ESPRIT Project No. 25 338

Work package H

User Access

Software Report
DH6.3
(Requirements, Design, and Implementation)

ID:	DH6.3	Date:	23.2.98
Author(s):	Elcha Triep, Michael Breu, Philip Scheideler (Fast e. V.)	Status:	Version 1.3
Reviewer(s):	H.-G. Stein, M. le Nouy	Distribution:	Project & Reviewers

Change History

Document Code	Change Description	Author	Date
DH5.2 & DH5.2	First version of document. Based on DH3 for UA V1.1	MBR & ETR	7.8.98
DH6.3	Update to UA V1.2	MBR & PSC	14.10.98
	inclusion of pattern stuff, inclusion of new gateways	MBR	31.1.99

1 Introduction

1.1 Purpose of this Document

A key objective of the FollowMe project is to free the user from a fixed desk-top. A user can deploy a software personal assistant (PA) [2] on the network which co-ordinates a set of task agents to accomplish missions. Both the agents and the users are mobile, i.e. the personal assistant can roam through the network while carrying out its mission, and the user can change his geographical location, and also the type of media, through which he interacts with its PA. The goal of the user access is to enable the information exchange between users and their personal assistant through a variety of different media without loosing quality of interaction. The user must be able to find his personal assistant and to communicate with it 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, each user's personal assistant maintains a diary that contains connectivity information.

This document describes the requirements, the design and the implementation of the User Access module and the device gateways. It is intended to convey the design rationale behind the User Access and the detailed architecture.

This document should enable to write FollowMe applications that use services of the user access.

1.2 Overview

The User Access provides facilities for mobile applications to exchange information with their mobile users and vice versa. The User Access consists of a main component that provides standard interfaces for FollowMe applications to interact with users, and a set of device gateways that provide an adequate layout rendering of this information.

Main requirements are

- the support of interactive (i.e. input/output) and passive (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 and the selected end-user device.

The design is driven by a Document Delivery Pattern. I.e. an application hands over a document to the user access for delivery. The document itself encapsulates the information, but also a description how this information is rendered to a layout suitable for a certain device.

The device gateway negotiates with the document for an appropriate layout rendering. A document provides a set of external representations of its information. If these representations are not suitable for the given device, the device gateway requests a standard representation of the information in XML ([10]) together with an accompanying style sheet in XSL ([8]), and renders both to an adequate layout.

Currently device gateways for WWW-browsers, Fax, SMS, email and java-enabled devices (based on swing) are implemented.

The following chapter describe briefly the requirements that have been addressed by the user access component. It is followed by an overview to the design patterns and the software components. 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.

The details of the user access code are documented with JavaDoc and are available from http://hyperwave.fast.de /FollowMe/.

2 Requirements to the User Access Component

The User Access component has to provide facilities for interaction between the agent and the user (see Fig. 1). There are two main classes of requirements that are addressed by the user access component:

- Communication requirements: i.e. what devices have to be supported, what are the facilities to exchange information between the user and the agent[1]?
- Layout rendering requirements: i.e. what facilities are needed to present the information on the selected device

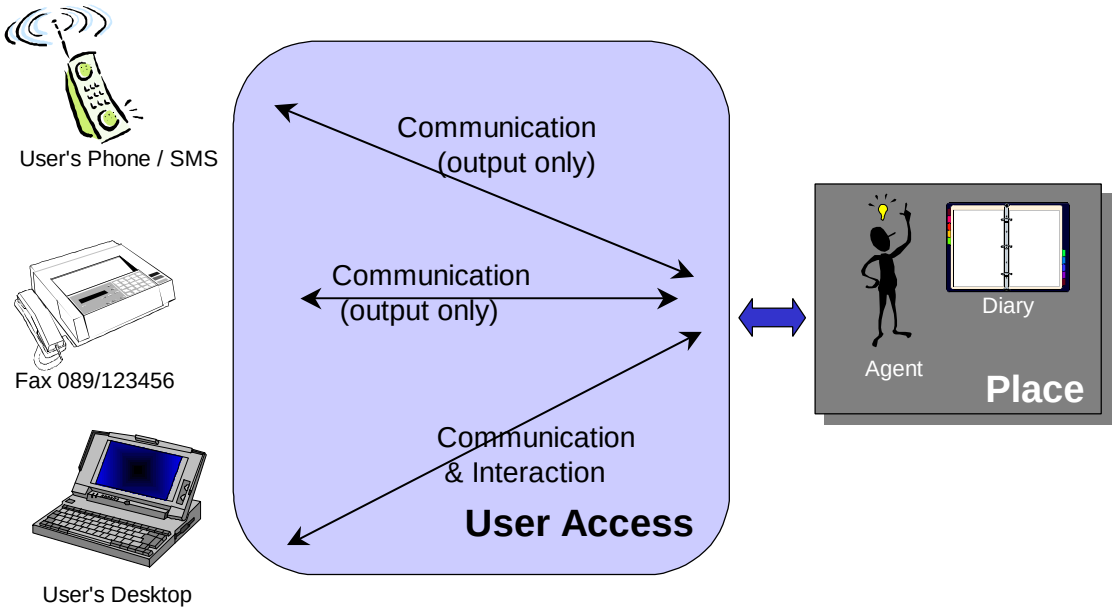

Fig. 1 *The User Access Component as a mediator between agents and devices*

2.1 Communication Requirements

Users interact with their agents through a variety of different media. A set of devices servicing these media types is supported. In the following we distinguish between passive (i.e. output only) and interactive (i.e. input/output devices). The following devices should be supported

[1] Please note that in the following the concepts of *agent*, *personal assistant*, and *application* could be used synonymously. We will mainly use the term agent, in some cases personal assistant if we want to emphasise the personalised nature of the agent. There is no difference from the usage point of view whether the user access service is used by a (mobile) agent, or any other flexinet-based application.

- Interactive Devices
  - web-browsers
  - Java enabled devices that support graphical output
- Passive Devices
  - fax-machines
  - audio output via phone
  - SMS (short message systems)
  - email

Additional device (types) should be pluggable into the system without the need to modify the existing agent's capabilities.

Initiation of a communication can either be triggered by an agent in order to contact its user, or vice versa.

When the user wants to contact its agent, he accesses it through an entry point provided by the user access component. The contact can e.g. be established from the user's browser. The browser contacts a server that acts as an entry point. An entry point provides two functionalities: It contacts an agent factory to create a new agent that gets in touch with its user, or it can lookup through a predefined agent directory an existing agent. In the latter case the entry point informs the agent to get in touch with its user.

The agent maintains connectivity information in the user's profile. When the agents wants to deliver information to its user, it has to lookup current connectivity information in the user's profile. This can be the user's business fax, or home fax, or the user's SMS gateway. The agent requests from the user access component to open a connection to this device. With this connection opened it can send to (and potentially interact with) the user.

2.2 Layout Rendering Requirements

The agent may wish to send the same information to the user through different devices and media types. Thus information can have different external representations, depending on the type of device. For example if an agent wants to deliver the information that a share's stock value has exceeded some set limit, this can be rendered in various ways to the user:

- On a web browser as a web-page with sophistical graphical layout elements
- As an email that containing a text note
- As a voice message via phone: "*Some shares in your portfolio have exceed a limit, please contact your agent*".

The knowledge of the agent about the specific characteristics of a device should be as small as possible. Thus the User Access has to provide a unique interface for different device/media types, and to bundle devices with similar layout capabilities into a device type.

To achieve this, information and layout must be separated: The information transferred is the same for all devices. The layout maps this information to an appropriate output format (see Fig. 2).

