International Journal of Software Engineering
and Knowledge Engineering
Vol. 14, No. 6 (2004) 565–602

World Scientific
www.worldscientific.com

CONCEPTUALIZATION, PROTOTYPING AND PROCESS OF HYPERMEDIA APPLICATIONS

ANTONIO NAVARRO, ALFREDO FERNÁNDEZ-VALMAYOR,
BALTASAR FERNÁNDEZ-MANJÓN and JOSÉ LUIS SIERRA

Dpto. Sistemas Informáticos y Programación,
Universidad Complutense de Madrid,
C/Professor José García Santesmases s/n, 28040 Madrid, Spain

Latest-generation hypermedia applications represent a new challenge in traditional design and development software engineering techniques. Although there is an impressive array of models to design hypertext applications, these models may not be specially suited for conceptualization-prototyping stages. In this paper, we provide a comprehensive software engineering approach for dealing with the conceptualization, prototyping, and process of hypermedia applications. This approach uses the *Pipe Model* to characterize hypermedia applications during the conceptualization stage, while prototyping is accomplished using XML and Java technologies. An XML-based representation of the Pipe structures is the input for a Java application that automatically builds the prototypes of the hypermedia application. This XML representation may reference to *Subordinate Processes*, i.e. compiled Java classes that implement a predefined interface and can be executed in the hypermedia application without interacting with the navigation. We also present the *Plumbing* and *PlumbingXJ* process models, two specializations of a well-known hypermedia process model, which integrate and manage the use of the Pipe model and its associated XML and Java technologies.

Keywords: Hypermedia-oriented model; hypermedia process model; hypermedia prototyping; XML; hypermedia; software engineering.

1. Introduction

More than fifty years ago Vannevar Bush described *Memex* [1], which can be considered as the first hypermedia system. Nowadays the dramatic evolution of digital systems has provided us with a capacity for processing information that would have been unthinkable in the past. Internet (and therefore hypermedia applications) is one of the more active areas in this evolution. Today, web sites are developed using advanced technologies and multitier architectures, producing much more complex and powerful applications than those built in the precedent years.

Because of this complexity, the use of software engineering techniques is mandatory during the development of these applications [2]. Several difficulties arise when traditional models and methodologies in software engineering are used for the design of hypermedia applications because they are not specially adapted

for representing the intrinsic characteristics of hypermedia systems: navigational structures and the use of multimedia contents [3]. Moreover, hypermedia applications have a number of requirements that do not appear during the development of non-hypermedia software [4], such as the necessity of handling both structured data (e.g. database records) and non-structured data (e.g. multimedia items), the support of exploratory access through navigational interfaces, a high level of graphical quality, the customization and possibly the dynamic adaptation of content structure, navigation primitives and presentation styles, and the support of proactive behavior (i.e. for recommendation and filtering).

In our opinion the process model proposed by Fraternali [4] contains most of the structural activities for dealing with most of the difficulties to be taken into account. This process model identifies the need for prototyping (as Nanard [5] does), but also includes a *conceptualization* phase prior to the prototyping phase. At the conceptualization stage the application is represented through a set of abstract models that convey the main components of the envisioned solution [4]. These models used in conceptualization can be understood as semantically closed abstractions of systems[a] [6]. The conceptualization-prototyping loop precedes the design-development stage, where the *final* application is designed and developed.

A number of *hypermedia-oriented models* [7] have been proposed to help designers to face their conceptualization task such as: Dexter [8], Amsterdam [9], HAM [10], Trellis [11], hipergraphs [12], Labyrinth [7], HDM [2], RMM [13], and OOHDM [14]. They are models with a different orientation.

Some of them are *system*-oriented models [2]. These models provide a data model for hypertext systems, environments that facilitate the creation of hypertext applications [13]. Dexter, Amsterdam, HAM, Trellis, Tompa hipergraphs and Labyrinth are some of these system-oriented models [2], [7].

Some others are mainly conceived as *application*-oriented models. These models provide a language for describing the information objects and the navigation mechanisms in hypermedia applications, like RMM [13]. HDM, RMM and OOHDM are some of the more relevant application-oriented models [15].

Notwithstanding this impressive array of hypermedia-oriented models, according to Nanard [5], the range of hypertext applications is so broad that no single formal design technique is relevant for designing all of them. For example, as described in the work of Wills [16], Balasubramanian rejected the application-oriented models RMM, HDM and OOHDM when designing a large-scale web site for a financial-management company because all of them require the application domain to be abstracted in the forms of entities or classes and relationships [17]. The problem is that according to Isakowitz, applications that have irregular (or dynamic) structures and high volatility (e.g. literary work or multimedia news service) may gain little from the use of the RMM and similar approaches [13].

