces	First version of document. No changes.	Triep	22.Jan.98
	Update	MBR	30.Jan.98
	Update to V1.0	Triep & MBR	18. Mar.98
	Inclusion of internal review remarks	Triep & MBR	6.May.98

1 Introduction

User Access is an independent service of FollowMe which is built on top of the MOW[4] and can be used by agents/applications to communicate to the user.

1.1 Status of this Document

This document describes version 1.0 of the design of the User Access service (UA). It also covers an overview of the API. The details of the API is documented in JavaDoc and will be provided with the source code. The API is still evolving, i.e. additional functionality will be added in later versions.

1.2 Overview

A key objective of the project is to free the user from a fixed desk-top, i.e. to support mobile users, allowing them to access their agent through a variety of different media without loosing quality of interaction.

FollowMe provides a personal assistant [2] that is mobile between various places in the internet/intranet and supervises the carrying out of selected missions. The user must be able to find and to communicate with his personal assistant interactively. Vice-versa the personal assistant must have facilities to send the user information about his missions, although he is not on-line. To this end, the personal assistant maintains a diary that contains connectivity information for each user.

The user access service is a component of the FollowMe architecture that provides standard means for agents/applications to interact with the user. Both, off-line delivery of documents to users via e.g. fax or phone and on-line interaction are provided.

Main requirements are

- the support of input/output and output only devices via common interfaces,
- the provision of a generic mark-up language to describe the contents and the layout of the rendered documents, and
- mechanisms to adapt the transmitted information and layout (i.e. the quality of service) to the performance of the system.

The following two chapters describe the requirements briefly and the architectural design. Additionally an overview to the API is given. The document concludes with an example of how this API can be used by a FollowMe mobile object.

2 Requirements

The analysis of the requirements [1] elaborated the objects and functionality of UA, and described them in the use cases, that are supported by the User Access work package. This section recapitulates these requirements briefly.

2.1 Purpose of the User Access work package

User access is in charge of the communication between FollowMe applications/agents and the user. The user access work package will provide a framework for handling the communication and a set of demonstration implementations for a wide range of devices types.

Since agents are typically executing while the user is offline, the agent must be able to actively send information to the user (e.g. by fax or phone). The user must also be able to contact his agent(s) and give them necessary input.

Two major groups of communication devices [1] are supported in response to the two types of communication required between agents and users:

- Output only devices (e.g. classical fax, mail, printers, SMS-Gateways).
- Input/output devices (e.g. a Web-browser): These devices can be used to interact with the FollowMe objects, i.e. to provide input from the user.

The approach is generic in the sense that in future more devices can be incorporated as they are required by applications. This will be achieved by a common API for all devices, and a common mark-up language that allows generic description of the information and layout for a set of devices.

2.2 Basic axioms of User Access

The following axioms form the basis of User Access architecture.

- UA is a service of FollowMe, which is built on the top of the MOW [4] and FlexiNet services like traders [3].

[1] This distinction between the two groups is due the fact that FollowMe was conceived to support only Java enabled devices, where the interfaces to the devices are rather similar. The rare presence of such devices on the market forced us to support non Java enabled devices, but the complexity of the input/output communication for such devices and in general the typical nature (output only) of non Java enabled devices, drove us to the decision of supporting them within FollowMe just as output devices. Special cases like normal http browser and telephones (interactive and non Java enabled) are considered separately.

- UA supports a mobile user, i.e. the user can be reached and can contact the system through different devices at different locations. The approach is generic, i.e. new devices can be added and supported progressively by FollowMe project.
- UA supports a Mark-Up language (see 3.1.3 later in this document) that provides standard means to define information and its layout. This mark-up language will be used by agents/applications as a common way to describe information and layout of the data to be sent to the user. The user access service provides wrappers to map documents in mark-up language into suitable formats of the output device.
- UA provides mechanisms to adapt the information and its layout to the performance of the transmission in order to achieve a certain quality of service.
- UA provides transparency of location in the sense that agents contact UA service and the service is in charge of finding a required device capable of carrying out the delivery of the information. It is supposed that not all UA services are capable of handling the same devices.
- The interactive communication between agents and users through UA is supported by input/output device gateways. The device gateways are explained later in this document.
- The user access service provides facilities to convert documents from one specific format to another that is better suited for transmission (e.g. text to fax, or text to speech[2]).
- An agent can use any local interface to interact with the user (e.g. java.awt). There will be no restrictions but also no support from user access for these interfaces. However the user access service provides an input/output facility, built on the top of UA API, based on AWT.

2.3 Typical Use Cases

2.3.1 User is off-line: Output only devices

The agent wants to send data to the user that is currently off-line. It looks up the diary in the personal profile to discover through which device, like fax, phone, SMS, etc., the user has to be reached and some characteristics of this device; that is valid at the time of transmission. The agent prepares a document with the information marked up with the data and its respective layout in the form of an XML/XSL document. Then, it contacts an available UA, which is in charge of opening a connection to the end user and sending the information provided by the Agent. The agent may receive an error message, alerting of some errors in the transmission or a transmission close, which indicates a successful transmission of data.

2.3.2 User contacts his personal agent through an entry point: I/O devices

The user contacts his personal assistant through an entry point to the system, e.g. the URL of an http-server that provides an entry point service. The entry point is attached to an input/output device gateway. The entry point offers the user two options: search for his personal assistant or creation of a new personal assistant for a new user.

If the user chooses the option ”search for the personal assistant”, UA looks up the agent via its name in an agent directory[3], sends an event to the found personal assistant, informing him that his user is on-line and giving the personal assistant the connection details (device type, format to be used, etc.). The personal assistant reacts by sending the relevant information (e.g. a form or an applet, or any other information) that will be processed via the connection and finally sent it to the user. UA service captures the responses of the user and notifies the agent through an event containing also the user responses.

[2] The implementation of advanced converters is not part of the user access work package.

[3] How this directory is managed, is in the responsibility of the application. E.g. for Bavaria-Online users there will be a Bavaria-Online-Agent-Directory that holds information about all Bavaria-Online Personal Assistants. The Entry Point may have to be tailored to look up personal assistants in this directory. How Personal Assistants are named is also in the responsibility of the application.

If the User chooses the option ”creation of a new personal assistant”, UA contacts the personal assistant factory and ask it for the creation of a new Agent for an on-line user. The factory creates the agent and (in general) it will move to the place associated with the entry point. The personal assistant gets an ”user is on-line”-event. The rest is as in the previous case.

For each on-line user, the capabilities of the end user device will be captured by UA and transmitted to the agent. Thus, the agent may build appropriate deliverables for his on-line user.

2.3.3 The user is able to run a local place

The user can run a local place. Either he has downloaded the place software before and started it, or he uses mechanisms to run a place in an browser applet.

