A Distributed, Self-Adaptive Model of Hypermedia System

Antonina Dattolo, Vincenzo Loia
Dipartimento di Informatica ed Applicazioni,
Università di Salerno, 84081 Baronissi (SA), ITALY.
e.mail: {antos,loia}@dia.unisa.it

Abstract

In this paper we discuss a distributed user model architecture suitable for hypermedia systems. In this architecture, the user knowledge is spread over a collection of autonomous actors, cooperating in a way which provides, through collaborative activities, a global, dynamic user model. Separate duties and knowledge, combined with asynchronous and concurrent processing, augment the adaptivity and the efficiency of the platform, avoiding the drawbacks of more traditional centralized approaches. Our goal was to enrich an existing hypermedia architecture with an appropriate user model activity, conceived and realized with the same design perspective of the underlying system, the actor-level design approach. The organizational structure has been defined by extending a previous hypermedia system model without redesigning the overall architecture. The prototype has been tested with success as an adaptive hypermedia interface in a logic object-oriented programming environment.

1 Introduction

The design of adaptive hypermedia systems (AHS) requires abilities to meet users' expectations at runtime; this need is becoming a crucial issue in more recent applications since the current technology provides an increasing amount of available electronic information and a more intensive integration of different media. The fact that nowadays the World Wide Web (WWW) [3] has shown hypermedia's potential to human society has fostered :

- the creation of new information repositories that are physically decentralized;
- the development of new information assistant strategies that support the user in exploring the jungle of repositories.

The second trend includes the research activity concerning the study of adaptive systems; most of the efforts accomplished by the scientific community have been spent in proposing metrics able to evaluate user cognitive state [29], in defining different forms of adaptivity (presentation and navigation) [6], in using normative user models (overlay [13, 22] and stereotype [30, 18] models) or prescriptive user models [21], in deepening the problems of orientation and comprehension [31], in implementing user-oriented querying assistance [1]. Additional work is necessary in order to define effective and general architectures able to support the previous methodologies. The usual trend is to separate the user model package from the hypermedia environment [22], centralizing the intelligence of the user model in a unique specialized module. The fact that information is disseminated over a web collides with this design approach, stimulating the investigation into new distributed hypermedia environments provided with distributed user modeling facilities. Our work is framed in this direction; its focus is to propose a new and more general way to conceive the adaptation process in hypermedia-based information systems. Principally two aspects differentiate our model from the other existing approaches:

- the user model and the related functionalities are completely modularized and distributed in the system.
- the process of collecting and analysing user behavior occurs during the hypermedia browsing activity and it is automatically exploited for new customized browsing.

The proposed framework is viewed as an "Open System" [16], an environment in which a continual flow of new information is originated from numerous actors. Actors benefit from massive concurrency; they own local data (acquaintances) and perform functions (scripts) concurrently. The decentralization of knowledge and tasks does not prevent the actors from managing global actions; in fact, designed collaborative duties may be adopted to coordinate their intentions. We adopted the Object-Oriented Concurrent

Proceedings of The Thirtieth Annual Hawwaii International Conference
on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

1060-3425/97 $10.00 (c) 1997 IEEE

Programming (OOCP, in short) paradigm to design in high-level fashion complete decentralized systems; it provides a useful organizational layer to develop open systems [5]. The main computational entity at the basis of this programming metaphor is the "actor" [2].

The results of this work are part of a larger project which is based on the study and realization of complex software systems within distributed architectures [9, 11, 14]. Recently, our attention has been focused, on one hand, on the problem of improving the design phases of distributed hypermedia architectures [10, 12] and, on the other hand, on distributed models to support user model management [25]. In this work we present results motivated by an attempt to unify the previous two goals in a unique framework.

The paper is organized as follows. Section 2 describes some issues of the user modeling in AHS. Section 3 introduces the architecture of our hypermedia system, highlighting the organization structure and the cooperation schemes. The software architecture of the developed prototype and its use in object-oriented logic programming are described in Section 4. Section 5 describes more in detail some actor categories, and illustrates the dynamic user recognition process. Concluding remarks summarize the advantages of the approach and some aspects that require further investigation.

2 Adaptive Hypermedia Systems

AHS are a new direction of research; it has become increasingly popular in the last five years. AHS are systems designed to learn by being used and to increase the functionality of the hypermedia by making it personalized. However, some recent work demonstrates that AHS techniques can be applied in a number of other application areas, as for example information retrieval hypermedia [19] or personalized information spaces [32]. Essentially, adaptivity is based on the accomplishment of two basic tasks:

- memorization of several protocols about (possibly) all occurring actions.
- analysis of these protocols and extraction of new user habits/preferences.

The learned knowledge is then applied in order to modify situation-dependent interfaces and information nodes.
Initially, AHS appeared as adaptive graphical interfaces able to support simple but frequent operations such as UNDO/REDO strategies, active help, predefined plans-of-actions schemes. With the evolution of hypermedia models, and with the enormous diffusion of hypermedia applications in different social and professional classes of users, the need of more efficient adaptive methods is now strongly required. Fortunally, adaptive hypermedia community has benefited from the long experience acquired by research on user (or student) modeling, especially in the field of artificial intelligence in education. The efforts accomplished in this area have been extremely useful to solve the difficult task of user's knowledge acquisition but minor attention has been paid in studying the possible implications on the hypermedia design architecture task. This was our main starting point: in fact, the work herein reported should not been considered as a new user modeling method designed for hypermedia applications but as a specialization of an open hypermedia model for adaptive user interaction.

