ESPRIT Project No. 25 338

Work package J

Pilot application 2

Description of the Working System

ID:	DJ4	Date:	01.03.99
Author(s):	**INRIA & TCM**	Status:	
Reviewer(s):		Distribution:	

Change History

Document Code	Change Description	Author	Date
DJ4	First version of the document.	INRIA & TCM	01.Mar.99

1 Introduction

This document contains the complete description of the second pilot application created by INRIA together with TCM. This application generates for a set of identified users electronic versions of the French daily regional newspaper Ouest-France. The major characteristics of this pilot application are:

- Each version of the newspaper is personalized, and reflects by different means the preference of each users (like specific center of interests);
- Each version of the newspaper is specific in the sense that it includes data whose content depends on the actual location of each user. Agents are typically used for performing the tasks of finding the right information;
- Each version may potentially be read using different type of browsers;
- The construction of versions is performed in a fully distributed manner;
- In general, the version created for a particular user is generated “near” this user. A mobile user will therefore force the creation of versions to follow him;
- The data-flows resulting from a workload-aware algorithm drives the construction of versions. Optimal routes form transferring data are followed when possible;

This paper presents the above features through the following sections. Section 2 presents an overview of the architecture of the application and highlights its major processing phases. Section 3 details the tool we use to extract articles from the actual Ouest-France Database before making them available to ETEL++. Section 4 details the hierarchy of objects representing the newspaper, and presents the typical composition relationship objects have together. Section 5 presents the principal internal data structures that are required by the application. Section 6 presents the retrieval and exploitation of User Profiles. Section 7 presents the workload-aware algorithm enforcing the deployment of the application. Section 8 details the assembling engine that creates personalized editions. Section 9 details the support for multiple terminals. Section 10 presents the use of Agents to obtain context-sensitive data.

2 An Overview of the Architecture of ETEL++

It is possible to identify four different entities that together compose the second pilot application. These entities are represented Figure 1. The first entity corresponds to the real database at Ouest-France. This database stores the articles produced daily by the journalists. We presented in Document DJ3 the structure of that database. The second entity is the ETEL++ Main Server.

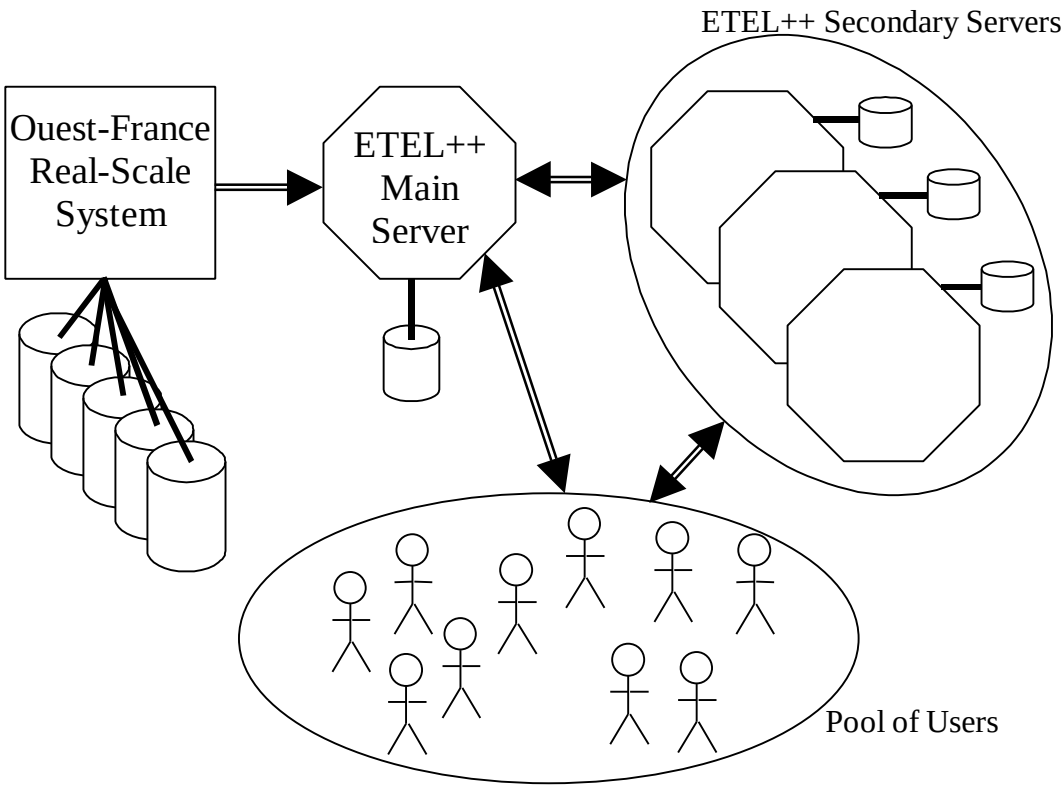

Figure 1: An Architecture Based on Four Entities.

This server is responsible for extracting the articles from the Ouest-France database. It also converts them in a format that is appropriate for ETEL++ related manipulations, and stores them in its local Information Space. The extraction of data and the associated conversion process are detailed next Section. The ETEL++ main server is responsible for delivering converted articles to the secondary ETEL++ servers. These servers correspond to the third type of entities. They are responsible for determining a subset of the whole readership of the newspaper and to assemble the editions for this subset in an appropriate way. They are also responsible for delivering editions to final users in a format that fit their terminals. These users correspond to the fourth and last type of entities. Final users connect to the ETEL++ service and request their newspaper either for immediate browsing or for off-line delivery.

There is at least one Information Space (IS) associated to each ETEL++ server. One IS stores the pieces of data that are important to have locally for the sake of performance. It is possible, however, that an ETEL++ server needs data that is not stored in its local IS. In this case, this server may access that data through the network in a Query-Shipping mode, or cooperate with the owner of the data that is needed to create a local copy, and therefore operate in a Data-Shipping mode. Such decisions, using either Query-Shipping or Data-Shipping modes is determined at run-time by a workload-aware algorithm cooperating with the Service Deployment workpackage.

Servers communicate via the MOW capabilities. Each server ignores distribution and knows about other servers by looking-up interfaces in a trader (a name service, referred to as the TrivTrader in the MOW documentation). During the construction of personalized editions, servers exchange messages carrying raw articles and their meta-data, pictures or elaborated articles after their adaptation to the requirements of a specific type of terminal.

The main ETEL++ server is the first one to propagate to other servers the editorial material. Subsequently, any ETEL++ secondary server can receive data either from the main server, or from another secondary server (a secondary server is therefore able to propagate data). This precludes the main server of being a bottleneck potentially hurting performance. Since articles can “travel” between any server, in any direction and irrespective of the role of the source and the destination, it is possible to draw a map of the flows of data that would best enforce the performance of the system. This feature is detailed later in this paper.

Articles are stored in Information Spaces as White Box objects (see IS documents).

3 Data Base Extractor

The first task that is performed is the extraction of data from the real Ouest-France Database and its conversion into XML.

The database of articles at Ouest-France can be divided in two parts. The first part is managed by a relational database management system, namely Sybase. This relational DBMS is responsible for managing the creation aspects of the paper-printed version of the newspaper, that is, journalists enter their articles in this database, graphic specialists decide upon the layout of images, etc. This database is therefore extensively used in read/write modes. It is nightly used by the publishing process to produce the paper edition that is delivered to subscribers each morning.

The second part is in contrast managed by an object-oriented DBMS. This part of the database is oriented towards read-only access by on-line subscribers, that is, by ETEL users. The object representation of articles, images and other editorial material is in this case more appropriate for navigation-based processing, searches in the archives or other full-text queries. This type of processing tasks can fully use object properties as inheritance, references, and can fully utilize the class schema. The data managed by this OODBMS (namely O2) is an object translation of the data managed by the RDBMS. Various indexes on both databases make possible and efficient the access to articles.

In our case, the database extractor simply selects articles on their date from the OODBMS database. The date is a sufficient criterion for the purpose of ETEL++. Once the relevant articles have been selected, a C++ program isolates each individual article and performs the following tasks:

- It creates a new, empty file, on a specific file system,
- Since each article is an object, it applies methods to extract the title, the headings, the summary, the body, etc… from the article. It writes in the created file each piece of data prefixed and suffixed by an XML tag as defined in the grammar (presented later in this section),
- It generates a request sent to the database to retrieve the keywords associated to the articles (the journalist who wrote the article has set the keywords). The result is also written in the file, enclosed with the appropriate tags,

- It generates another request to obtain geographical information about the current article: the area of coverage, the town(s), the region(s) and the country(ies) that have relationship with the article, with respect to the journalists specifications,
- It generates another request to fetch the pictures that may be associated with the article. If there is at least one picture associated to the current article, a new file is created, and the picture is written there. For simplicity, in case of multiple pictures associated to an article, the one having the largest importance (this is known from the analysis of meta-data) is kept, others are ignored.