A place provides a window as a special device that acts as an entry point. It enables the user to select between the two options: create new personal assistant, or search for a personal assistant. The agent is created/searched, and informed that its user is on-line. The agent can use standard user access mechanisms that are displayed in a standard window to contact its user locally.

3 Design

In the following, we describe the overall design of user access, its principles and the objects that define the service's architecture and its functionality. The interaction between User Access service with other FollowMe objects is explained through interaction diagrams.

3.1Architecture of User Access

3.1.1 Overview

Each host, participating in a FollowMe application, has a set of local *devices* attached (see e.g. host B in Fig. 1). Devices could be e.g. a fax card, voice modem, a web-server capable to display HTML on web browsers, or an applet that acts as an (sophisticated) input/output device. These *devices* typically establish a *connection* to the user's end device. The characteristics of these connections should vary depending of the device type which one is dealing with and the need of the users, this connection is represented by an instance of the connection class.

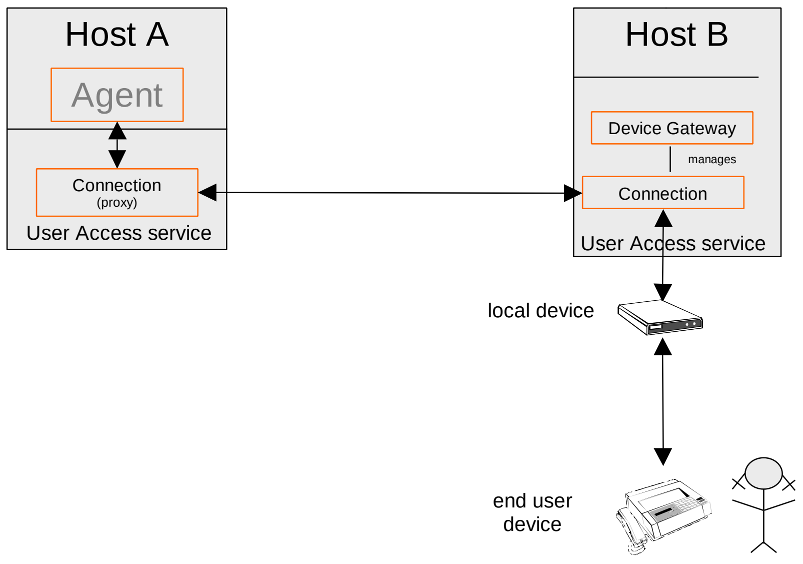


Figure 1 Basic communication schema between agent and end user

Regarding this model, User access supports the creation of *Connections* to the user's end devices, i.e. for an agent, the user is represented by a connection. A connection can be described as an open channel between agents/applications and users as long as they communicate. It is important to realise, that agents and users are able to close the connection and interrupt the communication between them at any point in time.

The advantages of having an open Connection between two entities which communicate with each other is to avoid wasting time and effort when opening the Connection every time one has to communicate with the other, to have the possibility of send more than one document or information piece at a time, to avoid to ask a trader or agent directory for an agent reference, every time the user needs the next data (that is quite often when using for example a Browser). This approach can also be used when the communication is one way, i.e. the agent has to report something to the user, send important information or just warn him about some fact : the agent opens a connection, sends the data through it and closes the connection.

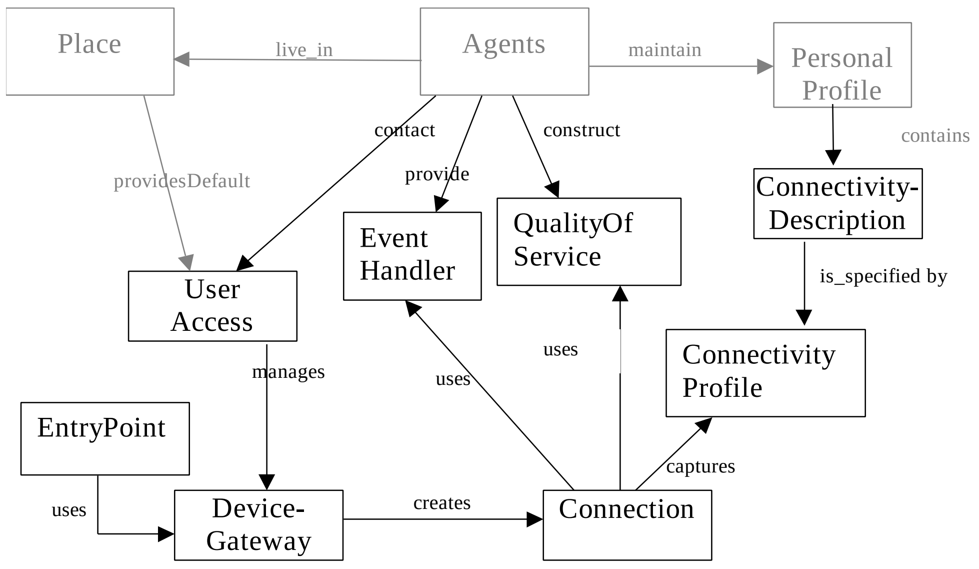


Figure 2 Domain Object model of main User Access objectss (Classes outside user access are shown in grey)

The arrows signify the relationships between the different objects related to User Access, they describe in an informal way the use of them by User Access. The names of the arrows do not represent the methods used to be accessed by the different object, instead the methods are depicted in 3.3 by interaction diagrams.

User access manages the local devices (fax card, modem, etc,) present at a host with the help of *Device gateways*[4] and it is capable to contact other user access service in case it cannot carry out the requirement of establishing a Connection to the end user, i.e. the devices which it manages do not respond to the required capabilities of the desired device or the local device with such capabilities is unusable at the moment the delivery has to be done.

Device Gateways support the standard capabilities of the devices attached to a host but may also provide support for additional features like translation from one format into another, e.g. text into fax format, XML into HTML, etc. Each device gateway is at least capable to transmit data that is marked-up as an XML document (see 3.1.3). The layout is defined by an XSL. This layout may be device specific or generic for a set of devices.

The Device Gateways are the abstract representation of the devices attached to a host with the capabilities of creating connections of its type, for example a fax gateway managed by user access is capable of create a connection for an agent that wants to send a fax to the user. It is also capable of translate an XML document into fax format.

The agent contacts a UA service and asks it for a Connection (e.g. via a fax device) to use. The Connection properties, i.e. capabilities of the device, like html/text, audio, etc., are specified by the connectivity description that is found in the profile. The user access service looks for a device gateway that can support the requirements given in the connectivity description. This can be either a local device gateway or also a remote device gateway[5] which is completely transparent for the Agent. The agent uses the Connection object to send the user information.