3 Actor-based Model of Hypermedia

Our framework of hypermedia is fully described in terms of autonomous and distributed actors. Each actor is a computational object living on autonomous knowledge and duties in a distributed, concurrent environment. Designed cooperation activities may arise when the need to attain a global goal stimulates the different actor categories to collaborate. The actor model adopted here extends the "classical" point-to-point actor communication protocol introducing more general and powerful schemes such as multicasting and broadcasting. A first attempt to define such actor-based hypermedia architecture appeared in [9]; recently a more complete description of this model and its applicability in a CSCW (Computer Supported Collaborative Work) target has been discussed in [12]. Here we are interested in extending the original architecture towards adaptivity. The primitive entity of our model is the actor; it serves as a small container of hypermedia information. The different media are stored in appropriate local resources (acquaintances), that are managed by local built-in scripts. Some of these scripts accomplish "social" tasks, such as the link management.
In this section we provide the general framework that is graphically represented in Figure 1. We characterize the different actor populations, highlighting their role in the hypermedia structure and their adaptive capability.

- *Structural Layer.* This section of the hypermedia architecture corresponds to the architectural model provided by the authors of the hypermedia. It is composed of two populations of actors, the *HypActors* and the *Collectors*. Although

Proceedings of The Thirtieth Annual Hawwaii International Conference
on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

1060-3425/97 $10.00 (c) 1997 IEEE

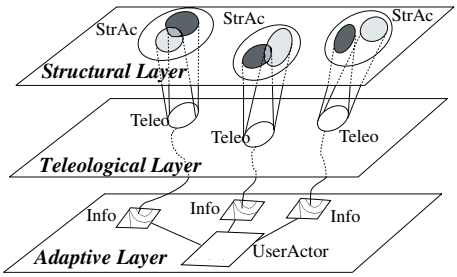

Figure 1: The hypermedia architecture.

these two actors are different in structure and behavior, they serve to compose the global organizational level of the hypermedia. Referring to the classical literature on hypermedia, the HypActors and the Collectors could correspond to atomic and composite nodes; the HypActor entity presents characteristics normally contained in nodes and links of classical hypermedia models (notecards, frames, nodes, slices [15, 17]). The Collector handles collections of HypActors and is useful to maintain and compose hypermedia views during browsing and query activities. The union of HypActors and Collectors represents the most general, complete user perspective of our hypermedia system. From now on, we use the term *StrActor* to indicate a generic object which belongs to this union and exists in the structural layer.

- *Teleological Layer.* This layer is composed of the population of *TeleoActors*. The role of this category of actors is to provide all possible dynamic user perspectives of the hypermedia. This task is carried out in a distributed and collaborative way. For this reason, each TeleoActor is specialized in the utilization of a unique StrActor and in the interfacing between the data/services provided by the StrActor and the user, according to the evolution of the preferences shown by the user during the browsing activity. The ability to shape the functionality of the StrActor is given through cooperation with the Adaptive layer (see below); in fact, as we will discuss in the next sections, the knowledge about the user is acquired by an external entity, the InfoActor. More precisely, this means that for each Stractor there are a unique TeleoActor and a unique InfoActor.

- *Adaptive Layer.* This level is composed of two actor populations. The *InfoActors* work as independent monitors of user behavior, in order to maintain a distributed, local user monitoring active on the net. Each InfoActor keeps a trace of the user activity and constitutes the main source of information useful to the TeleoActor to define wich view must be applied on the related StrActor.
  The *UserActor* plays the role of coordinator of the various InfoActors. Essentially, the InfoActors inform the UserActor, in a parallel and asynchronous way, about all the observations reported. The UserActor collects these different user profiles and recognizes a relevant user state; in this case, the UserActor communicates the new user behavior to the InfoActors.

We deepen our discussion by providing details about the actors cooperation activities.

3.1 Collaboration Activities

- Primitive collaboration: $StrActors \Leftrightarrow StrActors$.
  The framework of our hypermedia model is completely distributed. All the data and the services are spread over a web of autonomous StrActors. The structural and behavioral links between such entities correspond to a cooperation protocol defined between the StrActors. Key issues of hypermedia management, such as the versioning mechanism [8], are accomplished thanks to a concurrent problem solving strategy [9].

- Teleological collaboration: $StrActor \Leftrightarrow TeleoActor$, $InfoActor \Rightarrow TeleoActor$.
  The bridge between the hypermedia architecture and the user goals characterizes the Teleological layer. The ways by which the user interacts with each hypermedia node are provided by the scripts of the corresponding TeleoActor. Essentially, each TeleoActor furnishes various filters, which are made active by the user behavior. Each filter modifies the functionality of the related StrActor, by pruning/adding hypermedia views. This task is accomplished thanks to a collaboration activity between each couple of StrActor and TeleoActor. An important constraint to force this cooperation is given by knowledge about the user behavior which is acquired by the related InfoActor. This knowledge constitutes the basic information in the TeleoActor useful to define the filter to apply on its StrActor.