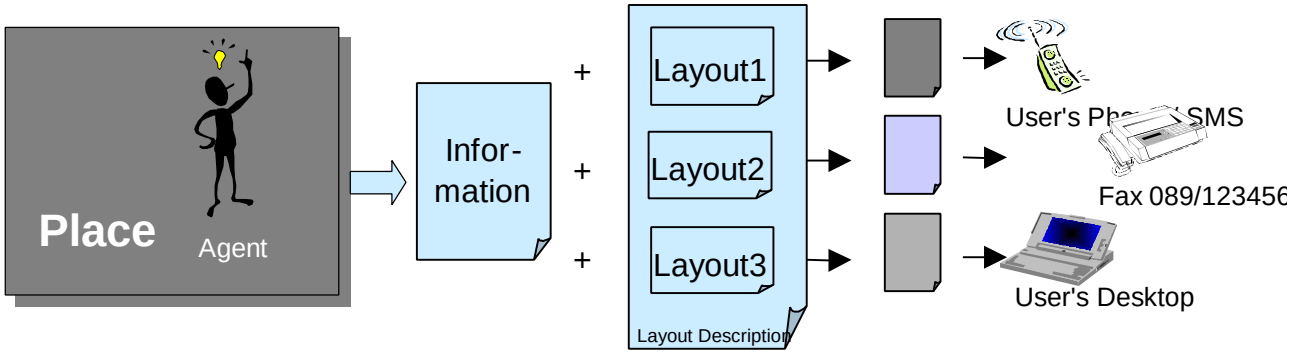

Fig. 2 *Separation of information and layout description*

The rendering into an appropriate external representation and the transmission is handled by a *device gateway*. A device gateway provides connections to a set of similar devices (e.g. a SMS device gateway, a fax device gateway, or an http device gateway). Each of these device gateways provides a certain set of capabilities.

The user access must provide mechanisms to adapt the layout description to different constraints as e.g. size of an output screen or black & white vs. colour output. The rendering of a layout can be quite resource consuming. Thus the transmission process must be separated into its two basic functionalities (see Fig. 3):

- rendering a layout by *converting* it into an appropriate representation suitable for that device, and
- *transmitting* the external representation to the end-user device.

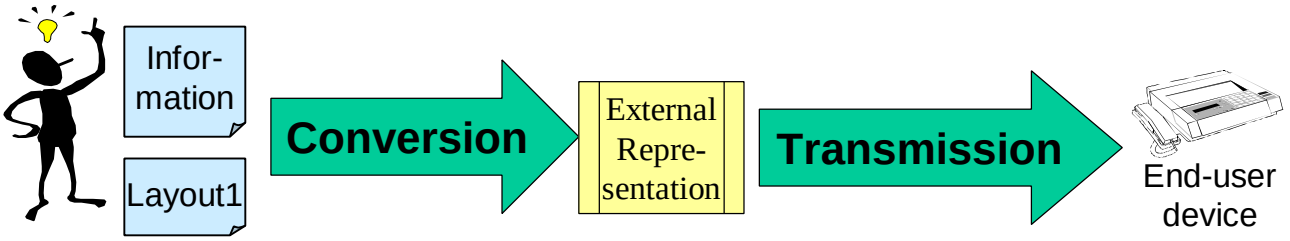

Fig. 3 *Conversion and transmission of information*

2.3 Typical Usage Scenarios

The following sections give typical usage scenarios of the user access components.

2.3.1 The Agent Wants To Deliver Information to an Off-Line User

The agent wants to send information to the user that is currently off-line. It looks up the diary in the personal profile to discover through which (potentially passive) device (fax, phone, SMS, etc.) the user can currently be contacted. The agent prepares a document (or a set of documents).

Then, it contacts an available UA component to open a connection to the end user's device and asks it to send the documents. The agent may receive an error message, alerting of some errors in the transmission or a transmission close message, which indicates a successful transmission of data.

2.3.2 The User Contacts his Personal Assistant through an Entry Point

The user contacts his personal assistant through an entry point to a FollowMe-based system. The entry point can e.g. be an URL of a http-server that provides an entry point service. The entry point is attached to an input/output device

gateway. An entry point typically offers the user two options: to search for his personal assistant or to ask an agent factory for the creation of a new personal assistant.

If the user chooses the option ”search for the personal assistant”, the UA looks up the agent via its name in an agent directory[2]. The agent gets informed that his user is on-line and gets the connection details (device type, format to be used, etc.). The agent reacts by sending the relevant information that will be processed via the connection and finally send it to the user. The UA captures the responses from the end-user’s device and delivers it to the agent component to handle it.

If the User chooses the option ”create a new personal assistant”, the UA contacts the personal assistant factory and asks it for the creation of a new personal assistant for an on-line user. The factory creates a personal assistant and can advise it to 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 the user access and transmitted to the agent. Thus, the agent may build appropriate deliverables for his on-line user.

[2] 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.

3 Design of the User Access

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

3.1 Architecture of User Access

3.1.1 Overview

A host, participating in a FollowMe application, can offer a set of local *devices* that support communication with the end user. Examples of such devices are fax cards, voice modems, 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 will vary depending of the device type and the specification of the agent.

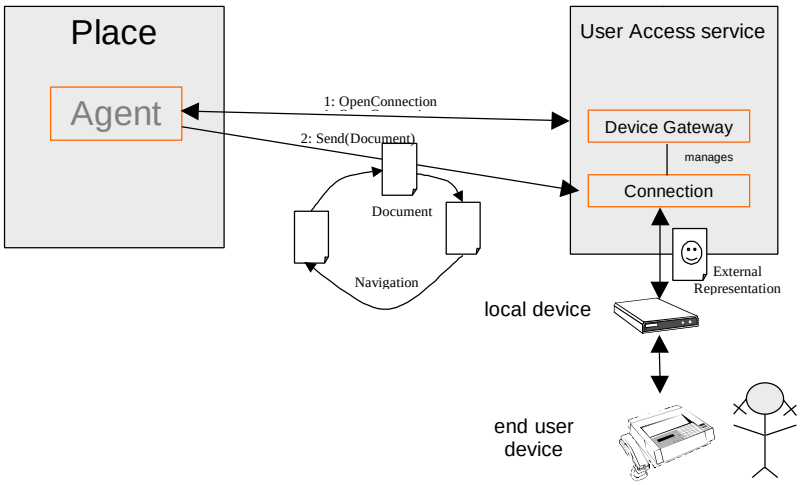

Fig. 4 *Basic communication schema between agent and end user*

Regarding this model, the main task of the UA component is to manage connections between agents and end-user devices. This includes the opening of a suitable connection, the transfer of documents, the support of navigation inside documents and the closing down of a connection. Both agents and users are able to close the connection and interrupt the communication between them at any point in time.

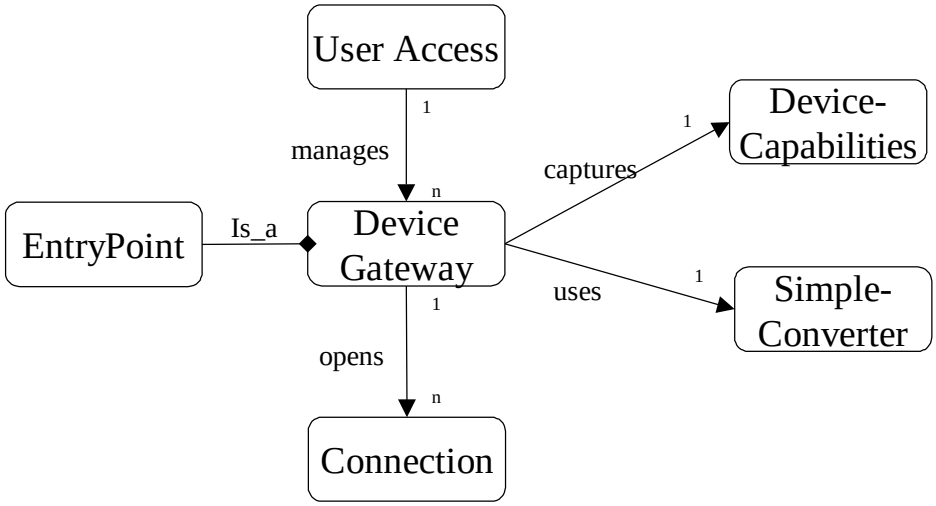

Fig. 5 *Class Diagram of the main User Access objects*

The *User Access* is a service that manages the local devices (fax card, modem, email-service, etc,) available at a host. Each device has a *Device gateway*[3] associated. The user access service can also relay to device gateways maintained by other user access services.

Device Gateways provide facilities to open connections between the local device and the end-user devices. They hold a description of the capabilities of the device, and provide a converter from the XML format to the external representation required by the device. An Entry Point is a special device gateway installed at known locations that allow the user to request a connection to an agent.