Objects stored in the Ouest-France database are much larger and much more complex than their "translation" into simple files as performed according to the above description. In particular, very complex layout information is ignored because ETEL++ does not try to produce personalized editions *that look like* the paper-based newspaper (this in contrast to ETEL. See Document DJ3).

The following text is a typical example of an article once transformed into an XML file after being fetched from the real Ouest-France database. In this example, the keywords are "*Quotidien, UNE, Spectacle*" (a possible translation could be: Miscellaneous News, Front-Page, Shows), the geographical information attached says that this article talks about something that took place in *Briec*, a small town in west Brittany.

```
<?XML VERSION="1.0" ?>
<!DOCTYPE article SYSTEM "article.dtd" >
<ARTICLE>
<CLASSEMENT>
      <NOM>aJng6loPPlki_19_RFD__19980616</NOM>
      <AUTEUR>Jean-Pierre Arceau</AUTEUR>
      <DATE>1998-06-16</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>
```

The extractor generates XML articles using the objects that have been selected in conformance to a grammar. This grammar, also called DTD, is given below.

```
<!ELEMENT ARTICLE (CLASSEMENT, DONNEES)>

<!ELEMENT CLASSEMENT (NOM, AUTEUR, DATE, LISTE_MOTSCLE, GEOGRAPHIE, POIDS)>
<!ELEMENT NOM (#PCDATA)>
<!ELEMENT AUTEUR (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT LISTE_MOTSCLE (MOTSCLE+)>
<!ELEMENT MOTSCLE (#PCDATA)>
<!ELEMENT GEOGRAPHIE (PAYS, REGION?, LOCALITE+)>
<!ELEMENT PAYS (#PCDATA)>
<!ELEMENT REGION (#PCDATA)>
<!ELEMENT LOCALITE (#PCDATA)>
<!ELEMENT POIDS (#PCDATA)>
```

```
<!ELEMENT DONNEES (SURTITRE, TITRE, CHAPEAU, RESUME, TEXTE, PHOTO, LEGENDE)>
<!ELEMENT TITRE (#PCDATA)>
<!ELEMENT SURTITRE (#PCDATA)>
<!ELEMENT RESUME (#PCDATA)>
<!ELEMENT CHAPEAU (#PCDATA)>
<!ELEMENT TEXTE (#PCDATA)>
<!ELEMENT PHOTO (#PCDATA)>
<!ELEMENT LEGENDE (#PCDATA)>
```

4 Object Hierarchy

This section presents the hierarchy of objects that is used to represent the articles manipulated by ETEL++. First, we detail the inheritance relationship between objects. Second, we detail a typical composition of objects.

4.1 Inheritance

ETEL++ uses three hierarchies of document classes. The first hierarchy is used to represent "intelligent" documents that are able to (i) answer queries on the meta-data describing the document and (ii) generate a new representation of themselves for delivery to a user terminal. Objects belonging to this hierarchy are said to be *active*. The second hierarchy is used to represent all the possible representation of documents that can be delivered to final users. Objects belonging to this hierarchy are said to be *passive*. Last, the third hierarchy is used to federate Active and Passive documents.

Regardless of the above hierarchies, a document can belong to one of the three following categories.

1. *Article*. If an ETEL++ document is a basic article (like the one presented in the previous section), then the ETEL++ document is said to be an *Article*. There is one *Article* created for every single article extracted from the database at Ouest-France.

2. *Rubrique*. If an ETEL++ document is referencing multiple *Articles*, then this ETEL++ document is said to be a *Rubrique* (this corresponds to a Column). There is one Rubrique per keyword found in the newspaper, and one per geographical information (see the XML DTD). Therefore, there might be in today's electronic edition a *Rubrique* from which all the *Articles* talking about Sport can be referenced, and another one from which all *Articles* talking about Paris or Munich can be accessed. It is also possible that a *Rubrique* referencing other *Rubriques* is created. For example, a (top) *Rubrique* may reference all other (bottom) *Rubriques* talking about economy, each (bottom) *Rubrique* referencing *Articles* about economy in a given country. In a nutshell, *Rubriques* can be nested up to an arbitrary level. It is the meta-data attached

to the *Articles* extracted from the database at Ouest-France that drives the nesting (or the absence of nesting) of *Rubriques*. A given electronic edition might, or might not, contain nested *Rubriques*.

3. *Front-Page*. The last category of ETEL++ document corresponds to front-pages. There is one specific front-page for each individual user. A front-page is in fact the entry point from which a user can browse its entire personal electronic edition. A front-page contains specific information about its associated user, references specific *Rubriques* and *Articles* according to the user's preferences.

These three categories of documents will be found in each hierarchy as detailed below.

4.1.1 Active Documents

An Active Document (therefore an Active *Article*, an Active *Rubrique* or an Active *Front-Page*) contains meta-data attached to the document, and is able to transform itself in other documents having different representations. An Active document is self sufficient in the sense that it contains, or can produce, all of the data that may be required by any processing task running inside ETEL++. An Active document is a fairly large object since it is an object representation of the XML data detailed above. It contains also additional information that helps the processing tasks.

A copy of a particular active document is needed on ETEL++ secondary servers that are instructed to generate a specific version (for example an HTML version) of this document. An active document is not needed on ETEL++ secondary servers that do not generate final representation of documents, but rather that obtain that final representation from another server that generated this representation on its behalf.
The hierarchy of Active Documents is represented on Figure 2.

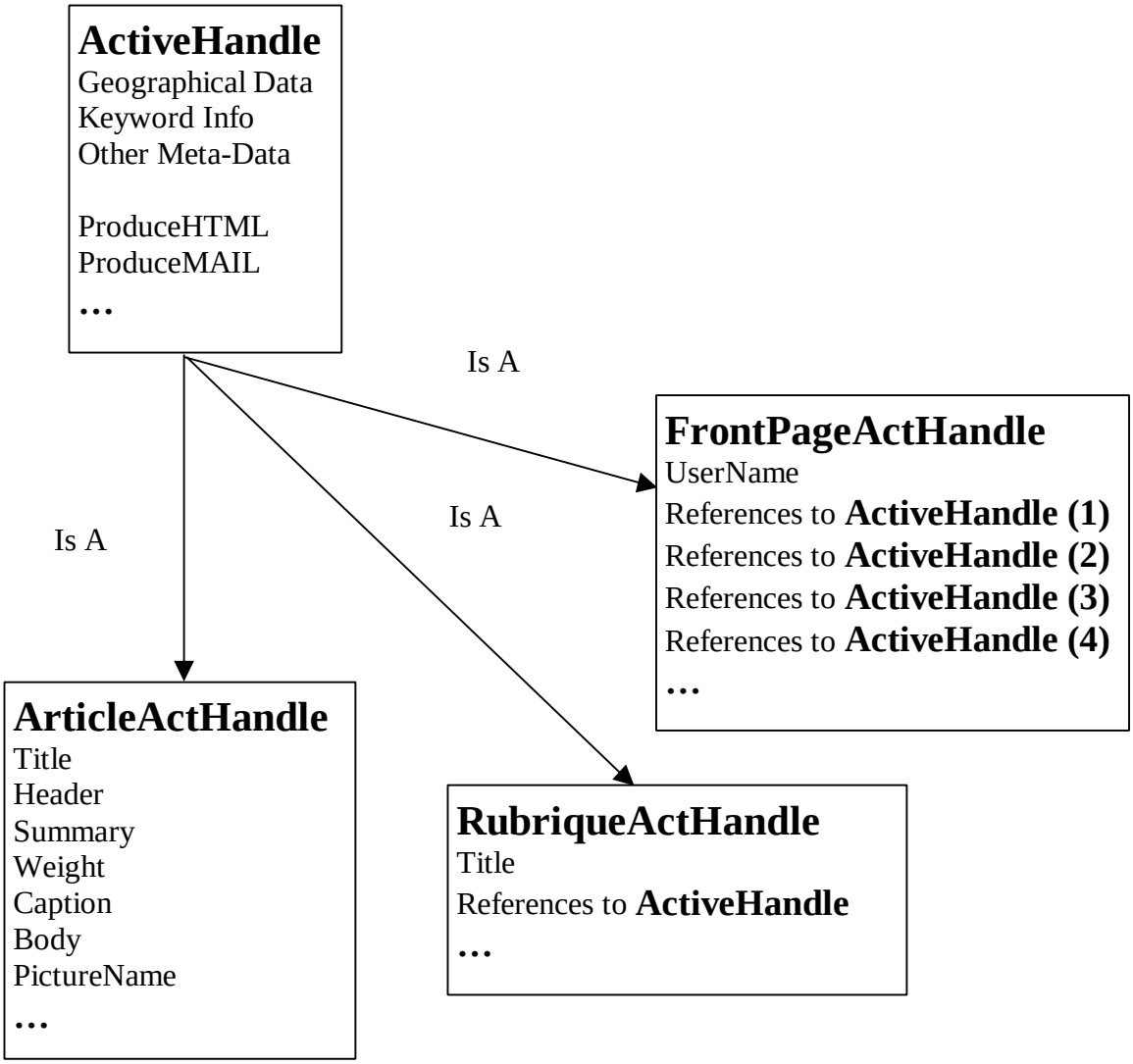

Figure 2: The Hierarchy of Active Documents

4.1.2 Passive Documents

Passive Documents (therefore Passive *Articles*, Passive *Rubriques* or Passive *Front-Pages*) correspond to the final representations of Active documents that are ultimately delivered to the end-users. A Passive Document is generated by an Active Document based on the information this Active document contains, together with some specific user-related information (like the terminal used for browsing). A Passive Document has no special capabilities, as opposed to Active documents. A Passive Document is simply able to display itself on the user's terminal as it is. A Passive Document is not able to adapt itself to the particular terminal used by a given