- Adaptive collaboration:
  $StrActor \Leftrightarrow InfoActor$, $InfoActor \Rightarrow UserActor$, $UserActor \Rightarrow InfoActors$.
  This collaboration level consists of three tasks.

Proceedings of The Thirtieth Annual Hawwaii International Conference
on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

The first is local knowledge acquisition about the user. This task is carried out through a collaboration activity between the StrActor and its related InfoActor. The InfoActor follows all the activity on the current StrActor and keeps a trace of the user actions. This task produces an initial knowledge about the user behavior which allows the TeleoActor to apply a first local filtering.
The second task consists in charging the InfoActors, which spy on the user browsing, to forward the most significative observations to the UserActor. This latter collects this information asynchronously. This means that, during the user monitoring, the InfoActors and the UserActor work in an independent way. The UserActor owns local duties which allow it to establish when and how the user model changes [28].
Finally, the third task occurs whenever the UserActor decides to change the current user model. In this case, the UserActor informs the InfoActors about the need to update their user perspective. This action is performed by exploiting concurrent message passing in order to improve the adaptivity of the system. As stated previously, once an InfoActor acknowledges this message, it communicates to its TeleoActor the necessity to modify the user vision of the related StrActor. This knowledge allows each TeleoActor to filter data and services on the corresponding StrActor, so that the user update occurs in a deeper way, taking into account not only the local actions performed on the single node but also the user behavior reported during the rest of the browsing.

4 Application

The described architecture has been applied in the development of a hypermedia system, named DiBlue, which is useful to support logic object-oriented programming in OPLA. OPLA [27] is a hybrid language originated from the marriage between Prolog [7] and CLOS [4]. Our experience consisted essentially in modifying the "traditional" OPLA hypermedia programming environment, named Blue [26], in order to obtain a distributed version, DiBlue [12]. In this work, we report the results obtained enhancing DiBlue with the adaptive model. Figure 2 depicts the software modules used in the realization of DiBlue.

The reader can note that the first layer (corresponding to the Blue realization) consists essentially of CLOS and CLUE [20] (this last to handle X-Windows objects), while the extension of Blue towards a distributed framework has been possible by using Hyper-

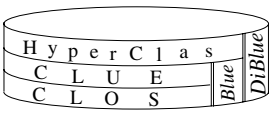

Figure 2: The DiBlue software platform.

Clas. HyperClas [9, 11] is an object-oriented language based on the top of CLOS; for this reason, its programming style is very similar to CLOS. The programmer embodies in the actor class definition all the information which characterizes the structural definition part, in the same way as CLOS programmers do for the class construct. The behavioral definition part is made explicit outside the actor construction. Essentially, the user specifies a set of external scripts which are linked to actors entities via dynamic bindings and inheritance mechanisms. The detachment between structure and behavior improves the flexibility and the efficiency of the software components. Due to a lack of space, we omitted more details about HyperClas, that can be found in [9, 11]. Here we briefly provide some features of DiBlue environment. The scenario of Figure 3 shows the situation in which the OPLA user requires information about the OPLA class `PRODUCT`. The interface in Figure 3 is organized in such a way

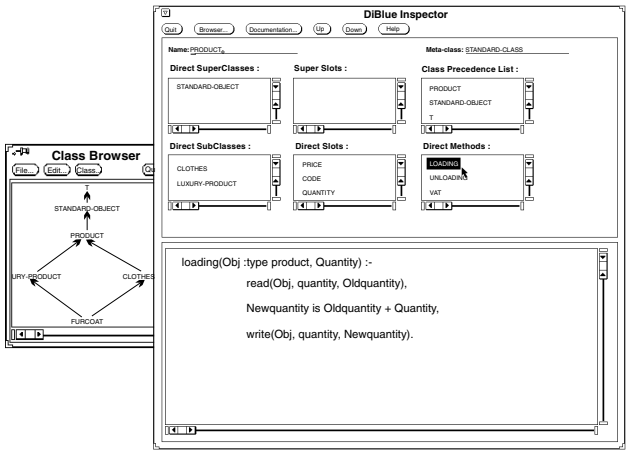

Figure 3: The DiBlue class browser.

as to provide meaningful information about the class, the superclasses and the subclasses, by specifying the data part and the procedural part, according to the inheritance mode. These data are shown in the upper pane, whereas the lower pane is dedicated to code editing. In particular, the method `loading` in Figure 3 is spotted when the user selects the method identifier in the upper `Direct Methods` area. For instance, if

Proceedings of The Thirtieth Annual Hawwaii International Conference
on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

1060-3425/97 $10.00 (c) 1997 IEEE

the user selects the *documentation mode* (see top-level banner), different class-based information is available in DiBlue; in Figure 3, on the left-most side, we show the `Class Browser` window, in which a graph provides the inheritance ordering that exists among the classes in relation with `PRODUCT`. The `Class Browser` window may change, by showing additional information, such as the slots, the methods, or both. For a sake of simplicity, in the following, we suppose that a generic user can see the class browers in only three different ways; let us denote these three views with A (only classes), B (classes with methods) and C classes with slots. In particular, the Class Browser window in Figure 3 shows the view A. In the next section, we will return back on the situation shown in Figure 3 to discuss adaptive evolution of the system.