A *connection* is a (temporary) channel to transport a set of (potentially complex) documents to the end-user's device. The transport can include conversion into appropriate external representations and transmission of referred documents (e.g. images). Also connections handle navigation through references to other documents and user responses (see 3.1.2).

There are two ways for an agent to obtain a connection. Either the agent contacts a UA service and asks it for a connection (e.g. via a fax device) to use. Or it gets a contactUser message from an entry point that contains the connection as an argument.

The connection properties, i.e. capabilities of the device, or the number of a telephone, are specified by the connectivity description that is found in the personal profile or constructed by the agent. 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 which is completely transparent for the Agent. The agent can then send documents through the connection object.

The information is encapsulated in documents. Information can have different external representations. A standard external representation supported by all device gateways is XML (see 3.1.2). Together with an associated layout description in XSL these representations can be translated into appropriate external representations supported by that device, as e.g. HTML, plain text.

A hook to support the Quality of Service (QoS) is provided in order to adapt the use of resources and network traffic to the performance of the system, the network, and the capabilities of the device. Via the QoS the layout of the rendered documents can be influenced. Thus the user may e.g. receive black and white data instead of coloured images depending on the agent/applications preferences, i.e. the agent can 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.

3.1.2 Document Model

Documents have to support three types of functionalities:

- Documents represent information sent from an agent to the end-user: The information can have quite a complex structure due to its multi-medial nature. It can contain text, pictures, audio, etc. Thus documents may refer to other

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

documents, containing individual pieces of the information. Each document can be rendered in different external representations which are described as MIME-types (see Fig. 6).

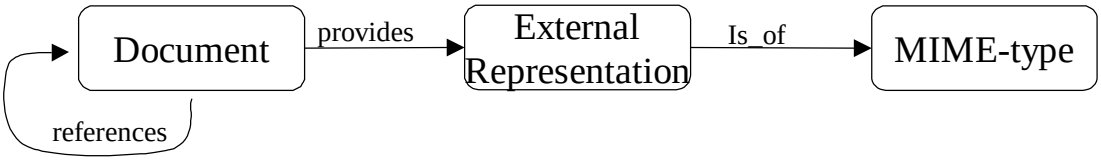

Fig. 6 *Class Diagram for Documents*

- Documents give access to layout descriptions. Documents provide e.g. XSL documents describing the layout of the information. The user access uses this description to render the information into the appropriate layout.
- Documents handle user feed-back, i.e. they manage the reaction of a user on this document. A user can trigger feedback by two types of actions: Following links to referenced documents, and by returning a set of input parameters requested (e.g. in an HTML-form).

A document gives access to different external representations (see Fig. 7). These representations can be already precompiled, or constructed on the fly. Device Gateways provide converters[4] that convert from an external representations to another external representation, suitable for this device gateway. If a document is not able to provide the appropriate external representation for a device, the XML representation is used to derive the appropriate external representation. These external representations are described by mime types (e.g. text/xml, text/html, image/gif). The relevant method is

```
public ExternalRepresentation getExternalRepresentation(MimeType mimetype)
```

A document can refer to other documents by an (internal) name. I.e. a document can be queried to give access to another document via the method

```
public Document getReferredDocument(String name, Properties params)
```

The name can refer to the result of the filling of a form by the user. In this case params is a list of extra parameters values given by the user. Thus a document can provide its own handling facilities for user feedback.

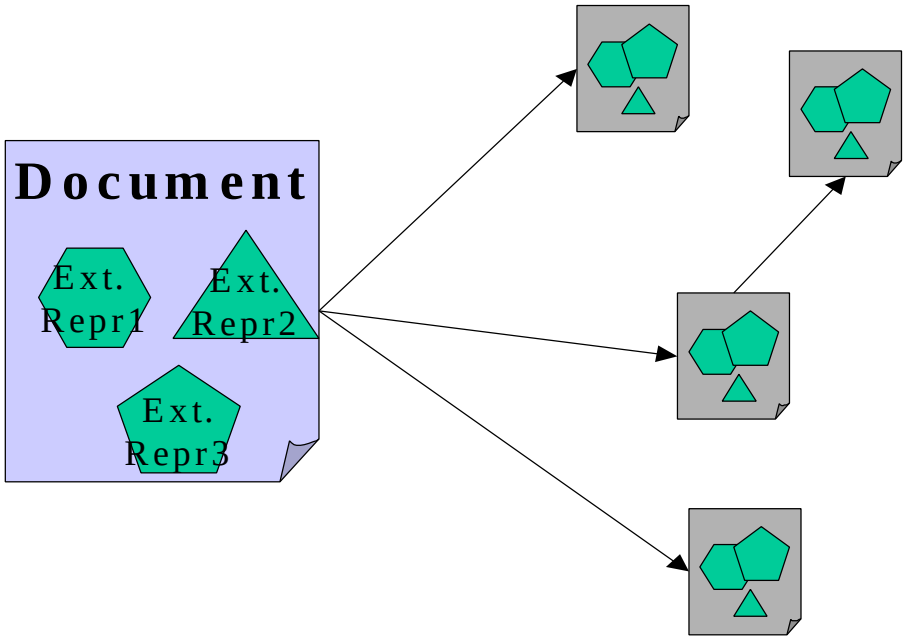

Fig. 7 *Structure of Documents*

Document is an interface, the implementation behind this interface is application depend. Thus documents can build quite a complex structure, e.g. contain subdocuments, refer to other documents.

[4] Converters implement the interface SimpleConverter. In future releases more complex converters may be defined.

Fig. 8 depicts the example of an issue of a newpaper. This issue is described as a document, which has two external representations in XML and HTML. These representations refer to articles as subdocuments[5] which in turn refer to documents that contain photos. These photos are represented in GIF-format.