user. To display newspaper data on a particular terminal, an Active document has to be contacted, and this Active document generates (on the fly, or off-line) the passive document that directly goes to the terminal without any further processing. Therefore, there are as many categories of Passive Documents than formats supported by ETEL++.

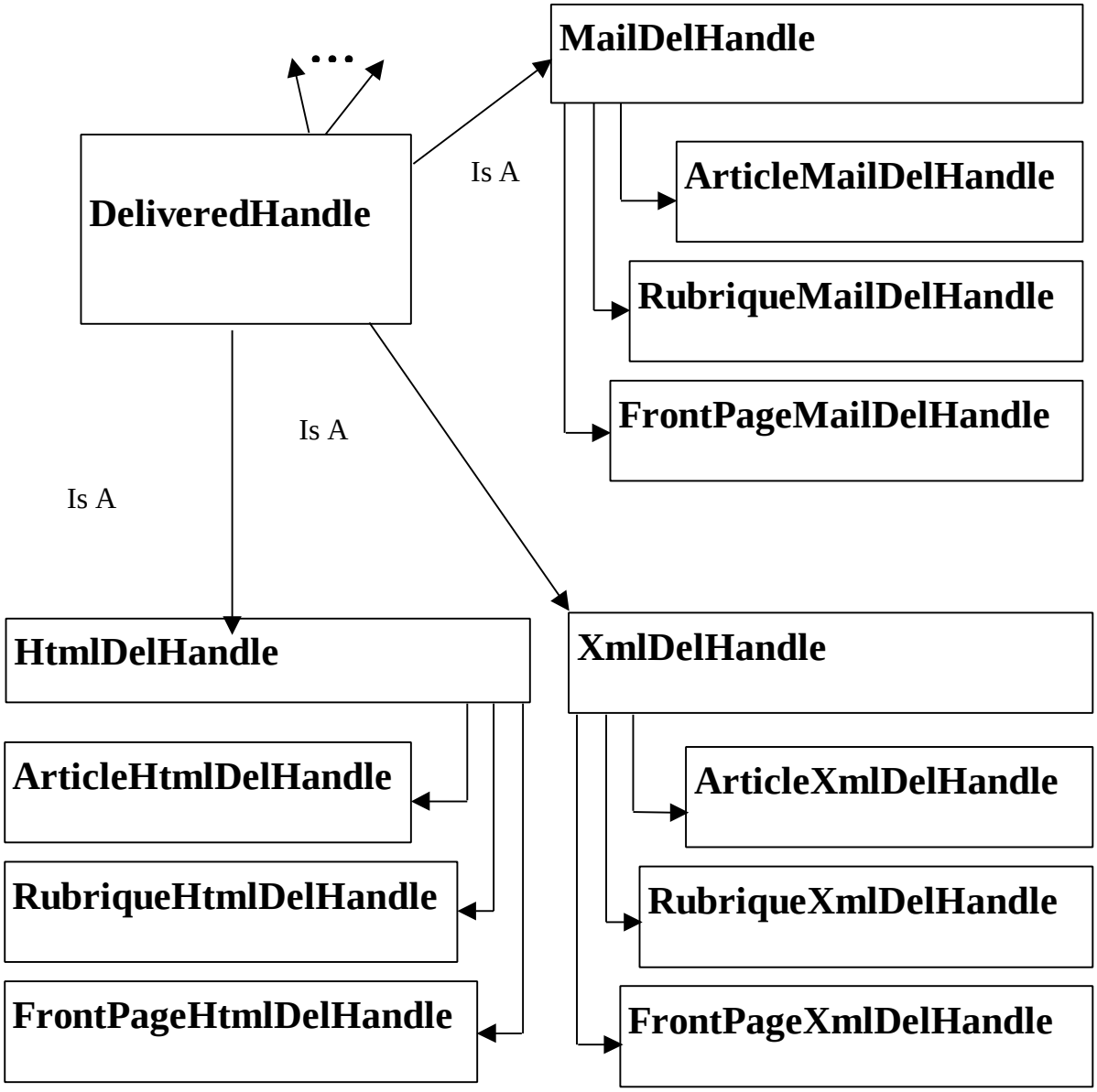

Figure 3: The Hierarchy of Passive Documents

The current implementation supports HTML-based browsers, Mail-based terminals and considers that the XML representation of data is a sort of final and passive representation. Having the XML data accessible as a Passive Object allows easy extension of the supported final

devices. In the implementation, Passive Documents are referred to as Delivered Documents since their ultimate goal is to be delivered on users terminals. The hierarchy of Passive documents is consequently as illustrated by the Figure 3.

4.1.3 Federating Documents

A Federating Document (there are 3 types of it: *Article*, *Rubrique* and *Front-Page*) is an object that is used to associate Active and Passive Documents. Federating documents are the ones whose identity is propagated inside ETEL++ pilot application. From such a document, specific methods allow access to the Active version, or to any of the Passive one. Strong encapsulation is enforced. Propagating solely the identity of federated documents (and not their sub-components) eases the implementation. The hierarchy of this type of Documents is as illustrated Figure 4.

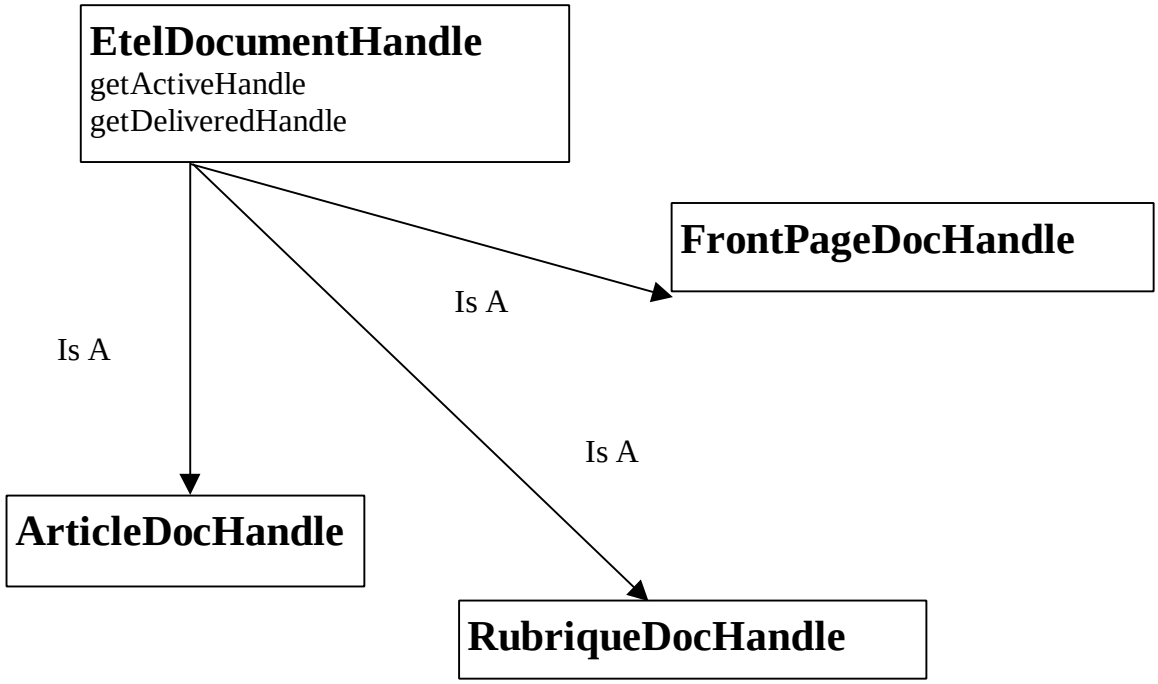

Figure 4: The Hierarchy of ETEL++ Documents

Although it is not represented in this Figure, but ArticleDocHandles, RubriquesDocHandles and FrontPageDocHandles implements the interface *Document* provided by FAST, and that makes possible the rendering of multi-terminals documents.