5 Adaptivity at Work

As discussed in Section 3, the bulk of our hypermedia system is composed of five actors categories, three of them dedicated for the user modeling task. This section presents more details about the inner capabilities of these last three actor populations, providing a more formal description of them and showing part of their functionalities through a practical example in DiBlue.

5.1 TeleoActor

The TeleoActors act as an adaptive interface between the Structural layer released by the hypermedia author and the user effective utilization. The existence of these actors allows one to adapt the hypermedia functionality according to the user behavior. Each TeleoActor provides a given view of the corresponding StrActor. This view changes by adding/deleting data and services of that hypermedia node.

The HyperClas code in Figure 4 shows the definition of the TeloActor object. Because our aim is to focus the reader's attention more on our framework than on the implementation details, the access mode instructions are substituted with a synthetic informal descripti on. It is important to note that in the data part of the TeleoActor we have a link with the relative StrActor. More precisely, when a StrActor instance is generated, automatically an instance of a TeleoActor is created and coupled with the corresponding StrActor via the local resource `struct` (containing properly the address of the StrActor instance). The same mechanism is applied to couple an instance of a TeleoActor with its corresponding InfoActor instance. Another important local TeleoActor resource is `services`. This acquaintance may be

```
(defclass TeloActor(Actor)
 (struct ...)   ;;to address the related StrActor
 (info ...)     ;;to address the related InfoActor
 (services ...);;to identify the allowable services
 (inSuggestion ...)  (brSuggestion ...)
 (cnSuggestion ...)  ;;to maintain the interface,
     ;;browsing and content user preferences
 (...)  )
```

Figure 4: The data part of the TeleoActor.

viewed as a frame which depicts all the possible services usable on the current StrActor (a list of possible calls to scripts associated with the StrActor). When a TeleoActor is created, the complete suite of services is addressed; during the interaction between the user and the system, each TeleoActor may alter such resources on the basis of designed information provided by its InfoActor. The local resources `inSuggestion`, `brSuggestion` and `cnSuggestion` serve to collect the user behavior changes, in terms of three basic action categories: interface, browsing and content. These resource are updated by the InfoActor `info`. In the windows of Figure 3, a TeleoActor is responsible for displaying the current Class Browser A.

5.2 InfoActor

Each InfoActor is equipped with enough knowledge to recognize the user action. This knowledge depends on the domain content in order to better control and understand the user cognitive behavior. The code in Figure 5 shows its definition.

When the InfoActor is created, it owns an indispensable amount of context knowledge which is given by the system designer and stored in the object `domain`. The user monitoring is stored in the four resources `inInfo`, `brInfo`, `cnInfo` and `msInfo`; these objects can be viewed as frames containing information to identify the user actions in terms of four basic categories: interface, browsing, content, measurements. Each of these classes depicts the local user activities. For instance, `msInfo` contains a sequence of numbers which quantify some user actions performed on the corresponding StrActor, such as the number of visits done by the user, the average value of the time spent during the visit, and the number of help activations required by the user. The InfoActor establishes communication schemes with its TeleoActor and the UserActor. The messages from the InfoActor to its TeleoActor enable the latter to be run-time up-

Proceedings of The Thirtieth Annual Hawwaii International Conference
on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

1060-3425/97 $10.00 (c) 1997 IEEE

```
(defclass InfoActor(Actor)
 (struct ...)   ;;to address the related StrActor
 (telo ...)     ;;to address the related TeloActor
 (usrActor ...);;to address the UserActor
 (domain ...)   ;;to specialize the behavior in the
     ;;current context
 (inSuggestion ...)  (brSuggestion ...)
 (cnSuggestion ...)    ;;to collect the user
     ;;preferences to send to the TeloActor
 (inInfo ...)  (brInfo ...)  (cnInfo ...)
 (msInfo ...)   ;;to keep trace of the user actions
 (inTrust ...)  (brTrust ...)  (cnTrust ...)
 (msTrust ...) ;;to record the InfoActor trust
     ;;values in the previous four slots
 (inHints ...)  (brHints ...)  (cnHints ...)
 (msHints ...) ;;to store the new user behavior
     ;;sent by the UserActor
 (...)  )
```

Figure 5: Data part of the InfoActor.

dated on the more recent user needs. This is possible thanks to the local InfoActor's acquaintances which provide useful information about user behavior changing, that it, `inSuggestion` specifies the last user interface choices, `brSuggestion` represents the user browsing modalities and `cnSuggestion` represents the user content expectations. We note that `inSuggestion` includes those aspects which in Dexter [15] are stored in the *Presentation Specification* area of any component. For these reason, as said before, a TeleoActor is responsible for displaying different views for the same StrActor, but a InfoActor is responsible for informing that TeleoActor about the meaningful view for the user (in Figure 3, the Class Browser A).