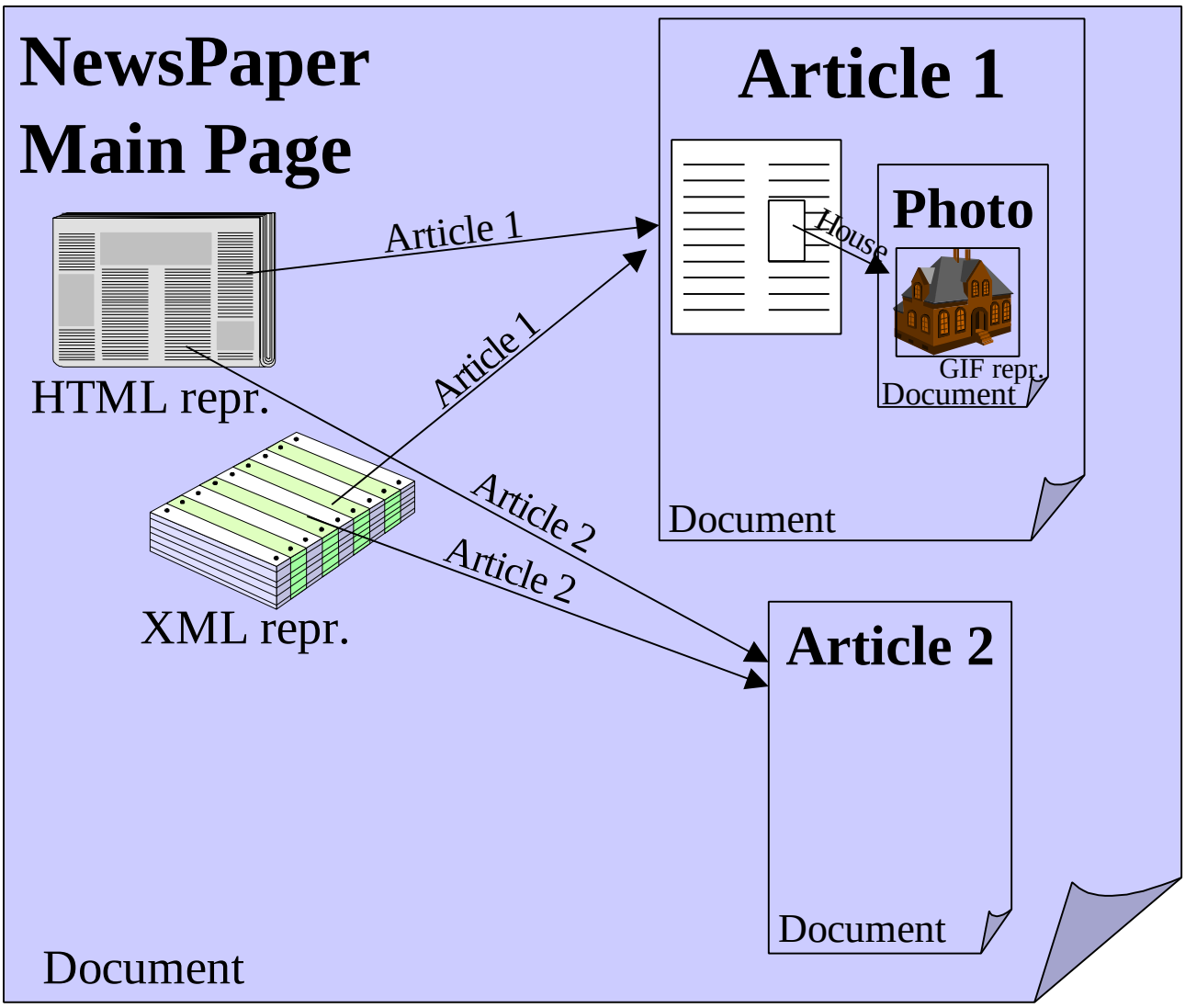

Fig. 8 *Example of a network of documents*

3.1.3 Interaction between Connection and Agent

The interaction between agents and connections falls into three phases:

- Obtaining a connection through an open request to the user access or a device gateway , or through the “contactUser”-message emitted by an entry point.
- Sending of documents, navigation through documents
- Closing a connection

There are two ways to obtain a connection:

- The agent actively requests from a user access service the opening of a connection with the `openConnection(ConnectivityDescription ConDes)` method. Argument to this method is the ConnectivityDescription which describes how the user can be reached, i.e. the device type to be used, and the number. (Fig. 9)

[5] Whether these documents are subdocuments or separated documents is just a conceptual view. From the user access point of view they can be independent documents.

- The user contacts an EntryPoint. The entry points are located at well known hosts for each device type through which the user can contact FollowMe, e.g. an http-address if the contact is through a browser. This EntryPoint interface is in charge of opening a Connection and it calls the method `public contactUser(Connection C)` in the interface of the agent. contactUser carries as a parameter the connection via which the user is online. The agent can then use this connection to interact with the user. (Fig. 10)

The agent uses the `send` method of the connection to ask for the transmission of a document. The transmission can involve the access to referenced documents. However the agent is not directly involved in the transmission of these documents, because the documents themselves handle the access. The effect of multiple sends is device dependent. On screen oriented devices (e.g. http-devices, awt-devices) it will in open another window for each document. On text oriented devices (e.g. mail, fax) it will result in a concatenation of the documents.

The sending involves both the transformation into the correct layout and the physical transmission to the user's end-device. These two steps can be taken appart by using a SimpleConverter to render the external representation and using the method `public abstract void sendraw(ExternalRepresentation er)` to physically transmit the external representation.

A connection can be closed by the agent with the close-method. Whether this immediately closes the connection or whether the physical transmission of documents is still ongoing, is device dependent.

In order to intercept the closing of a connection by the user, or by enforcement of the user access, an agent can install a `ConnectionListener`. This listener can receive two types of events: Failure of transmission and closing of the connection.