We could propose the use of system-oriented models for the conceptualization of these hypermedia applications with irregular structures. However, these system-

[a]Collections of connected units that are organized to accomplish a specific purpose.

oriented models, like those previously mentioned, provide an application representation that is tied to the underlying data model of the hypertext system. In our opinion, these models not only characterize the conceptualization-prototyping stages, but also constrain the design-implementation stages. This problem is related to that found by Barry [15] when analyzing the low impact of application-oriented models in industry, because the model concentrates on the production of a design representation rather than assisting the design process. Moreover, other authors such as Isakowitz consider that systems oriented models are of little value in modeling hypermedia applications [13].

Note that we are not arguing that hypermedia-oriented models are useless. In fact, they are helpful contributions when designing hypermedia applications. We have only identified a stage in the development of some hypermedia applications where a new model would be useful.

Taking into account several works in the hypermedia domain we can conclude that this new model must fulfil, at least, the following four requirements:

(i) If the model is focused on applications with irregular structure [13], then such a model should be focused on the common structure that appears in every hypermedia application: links between contents. In this way, contents represent units of information while links indicate relations between those units [18], [19].
(ii) As Schwabe [14] and Koch [20] identify, the model should describe the navigational items of the application.
(iii) If this model serves as the basis for prototyping, an easy transition from model diagrams to running applications would be desirable [5]. This requirement makes the definition of a common *default browsing semantics* necessary [2] [11].
(iv) Moreover, due to the nature of actual hypermedia applications, the characterization of dynamic linking would be a valuable characteristic [7][21].

The solution that we propose to help characterize the conceptualization stage is the *Pipe Model*, an application model able to:

(i) Capture the relationships between elements in the domain using a graph, called *Contents Graph*.
(ii) Describe the navigation desired for the contents using another graph, called *Navigational Schema*. This navigational schema is defined independently of (without taking into account) the contents graph. The relationships between both graphs are defined using the so called *Canalization Functions*.
(iii) Provide a default browsing semantics that, using the Pipe graphs and functions, is able to describe the behavior of the application when the user traverses a link.
(iv) Enable the abstract characterization of dynamic links using the contents graph.

Note that although other approaches such as OOHDM, UWE [20] or WebML [22] seem to have modeling primitives with more abstraction power, entities/classes

and their relationships are not more powerful abstractions than informational units of content and their relationships (or links). They are simply tailored for different domains. For example, if we need to model a hypermedia literary work, the entity/class primitives, as Isakowitz states, are of little use, and in our opinion, a graph is one of the clearest, simplest and most formal representation that can be provided. Moreover, although Pipe does not explicitly define a notation to represent entities and their relationships if there is a need to characterize this type of elements in the application domain, a suitable model, like RMM, can be used. Later, a Pipe contents graph can refer to the instances of these entities/classes using the *generation function g*, presented later. In this way, for example, if an index is represented using an RMM index guided tour node [14], the generation function can be used to select and link the elements that compose this guided tour.

The inclusion of the Pipe model into the Fraternali process model (with some influences of Ginige/Lowe's process model [16]) generates a new process model called *Plumbing*. We have specialized Plumbing with a specific prototyping technique where XML [23] documents represent the hypermedia application that is described using the Pipe model, and where a Java hypermedia system (the *Automatic Prototypes Generator*, APG) generates the prototypes used at the prototyping stage. This process model is called *PlumbingXJ*.

This paper gives a brief review of the work related to our approach. It presents the Pipe model (omitting its formal component), and shows how Fraternali's process model is particularized to obtain Plumbing and PlumbingXJ process models. Throughout these sections we will use a case study to demonstrate the applicability of our approach. Finally we present our conclusions and ongoing work.

2. Related Work

In this section several approaches to the modeling of hypermedia applications, hypermedia process models and the use of markup languages in hypermedia development are reviewed.

2.1. *Hypermedia-oriented models*

The *Dexter Hypertext Reference Model* [8] was the first attempt to unify and clarify hypermedia technology, and it has become a reference model to compare with any other system-oriented model. Its most important feature is the division of a hypermedia application into three layers: *within-component*, *storage* and *runtime*. The first layer represents the contents as independent elements. The second layer represents the linked elements of a hypertext application. Finally, the third layer represents the running application. Note that according to the Dexter Model, an application can be characterized providing the data in the within component layer, the storage and the run-time layer. By using a contents graph, the Pipe model characterizes the application data, which in the Dexter model are considered in the within-component layer. The presentation specifications that relate the storage

and the runtime layer, thus allowing for the presentation of the application, is described using a navigational schema and canalization functions. Finally, the default browsing semantics allows for the description of the behavior in the run-time layer.