The second communication activity is provided by the collaboration with the UserActor; here we discuss the messages from InfoActors to UserActor, deferring the other communication in the next subsection. The InfoActor possesses local and temporary user perspectives in the acquaintances `inInfo`, `brInfo`, `cnInfo` and `msInfo`, which are respectively the result of the tracing activities locally performed by the scripts `trace-in`, `trace-br`, `trace-cn` and `trace-ms`. These four typologies of information, together with the corresponding trust values `inTrust`, `brTrust`, `cnTrust` and `msTrust` are sent to the UserActor. As we will see in the next subsection, the UserActor collects this information asynchronously and establishes when and how the user model changes. The application of these changes induces the adaption of the local InfoActor knowledge to the new user perspective. This updating step consists of modifying the local acquaintances `inTrust`, `brTrust`, `cnTrust` and `msTrust` in order to dynamically vary the relevance of corresponding InfoActor observations. The rule applied to change these value is similar to that used in [23]. Let s be the suggestion `brInfo` (or respectively `inInfo`, `cnInfo`, `msInfo`), provided by the InfoActor to the UserActor and let p be the suggestion chosen by the UserActor and considered as relevant amongst all the suggestions received by all the activated InfoActors. The formula used to compute the new trust value `brTrust` (or respectively `inTrust`, `cnTrust` and `msTrust`) is:

$$trust = clamp(0, 1, trust + \delta_{s,p} * (\gamma * trust * (1 - wSum)))$$

where

$$\delta_{s,p} = \begin{cases} +1 & \text{if suggestion } s = \text{UserActor preference } p \\ -1 & \text{if suggestion } s \neq \text{UserActor preference } p \end{cases}$$

and the $trust$ maintains the old trust level in `brTrust` (or respectively in `inTrust`, `cnTrust`, `msTrust`) of the InfoActor, $wSum$ represents the corresponding value in `wBrTrust` (or respectively in `wInTrust`, `wCnTrust` and `wMsTrust`) provided by the UserActor, γ is the trust learning rate, and the function $clamp(0, 1, v)$ ensures that the value of v always lies in $(0, 1]$.
The rationale behind the modeling above is the following. The previous formula works in such a way as to increase (respectively decrease) the trust related to the four local slots `inInfo`, `brInfo`, `cnInfo` and `msInfo`, when the information contained in them, corresponding to the suggestions sent previously to the UserActor, has (respectively, has not) been effectively taken into account by the UserActor as meaningful for a new user behavior. The amount by wich the trust value rises or falls depends on the confidence of the other InfoActors in the suggestion provided by the current InfoActor. That is, if the suggestion of the InfoActor is not taken in account by the UserActor and the average trust ($wSum$) expressed by the other InfoActors is high, then the trust value should be penalized less heavily than for an incorrect suggestion with a lower average trust value. This inverse ratio is captured by the value $(1 - wSum)$. The formula to update the trust values of the single InfoActors constitutes the more relevant part of the script `update-trust`.

5.3 UserActor

Although the goal of the InfoActors is to observe local

Proceedings of The Thirtieth Annual Hawwaii International Conference
on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

user actions, the main task of the UserActor is to study these actions in order to infer new, general user preferences. Once this information has been customized by the corresponding InfoActors, the new habits would be sent to the corresponding TeleoActors. The UserActor is hence a collector-like actor, since it must organize the knowledge provided by collections of InfoActors. In Figure 6 a code description of its data part is shown. The cooperation activity that allows an adaptive evo-

```
(defclass UserActor(Actor)
 (infos ...)          ;;to address the InfoActors
      ;;responsible of the current user view
 (futureInfos)        ;;to address the InfoActors
      ;;interested in update the user model
 (inInfo ...)  (brInfo ...)  (cnInfo ...)
 (msInfo ...)         ;;to store the different local
      ;;user actions provided by the InfoActors
 (inTrust ...)  (brTrust ...)  (cnTrust ...)
 (msTrust ...)        ;;to maintain the trust values
      ;;user related to the four previous slots
 (wInTrust ...)  (wBrTrust ...)
 (wCnTrust ...)  (wMsTrust ...)
      ;;the normalized weighted user sum of trust
      ;;values exactly related same suggestions
 (gwInTrust ...)  (gwBrTrust ...)
 (gwCnTrust ...)  (gwMsTrust ...)
      ;;to contain the highest trust values
 (gInHints ...)  (gBrHints ...)
 (gCnHints ...)  (gMsHints ...)
      ;;to store the user features corresponding to
      ;;the previous highest trust values
 (...)  )
```

Figure 6: Data part of the UserActor.

lution of the hypermedia is obtained by a continuous, asynchronous information flow between the InfoActors and the UserActor. In more detail, the InfoActors in a parallel and asynchronous way return local user views to the UserActor (the slots `inInfo`, `brInfo`, `cnInfo` and `msInfo`) together with the related trust values (`inTrust`, `brTrust`, `cnTrust` and `msTrust`). For example, let us suppose that, from the situation shown in Figure 3, the programmer accomplishes meaningful operations, detected by some InfoActors and envoied to the UserActor, as shown in Table 1. Table 1 shows the asynchronous information flow from eight InfoActors (`Info1`, `Info1`, ..., `Info8`) to the UserActor. More in detail, first column contains the time in minutes (starting from 0, that it the first clock signal

time	InfoActor	suggestion	trust
0			
0.5	Info1	B	0.5
0.8	Info2	C	0.6
1.7	Info3	B	0.1
2	Info4	C	0.38
2.7	Info5	C	0.75
2.7	Info6	A	0.7
3	Info7	C	0.8
3.5	Info8	C	0.9

Table 1: Asynchronous information flow from InfoActors to the UserActor.

after the last threshold fixing) in which the UserActor receives the couples (preference, trust) (respectively third and fourth columns) from the activated InfoActors (second column).