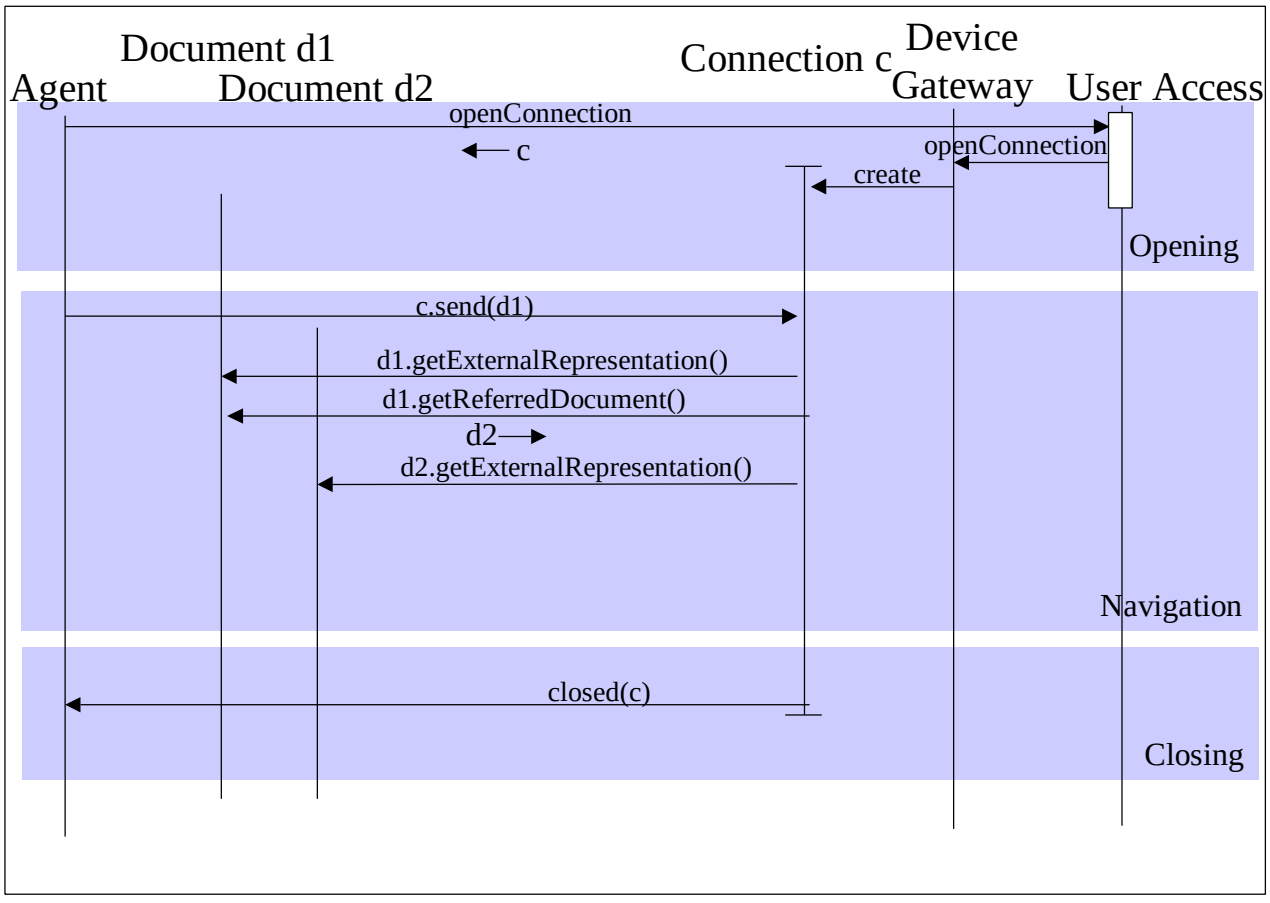

Fig. 9 *Interaction diagram: agent actively opens a connection*

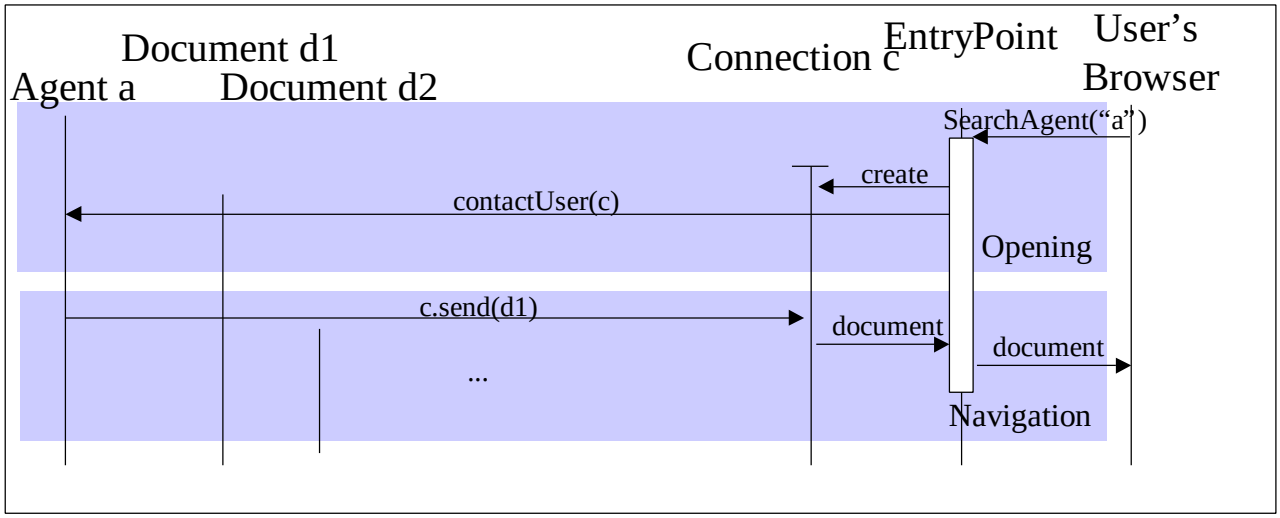

Fig. 10 *Interaction diagram: A user requests an Entry Point to search and to contact agent "a"*

3.1.4 Generic Mark Up Language XML/XSL

The main motivation to use a mark up language to represent data and layout was born from the requirement 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 uniform 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 ("extended style language") emerged as the complement of XML to be used to describe in a so-called style sheet the layout of the XML data.

Both XML and XSL are W3C recommendations or W3C working draft. The language definition is not reproduced here, but can be found at [8], [9] and [10].

Use of XML/XSL by agents/applications

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

The XML document can hold arbitrarily structured (textual) information (see [10] or e.g. the example in section 5.2).

The layout of the information is defined in an XSL style sheet (see [8] or [11]). XSL supports different modes. These modes are basically used to discriminate between different layout descriptions:

" "(default):	for devices which can understand full HTML, e.g. Browsers
"text":	for devices which can understand only text, e.g. some fax devices, e-mail
"staticHtml"	for devices that support rendering of HTML-structured text and graphics like a fax
"shortText" :	for SMS and similar devices with limited text capabilities
"audio" :	for devices which can only work with sound, like a standard phone

Via the mode-method of the QualityOfService-Interface the agent can provide an alternative mode in which the layout should be rendered. This could e.g. allow for the replacement of coloured images by textual descriptions, if the network bandwidth is low.

The connection transparently applies converters that combine the XML document and the XSL description 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 style sheet can already be tailor-made for this device type. Otherwise the XSL style sheet defines a set of different layouts for all devices to which a certain piece of data may be 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. If no tag is given, the name "index.xsl" is assumed. After parsing the XML-file the XSL-file is requested from the document (via the referredDocument-method), and both applied to convert to the appropriate layout.

3.1.5 Integration of new supported devices types

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.

New device gateways that can be assigned one of the layout descriptions above (3.1.4) , can be integrated into a FollowMe application without any reprogramming of the agents. If however devices with new descriptions are added, then layout files have to be extended in order to take advantage of these new device types.

3.2 Design Patterns

The design pattern of *Document Delivery* governs the architecture of the user access. A specialisation is the *XML document delivery*. These patterns are described in this chapter following the structure proposed by Gamma et al. [13].

3.2.1 Document Delivery

Pattern Name: Document Delivery

Context: Multi-medial presentation of complex information

Problem: Information needs to be delivered to users making use of a variety of devices. The details of which kind of devices are available to a user change with the time and the user's location, and indeed the user's preferences for how information is delivered may change for arbitrary reasons. The information must be rendered on these devices in an adequate quality.

Solution: A uniform model is adopted in which information can be encapsulated in an inter-linked set of documents. Each document represents a certain piece of the information. A document supports methods to render this piece of information in several external representations (graphic, text, audio, ...), depending on the capabilities of the device.

The external representation may refer to other documents that maintain other linked pieces of information.

Devices are encapsulated by device gateways. Device Gateways encapsulate the knowledge of the capabilities of the devices and provide connections to the devices through which documents are rendered to the users. According to the available capabilities the most appropriate external representation of a document is selected. Interlinked information are rendered by querying the referred documents.

This model allows for an information centred design. The representation of this information can be provided either at compile time, or at delivery time. The encapsulation of a variety of device types into device gateways allow a uniform interface for rendering of data.

Cross reference: Document Delivery is an instance of the pattern Self Interpreting Data in [14]. A document is an object that understands how to render itself in different contexts. Particular instances of the pattern are *XML based Document Delivery*.

The Components

The component architecture assumes that a client object wishes to deliver something to a user. The client contacts the User Access component for the user's end device and requests a connection satisfying some criteria (e.g. suitable for

delivering at a particular time of day and a particular volume of information) (see Fig. 4). The User Access creates a Connection to be used by the client to communicate with the user and returns a reference to the Connection together with a description of the Connection which may impose limitations on messages (e.g. message size). The client constructs a set of documents suitable for the Connection (it is assumed that it is the client that knows how best to construct, edit, précis etc. the information to be delivered). The Connection is an abstraction which hides the details of the use of real devices including any possible multiplexing, queuing etc.

3.2.1.1 Variations:

- Documents may be stubs. By asking them to provide the appropriate external representation the virtual device causes the deliverable to collect the actual data from a remote file store.
- Documents may be created dynamically, depending on the feedback from the user. When this method is called by a virtual device the deliverable may act as an applet and an interactive session is started with the user using the local device for computation.

3.2.2 XML-based Document Delivery

Pattern Name: XML-Based Document Delivery

Context: Multi-medial presentation of structured information

Problem: Different devices may support different layout mechanisms to represent textual information, e.g. either as HTML for a web browser, as plain text in an email, as short message in SMS, RTF in MS Word. The same information must either be stored and exchanged in different layouts or it is stored and exchanged in one unique format and from this format the different layouts are derived.

Solution: The structured information is stored in an document, which supports the representation as XML-text. The XML-text holds a reference to an XSL document. This XSL document contains a set of layout definitions (so-called modes). Each layout definition contains a set of rules, that describe how the structured information is represented in the appropriate format.

The Components

The client sends a document to the connection. The connection requests an XML representation of the document and retrieves also the XSL layout definition as a referred document. The XML and XSL documents are given to a converter that produces the final layout in the appropriate mode.

3.3 Overview to Classes and Interfaces of the User Access Component

The relationship between the classes can be found in Fig. 5 on page 8. A detailed javadoc documentation of all classes is included in the source code.

3.3.1 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 top of FlexiNet services. From the agent point of view, it acts as a factory of Connections which will take place between agents and user. An User Access component typically registers at a know FlexiNet trader with "UA@nnn.nnn.nnn.nnn" where nnn.nnn.nnn.nnn is the IP-number of this host.

The main method of the UserAccess is

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

- ConDes – a connectivity description of how the user may be reached (device type + number (IP address, phone number, etc.) + hostname of user-access component). It encapsulated also the required capabilities (audio, text/html, display size, etc.) of the device to be used in the ConnectivityProfile object.

3.3.2 Class ConnectivityDescription

The class ConnectivityDescription represents a description of how the user's end device can be reached. Typically this is a device type specification together with an identification of the device, e.g. a telephone number. It will be used internally by User Access and provided as parameter when sending data through the Connection by the agent.

It consist of three components :

- Device type (e.g. fax, phone)
- Number (e.g. telephone number, fax number, sms number)
- DeviceCapabilities

The ConnectivityDescription provides access to an object called DeviceCapabilities.

3.3.3 Interface DeviceGateway

A DeviceGateway is an object that is able to establish a virtual connection to the user's end device. Through this connection information can be transmitted in an appropriate representatcion. A Gateway may be more than a simple device driver since it can also do format adaptations, etc. Examples for DeviceGateways are FaxModems, VoiceModems, Interfaces to a WebBrowser, Applets, etc. The DeviceGateway is created by the UserAccess and its main purpose is to establish a connection to the end-user.