[4] Device gateways can be compared to device drivers in standard operating systems

[5] The selection of the device gateway will be in future implementations supported by a trader that knows the capabilities of various device gateways and selects an appropriate one.

In the current version the capabilities already contain the location of the device gateway to use.

Connections are described by their capabilities, e.g. the mime type of documents it is able to display (e.g. text, gif, wav-audio, …) and also the parameters of the end device as e.g. the display size, colour capabilities, text length supported (e.g. for SMS); this information is specified by a Connectivity Profile.

Some Quality of Service (QoS) will be provided in order to improve the use of resources and network traffic. Regarding that, the agent/application construct a quality of service object, which represent the desired rules to be applied depending on available resources and network traffic. This QoS will be used for the Connection to send the user data appropriately, e.g. when the network load is too big, the user may receive black and white data instead of coloured images depending on the agent/applications preferences, i.e. the agent has to decide if it is preferable that the user receives a coloured image at a very low speed or if it is better to send the user just black and white pictures.

As already mentioned, agents send the user data through the connection. In order to ask the agent for the next data, the connection uses an EventHandler which is provided for the agent. This EventHandler provides input and output parameters that will be used to receive data requirements from the connection through the input parameter and to answer/respond the connection using the output parameter (see 3.1.2).

3.1.2 Interaction between Connection and Agent

The Information to be sent to the user can have quite a complex structure due to its multi media nature. It can contain text, pictures, audio, etc and accompanying layout information. To allow the transmission of multi-part documents, the interaction between a Connection and the agent works on a client/server paradigm using the Java-servlet mechanism [5].

The agent exports the interface of an event handler to the connection. The connection can then use the EventHandler to request further parts of the multi-part document and to signal events back to the agent (see Fig.3)

This approach solves the problem of the agent sending all parts of a complex document in advance. Also an interactive communication can be modelled. The connections are able to request dynamically for the next data.

The motivation behind this is to have an agent attending permanently the needs of his users as soon as they connect the system and contact his personal agent, taking in consideration that the personal assistant/agents are the ones who provide the user with access to the information stored in IS or to the information retrieved by agents/applications. Thus, once the user has contacted his agent, the agent sends him a document and the user can inquire for the next data (next document, an image, etc.). This way, the agent acts as a server and the connection as its client.

An EventHandler implements a method called `service(FollowMeServletRequest request, FollowMeServletResponse response)`. In simple terms the FollowMe`ServletRequest` describes the requested piece of information (e.g. an URL describing a file "agent.xml", an image, next document, next page of a document, etc.), the FollowMe `ServletResponse` is an object that accepts an input stream with the requested information.

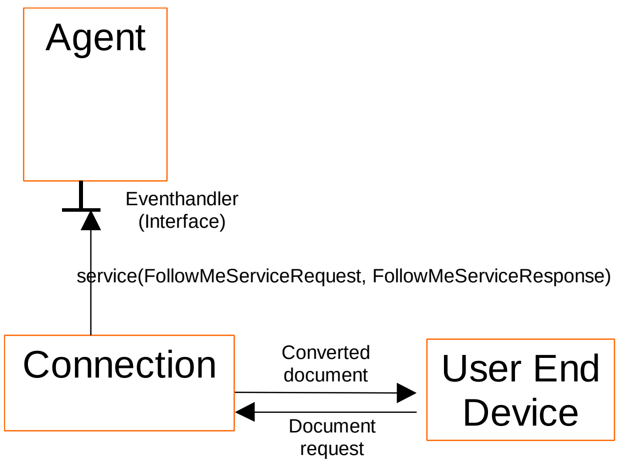


Figure 3 Interaction between Agent and Connection

Basically there are two ways to open a connection:

- The agent actively requests from user access the opening of a Connection with the openConnection method. Argument to this method is the ConnectivityDescription which describes how the user can be reached, i.e. the device type to be used, number, IP address and its capabilities, like format (text/html, jpeg, etc) and properties like screen size, maximal text length supported, etc. These capabilities are encapsulated by an object called ConnectivityProfile, which will be stored in the User Profile associated to each device the user can be reached through, i.e. each possible device present in the User Profile will contain a reference to an object called ConnectivityProfile that describes the exposed features.
- The user contacts an EntryPoint interface. This interface will be provided at well known hosts for each media through which the user can contact FollowMe, e.g. the user call a well known number if he wants to reach FollowMe through telephone, or an http-address if the contact is through a browser. This EntryPoint interface is in charge of opening a Connection and it sends the PersonalAssistant of the user setConnection () indicating a "user_is_online" event . set-Connection() carries as an attribute the connection via which the user is online and the ConnectivityProfile of the user end devices, which was captured by the EntryPoint. The agent can then use the `send` method to interact with the user, providing the Connection with an EventHandler reference through which the Connection can ask the agent for the next data.

The `Eventhandler` is used to retrieve document components of a structured document from the agent and to display it on the end user device. But it is also used to receive feed-back from the end-user device (e.g. the contents of an HTML form returned from the user).

In this case the `ServiceRequest` contains the data transmitted back to the agent. The `Eventhandler` has to analyse it and to send the respective response, if applicable.

Asynchronous events like setfocus, onclick, etc. are transmitted by the method eventReceiver(Event). Not all interactive devices (e.g. WWW-browsers) will support asynchronous events. Such asynchronous events are related mainly to small actions made by the user which does not necessary involve the requirement for the next data. Instead, they reflect the behaviour of the user by using/reading, etc. the data. To capture such events from the user end device, we use the Java event handling mechanism.

3.1.3 Generic Mark Up Language XML/XSL

The main motivation to use a Mark Up language to represent data and layout was born from the idea to have a general, common way to represent data and layout that has to be delivered to different users and devices with different capabilities like audio, html/text, etc.

Why a mark up language to solve this problem?

Some way to represent structured data was needed, some way to write documents in a form that allows to describe their own grammar, i.e. describe elements and the structural relationships that those elements represent. We need extensibility, allowing agents to create their own elements representing data, managing of structured data with a high level of nested complexity, some common way to describe different views of the same data. XML emerged as a standard which provides such features. In parallel, XSL emerged as the complement of XML to be used to describe the layout of the data contained in an XML document

Well-Formed XML Documents

A textual object is a well-formed XML document if:

1.Taken as a whole, it matches the production labelled document.

2.It meets all the well-formedness constraints given in this specification.

3.Each of its parsed entities is well-formed.

Matching the document production implies that:

1. It contains one or more elements.
2. There is exactly one element, called the root, or document element, no part of which appears in the content of any other element. For all other elements, if the start-tag is in the content of another element, the end-tag is in the content of the same element. More simply stated, the elements, delimited by start- and end-tags, nest properly within each other.

As a consequence of this, for each non-root element C in the document, there is one other element P in the document such that C is in the content of P, but is not in the content of any other element that is in the content of P. P is referred to as the parent of C, and C as a child of P.