For a sake of simplicity, the sent messages are related only to the three Class Browsers, A, B, and C. The UserActor evaluates such information starting from the values contained in this case in `brTrust` and gets, for each component inside `brTrust`, a normalized weighted sum (`wBrTrust`) by means of the execution of the script `return-trust-wsum`. The highest trust sum determines the relevant user preference to take into consideration. If this value is higher than the corresponding (current) meta-net threshold, contained in `gwBrTrust`, then this slot is updated with the new higher value. This corresponds to a new current meta-net threshold. Following the evolution of the data contained in Figure 1, the graph in Figure 7 synthesizes the progress of trust functions pointing out the user preference change at time 3.5 minutes. In Figure 7 is evident that the Class Browser A remains current for 3.5 minutes; at this time, `wBrTrust(C)` overcomes `wBrTrust(A)` and the first becomes the new current meta-net threshold `gwBrTrust`. This situation provokes the decision of the UserActor to modify the user preferences from the scenario A (left most window in Figure 3) toward scenario C (Figure 8). Now, the UserActor communicates the deduced meaningful user changes, the slot `gBrHints` to the interested InfoActors `futureInfos` [1]. This action is performed by the script `propagate-changes`. Now, because a new threshold is established, it is necessary to update the local, distributed trust contained in the sender In-

[1] The `futureInfos` identifies the collection interested in updating the user model; in particular, it is defined as the union of the current active InfoActors and their frontier, extended k times by means of an iterative process (where k is a natural number related to the specific application).

Proceedings of The Thirtieth Annual Hawwaii International Conference
on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

1060-3425/97 $10.00 (c) 1997 IEEE

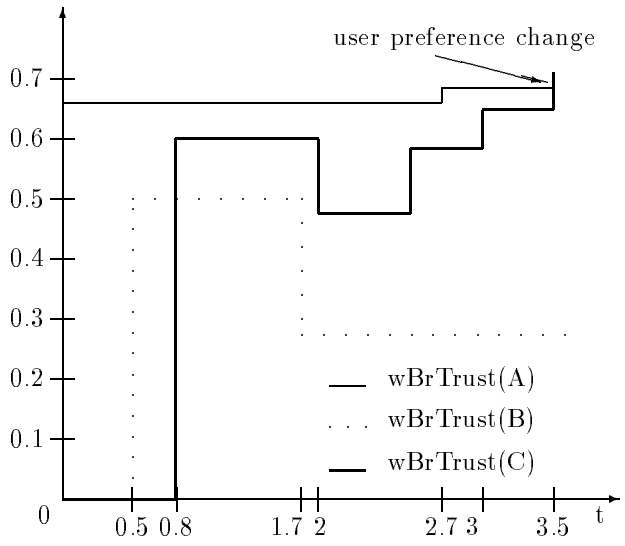

Figure 7: Three trust functions; the function `wBrTrust(C)` invokes the user preference change at time 3.5 minutes.

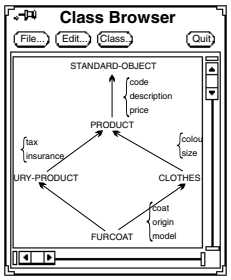

Figure 8: An example of the browsing mode C.

foActors. This is done by sending the multicasting message `update-trust` to each InfoActor specified in `infos`. Naturally, the whole process is accomplished also for the other three typologies of suggestion (interface, content, measurements). To better explain this behavior we visualize the adaptive collaboration in Figure 9. Figure 9 visualizes the cooperation between the UserActor and the InfoActors. In this example, the InfoActors `Info1`, `Info2` and `Info3` represent the resource `infos`, while the InfoActors `Info3`, `Info4`, `Info5` and `Info6` are the `futureInfos`. Each InfoActor in `infos` spies the user browsing and sends a local and partial user view to the UserActor. In Figure 9, the InfoActor `Info1` reports special user interest towards a given interface format, `Info2` illustrates a favourite browser activity and `Info3` underlines the user choice about a certain topic. This information

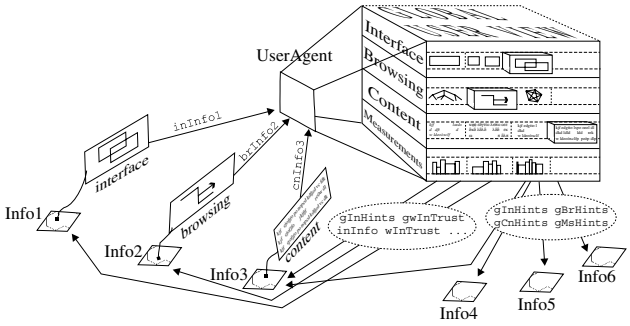

Figure 9: Cooperation activity between UserActor and InfoActors.