4.2 Composition

Objects belonging to the hierarchies presented above have to be composed to represent all the editorial material originating from Ouest-France. A possible composition of objects may be as represented Figure 5.

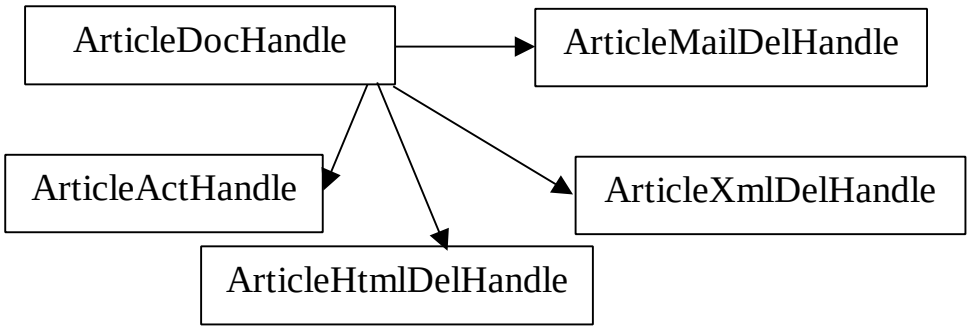

Figure 5: A Typical Composition of ETEL++ Objects

5 Internal Data Structures

We give in this section an overview of the main data structures that are used by the application.

5.1 Inverted Lists

ETEL++ requires several inverted lists for performing the assembling phases of the newspaper. The inverted lists are generated as a side effect of the parsing of Articles. During parsing, the XML tags *MOTSCLE*, *PAYS*, *REGION* and *LOCALITE* (this stands for KEYWORD, COUNTRY, REGION and TOWN) trigger the insertion of the associated data into a specific list (there are four such lists). When all the articles have been parsed, four inverted lists are built using the data stored in each list. The first inverted list enables to know the reference of all articles given a specific keyword. The second returns all the articles with a specific country, the third returns all the articles with a specific region, and the last inverted list returns all the articles with a specific town.

These inverted lists are used during the assembling phase of the personalized editions. Given the profile of a particular user (and thus knowing the keywords he chose), it is possible to know all the articles that have to be assembled for this user. In addition, it is possible to use these lists to built Rubriques (as explained in the previous section) for referencing all articles with a given keyword or a given geographical criterion. Rubriques that are newly created are also inserted into the inverted lists.

These inverted lists are created within the memory of the main ETEL++ server, and also within each secondary ETEL++ server. They differ, however, by the number of articles each reference. At the main ETEL++ server, *all* articles associated to the edition for a given date are referenced. In contrast, only the articles that are needed by a secondary ETEL++ server are referenced from such a server. The main ETEL++ server is therefore able to build a global view of the ultimate location of articles once the ones needed by each individual secondary server will be transferred. This global view is required to decide adequately upon the flow of data to enforce for performance. In contrast, each secondary ETEL++ server has only a limited view on these

articles, and does not waste a significant amount of memory in storing useless information in its internal data structures.

5.2 User Profiles

The goal of ETEL++ is to build personalized editions for its users. It therefore needs to maintain data structures that describe every single user. The data that characterize each user is referred to as its profile. It is important to note that the profile of a user must be accessible even if the user is not connected to the FollowMe environment. Therefore, user profiles are stored in a particular place that may be interrogated by ETEL++ processing tasks any time. The profiles may be created or updated by users when these connect. The profile of a user contains its name that acts as a unique identifier. It contains the keywords that the user selected as its center of interests (there are thematic keywords, and also geographical keywords) and also stores the possible location from which the user might connect the next time this user will want to access its personalized edition. This last information corresponds to what could be stored in the electronic diary of a user. By exploiting this information, ETEL++ is able to determine the server among all ETEL++ servers that will generate the newspaper for each user. In addition, the appropriate server can determine, by analyzing the data stored in each profile, what are all the keywords set for all the users it deals with, and therefore can know which articles it has to manipulate, and in which ways.

5.3 Workload Information

For the purpose of Service Deployment, ETEL++ needs to retrieve information about the workload of each server, and subsequently to determine routes for data flows that would best enforce the global performance of the application. Workload information is obtained via the interfaces of the Service Deployment workpackage. Each server (either the main server or any secondary server) can be decomposed in three parts, one devoted to take care of all the processing of editorial material, one dedicated to deal with all agent issues, and one dedicated to manage the performance of the server. This last part works with a local performance monitor that returns upon request values reflecting the usage of the local resources. Any other server may get a view of the current workload of a remote server by interrogating its performance-related part. The main ETEL++ server typically performs this before deciding on the routes for data flows. It therefore maintains a data structure in which all workload information of remote sites are kept. The usage of this data structure is further detailed Section 7.

6 Accessing User Profiles

We outline in this section the management of User Profiles. This management relies extensively on one of the features of the Agent Framework provided by UWE. The management of User Profiles uses the ability of specifying *properties* on data and also the *querying facilities* that are possible thanks to the trader provided by UWE. In addition, the management of Profiles uses the notion of Information Spaces proposed by APM.

We first motivate the need for persistent profiles before presenting the properties that are set, and the queries that are posed on the UWE trader. Finally, we present the interface.

6.1 Persistent User Profiles

User profiles are used to store data about each individual user who may not be continuously connected to the network. Therefore, profiles must be kept beyond the lifetime of a user connection, that is, they must be made persistent. Another motivation for this need is the requirement of being able to access any profile any time for the purpose of newspaper construction. Not only profiles must be persistent, but also they must be stored in a location that is independent of users.

There is therefore a User Profile manager that keeps a database of profiles, and that is accessed by users to browse and/or update their data, or that is accessed by read-only tasks to retrieve data that is store.

Users are transparently connected to this profile manager. Upon request, their profile is displayed, and they are free to update it. To display a profile, the internal processing contacts the trivial trader set by APM in order to know the reference of the trader that is provided by UWE. It then contacts this last trader and feeds-in the name of the user that has been entered at connection time. It is therefore possible to get a reference on the persistent object that represents the profile, to fetch it and to create adequate objects in memory that supports profile manipulations.
Profiles are managed as White Box Objects. This makes their updating more flexible and simple for the Profile Manager since updates are automatically trapped and propagated to the stored

copy. In addition, using white Box objects increases the level of resilience to failures since APM copes with certain failures.

The creation of a profile is enforced when a user connects for the first time to the ETEL++ application. A typical template profile is instantiated and proposed to this new user.

6.2 The Use of Properties

When first bootstrapped, the Profile Manager creates a new information context within the UWE trader. This context is devoted to store information about the profiles that will be later created by users. Each time a new profile is created, a new record is inserted in this context. A record keeps the minimal amount of data that is needed to determine, in later phases of processing, the ETEL++ server that needs to read the associated profile. A record contains a reference to the persistent profile, a string that stands for the name of the user, and a string that stands for the (possible) location of the next user connection. Both the user name and the location are defined as *properties*. Therefore, it is possible to send a query to the UWE trader to get the specific record characterizing a user given its name, or given a location.

When a user wants to access its profile, his name is used to access the trader as the property for the query.

When a given ETEL++ server wants to know who are the readers it will have to work for, it builds a query in which its own location is specified, and sends it over to the UWE trader. In turn, it receives back all matching records, that is, all records stored in the ETEL++ context in the UWE trader that have the same location as a set property. It is possible, however, that the specified location of the next connection for a user does not corresponds to the actual location of any ETEL++ server. To face this problem, the Profile Manager stores geographical information specifying triplets (Town, Region, Country) for a large set of predefined locations. For example, the actual locations of ETEL++ servers might be *Munich* and *Rennes*, while the location set for a particular user is *Paris*. In this case, the actual location of ETEL++ servers is extended to incorporate the facts that *Rennes* is in *Brittany* in *France*, and that *Munich* is in *Bavaria* in *Germany*. The triplets are therefore (Rennes, Brittany, France) and (Munich, Bavaria, Germany). Even though the user specified *Paris*, this information is internally augmented to reflect the facts that *Paris* is in the *Ile De France* Region in *France*. When determining the server that will take care of this user, a first matching on town names is attempted. If no matching is found, a matching on Region names is tried before ultimately performing a matching on the country name. With the example above, the server located in *Rennes* will service the user because a matching on the country name was found. Of course, this approach works only for a predefined set of registered locations for both ETEL++ sites and users.