Main method are:

```
public Connection open(ConnectivityDescription ConnDescr);
```

Opens a logical connection to the end-user. If the device is output-only, the output may be spooled, i.e. not transmitted immediately, but with some delay. ConDescr indicates how can the user be reached (device + Number (IP address, phone number, etc.)

```
public void destroy();
```

Destroys the DeviceGateway. Open connections will not be destroyed, however no new connections can be established.

```
public String getType();
```

Returns the Type of the device (fax, sms, email, ...)

```
public DeviceCapability getCapability();
```

Returns the supported Capabilities

```
public DE.fast.followme.useraccess.SimpleConverter getSimpleConverter();
```

Returns a simple converter that converts a document to an appropriate external representation for this device.

3.3.4 Interface Connection

The connection represents a temporary "channel" to the end user's device, e.g. a fax machine, phone, Browser, etc. through which information can be exchanged.

A connection is requested via the UserAccess component and opened by a DeviceGateway that is identified by the Connectivity Description. The output is initiated via the send method.

The feedback of the user is reported via the getreferredDocument method to the rendered document. Abnormal Events like closing or failure can be received by attaching an ConnectionListener.

Major methods:

```
public abstract void send(Document d) throws DeviceError
```

Advises the Connection to send the document to the end-user. This could mean that the user navigates through other documents (e.g. images) referred to by this document.

```
public abstract void sendraw(ExternalRepresentation er) throws DeviceError
```

Used to send an external representation through the connection. It is solely in the responsibility of the caller that the external representation is in a form that can be transmitted to the recipient.

```
public abstract void close()
```

Used to close a connection.

```
public abstract void addConnectionListener(ConnectionListener qs)
```

Adds a ConnectionListener to get call-back on certain connection events (close, failure).

```
public abstract void removeConnectionListener(ConnectionListener qs)
```

Removes a ConnectionListener

```
public abstract void setQualityOfService(QualityOfService qs)
```

Sets the quality of service

3.3.5 Interface Document

A document is a piece of information that can render itself into a set of external representations. It manages references to other documents. Documents can contain text-like information, images, video, etc. A device gateway first checks whether the document supports its favourite representation (e.g. HTML for Web-browsers). If not it looks up the XML represnetation fo the information.

Major methods:

```
public abstract ExternalRepresentation getExternalRepresentation(MimeType mimetype)
throws MimeTypeNotSupported
```

Retrieves the external representation of a document

```
public abstract MimeType[] getsupportedMimeTypes()
```

the MimeTypes that this document supports. e.g. {"text/html", "image/gif"}.

```
public abstract Document getReferredDocument(String name, Properties params)
```

returns the document on which the string name refers to. The result can be e.g. an image. The returned document can be null. The semantics of this is device-dependent. E.g. a window presenting this document is closed.

3.3.6 Interface ExternalRepresentation

Representation of a document that is exchanged between the application and the connection.

Major methods:

```
public abstract InputStream getInputStream() throws IOException
```

Retrieves the external representation of a document as an (arbitrary long) byte stream.

```
public abstract MimeType getMimeType()
```

returns the MimeType of the document, e.g. text/html or image/gif .

3.3.7 Interface SimpleConverter

Simple converter that constructs a new external representation (on basis of existing representations).

Major methods:

```
public abstract ExternalRepresentation convert(Document d)
```

Constructs an external Representation of a document

```
public abstract MimeType[] requires()
```

returns the set of MimeTypes-representations to construct new external representations e.g. xml/text.

```
public abstract MimeType constructedType()
```

the mime type of the constructed representation

3.3.8 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.

4 Device Gateways

This chapter lists the device gateways that have been implemented by the user access components.

The main class of the user access is DE.fast.followme.useraccess.UserAccessStart. The main-method takes as an argument the name of a property-file that defines what devices are attached to this user access and the properties of theses devices. Each device is defined by a line "`Device<n>=<devicetype>`", where `n` is a running number, and `devicetype` is a string defining the type of the device. The property-file must also define the name of the class file with `<devicetype>.class = <classname>`.

Currently implemented Device Gateways are

- Http-Entry Point for WWW-Interfaces like Web-Browsers
- Fax Gateway for Fax-Delivery
- Email Gateway for text mail messages
- SMS Gateway for delivery of short messages to a GSM based cellular phone
- Swing Device Gateway for local document delivery
- Offline HTML Device Gateway for producing an offline readable set of HTML-documents

4.1 Http-Entry Point

The Http-Entry Point employs servlet mechanisms [6] to define a web-based entry-point for FollowMe. It allows access to a FollowMe application via Web-browsers. Since the entry point to the system is application dependent the two abstract methods

```
abstract protected void createNewAgent(HttpServletRequest req, HttpServletResponse res)

abstract protected void SearchAgent(HttpServletRequest req, HttpServletResponse res)
```

of the Class DE.fast.followme.useraccess.device.httpdevice.HttpEntryPoint must be instantiated, depending on the application. E.g. CreateNewAgent may contact an agent factory to create a new (application specific) agent, SearchAgent looks up an agent in a application specific directory.

Implementing Class	abstract class DE.fast.followme.useraccess.device.httpdevice.HttpEntryPoint
used layout description	"" (default), interactive device

Start-up properties	`#'----------------------------------------------------------------`
	`#' Configuration of an http device`
	`#'----------------------------------------------------------------`

	Device2=http #' #' class name of device gateway #' http.class=DE.fast.followme.useraccess.device.httpdevice.BavariaOnlineHttpEntryPoint #' #' http.port: port to serve #' http.port=8888 #' #' http.documentDir: directory where to find the html-documents #' http.documentDir=c:\\Daten\\Michael\\followme\\testenv\\httpGWdocuments #' #' http.welcomeFileName: name of welcomeFile in documentDir #' first page to be presented to the user #' http.welcomeFileName=\\Welcome.html #' #' http.errorFileName: name of error file in documentDir #' page to be presented to the user in case of internal errors #' http.errorFileName=\\Error.html #' #' http.waitFileName: name of wait information page in documentDir #' page to be presented to the user if user access is #' waiting for answer from agent #' http.waitFileName=\\Wait.html #' #' http.controlFileName: name of control information page in documentDir #' page to steer the opening of additional browser windows #' http.controlFileName=\\control.html #' #' http.closeFileName: name of page to send, to close window # page to be presented a document window is closed. # Mainly for debugging purposes, the window does not be closed at once #' http.closeFileName=\\close.html #' #' http.servlet.followme.code: class of servlet to load #' The servlet forwards all http-requests to the connection #' http.servlet.followme.code=DE.fast.followme.useraccess.device.httpdevice.HttpEntryPointServl et #' # #' http.servlet.followme.initArgs: Initialisation arguments for servlet #' not used now #' http.servlet.followme.initArgs=\ #' #' The http-EntryPoint can be used in two modes: #' if pollingSupport = true, a control window in netscape is opened to poll #' the UA every 10 seconds for commands, as opening #' a new window #' if pollingSupport = false, only interactions will happen when the user is #' loading a new page. http.pollingSupport = false #' #' The http-Entrypoint provides localized error messages (provided, the #' localized ListResourceBoundle was implemented for this language) #' Language http.localeLanguage=de #' Country http.localeCountry=DE

4.2 Fax Device Gateway

The Fax Device Gateway is capable to send faxes to a fax device. It uses the “staticHtml” layout mode to render the documents. Unfortunately it is currently not possible for a java process to print faxes without user-intervention. Therefore the current implementation uses a Netscape-Browser to print to a Windows-NT based fax device.

Implementing Class	DE.fast.followme.useraccess.device.faxdevice.FaxGateway2
used layout description	“ staticHtml”, passive device
Start-up properties	#'-- #' Configuration of a fax device #'-- Device3=fax

	#' #' class name of device gateway #' fax.class=DE.fast.followme.useraccess.device.faxdevice.FaxGateway2 #' #' This fax gateway looks up an http-gateway to render the relevant document #' to a web browser, which prints it onto a fax printer. #' #' fax.command: How to start Netscape with the printing option? #' fax.command=C:\\Programme\\Netscape\\Communicator\\Program\\netscape.exe -P\"followme\"; #' #' fax.faxprog: How to start the fax program #' fax.faxprog=C:\\WINNT\\FAXmaker\\SENDFAX.EXE #' #' fax.httpConnectionHost: on which host resides the user access maintaining #' the http gateway? #' fax.httpConnectionHost=194.59.176.168

4.3 Email Device Gateway

The Email Device Gateway sends an email to an email address. The email is rendered in “text”-mode. It uses the SMTP protocol to forward the mail to a mailhost for further processing.