(the resources `inInfo1`, `brInfo2` and `cnInfo3`) is collected by the UserActor in the acquaintances `inInfo`, `brInfo` and `cnInfo`, respectively. Then, the main UserActor activity consists in analyzing such data as part of the Global User View (GUV). As a result of this process, the UserActor updates the GUV and, if a new user state is detected, as shown in Figure 9, then it sends:

- the new user preferences (in the most general form, the slots `gInHints`, `gBrHints`, `gCnHints` and `gMsHints`) to each InfoActor in `futureInfos`;
- the information (the slots `gInHints`, `gwInTrust`, `inInfo`, `wInTrust`, `gBrHints`, ...) necessary to update the trust values of all the InfoActors contained in `infos`.

The previous actions are performed by the script `propagateChanges` in Figure 10.

```
(defmethod propagate-changes(self UserActor)
 (send (get-futureInfos self) :spray
  :selector update-preferences
  :args (get-gInHints get-gBrHints
      get-gCnHints get-gMsHints self))
 (send (get-infos self) :spray
  :selector update-trust
  :args (get-inInfo get-wInTrust
     get-gwInTrust get-gInHints ... self)))
```

Figure 10: Distributed, concurrent user model update.

The previous HyperClas code exploits the multicast message passing protocol via the keyword `:spray` in order to gain concurrency in informing the InfoActor about the user model changes. At the execution

Proceedings of The Thirtieth Annual Hawwaii International Conference
on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

1060-3425/97 $10.00 (c) 1997 IEEE

of this script, a distributed, parallel update of local user models occurs. The local treatment of the updating process is delegated to each InfoActor belonging to `futureInfos` through the activation of a specific script, `update-preferences`.

6 Final Remarks and F uture W ork

Our goal was to enrich an existing hypermedia architecture [9] with an appropriate user model activity, conceived and realized with the same design philosophy of the underlying system. Previous experiences in crafting distributed cognitive diagnostic systems [25] helped us in defining the overall architecture.

This approach leads to several advantages:

- organizational structure: the user recognition is accomplished through a cooperation activity determined by an actor-based knowledge acquisition process. The specialization of InfoActors knowledge/duties for each hypermedia node allows the system to better handle local observations on the user, personalizing metrics/strategies that depend on the various user characteristics. Such an organizational approach makes flexible the architecture of our model, enabling an extension of the functionality without affecting its inner features.

- cooperation modes: the cooperation protocols among the actor populations allow the system to gain flexibility and adaptivity, while avoiding rigid schemes which do not easily support the dynamism and the evolution of the user.

- conflicts: no conflict can arise between InfoActors, because the presence of a unique UserActor leads to a centralized management of the user recognition preferences or habits. This does not decrease the level of potential distributed operations. In fact, we are currently developing a collaborative, multi-user version [23] of our architecture, where for each user we have an autonomous UserActor, in order to make better use of the concurrency of the model.

- distributed, decentralized computation: concurrency is introduced as software design metaphor of the hypermedia system.

The architecture has been applied in the development of a hypermedia system that is useful for supporting an object-oriented logic programming system [11, 27]. Some problems exist in the current version:

- too much work for the application engineer: the knowledge specification for each InfoActor requires substantial effort. W e are investigating the possibility of automating their construction by applying a meta-level scheme definition.

- few reasoning mechanisms are used by the UserActor: currently a threshold-based deduction strategy is supported. This is due to the fact that our initial experimental effort was focused on providing a general framework with a high degree of flexibility. The deduction strategies of the UserActor can be enriched by defining additional scripts without expensive design effort [23, 28].

Finally, the generality of our architecture permits one to image its future exploitation as not limited to hypermedia environments; for this reason, with the necessary (minor) modifications, the application of the model to other kinds of systems, such as DBMS or CASE systems, represents an interesting future step.

References

[1] K. Aberer, W. Lkas, A. L. Furtado. Designing a User-Oriented Query Modification Facility in Object-Oriented Database Systems. *Proc. of the VI CAiSE*94*, Utrecht, The Netherlands, June 1994, LNCS (811), pp.380-393, 1994.

[2] Agha, G. & Hewitt, A. (1988). Actors: A Conceptual Foundation for Concurrent Object-Oriented Programming. In *Research Directions in Object-Oriented Programming*, MIT Press, Cam bridge, MA, pp.49-74.

[3] T. Berners-Lee, R. Cailiau, J. Groff, B. Pollermann. World-Wibe Web: The Information Universe. *Electronic Networking*, **2**(1):52-58, 1996.

[4] D. G. Bobrow, L. DeMichiel, R. P. Gabriel, G. Kiczales, D. Moon, S. Keene. CLOS Specification; X3J13 Document 88-002R. ACM-SIGPLAN Not. **23**, 1988.

[5] J. P. Briot, L. Gasser. Connections between Object-Based Concurrent Programming and Distributed Artificial Intelligence. *IJCAI Workshop on Objects and Artificial Intelligence*, Sydney, Australia, August, 1991.

[6] P. Brusilovsky. Methods and techniques of adaptive hypermedia. Special issue of *User Modeling and User Adapted Interaction*, **6**(2-3), 1996.