The language definition is not reproduced here, but can be found at [8], [9] and [10].

Use of XML/XSL by agents/applications

The model described in 3.1.2. is capable to transmit any type of information to the end user device, provided the device gateway that supports this type of information (i.e. the respective Mime-type).

Agent are ask to send UA/Connections data and layout coded in XML/XSL format respectively.

The XML document can hold arbitrary structured information (see [9] or e.g. the example in section 4).

The layout of the information is defined in XSL (see [8] or [10]. The XSL file defines for one or for several different devices the layout format. The Connection transparently applies converters that combine the XML and the XSL data and produces a format suitable for the end device (e.g. HTML, RTF, plain text, etc.).

If the device type is already known, the XSL can already be tailor-made for this device type. Otherwise the XSL defines a set of different layouts for all devices to which a certain piece of data is transmitted. The different layouts are defined in different modes of the XSL layout rules.

The corresponding XSL-file is referenced in the XML-file by the <?xml-stylesheet href="..."?> tag. After parsing the XML-file the XSL-file is requested from the agents, and both applied to convert to the appropriate layout.

An example of an XML file ”newspaper.xml” and an XSL file ”newspaper.xsl” can be found in section 4. The XML file holds a very simple XML representation of a newspaper containing articles with title, text and pictures. The XSL file provides rules to translate them into HTML format.

The translation rules support two modes ”default” and ”textonly”: The default mode translates the layout into an HTML format with <IMG>-tags for pictures. The textonly mode translates the layout into an HTML format with alternative text instead of the <IMG>-tags.

XML/XSL converted to device specific data

As already mentioned, the agent/application asked the UA service to open a Connection to send the user data , i.e. to send data to a specific device with specific characteristics. The characteristics of the Connection are encoded in the ConnectivityDescription throw the definition of a device to be used and the properties and format the user end device can handle.

Once the agent/applications have sent the XML/XSL data to the Conenction (e.g. FaxConnection, MailConnection), the Conenction process the data using a XML/XSL converter[6]. This conversion implies to parse the two files, reporting errors(e.g. XML file not available) and refusing to proceed when the documents are not well form and converting the XML/XSL data to a document adapted to the necessary format (e.g. HTML for Browser, plain text for mail). Such process will be done internally in the connection throw the method process of the class XMLXSLConverter.

process(inXML, inXSL, out, RequiredEnd, mode)

where inXML, inXSL are objects that represent the XML and XSL files respectively , out is an object that represent the specific device type converted document, RequiredEnd indicates the required format (e.g. ASCII for mail, HTML for Browser) and mode indicates which mode specified in the XSL document has to be applied (e.g. plain text mode, black and white mode for images). The modes are related to the QoS, e.g. if the QoS rules indicates that at a specific network load rate, the black and white images have to be send instead of the coloured images, the mode will signalised a black and white mode. Then the converted document will be send to the user. The process to send the user the data is dependant of the device to be used, e.g. a MailConnection will use a SMTP class to send a mail. The class provides a method to open a socket for a mail server an send through it a mail.

Is it possible using XML/XSL to describe every data and view for every device?

As already mentioned, the design of user access uses XML/XSL as a way to describe the information and it layout which will be send the user by the agent. Some special cases like applets will be consider later and probably an extension to the supported

[6] A XML/XSL converter called “docproc” written purely in Java is used to convert XML/XSL data into a device specific format. Docproc was written by Sean Rusell , http://javalab.uoregon.edu/ser/software/docproc/index.xml. The converter supports the conversion to HTML and ASCII formats, but it can be extended to support additional features. Fast e. V. modified it in order to adapt the converter to the FollowMe needs.

XML will be done as far as XML is not adequate to handle with applets. A more complete User Access may support services that delivered other data not codified under XML/XSL, but in the short time not support for such services is provided.

3.1.4 Quality of Service

Quality of Service (QoS) issues can affect environment components including devices, networks and operating systems. For example, disks have small amounts of space, networks have limited bandwidth for data transport; operating system have a finite amount of processing resources.

The Quality of Service starts the adequate transmission and presentation of the document.

If the Device Gateway used to transmit data is remote, User Access may install/use a local *Connection* that acts as a Connection proxy by means of filtering the data to avoid unnecessary transmission, i.e. UA install a normal Connection which will filter data when appropriate before it sends the data to the end user, as expected, this filtering happens locally where the Agent is living . This is transparent for the agent. Through this mechanisms the transmission of data can be adapted to the performance of the system and the network (e.g. the quality of the layout adapted to the throughput of the system).

To achieve this, the agent may provide in the XSL-layout a set of different data presentations and layouts, e.g. a full colour mode, a black and white mode, a text-only mode, or an audio mode. Depending on the network load at transmission time and the capabilities of the end device one of these modes is selected and the data adapted before transmission

The agent can provide a quality of service object when sending data through the Connection The QoS object defines rules to be applied according network traffic, disk space of the end user devices, display possibilities of them, etc. Before the transmission of the deliverable, the object is asked for the most appropriate mode according the available resources to transmit/receive the data, where the monitoring of such resources will be performed by Service Deployment services [10]. With this mode the XSL-layout rules are applied with on the XML-file.

If no QoS object is provided, User Access applied the default mode reflected in XSL file without taking care off improvement possibilities, filtering of unnecessary data.

A *Filter object*, which is an internal object of UA, provides functions to filter out unnecessary data before transmission. Thus the transmitted data can e.g. be adapted to the capacity of the network.

3.1.5 New supported devices and their integration to FollowMe

The long term strategy to allow the dynamic incorporation of new gateways and capabilities include that agents do not need to be reprogrammed to take advantages of them, i.e. they send a delivery (data, layout) and UA will adapt them, as best as possible, to be delivered to the end device. This approach has a lot of technical difficulties, due to the wide range of possible devices to be incorporated, different platform, not enough Java techniques available in the market etc.

In the short term, the idea is that each new supported device's (when talking internally about UA, each device is a device gateway), classes have to be copied intothe existing directory structure and the Property file reedited manually to include there the name of the new device gateway supported, the name of the class and its parameters. This file is present in the User Access directory for each UA; it contains all the devices supported in the host where the useraccess service is functioning with its respective classes and the parameters needed forthe functioning of that devices. The inclusion of the new device in this file, will be noted by a new initialisation of the UA and it will be register the new device gateway with a trader, then every agent is able to use it. Unfortunately, the agents programmers haven to be told about the presence of a new supported device.