6.3 Interface

Figure 6 presents a screen capture of the interface that is proposed to users for managing their profiles. This Figure can be divided into three main parts.
On the left, a sub-window gives a list of the thematic keywords defined by Ouest-France, and that are available to users for selection. When a keyword is selected, a tick is inserted in the check-box. This list is a short subset of all the possible keywords that may characterize the articles produced by the journalists. We voluntarily limited the length of this list for simplicity, and the keywords that we kept are the ones that offer a good compromise between accuracy (and therefore only a small number of articles match a given keyword) and generality (and therefore enough articles match also). In fact, it corresponds to the upper levels of the hierarchy of keywords that are defined by Ouest-France. Note that for ETEL the selection of keywords is based on a complex tool presenting nested hierarchies of keywords.

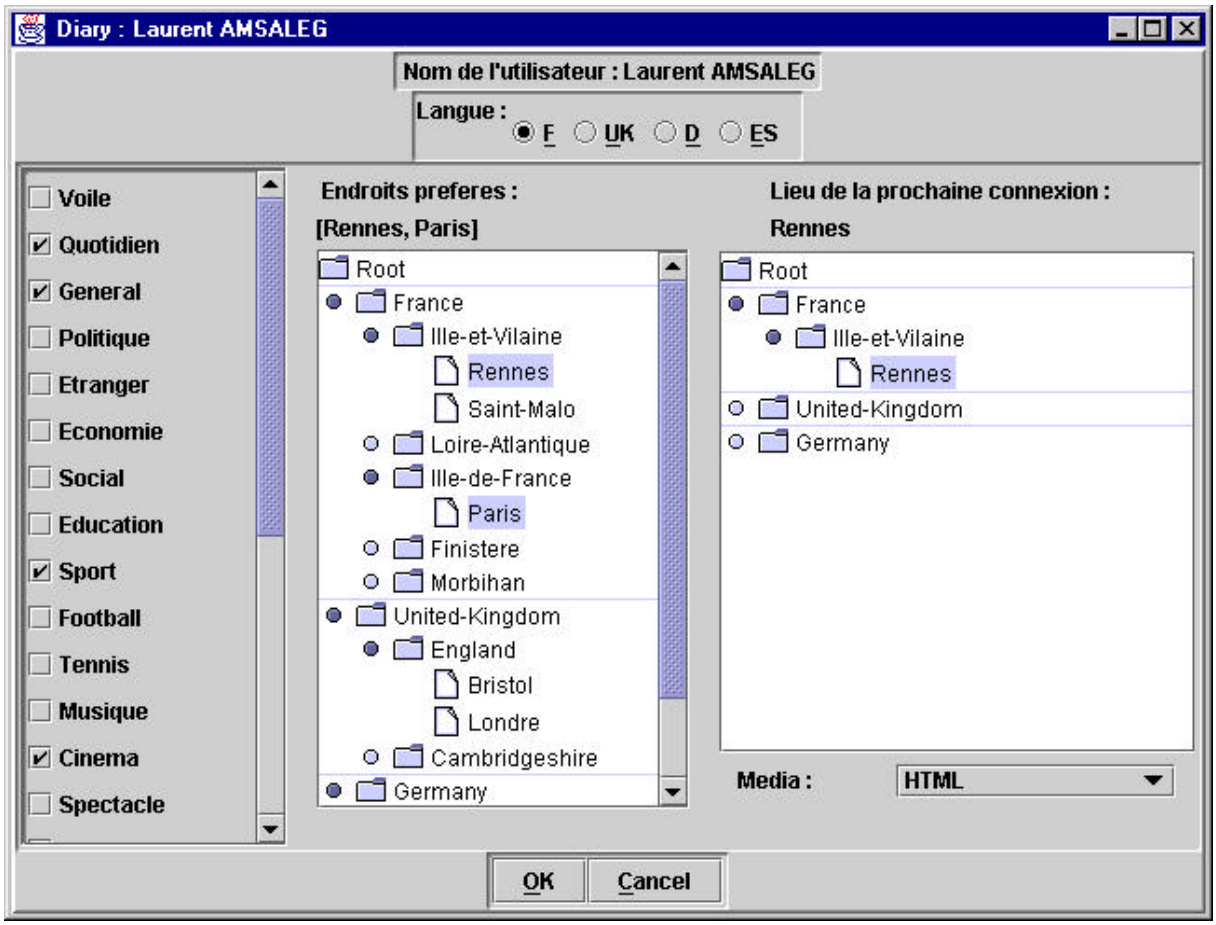

Figure 6: Interface of a User Profile

The center of the Figure is occupied by another sub-window in which a user may specify the towns, regions or countries of interests. In the case illustrated by the Figure, the user is interested

in reading articles talking about *Paris* and *Rennes*. This information corresponds to the geographical keywords a user may set.

On the right of the Figure, the sub-window offers means to specify the possible location of the next connection. Just below this sub-window, the user can set the type of terminal he is likely to use during the next connection session. In the case illustrated, the user informs ETEL++ that he may be connected from Rennes via an HTML-based browser. In turn, ETEL++ will ask the server that is close to Rennes to produce the edition for that user, and will require that server to create Passive HTML objects.

7 Deployment of ETEL++

This section presents the algorithms enforcing the deployment of ETEL++. We first describe the workload information used to determine the current burden placed on the various servers that can be used to build personalized editions. We also describe the ways this information is used. We then summarize our conclusions regarding the mining of user profiles, which appears as a promising technique when multiple co-localized servers are used.

7.1 The Basic Trade-Offs

As presented in the previous sections, a document managed by ETEL++ can either be an Active Document, or a Passive Document. An Active Document is a fairly large object while each Passive Document is smaller. An Active Document, however, is able to generate all possible Passive Documents by transforming itself into the appropriate representation(s). If several Passive Documents are generated by the Active Document, it is likely that, in general, the total size of all Passive Documents that have been generated exceeds the size of the active Document itself. For one Document, it is therefore cheaper to send the Active part of the document if several documents have to be generated. Generating Passive Documents from Active ones consumes some CPU power. The machine that receives Active Documents uses therefore more CPU-power to transform these documents into Passive Documents than another machine that would receive pre-calculated Passive Documents.

It is therefore possible to isolate the following trade-off: CPU-consumption can be traded-off against network-consumption by sending either Active Documents or Passive Documents. This trade-off is at the root of the deployment of ETEL++, as detailed in the sections below.

7.2 Workload of Servers

There are two major sources of costs when creating personalized editions. The first cost is due to the numerous transfers of data between sites. This cost is somehow purely related to the performance of the network, and the current load of links between sites might hurt performance.

The second cost is due to the transformation of Active Documents into their final representation that fit users terminal. These transformations are somehow purely related to the CPU power of each site, since transformations are CPU-intensive calculations. Therefore, the two dominant costs for ETEL++ are related to the (i) network workload and (ii) the CPU workload of each site. These two criterion are consequently the ones used for determining the best routes for the flows of data, and also for determining where transformations should take place.

CPU and network load information are obtained by querying the performance monitor that monitors the usage of resources at each site. As explained in the documents related to the Service Deployment workpackage, several types of resources can be specified and observed by a Performance Monitor. In the case of ETEL++, only two resources are defined: the CPU and the network links. The observation type set on these resources is aperiodic, that is, the current amount of the resource consumption is determined and immediately returned upon request. Together with the current values of the CPU and network load, it is possible to ask the performance monitor to return the maximum load the CPU and the network can absorb.

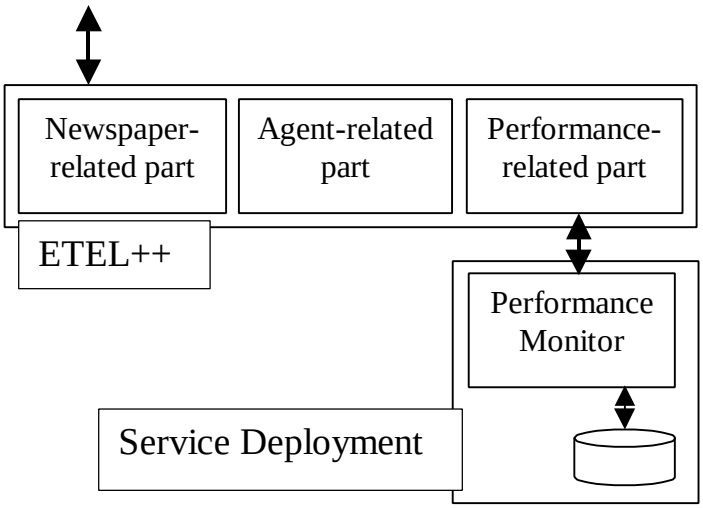

Figure 7: Internal Architecture of ETEL++ Servers

Comparing these maximums to the current values makes possible to determine the load percentage of each resource. If a CPU load factor is above a given threshold, then the corresponding machine is said to be CPU bounded. In this case, ETEL++ considers this machine has not enough CPU power to transform articles into the final representations required for user delivery. The same also applies for the network. If the available bandwidth between two machines is scarce, then it is undesirable to transfer between them a large volume of data. In this case, ETEL++ considers the machine at the end of the link to be network bounded.