Implementing Class	DE.fast.followme.useraccess.device.maildevice.MailGateway
used layout description	“text”, passive device
Start-up properties	#'-- #' Configuration of an email device #'-- Device1=mail #' #' class name of device gateway #' mail.class=DE.fast.followme.useraccess.device.maildevice.MailGateway #' #' mail.sender: Who should be the sender of the mail? #' mail.sender=FollowMe UserAccess FAST Munich <UA@followme.fast.de> #' #' mail.mailhost: Mail host that supports the SMTP-protocol #' mail.mailhost=fast.fast.de #'

4.4 SMS Device Gateway

The SMS Device Gateway sends an SMS message to a telephone number. The documents are rendered in “shortText”-mode. The access to the different SMS supporting networks is provider dependent. For each provider the abstract class DE.fast.followme.useraccess.device.smsdevice.SMSBasicConnection has to be extended for the method

```
protected abstract void sendSMS(String telnumber, String message);
```

This method defines how to send a message to a specific network.

At start-up time a specific property file (sms.propertyfile) is loaded. The SMS property file contains a mapping between prefix codes and concrete Connection classes that implement the sending of a message. Also the sms property file contains further initialisation parameters.

Implementing Class	DE.fast.followme.useraccess.device.smsdevice.SMSGateway
used layout description	“shortText”, passive device
Start-up properties	#'-- #' Configuration of an SMS device #'-- Device4=sms #' #' class name of device gateway #'

	sms.class=DE.fast.followme.useraccess.device.smsdevice.SMSGateway #' #' sms.propertyfile Configuration file for the SMS gateway #' sms.propertyfile=W:\\followme\\fast\\TestStarts\\smsgateway.properties

4.5 Swing Device Gateway

The Swing Device Gateway uses swing-classes ([12]) to render documents directly in a window of a java virtual machine. It is based on the HTML-EditorKit of swing. Currently this editor-kit supports HTML3.2.

Implementing Class	DE.fast.followme.useraccess.device.swingdevice.SwingEntryPoint
used layout description	" " (default), interactive device
Start-up properties	#'--- #' Configuration of an swing device #'--- Device5=swing #' #' class name of device gateway #' swing.class=DE.fast.followme.useraccess.device.swingdevice.SwingEntryPoint #' #' if displayEntryPointWindow = true then a entry point window is opened #' at start up #' swing.displayEntryPointWindow=true #'

4.6 Offline HTML Device Gateway

This gateway saves a set of document as HTML-documents to the local file system, in order to make it readable offline. A user can e.g. use an agent to download the latest newspaper onto its notebook in order to read it offline.

The device gateway takes the initial document, and computes the transitive hull of all documents referred to. It then converts these documents to HTML and saves it to a given file location.

Implementing Class	DE.fast.followme.useraccess.device.offlinehtmldevice.OfflineGateway
used layout description	" " (default), passive device
Start-up properties	#'--- #' Configuration of an offline html device #'--- Device6=offline #' #' class name of device gateway #' offline.class=DE.fast.followme.useraccess.device.offlinehtmldevice.OfflineGateway #' #' define a list of those HTML tags that should be replaced by local references #' offlinehtml.tag.a=href offlinehtml.tag.img=src offlinehtml.tag.body=background offlinehtml.tag.embed=src offlinehtml.tag.bgsound=src offlinehtml.tag.table=background offlinehtml.tag.tr=background offlinehtml.tag.td=background offlinehtml.tag.th=background offlinehtml.tag.head=profile offlinehtml.tag.link=href offlinehtml.tag.blockquote=cite offlinehtml.tag.area=href offlinehtml.tag.input=src offlinehtml.tag.frame=src

	offlinehtml.tag.iframe=cite offlinehtml.tag.object=data offlinehtml.tag.layer=src offlinehtml.tag.script=src

5 Examples

5.1 Example Use of the User Access by an Agent

The following is a simple Agent that, when contacted by the user retrieves the current value of the Oracle Shares and sends it to the user. For details see the source file of DE.fast.followme.tests.Orcl in the source code distribution.

This example creates the respective documents by loading the respective XML and XSL templates from the file space and forwards them to the user-access

```
public class Orcl extends MobileObject implements OrclIface
{

	public Orcl ()
	{
		// internal creation stuff
	}

// ...

	/**
	*  rudimentary implementation of contactUser
	*/
	public void contactUser(Connection c)
	{
		try
		{ // create a new Document that contains the Orcl-value
			c.send(new OrclDoc("index.xml"));
			Debug.trace(this, "Initial document created and sent");
		}
		catch (Exception e) {
			Debug.trace(this, "Exception: " + e.getMessage());
			e.printStackTrace();}
	}

	// returns a report to a device described in the ConnectivityDescription

	public void makeReport(ConnectivityDescription ConnDescr)
	{
		try
		{ // find the FlexiNet-trader
			FNetTrader trader = UK.co.ansa.flexinet.test.FNetTest.getTrader();
		  // find the UserAccess
			UserAccess UA=(UserAccess)trader.get

("UA@"+InetAddress.getByName(ConnDescr.getHost()).getHostAddress());
		  // request the opening of the connection
			Connection c= UA.openConnection(ConnDescr);
		  // send document
```

```
                    c.send(new OrclDoc("index.xml"));
             }
             catch (UK.co.ansa.flexinet.core.FlexiException  e)
             {Debug.trace(this, "Error FlexiException " + e);e.printStackTrace();}
             catch (java.net.UnknownHostException  e)
             {Debug.trace(this, "Error UnknownHostException " +
e);e.printStackTrace();}
             catch (Exception  e)
             {Debug.trace(this, "Error Exception " + e);e.printStackTrace();}
      }
}

/**

class OrclDoc implements Document

{
      String fname;
      MimeType mimetype;
      Hashtable parameter;

      public OrclDoc(String name)
      {
             Debug.trace(this, "Now creating document " + name);
             if (name.equals("")) name = "index.xml";

             mimetype =
DE.fast.followme.useraccess.device.httpdevice.HttpUtils.guessContentTypeFromName(name);

             if (name.equals("index.xml") || name.equals(""))
             {
                    /*  Create an XML-file that contains the right values */
             }
             else /* all other documents are read from the filespace
             {
                    mimetype =
DE.fast.followme.useraccess.device.httpdevice.HttpUtils.guessContentTypeFromName(name);
                    fname = DE.fast.followme.tests.Orcl.DocDir+name;

             }

      }

      /**
      * Retrieves the external representation of a document
      */
      public ExternalRepresentation getExternalRepresentation(MimeType mt)
      {
             return new ExternalRepresentation()
             {
                    public  InputStream getInputStream() throws java.io.IOException
                    {
                           try
                           { // FlexiNetInputStream3 is an InputStream that
                             // works through FlexiNet Bindings
                                 return new FlexiNetInputStream3(...)
                           }
                           catch(IOException io_e)
                           {
                                 throw new java.io.IOException("IOException:"+io_e);
                           }
                    }

                    public MimeType getMimeType()
                    { return mimetype;}
             };
      }


      /**
      * returns the list of supported MIME-Types
      */

      public MimeType[] getsupportedMimeTypes()
```

```
        {
                return new MimeType[] {mimetype};
        }

        public Document getReferredDocument(String name, java.util.Properties params)
                                                throws DocumentNotFound
        {
                if (name.equals("close")) return null;
                else    return new OrclDoc(name);
        }

}
```

5.2 How to use XML and XSL

This section gives an extensive example on how to employ XSL-techniques to define layouts that are rendered by the user access.

It assumes familiarity with the XML and XSL specifications ([10], [8]). More information about XSL is available at [11].