Example of a typical Property-file :

```
Device1=fax
fax.class=DE.fast.followme.useraccess.device.faxdevice.FaxGateway
fax.faxprog=C:\\WINNT\\FAXmaker\\SENDFAX.EXE
Device2=mail
mail.class=DE.fast.followme.useraccess.device.maildevice.MailGateway
mail.mailhost=194.59.176.1
Device3=http
http.class=DE.fast.followme.useraccess.device.httpdevice.HttpGateway
```

3.2 Overview to Classes and Interfaces

For a complete overview on the API see the documentation on http://www.fast.de/FollowMe/UserAccess/doc .

3.2.1 Interface UserAccess

The UserAccess

- manages a set of local device gateways
- is able to find and to establish contact to remote device gateways

A User Access is a service built on the top of FlexiNet services (UA uses some methods of FlexiNet/MOW, like traders) which support mobility of users. From the agent point of view, it acts as a factory of Connections which will take place between agents and user.

The main method of the UserAccess is

```
public Connection openConnection(ConnectivityDescription ConDes)
   throws DeviceNotAvailable, DeviceError, DeviceTemporaryNotAvailable
where
```

- ConDes - indicates how can the user be reached (devicetype + Number (IP address, phone number, etc.) + host). It encapsulated also the capabilities (audio, text/html, display size, etc.) of the device to be used in the ConnectivityProfile object.

3.2.2 Class ConnectivityDescription

The class ConnectivityDescription represents a description of how can the user be reached. It will be used internally by User Access and provided as parameter when sending data through the Connection by the agent.

It consist of four components :

- Device type (e.g. fax, phone)
- Device number
- Place (host, e.g. @followme.fast.de) : it will be not necessary when User Access Trader is implemented.
- ConnectivityProfile

In future versions of user access the explicit naming of the host, where a device is attached, will be replaced by a trader that allows to look for the most appropriate Device Gateway.

The ConnectivityDescription provides access to an object called ConnectivityProfile. This object represent some capabilities, formats and properties of the device the user has to be reached through and is related to each device the user will be reached through, e.g. if the user has to be reached through a telephone, the User Profile contains a tag indicating

<phone> 4545666

<ConnectivityProfile> ..... </ConnectivityProfile>

</phone>

```
public void  setDeviceType(String Type)
public void  setDeviceNumber(String Type)
public void  setDeviceHost(String Type)
public void  setConnectvityProfile(Format, Properties)

public String  getDeviceType()
public String  getDeviceNumber()
public String  getDeviceHost()
public ConnectivityProfile  getConnectivityProfile()
```

3.2.3 Interface Connection

The connection represents from the agent point of view a "communication line" to the end user device, e.g. a fax machine, phone, etc.

The agent send data through the connection and the connection send the data to the end user.

A connection is created by a DeviceGateway. The output, i.e. the deliverable is send via the send method. The input is received via the event handler.

The major method is

```
public abstract void send(Object deliverable,
                           EventHandler eventHandler,
                           QualityOfService QServ) throws DeviceError)
```

where

- deliverable - is an Object indicating the data to be send to the user. Examples of such objects can be XML documents, a String containing a name indicating the required data, another file, etc.
- eventHandler –is the agent's EventHandler through which the Connection can ask them for the documents to be sent.
- QServ - is the quality of service object . It will be defined by an agent/application which wants to implement some Quality of service when interacting with the user.

The agent can close the connection using the closeConnection method.

```
public abstract void closeConnection()
```

3.2.4 Interface DeviceGateway

A DeviceGateway is a Device or a Gateway to a device that is able to establish a connection to the end-user. A Gateway may be more than a Device since it can also do format adaptations, etc.

Examples for DeviceGateways are Fax device gateways, phone device gateways, http device gateways, etc

The DeviceGateway allows opening of connections. It can be thought of as a factory of Connection of the type of the device gateways.

This class is sub-classed according to the different devices types that are supported by FollowMe. It will be extended/subclassed by devices that will be supported in the future by FollowMe. This class is in normal cases not visible for agents.

3.2.5 Class Connectivity Profile

A user end device is described by its format capabilities like text, graphic, jpeg, etc and the properties of device like screen size, resolution, etc. The format is described by the MIME types supported by a device, e.g. a fax could support text/plain.

The ConnectivityProfile is related to every device the user can be reached through. E.g. if the user can be reached through telephone, mail and a normal Browser, there is a ConnectivityProfile object in the user profile for each of these devices which represent the characteristics and capabilities of such devices.

Example of a connectivityProfile as part of the User Profile

```
<ConnectivityProfile>

         <format> text/html </format>

         <format> audio/wav </format>

        <format> graphic/jpeg </format>

        <properties> <screen size> 100:200l </screen size> </properties>
```

......

</phone>

```
public void setFormat(Format capabilities)
public void setProperties(Properties capabilities)
public Format getFormat()
public Properties getProperties()
```

3.2.6 Interface EventHandler

The EventHandler is an object that handles service requests from a connection. It works like a simplified servlet [6]. Its main purpose is to accept service requests from the Connection, analyse them and to return the appropriate subdocument. Its most important method is the service method:

```
public abstract void service(FollowMeServletRequest req, FollowMeServletResponse res)
```

- The FollowMeServletRequest describes the service requested. It can be queried for the name of the document and potentially parameters of the service request (e.g. if a form is filled out by the user).
- The service method uses the FollowMeServletResponse to return a document component to the Connection. The FollowMeServletResponse can be set to a Mime-type and a dedicated outputstream that contains the document.

3.2.7 Interface QualityOfService

The interface QualityOfService is an interface to an object provided by agents to provide rules to be applied according the available resources to send/receive data. It will be used internally by user access to send the user appropriate data by taking decisions according to agent/application provided rules.

3.2.8 Interface EntryPoint

EntryPoint will be used internally by user access. It provides an entry for user who want to contact FollowMe.

It is able to show the user an initial page, window, etc. according the device type through which the user connects and to capture personal agent name and some characteristics of the user end device. It will open a connection and call the agent with a contact-agent (setConnection [2]) method, giving him the connection handler and the characteristics of the user end device. With these data, the agent can initiate with its user an interactive communication.

It will be sub-classed according the different devices, the user is connected, e.g. telephone number when the user is calling FollowMe, http URL when the user connects through a browser.

3.3 Interaction Diagrams

The following two diagrams demonstrate the basic control flow between the Agent and user access in the off-line and the on-line case.