A machine can be solely CPU-bounded, solely network-bounded or both. In the first case, this machine will not trigger the transformation of Active Documents into Passive ones, but will rather try to obtain a copy of the relevant document after having asked another non-CPU-bounded machine to do the transformations on its behalf. In the second case, a network-bounded

machine will rather get a copy of Active documents, and subsequently generate all the transformed versions to match users's expectations. ETEL++ assimilates the case of a machine that is both CPU- and network-bounded to the case of machines that are solely CPU-bounded. It is also possible that a machine is not bounded at all. In this case, ETEL++ sends active documents, since this tends to minimize bandwidth consumption.

Before initiating the computation of the data flow map, all servers (the main ETEL++ server as well as all secondary servers) can therefore be classified in NO-BOUNDED, CPU-BOUNDED or NETWORK-BOUNDED.

Servers that are in the NO-BOUNDED class receive by default Active Documents. They are supposed to have enough CPU power and enough network resources to create the personalized editions for their users in the relevant formats. Such a site may possibly forward to other sites either a subset of the Active Documents it received, or a subset of the Passive documents it generated.

Servers that are CPU-BOUNDED receive from other servers the Passive version of the documents they need for their users. This means another server (that is not bounded) gets the active articles each bounded server would have needed, and performs the transformations each bounded server would have performed. Once the transformations performed on the behalf of each CPU-bounded server are complete, each bounded server fetches from the relevant (unbounded) remote server the Passive Documents this server pre-computed, and delivers the fetched documents to the end users associated to the site. A CPU-BOUNDED site may possibly forward Passive articles to another CPU-BOUNDED site.

Servers that are NETWORK-BOUNDED receive Active version of Documents since we assume that the size of one Active Document is smaller then the global size of all the corresponding Passive Documents (our experiments proved this was generally true). Such a server, however, can not send any data to another server since it is bounded.

When a server receives some Documents (Active or Passive), its network consumption is incremented to reflect the new load that is imposed on the links. It is therefore possible that after having decided a specific non-network-bounded site will receive data, this site becomes network-bounded. The network consumption of the sending site is also incremented (it may therefore become bounded as well).

When a server transforms Active into Passive Documents, its CPU load is also incremented to reflect the new load imposed on the processing unit. As for the network case, it is possible that a non CPU-bounded machine becomes CPU-Bounded once it has been decided that this machine will transform Documents.

Therefore, non-bounded servers may become bounded as a side effect of the decisions that are taken when deciding where articles are transformed, and what kind of data is transferred between sites.

The policy that drives the deployment of ETEL++ is based on this mechanism. This algorithms runs at the main ETEL++ server. It needs 3 steps:

1. All servers (the main server itself and all secondary servers) are contacted and asked to return their workload information. The information that is returned is a tuple containing the current consumption of the CPU and the network and the maximum load for both.

2. For all the servers except the main ETEL++ server: if sending active documents to this server would make it CPU-bounded, then this server is inserted in a special list keeping all servers that can not transform active documents. The least network-loaded server (the ratio current-load / max-load) is searched. This server is the source while the current server in the loop is the destination. Active articles will be sent from the source to the destination. The network consumption of the source server and of the destination server are incremented, the CPU consumption of the destination server is also incremented.

3. For all the servers that are kept into the special list, find the server that is both the least CPU-loaded and the least network-loaded. This server will act of the behalf of the bounded server, that is, it will receive the active articles the bounded would have received (except those the server already has) and transform them as the bounded server would do. Then, the server would send the transformed articles to the bounded one. CPU and network consumptions are also incremented during this step.

After the completion of this algorithm, the main ETEL++ server possesses a map of the data flows between servers. This map takes into account the actual load that is imposed on servers, and therefore enforces a deployment of ETEL++ that is performance aware. Different loading factors will result in different maps, in which the flow of data will not be identical.

This feature will be demonstrated.

In the current implementation of ETEL++ (early March), the complete Service Deployment framework for defining the characteristics of observations is implemented and operational. It is actually used by the prototype that will be demonstrated. Querying the hardware resources, however, is not complete mainly due to the complexity and lack of clarity of the interfaces available on Windows platforms (the ones we currently use for development). This missing part consists of the lowest levels of the Service Deployment work.

Indeed, ETEL++ is today using the load-balancing policy described above. This policy exploits the results delivered by the Service Deployment workpackage. The detection and the observation of values is performed by the analysis of configuration files, and not however by querying real hardware resources. The observation framework, however, completed last summer, is fully operational, as detailed in the document DG3. The core of the Service Deployment is ready, that is, tools for performing periodic observation and for enabling general definition of what is a resource and how to observe it (indeed, the configuration files we use *are* a resource). The

validity of what we already developed is demonstrated by the policy that actively runs inside the Pilot application and that drives the flows of data.

7.3 Mining User-Profiles

In general, Data Mining Techniques are used to discover hidden relationships between data items. The discovered relationships are traditionally used to enforce for example new marketing strategies. In the case of ETEL++, Data Mining Techniques were thought to be applicable to user profiles, in order to discover hidden similarities between profiles, and then to use this valuable knowledge to enforce Quality of Service requirements. Indeed, Data Mining techniques were thought to be used both in ETEL and ETEL++.

Applying Data Mining techniques to user profiles in the environment of an electronic press service was the research topic of a (completed) Ph. D. work. This thesis motivated the need for data-mining, implemented new mining and clustering algorithms and evaluated their performance. The basic idea investigated in this work is to determine, from the mining of profiles, group of users (in this case readers) that have similar, or close, center of interest. Once user groups have been determined, another algorithm presented in the thesis assigns groups on each server of a cluster of servers running at Ouest-France in an optimal way. As a result, servers better cope with scale, since mining and clustering together tend to maximize the buffers hit ratio when servers service data pages to users.

The architecture of ETEL++, however, relies on multiple single-servers, instead on clusters of servers, as ETEL. Grouping users on servers is therefore more delicate. In particular, we found that the geographical location of a user was a dominant factor. That is, it was generally more efficient to generate the newspaper *close* to users that need it, instead of generate it on a *distant* server on which mined groups were assigned. Therefore, the generation of ETEL++ editions is driven by the actual location of users, while ETEL relies on mining and clustering techniques.

8 Generation of Editions and the Assembling Engine

We outline in this section the process used to build personalized editions. We first restate the context in which personalized editions are built. We then present how we produce articles, rubriques and front-pages that are ready to go to end-users.

8.1 Prerequisites

Each ETEL++ server is possibly in charge of building personalized editions. Determining which servers have to build editions is the role of the deployment algorithm that uses the workload of each machine for that purpose. Some servers build editions solely for the readers that have been identified as serviced by that server. Some servers build editions for themselves, and also for other servers that are bounded. In this case, each building server asks the relevant bounded servers specific information that is merged with their own. They therefore generate editions given an extended information set, and then forward the documents they have generated on behalf of others to the relevant servers. Last, some servers do not generate editions but simply wait to receive editions that have been generated by another server.

A server knows which articles it needs after having analyzed its readership. The readership of a server consists of a subset of all existing user profiles. This subset is determined at run-time, and the location of the next connection of users is the criterion used to identify which profiles are “attached” to a server. Profiles give, among other things, the center of interests of a user expressed by the means of keywords. There are thematic and geographic keywords. A profile contains also information about the device that the user is likely to use for the next connection.

Keywords, used together with the inverted lists of articles, make possible to precisely know which articles a server needs to satisfy its readers. Therefore, when a server works on behalf of other bounded-servers, it asks those servers for their readership, and merges those readerships with its own. Once personalized editions have been generated, such a server is able to know, using again the keywords obtained from other servers, what it needs to send them.

To summarize, at the time a server is fully capable of generating personalized editions it knows (i) the keywords of its (possibly extended) readership, (ii) the set of corresponding articles, (iii) the final formats of the editions and (iv) the articles it may have to forward to other bounded servers.

Generating personalized editions is a complex process that can be decomposed in three main parts. The first part consumes articles and generates their final versions. The second part also consumes articles, but generate summary pages that we call Rubriques. Last, the third part consumes Rubriques and Articles, and generates personalized front-pages that are private to each individual user.

8.2 Preparing Final Articles

When a server initiates the generation of editions, it first gets the Active Articles it needs. These Active Articles are obtained from the server that is indicated by the deployment algorithm (it may be the Main ETEL++ server, or any other non-bounded server as well). It then stores a copy of these Active Articles in its local Information Space. It starts a loop that reads-in each Active Article and applies the treatments that follow.