[7] W. F. Clocksin, C. S. Mellish. *Programming in Prolog*, Springer-Verlag, 1981.

Proceedings of The Thirtieth Annual Hawwaii International Conference on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

[8] A. Dattolo, A. Gisolfi. Analytical Version Control Management in a Hypertext System. *Proc. 3th Int. Conf. CIKM'94*, 132 -139, Nov. 29 - Dec. 2, NIST, Gaithersburg, MD, ACM Press, 1994.

[9] A. Dattolo, V. Loia. Hypertext Version Management in an Actor-based Framework. *Proc. of the VII Intern. Conf. CAiSE*95*, Jyväskylä, Finland, June 12-16, LNCS (932), pp.112-125, 1995.

[10] A. Dattolo, V. Loia. Conceiving Collaborative Version Control for agent-based conceived Hypertext. *Proc. of the Workshop on The Role of Version Control in CSCW Applications - ECSCW'95*, Stockholm, September 10, 1995.

[11] A. Dattolo, V. Loia. Agent-based Design of Distributed Hypertext. *Proc. of the 11th ACM Symposium SAC'96*, Philadelphia, Pennsylvania, February 17-19, pp.129-136, 1996.

[12] A. Dattolo, V. Loia. Collaborative Version Control in Agent-based Hypertext Environment. *Information Systems*, **21**(2):127-145, 1996.

[13] F. De Rosis, N. De Carolis, S. Pizzutilo. User tailored hypermedia explanations. *INTERCHI'93*, Amsterdam, April 24-29, pp.169-170, 1993.

[14] A. Gisolfi, V. Loia. Designing Complex Systems within Distributed Architectures. *Int. J. of Applied Art. Intelligence*, **8**(3):393-411, 1994.

[15] F. Halasz, M. Schwartz. The Dexter hypertext reference model. (K. Grønbæk and R. H. Trigg Eds.) *CACM*, **37**(3):30-39, 1994.

[16] C. Hewitt. Open Information Systems Semantics for Distributed Artificial Intelligence. *Artificial Intelligence*, **47**(1-3):79-106, 1991.

[17] Isakowitz, T., Stohr, E. A., Balasubramanian, P. RMM: A Methodology for Structured Hypermedia Design. *CACM*, **38**(8):34-44, 1995.

[18] C. Kaplan, J. Fenwick, J. Chen. Adaptive Hypertext Navigation Based on User Goals and Context. *User Modeling and User Adapted Interaction*, **3**:193-220, 1993.

[19] M. Katsumoto, M. Fukuda, Y. Shibata. The Kansei Link Method for Multimedia Database. *Proc. of the 10th Int. Conf. ICOIN-10*, pp.382-389, 1996.

[20] K. Kimbrough, O. Lamott. Common Lisp User Interface Environment. Texas Instruments Inc., July, 1990.

[21] A. Kobsa. Preface to *User Modeling and User-adapted Interaction*, **1**(1), 1991.

[22] A. Kobsa, D. Müller, A. Nill. KN-AHS: An Adaptive Hypertext Client of the User Modeling System BGP-MS. *Review of Information Science*, **1**(1), July 1996.

[23] Y. Lashkari, M. Metral, P. Maes. Collaborative Interface Agents. *Proc. of AAAI'94*, 1994.

[24] H. Lieberman. Concurrent Object-Oriented Programming in Act 1. *OOCP'87*, pp.9-36, 1987.

[25] V. Loia. Distributed Diagnostic Reasoning: a new approach to student modeling. *Proc. of the IV Intern. Conf. on User Modeling, UM94*, Hyannis, Massachusetts, August 15-19, pp.37-41, 1994.

[26] V. Loia, M. Quaggetto. CLOS: a key issue to bridge the gap between object-oriented and logic programming *Proc. of the 5th Inter. Conf. SEKE-93*, San Francisco, June 16-18, pp.62-69, 1993.

[27] V. Loia, M. Quaggetto. The Opla system: designing complex systems in an object-oriented logic programming framework. *Computer Journal*, **39**(1), 1996.

[28] P. Maes. Social Interface Agents: acquiring competence by learning from users and other agents. *Software Agents - Papers from the 1994 Spring Symposium*, AAAI Press, pp.71-78, 1994.

[29] N. Mathé, J. Chen. A User-Centered Approach to Adaptive Hypertext based on an Information Relevance Model. *Proc. of the IV Intern. Conf. on User Modeling, UM94*, Hyannis, Massachusetts, August 15-19, pp.107-114, 1994.

[30] E. Rich. Stereotypes and User Modeling. *User Models in Dialog Systems*, A. Kobsa and W. Wahlster Eds., Berlin, Springer-Verlag, 1989.

[31] M. Thüring, J. Hannemann, J. M. Haake. Hypermedia and Cognition: Designing for Comprehension. *CACM*, **38**(8):57-66, 1995.

[32] J. A. Waterworth. A pattern of islands: exploring public information space in a private vehicle. *Multimedia, Hypermedia and Virtual Reality.* LNCS, pp.266-279, 1996.

Proceedings of The Thirtieth Annual Hawwaii International Conference on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

1060-3425/97 $10.00 (c) 1997 IEEE