3.3.1 The User is off-line

This case corresponds to the use case of section 2.3.1. Figure 4 depicts the diagram class of the involved objects

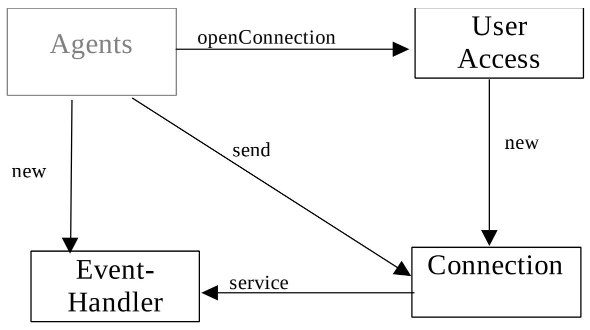


Figure 4 Involved Objects (external view)

When the agent wants to send a document to the user, it first looks up the ConnectivityDescription from the User Profile. It asks the local user access (the default user access attached to the place where the agent lives) to open a Connection according to the given connectivity description. Next, the Agent creates a new eventhandler that will manage the data requests from the Connection. It asks the connection to send a document (the deliverable object) to the end-user, providing it with the eventhandler and some QualityOfService-object[7].

The Connection uses the service method of the eventhandler to get the document, service haS two parameter, FollowMe-ServletRequest to express the input parameter, i.e. what the user/Connection from will next require the agent and FollowMe-Servlet Response where the agent write the desired data.

The transmission finisheswith the call of the function closeClonnection(), caused by agent or user or transmissionError().

The following diagram shows the interaction of the FollowMe object for the situation described.

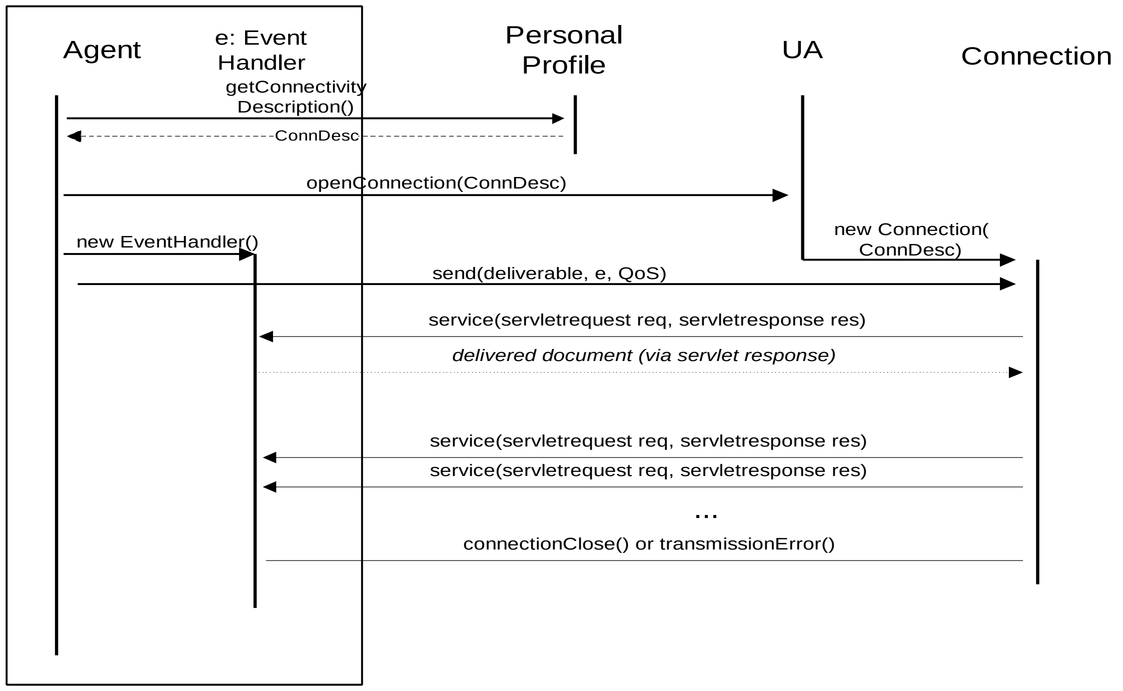

Agent
e: Event Handler
Personal Profile
UA
Connection
getConnectivity Description()
ConnDesc
openConnection(ConnDesc)
new Connection(ConnDesc)
new EventHandler()
send(deliverable, e, QoS)
service(servletrequest req, servletresponse res)
delivered document (via servlet response)
service(servletrequest req, servletresponse res)
service(servletrequest req, servletresponse res)
...
connectionClose() or transmissionError()

Figure 5 Sequence chart (external view)

Note that in the diagram servletrequest is the FollowMeServletRequest and servletresponse is the FollowMeServletResponse already described. These short names are used just for visual purposes in the diagram.

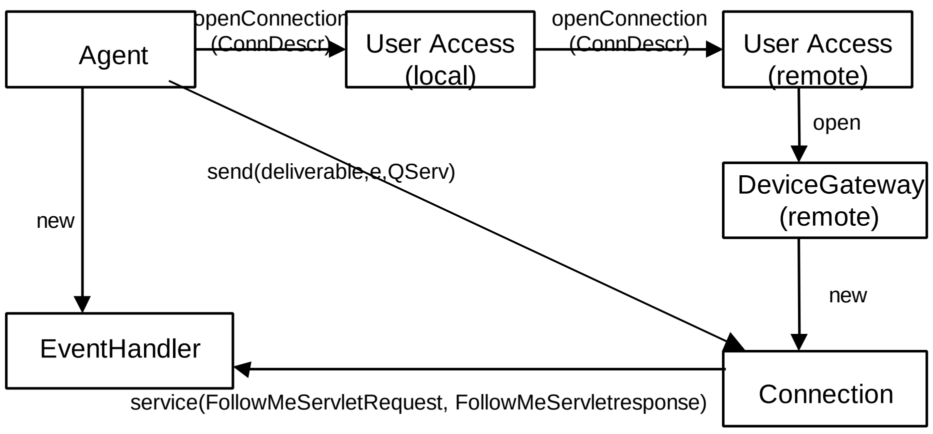
From the diagram, it can be seen that internally the local user access analyses the connectivity description and identifies either a local device gateway or asks a remote user access to identify an appropriate device gateway (see 3.1.1). The devicegateway is then asked to create a new Connection to the End user device and to transmit the document.

[7] a detailed description of the quality of service interface is still to be done.

The connectionClose can be produced by the Agent or by the Connection, when it terminates or finds an transmission error .