For one Active Article, the inverted lists are queried in order to know the formats in which the current Article is likely to be browsed. For each final format, the Active Article is asked to produce the relevant version. The produced versions are stored in the local information space of the server for later delivery to users or to other servers. Versions are produced with respect to a complex set of rules that are partially exposed in the next section.

8.3 Preparing Final Rubriques

Once all Active Articles have been transformed, Rubriques are produced. To produce Rubriques, the algorithm uses the inverted lists of keywords built from the keywords found in the readership associated to this server (there are thematic keywords, and also geographic keywords). For each keyword, an Active Rubrique is created. In addition, all the Active Articles that contain the current keyword are identified. The data of the new Rubrique is deduced from the analysis of the titles of the Articles, and references to these articles are set in the Rubrique. In addition, the Rubrique is inserted into a special inverted list of Rubriques in which a Rubrique and the keyword that triggered its creation are associated.

Once the Active Rubrique has been created, it is converted into the appropriate formats. Since a Rubrique references Articles, it is possible to know the formats in which a Rubrique has to be converted by checking the formats in which the articles the Rubrique references have been converted. Rubriques are, as Articles, stored in the local information space of the server.

8.4 Determining the Main Articles and Rubriques

Once all Articles and Rubriques have been generated, the server accesses the inverted lists to determine what are the articles Ouest-France wants to see on the front-page of each personalized edition. This is an editorial constraint, that is, irrespective to any possible keyword any user has set, Ouest-France forces the system to display several articles on the front-page. Achieving this is made possible by the joint use of the weight set on articles and of special, “invisible” to users, keywords.

These special keywords are used to isolate the corresponding articles, and also the corresponding rubriques, if any. The isolated documents are further processed. They are sorted with respect to their weight, and it is possible that their final representation is patched to reflect their crucial importance to Ouest-France (like changing the font-style they use to enforce bold character sets). A list on these articles is built.

8.5 Preparing Final Front-Pages

After having prepared all Rubriques, the Server processes every user profile. For one profile, the server determines the name of the user, the keywords the user has set and the adequate delivery format. It then creates an Active Front-Page. Once created, the server loops on all the keywords the current user has set and queries the list of main articles to determine the ones that will be inserted in this user’s edition. The new created front-page references some of the main articles, and references also the Rubriques that corresponds to the keywords set for that user. The references that are inserted in the front-page are independent of the actual format of the final delivery. The front-page references Documents, and the adequate passive version (for the web, for a mail, …) is chosen at connection time. Front-pages are stored in the local information space of the server.

8.6 Forwarding Editions to Bounded Servers

Because some servers may be performance-bounded, they do not generate or assemble editions, but rather receive pre-computed data from another server. In this case, the server that computed the edition on the behalf of some other bounded server(s) contact these servers and informs them that their editions are ready. In turn, bounded servers directly access the local information space of the server that has performed the computations, and copy the relevant data in their local information space. To optimize data transfers, they copy only the Passive piece of data that they ultimately need to deliver to their own users. In the case of a discrepancy between what has been pre-computed and what the users actually wants (for example, the diary specified FAX, and the current connection is HTML), the server is capable of contacting the one that computed the version, and may ask for more processing.

9 Coping with Multi-terminal Editions

We outline in this section the way we produce editions that can be displayed on multiple terminals. Producing multi-terminal editions is possible because ETEL++ uses the features of User Access provided by FAST. The User Access workpackage provides means for defining the rules to enforce during the transformation of XML data to terminal-specific data, and means to instantiate connections.

The current prototype of ETEL++ supports two different types of terminals: web-based terminals and Mail-based terminals. We therefore present in the next sections the use of FAST software, and then the processing tasks that produce editions for such media. We also briefly outline how additional media could be incorporated into ETEL++.

9.1 Using User-Access

To make possible the production of multi-terminal editions, ETEL++ documents implements the Document interface defined in the *DE.fast.followme.useraccess.Document* java file. As described in the User Access documentation, this interface requires the implementation to provide several methods that (i) return what Mime types are supported, (ii) return the representation of a document given a mime type and (iii) return the documents that are referred by a given document. In the case of ETEL++, the supported mime types corresponds to editions for HTML, Mail and XML (this last type is not delivered to users, but can be used to extend the number of devices that are supported).

As mentioned Section 4, only the identity of ETEL++ Documents is propagated thorough the application and therefore only ArticleDocHandles, RubriquesDocHandles and FrontPageDocHandles need to implement the interface provided by FAST. It should be apparent that ETEL++ relies on the pre-computation of electronic versions. Therefore, the documents in the appropriate format that have to be delivered to a user are generally available before the connection is opened. It is possible, however, that the mime type of a newly opened connection differs than the one specified via the setting in the Profile. In this case, ETEL++ has to compute on the fly the version to deliver. This is always possible, from any server, because the Active

version of every document is always accessible at least on one site, and because every ETEL++ document that have traveled between servers always maintained a reference to its active part (the Active Object), even if it is stored on a remote server.

If the user requests a document in a format that was not planned *a priori* for delivery, our implementation of the interface provided by FAST will access the (possibly remote) Active version of the requested document, and will enforce its transformation given the mime type in argument. The processing that is triggered is identical to the one that would have been executed if the transformation into this particular media would have been planned. As a result, a new Passive object is created, and its external representation returned to the user. If the active object is remote, the response time for building on-the-fly the appropriate passive object may be time consuming.

9.2 Web-Based Editions

Editions built for Web-base terminal can be decomposed in two parts. The first part is made of a *style sheet* in which the look of the editions is specified. Style sheets rely on the Cascading Style Sheets (See http://www.w3.org/Style/css/). This style sheet makes possible the definition of precise, elaborated rendering that is independent of the data to render. 23 different styles have been defined for the rendering of ETEL++ web-based editions. The following presents the file that is actually used.

```
BODY {font-family: serif; background: white;}
A:link {color: blue;}
A:visited {color: purple;}
.logo{
      border-width: 0;
      padding-top: 0.0cm;
      padding-bottom: 0.3cm;
      color: white;
      text-align: center;
}
.logoUne{}
.lienTitreArticlePrincipal{
      font-size: medium;
      font-weight: 100;
      font-style: bold;
      color: red;
      padding-bottom: 0.5cm;
}
.lienTitreArticleUne{
      font-size: medium;
      color: black;
      padding-bottom: 0.3cm;
}
.lienTitreGeographieUne{
      font-size: medium;
      color: green;
      padding-bottom: 0.3cm;
}
.lienTitreRubriquePerso{
```

```
	font-size: medium;
	padding-bottom: 0.3cm;
}
.titreArticlePrincipal{
	padding-left: 0.2cm;
	text-decoration: underline;
	padding-top: 0.2cm;
	padding-bottom: 0.8cm;
	color: white;
	font-size: x-large;
	font-weight: 600;
	text-align: left;
}
.titreArticle{
	padding-left: 0cm;
	text-decoration: none;
	padding-top: 0.1cm;
	padding-bottom: 0.8cm;
	color: black;
	font-size: large;
	font-style: bold;
	font-weight: 900;
	text-align: left;
}
.titreRubrique{
	text-decoration: none;
	color: gold;
	padding-top: 0.5cm;
	padding-bottom: 0.5cm;
	font-style: bold;
	text-align: center;
	font-size: large;
}
.titreArticlesDansRubrique{
	text-decoration: underline;
	font-family: arial;
	width: 100%;
	padding-top: 0.3cm;
	padding-bottom: 0.3cm;
	padding-right: 0.3cm;
	padding-left: 0.3cm;
	font-size: large;
}
.chapeauArticleDansRubrique{
	padding-left: 0.5cm;
	padding-right: 0.5cm;
	padding-bottom: 0.3cm;
	text-align: justify;
}
.cadreTitreRubrique{
	width: 100%;
	background: lightgrey;
	border-width: 1;
	  border-color: white;
}
.cadrePhotoArticle{
```

```
      width: 200;
      padding-left: 5;
      padding-right: 5;
      padding-bottom: 5;
      padding-top: 5;
      align: right;
}
.cadrePhotoArticlePrincipal{
      width: 50%;
      padding-left: 15;
      padding-right: 15;
      padding-bottom: 15;
      padding-top: 15;
}
.cadreArticlePrincipal{
      width: 110%;
      background: slategray;
      color: white;
      border-width: 1;
      padding-top: 15;
      margin-top:2cm;
      border-color: white;
}
.resumeArticlePrincipal{
      text-decoration: none;
      font-family: arial;
      color: white;
      padding-right: 0.2cm;
      padding-left: 0.2cm;
      font-size: 110%;
      float: left;
      width: 45%;
      text-align: justify;
}
.legendeArticlePrincipal{
      width: 250%;
      text-decoration: none;
      font-family: arial;
      color: white;
      padding-left: 0;
      text-align: left;
      font-size: medium;
      font-style: italic;
}

.photoArticle{}

.legendeArticle{
      text-decoration: none;
      font-family: arial;
      color: black;
      padding-left: 0.1cm;
      text-align: justify;
      font-size: small;
      font-style: italic;
}
```

```
.chapeauArticle{
	text-decoration: none;
	font-family: arial;
	color: black;
	padding-right: 0.3cm;
	font-size: 90%;
	float: left;
	width: 75%;
	text-align: justify;
}

.texteArticle{
	width: 100%;
	font-size: 100%;
	text-align: justify;
	text-indent: 0.5cm;
	padding-top: 0.8cm;
	padding-right: 0.5cm;
	padding-left: 0.5cm;
}
.surtitreArticle{
	font: 150%/150% sans-serif;
	color: black;
	text-align: left;
	padding-left: 0cm;
	padding-bottom:.5cm;

}
.surtitreArticlePrincipal{
	font-size: 120%;

	font-family: sans-serif;
	color: black;
	text-align: right;
	padding-right: .5cm;
	padding-top:1cm;
	padding-bottom:.1cm;
}
.aLaUne{
	position: absolute;
	top: 85px;
	left:7px;
	color: yellow;
	text-align: left;
	white-space: pre;
	background: slategray;
	padding-left: 5;
	padding-bottom:.1cm;
	font-size: x-large;
	font-family: arial;
	font-style: italic;
}
```