The debugging of complex XML and XSL statements can be quite troublesome. To check XML and corresponding XSL-files there is a java class DE.fast.followme.tests.XMLtest. It can be started by:

java DE.fast.followme.tests.XMLtest [-M<mode>] <xml-filename> [<xsl-filename>]

5.2.1 XML Example

The following example shows a (simplified) XML representation of a newspaper (taken from the ETEL++ pilot application). The newspaper contains a set of articles. Articles contain a classification <CLASSEMENT> and the body <DONNEE>. The body contains the title <TITRE>, subtitle <SURTITRE>, etc. followed by the text <TEXTE>. The body can contain text and photos.

```
<?xml version="1.0" ?>
<?xml-stylesheet href="newspaper.xsl" ?>
<NEWSPAPER>
<ARTICLE>
<CLASSEMENT>
<NOM></NOM>
<AUTEUR></AUTEUR>
<DATE></DATE>
    <LISTE_MOTSCLE>
<MOTSCLE>Quotidien</MOTSCLE>
<MOTSCLE>UNE</MOTSCLE>
<MOTSCLE>Spectacle</MOTSCLE>
    </LISTE_MOTSCLE>
<GEOGRAPHIE>
<PAYS>France</PAYS>
<REGION>Bretagne</REGION>
<LOCALITE>Briec</LOCALITE>
</GEOGRAPHIE>
<POIDS>50</POIDS>
</CLASSEMENT>
<DONNEES>
<SURTITRE><![CDATA[Portes ouvertes a l'espace Xavier-Rousseau, samedi]]></SURTITRE>
<TITRE><![CDATA[Decouvrir les activites de l'EXR]]></TITRE>
<CHAPEAU><![CDATA[Pour la premiere fois, l'Espace Xavier-Rousseau ouvre ses portes au
public, samedi 20 juin, pour faire decouvrir les activites menees par les differents
animateurs. Un spectacle de danse cloturera cette journee, salle de La
Bayard.]]></CHAPEAU>
<RESUME><![CDATA[Decouvrir, regarder, jouer, participer ou bien applaudir... Voila ce
qui est propose par l'Espace Xavier-Rousseau, le samedi 20 juin, lors de l'operation
 portes ouvertes .
De 13 h a 18 h, que vous te journee, salle de La Bayard.]]></RESUME>
```

```
<TEXTE><![CDATA[Decouvrir, regarder, jouer, participer ou bien applaudir... Voila ce
qui est propose par l'Espace Xavier-Rousseau, le samedi 20 juin, lors de l'operation
portes ouvertes .
De 13 h a 18 h, que vous soyez jeunes ou moins jeunes, vous serez accueillis par les
permanents et animateurs de l'association qui ne manqueront pas de vous presenter leurs
ateliers. Qu'il s'agisse d'activites d'expression corporelle, telles que theatre,
danse, de musique ou bien manuelles, le choix est important.
Expos et videos
 Des expositions presentant les activites randonnee, anglais, yoga, peinture... seront
proposees, ainsi que des videos sur les cours de gymnastique adultes et retraites,
cours d'eveil de danse, explique un des permanents de l'association. Toutes les
activites ne se derouleront pas dans nos locaux. Par exemple, une seance d'aquagym aura
lieu de 14 h 30 a 15 h 30 a la piscine.
En effet, il sera possible de decouvrir le billard, a la Maison des associations, de 14
h a 18 h. Les amateurs d'escalade devront se rendre place de la Gare, sur le parking de
la salle Jean-Lenoir, entre 13 h 30 et 15 h ou de 16 h 15 a 18 h.
Un spectacle de danse intitule  Les coups au coeur des Tarabiscotes , salle de La
Bayard, a 20 h 30, cloturera cette journee. Pendant ce temps l'orchestre  La Medina
participera a la Fete de la musique et se produira devant les locaux de
l'association.]]></TEXTE>
<LEGENDE>Une apres-midi  portes ouvertes  est organisee a l'Espace Xavier-Rousseau,
samedi 20 juin.</LEGENDE>
<PHOTO>al04g207_19980616.jpg</PHOTO>
</DONNEES>
</ARTICLE>
</NEWSPAPER>
```

The first line contains an indication that this is an XML document and a version number.

The following line defines a reference to the corresponding xsl-document. If this reference is omitted then the reference “index.xsl” is assumed.

5.2.2 XSL example

The following example shows an XSL file that maps the previous newspaper to different results depending on the required mode. Supported modes are

“ ”(default):	for devices which can understand full HTML, e.g. Browsers (and fax gateways, because the specific mode “staticHtml” is not defined.
“text”:	for devices which can understand only text, e.g. mail
“shortText” :	for SMS and similar devices with limited text capabilities

The example constructs three types of layout:

The HTML layout (which is also a staticHtml layout) represents the newspaper with all articles and images. The text layout contains the title, the bodies, and for each photo only the alternative text. Finally there is a shortText layout that presents only the headings of the top important articles. [6]

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<!-- the root rule for HTML -->
<xsl:template match="/">
    <html> <!-- here comes lots of HTML stuff surrounding the body -->
     <head> <title>FollowMe - NewsPaper 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/>
             <!-- here comes the contents: Write only the data -->
```

[6] Please note that the XSL syntax follows the syntax of the XSL proposal [8]. The XSL working group has established a working draft [9]. Although this version has not changed in spirit, it has substantial changes in the syntax. As soon as java-based converters are available, the user access will provide a migration path to the new version.

```
             <xsl:for-each select="ARTICLE">
                    <xsl:apply-templates/>
             </xsl:for-each>
          </td></tr>
         </table>
    </body>
    </html>
</xsl:template>

<!-- the root rule for short text (SMS): Only produce the headings of the top news
     separated by comma -->

<xsl:template match="/" mode="shortText">
       Top headings:
       <xsl:for-each select="ARTICLE">
       <xsl:for-each select="DONNEES">
             <xsl:for-each select="TITRE">
                <xsl:apply-templates mode="shortText"/>,
             </xsl:for-each>
       </xsl:for-each>
       </xsl:for-each>
</xsl:template>

<!-- the root rule for mode text -->
<xsl:template match="/" mode="text">
       <xsl:apply-templates mode="text"/>
</xsl:template>

<!-- standard mode -->

<xsl:template match="DONNEES">
          <xsl:for-each select="TITRE">
             <H1><xsl:apply-templates/></H1>
          </xsl:for-each>
          <xsl:for-each select="SURTITRE">
             <BOLD><xsl:apply-templates/></BOLD>
          </xsl:for-each>
             <!-- xsl:foreach ... further elements ... -->
          <xsl:for-each select="TEXTE">
             <HR/><FONT SIZE="16"><xsl:apply-templates/></FONT><BR/>
          </xsl:for-each>
</xsl:template>


<xsl:template match="PHOTO">
       <IMG SRC='{.}' alt='Photo for this article' ALIGN="RIGHT"/>
</xsl:template>

<!-- text mode -->

<!-- In text mode: Just give Title and text body process the articles -->
<xsl:template match="ARTICLE" mode="text">
          <xsl:for-each select="DONNEES/TITRE">
             <xsl:apply-templates mode="text"/>
        </xsl:for-each>:

         <xsl:for-each select="DONNEES/TEXTE">
             <xsl:apply-templates mode="text"/>
        </xsl:for-each>:
</xsl:template>

<xsl:template match="PHOTO" mode="text">
         <br/>--- Photo cannot be displayed ---<br/>
</xsl:template>
</xsl:stylesheet>
```

6 References

[1] DH2, User Access Requirements, M. Breu, A. Sindermann, E. Triep, December 1997.
[2] DD6.2, DE5.2, DF5.1 & DF5.2: Agent Framework: Software Reports, S. Battle, N. Taylor, J. Tidmus, M. Yearworth, October 1998
[3] FlexiNet Documentation (by ANSA)
[4] DB7.4 & DB 8.3: "Mobile Object Workbench Software Report".
[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] Extensible Style Sheet Language (XSL), Version 1.0, W3C working draft, 18. Aug. 98, http://www.w3.org/TR/WD-xsl
[10] XML: Extensible Markup Language (XML) 1.0, W3C Recommendation 10-February-1998 (http://www.w3.org/TR/1998/REC-xml-19980210)
[11] http://www.w3.org/Style/XSL/
[12] The Swing Connection, http://java.sun.com/products/jfc/tsc/
[13] Design Patterns Elements of Reusable Object Oriented Software. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Addison Wesley 1995. ISBN 0-201-63361-2
[14] DA 1.3, Architecture Release 1.3, Will Harwood