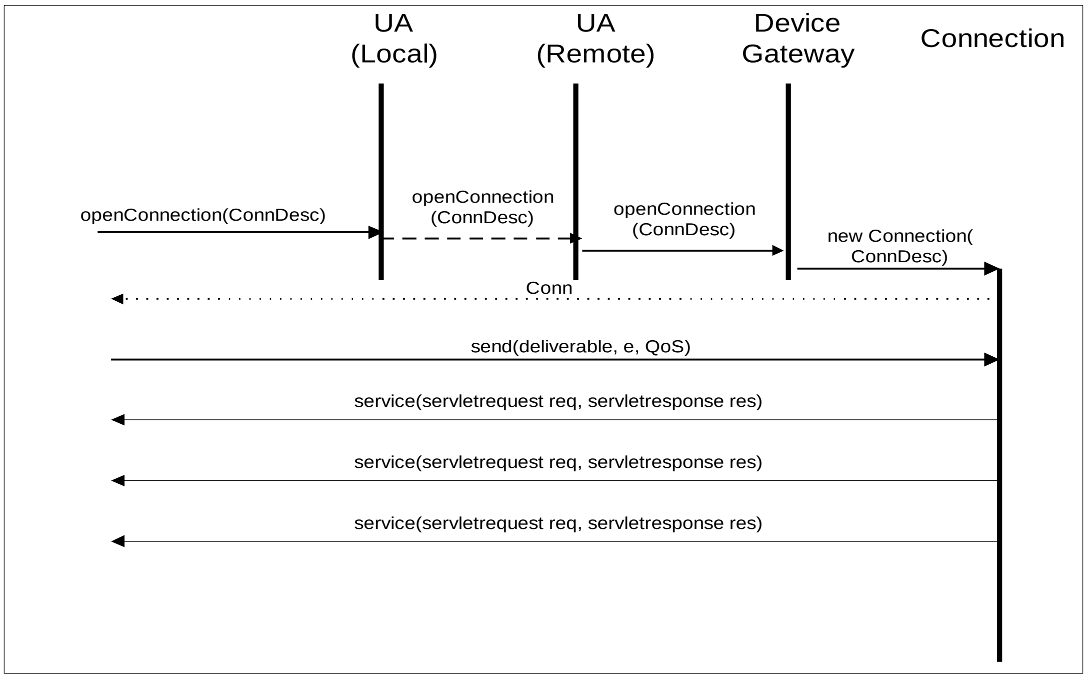
The following class diagram shows the internal functioning of the user access.

Figure 6 Involved Objects (internal view)


Agent
openConnection (ConnDescr)
User Access (local)
openConnection (ConnDescr)
User Access (remote)
open
DeviceGateway (remote)
send(deliverable,e,QServ)
new
new
EventHandler
Connection
service(FollowMeServletRequest, FollowMeServletresponse)

The next diagram shows the internal interaction within the user access.

Figure 7 Sequence Chart of internal interaction


UA (Local)
UA (Remote)
Device Gateway
Connection
openConnection(ConnDesc)
openConnection (ConnDesc)
openConnection (ConnDesc)
new Connection(ConnDesc)
Conn
send(deliverable, e, QoS)
service(servletrequest req, servletresponse res)
service(servletrequest req, servletresponse res)
service(servletrequest req, servletresponse res)

3.3.2 User is Online

This case corresponds to the use case of section 2.3.2

The picture shows the interaction between FollowMe object when the user is on-line.

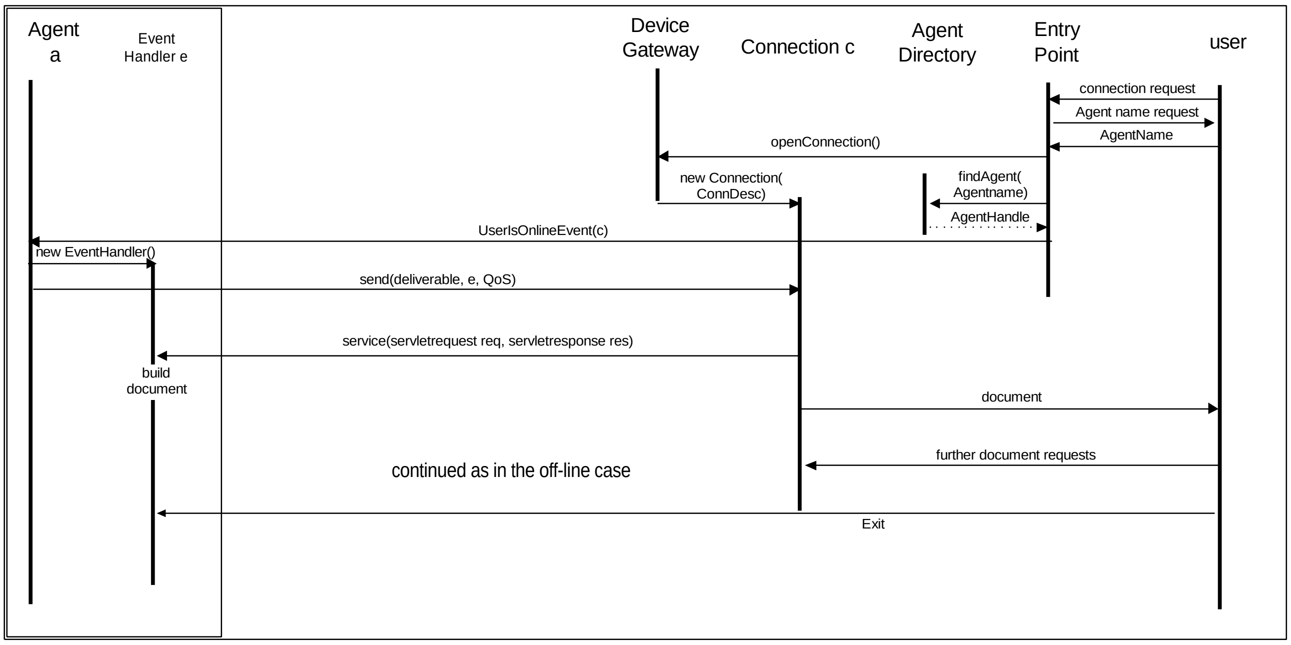


Figure 8 Sequence Chart when the user is on-line

If the user wants an on-line connection to its agent, it first has to contact an entry point (e.g. a public URL). The entry point inquires the name of the agent and looks it up in an agent directory. It also asks the associated device gateway to open a dedicated connection to the user.

After the establishing of the connection, the entry points sends a handle to this connection to the agent. The agent can then initiate an interactive communication with the end user by invoking the send method on the Connection, giving it the initial document/data/menu to transmit, the eventhandler dedicated to handle this communication and the QualityOfService object.

The communication continues as in the off-line case.