9.3 Mail-Based Editions

When ETEL++ constructs editions for Mail-based terminals, images and not delivered. Only the title of the main articles together with their summary is delivered, and no article body is sent. We could have designed another type of delivery, but we decided to limit ourselves to simple ASCII and to a small amount of data.

9.4 Supporting Additional Media

To support an additional media, it would be necessary to define a new set of Passive Documents (Articles, Rubriques and Front-Pages), and to connect this set to the existing hierarchy of Passive Documents as presented Section 4. It would be necessary to implement an additional method in Active Objects that would be responsible for producing the Passive data from the Active one, given certain layout rules. It would also be necessary to modify the implementation of the FAST interfaces in the ETEL++ Documents to deliver the appropriate external representation given the new mime type taken into account.

The new objects compatible with this new device would be transparently taken into account by the workload-aware algorithm, and would travel between servers as other objects do.

10 Using Agents to Obtain Context-Sensitive Data

The notion of context-sensitive data has been presented in the Document DJ2. In a nutshell, context-sensitive data refers to the ability to deliver information whose content differs depending on a specific context, such as the physical location of the user. With context-sensitive data, a user can have access, via the newspaper, to the current list of movies that are played in the town from which he connects ETEL++. Obviously, this list differs if the user is in Paris or London. Accessing weather forecasts is another straightforward application of context-sensitive data: instead of always including the weather of a fixed place, the electronic newspaper displays the weather that corresponds to the actual location of the user.

The Agent Framework provided by UWE makes possible the definition and use of Agents together with Mission Scripts. ETEL++ uses both. Agents are interesting in our context because of their autonomy, their goal directed behaviors and their asynchrony. Missions are interesting because they are simple to write and also because they are interpreted. It is therefore easy to change the behavior of a mission without recompiling the Agent Java classes that execute the mission or the ETEL++ Java classes that set the appropriate parameters or make use of the mission's results.

We present in this section the way ETEL++ uses agents and Missions to extract weather forecast information from real Internet data-providers. We first present our specific objects that encapsulate the behavior of real Internet Weather Forecast providers. We then detail the context that had to be created in the Agent Framework trader. We then are present several missions that we wrote.

To keep our code simple, we solely focused on context-sensitive data that are compatible with Web-browsers. Therefore, the weather forecast is available only to the users that set HTML as their type of media. A message saying the information is not available is returned for other terminal types.

10.1 Encapsulating Internet Weather Forecast Providers

In order to return to users the weather that correspond to the area from which they connect, ETEL++ needs to interact with real Weather Forecast providers that are on Internet. There is a great number of such providers, but most of them can be classified as follows:

- A first type of providers does deliver forecast for a single location. The URL that has to be used remains the same from one day to the other, but the information that is accessible from that URL is often updated daily. For example, http://www.meteo.fr/temps/index.html works this way.
- A second type of providers does deliver forecast for multiple locations, however their URL is fixed and a cgi-script requires additional parameters before constructing on-the-fly the correct page. For example, the forecast service of the Washington Post (see http://www.weatherpost.com/) works this way. This unique entry points requires the user to enter the name of a town, feeds this name to a script and displays the resulting page.
- A third type of providers does also deliver forecast for multiple locations, however, the URL to use to get the forecast for a specific location is directly constructed from geographical information attached to the location. The Yahoo weather forecast service works this way. To get the forecast for Rennes via Yahoo, the URL to use is http://weather.yahoo.com/forecast/Rennes_FR_c.html, while for Munich it is http://weather.yahoo.com/forecast/Munich_DL_c.html.

Each provider has therefore to be queried in a specific way. To facilitate the use of agents and Missions, we created a particular class devoted to encapsulate Internet Weather Forecast Providers, and that exhibits a simple, unique and uniform interface for access. This class hides the way input parameters have to be manipulated in order to query each provider in the appropriate way.
The interface of this class is:

```
package fr.inria.Etel.Agents;

public interface WeatherInternetProvider {
        // Displays in a web-browser the site that gives the weather forecast
        public void display();
        // Return a string representation of the URL of the weather forecast site
        public String returnURL();
        // Set appropriate info for latter access
        public String setGeoInfo(String worldZone, String country, String countryCode,
                                 String townName, String degreeUnit);
}
```

We manually identified several Internet forecast providers and created instances of the above object. There are *at least* as many instances of these objects as towns for which ETEL++ may return the weather forecast. Note that automatic identification of such sites is not yet possible since these sites often lack of meta-data describing their role. In addition, determining the ways input parameters had to be manipulated for preparing appropriate queries could not be automated. For example, we had to analyze the source code of several web pages in order to determine what were the relevant variables to set before invoking the cgi-script (see the source code of the page http://www.weatherpost.com/).

We also created a specific Information Space in which these objects are stored as White boxes. We can therefore access these objects any time, from any site.

10.2 The Contexts

To use agents, we had to define a specific context in the UWE trader. This context has particular properties describing the purpose of the context (here Weather forecast). We inserted all the objects mentioned in the above section in this context. Each object inserted is an offer, according to UWE's vocabulary. Therefore, when inserting an object, we were also setting specific properties to make possible the selection of the appropriate offer(s) when searching for a particular offer fulfilling a specific request. The main property we set is the town name, making therefore possible to determine the set of objects that encapsulate all internet forecast providers that returns the weather for that town.

The context defined to keep ETEL++ specific information about forecast needs stores in addition information about missions, as presented below.

10.3 Missions

Missions are interpreted scripts consumed by agents and that define their goal. In our context, the missions have the general goal to determine among the possible objects the one that should be chosen to deliver the required information. All missions need a town name as argument (we assume no duplicates). Each mission accesses the trader of UWE, gets a set of offers that match, determine which object from this set is to be used, and return the interface of that object to the mission launcher.

Missions are interesting to ETEL++ because interpretation makes possible simple, fast prototyping of new selection criteria without touching any part of the application. We therefore wrote several missions that we present now:

1. Availability-aware mission. The first type of mission we implemented was selecting the site to use with respect to an availability criterion. Each element of the set is accessed, and if no answer is returned after a time-out ring, the next site is tried.

2. Performance-aware mission. The second type of missions launches in parallel several HTTP requests to get the data referenced by each URL. It returns the interface of the object associated to the first request that answered, others are killed.
3. Content-Aware missions. We are currently trying to implement a simplistic parsing of the data that is returned by de-referencing each URL to check for the time the associated information has been updated, and we keep the most recent update while throwing away others. This tries to return the most up-to-date weather forecast information. Our in-progress implementation is complex because the way sites specify updates differs greatly from one to the other, and lacks sometimes. This third type of missions can not therefore work for any arbitrary site.

Missions are registered in the trader of UWE together with a short description (like: “return URL on internet site giving weather forecast, given a town name”). In addition, the specific role of each mission is specified as a property.

Today, which mission has to be used is decided when the ETEL++ bootstraps its daily construction of editions. Therefore, before editions get assembled, each server discovers by analyzing its keywords that it has to deal with context-sensitive data. It creates an agent place, initiates an agent and gives the mission name to execute. Once the mission is over, it includes the URL that can be returned by the interface of the identified object in the HTML pages of the relevant users.

It is possible to implement other mechanisms in which the type of mission to execute could be decided by a user. A user requiring the most up-to-date information could check a box in its profile, triggering a specific treatment. This is currently investigated.