4 Examples

4.1 Usage of User Access by an Agent

- Agent who wants to send data to an output only device fax. The agent delivers a report which shows the curve of the Oracle shares organised in a small table.

```
Agent
UserProfile profile = personalAssistant.getProfile();
ConnectivityDescription connDescr=profile.getConeDesc(...);
// e.g. fax+74612041@fast.de
try
  {
  UserAccess UA=(UserAccess) thisPlace.getProperty("UA");
  EventHandler eh=new MyEvH();
  Connection fc=(Connection)UA.openConnection(ConnDescr);
  fc.send("orcl.xml",eh,null);
  }
catch (Exception e){System.out.println("Exception: " +e);e.printStackTrace();}

Agent's EventHandler //example of an agent EventHandler
public void service(FollowMeServletRequest req,FollowMeServletResponse res)
 {

PrintWriter out=new PrintWriter(res.getOutputStream());
 System.out.println("Request:"+req.getParameter("Name"));
 if(req.getParameter("Name").equals("orcl.xml"))
  {
  try
   {

   String value = stockService.getValue("ORCL");
   out.println("<?xml version = \"1.0\"?>\n");
   out.println("<?xml-stylesheet href=\"orcl.xsl\"?>");
   out.println("<Oracle>\n");
   out.println("<Stock>");
   out.println(" <Name> "+ORCL+" </Name>");
   out.println(" <Value> "+value+" </Value>");
   out.println("</Stock>\n");
   out.println("</Oracle>");
   }
  catch(...)
   {
   System.out.println("Exception:"+...);
   }
  }
 if(req.getParameter("Name").equals("orcl.xsl"))
  {
  out.println("<xsl>\n");
  out.println("<rule>");
  out.println("<root/>");
  out.println("\t<HTML>");
  out.println("\t<BODY>");
  out.println("\t<table table-width=\"60 mm\"><TR><TH>Name</TH><TH>Value</TH></TR>");
  out.println("\t<children/>");
  out.println("\t</table>");
  out.println("\t</BODY>");
  out.println("\t</HTML>");
  out.println("\t</rule>\n");

  out.println("<rule>");
  out.println("<target-element type=\"Stock\"/>");
  out.println("\t<TR>");
  out.println("\t<children/>");
  out.println("\t</TR>");
  out.println("\t</rule>\n");

  out.println("<rule>");
  out.println("<target-element type=\"Name\"/>");
  out.println("\t<TD>");
  out.println("\t<I><children/></I>");
  out.println("\t</TD>");
  out.println("\t</rule>\n");

  out.println("<rule>");
  out.println("<target-element type=\"Value\"/>");
  out.println("\t<TD>");
  out.println("\t<B><children/></B>");
  out.println("\t</TD>");
  out.println("\t</rule>\n");

  out.println("</xsl>");
  }
out.close();
 }
User Access //user access inside
public Connection OpenConnection(ConnectivityDescription ConnDesc)
{....
             DeviceGateway Dev = getDevice(ConnDesc.getDevType);
      return Dev.open(ConnDesc);

DeviceGateway
public Connection open(ConnectivityDescription ConnDescr) {
 FaxConnection fc;
 fc=new FaxConnection();
 fc.init(ConnDescr.getNumber());
 return(fc);
 }
```

4.2 XML example

The following example shows a very rudimentary XML representation of a newspaper, containing articles with title, text and pictures.

```
<?xml version="1.0" ?>
<?xml-stylesheet href="newspaper.xsl"?>

<!DOCTYPE newspaper [
      <!ELEMENT newspaper (article*)>
      <!ELEMENT article (title, body)>
      <!ELEMENT body (PCDATA | IMG )*>
      <!ELEMENT IMG (#PCDATA)>
      <!ATTLIST IMG SRC CDATA #IMPLIED>
      <!ATTLIST IMG ALT CDATA #IMPLIED>
      ]>
```

```
<newspaper>
 <article>
   <title>This is the Title of the first article</title>
  <body>
     This is some text
     <IMG SRC="http://www.fast.de/FollowMe/grafik/followMe_LogoT.gif"
        ALT="Photo cannot be displayed">
       This could be some comment on picture 1
     </IMG>
     Some more text
  </body>
 </article>
 <article>
   <title>This is the Title of the second article</title>
  <body>
     This is some text for the second article
     <IMG SRC="http://www.fast.de/FollowMe/grafik/followMe_LogoT.gif"
            ALT="Photo cannot be displayed">
       This could be some comment for picture2
     </IMG>
     Some more text for second article
  </body>
 </article>
</newspaper>
```

4.3 XSL example

The following example shows an XSL file that maps the previous newspaper article to HTML. The mapping is done in two variants:

- mode default: Show pictures
- mode textonly: Show alternative text instead of pictures

This version only works with docproc and a specialised FollowMe backend.

```
<xsl>
      <rule>
         <root/>

    <html>
     <head><title>FollowMe - Test Page</title></head>
    <body background="http://www.fast.de/FollowMe/grafik/musterL.gif"
          link="#0000ff" vlink="#ff0000" alink="#800000">
       <table border="0" width="600">
       <tr>
          <td width="227" height="114" valign="top">
            <img src="http://www.fast.de/FollowMe/grafik/followMe_LogoT.gif"/>
          </td>
          <td width="365" valign="top">
                  Example of a nice newspaper<br/>

           <children mode="textonly"/>

         </td></tr>
        </table>

   </body>
    </html>
       </rule>

<!-- Default Rules  -->

       <rule>
         <target-element type="article"/>
          <children/>
       </rule>

       <rule>
         <target-element type="title"/>
          <H1><children/></H1>
       </rule>

       <rule>
         <target-element type="body"/>
          <HR/><children/>
       </rule>

       <rule >
        <target-element type="IMG"/>
        <external-graphic src='=attributeString("SRC")' alt='There is a photo'/>
         <br/><emph><children/></emph><br/>
       </rule>

       <rule mode="textonly">
         <target-element type="article"/>
          <children/>
       </rule>

       <rule mode="textonly">
         <target-element type="title"/>
          <H1><children/></H1>
       </rule>


       <rule mode="textonly">
         <target-element type="body"/>
          <HR/><children/>
       </rule>

        <rule mode="textonly">
         <target-element type="IMG"/>
          <br/><eval>attributeString("ALT")</eval><br/>
          <br/><emph><children/></emph><br/>
        </rule>

</xsl>
```

5 References

[1] DH2, User Access Requirements, December 1997.
[2] FollowMe Work package D ”Autonomous Agents”
[3] ANSA’s FlexiNet.
[4] FollowMe Work package B ”Mobile Object Workbench documentation”.
[5] Servlet Tutorial, Sun Microsystems, http://java.sun.com/products/jdk/1.2/docs/ext/servlet/servlet_tutorial.html
[6] The Java Servlet API, http://java.sun.com/marketing/collateral/servlets.html
[7] Introduction to the New AWT Event Model, Sun MicroSystems, http://java.sun.com/docs/books/tutorial/ui/components/eventintro.html
[8] XSL: A Proposal for XSL (http://www.w3.org/TR/NOTE-XSL.html)
[9] XML: Extensible Markup Language (XML) 1.0, W3C Recommendation 10-February-1998 (http://www.w3.org/TR/1998/REC-xml-19980210)
[10] XSL Tutorial: http://www.microsoft.com/xml/xsl/tutorial/tutorial.htm