The *Amsterdam Hypermedia Model* [9] is based on Dexter's layers and expands them with synchronization and *context* capabilities. The need for synchronization appears in modern hypermedia application where multimedia contents are involved. The context capabilities appear when the simultaneous visualization of several contents (of the same or different nature) is needed (e.g. the HTML frames). Pipe is able to represent the notion of context using a navigational schema. At the same time, this navigational schema is used to attach the synchronization information. Although Pipe's synchronization characteristics are not as powerful as the characteristics of the applications that are described using the total expressive power of the Amsterdam model, it is powerful enough to depict the usual timing information that appears in most hypermedia applications.

The *Hypertext Abstract Machine (HAM)* [10] is an abstract representation of the storage layer of a hypermedia application. The Pipe representation of the storage layer coincides with the HAM approach, but Pipe provides an explicit representation of the more complex graphical user interface and context characteristics that HAM cannot represent.

The *Trellis Model* [11] is a hypermedia model that tries to characterize the simultaneous presentation of several contents using a Petri net. Although valid for earlier applications, it cannot support the modern context user interfaces and the complex navigation capabilities that they induce. The most important feature of this model (present also to a certain extent in HAM) is the browsing semantics represented by the Petri net. The Pipe browsing semantics is defined via the *link activation function* that indicates the changes in the hypertext when the user traverses a hyperlink. The *Hypergraph Model* [12] is a similar approach to that of the Trellis model. In this case, the Petri net is replaced by a hypergraph, but the final browsing semantics relies on several primitives defined by the model.

The *Hypertext Design model (HDM)* [2], the *Relationship Management Model (RMM)* [13] and the *Web Designing Language (WebML)* [22] are design methodologies that define a relational hypermedia-oriented model. They provide several access structures that define the different navigational alternatives when added to Entity-Relationship [24] (or extensions) diagrams. These models are appropriate for providing a hypertextual access to a database, where the hypertextual relationships are (basically) derived from the foreign keys of the relational schema. In other hypermedia applications these models do not fit well [13]. Although it is not a relational model, Pipe is also able to represent relational hypermedia applications using the graph representation of the instances derived from the relational schema [22].

In the former hypermedia-oriented models the selection of an anchor was the only event that could occur. The *Labyrinth Model* [7] is prepared to respond to any event that may occur in an application, *pure* hypermedia or *Computational Hypermedia Applications (CHAs)* [25]. This is a more powerful approach, but with

a significant shortcoming: there is no default browsing semantics, and therefore the automatic generation of applications based on the Labyrinth model representation is more complicated. On the contrary, although the Pipe model can only represent the anchor selection event, it has a default browsing semantics.

Although the *Object-Oriented Hypermedia Design Methodology (OOHDM)* [14] identifies a set of classes and relationships between them, in the same manner as HDM or RMM methodologies, it presents more powerful object-oriented capabilities. It describes hypermedia applications by means of three main activities, with their corresponding diagrams (or schemas). The *conceptual design* characterizes the underlying classes and their relationships in the application domain. The *navigational design* characterizes the navigational structure of the application. Finally the *abstract interface design* describes the user interface of the application. OOHDM is a powerful methodology suitable for representing CHAs, but like HDM and RMM methodologies, it does not fit well in applications where the domain is not naturally abstracted in the form of classes and relationships [17]. Regarding OOHDM, in Pipe, the contents graph characterizes the elements of the application domain and their relationships. It is similar to the OOHDM conceptual schema, but in the case of Pipe, *objects* (or elements) are described, not *classes*. The Pipe navigational schema is similar to the OOHDM navigation chart, but again expressed in terms of elements. Because Pipe manages elements, not classes, there is no need to define OOHDM navigational classes or OOHDM navigational schemata. On the contrary, a mapping between the Pipe contents graph and the Pipe navigational schema must be defined. Precisely, the canalization functions define the map that relates both structures.

There are several Web design methods influenced by OOHDM such as the *UML-based Web Engineering approach (UWE)* [20] and the *Web Site Design Method (WSDM)* [26] that are tailored for Web Engineering. Consequently, the conclusions extracted from OOHDM are also applicable.

Other approaches like the *Object-Oriented Hypermedia method (OO-H)* [27] or the *Object-Oriented Web-Solutions modeling approach (OOWS)* (an extension of OO-H) [28] are also based on OOHDM and tailored for Web Engineering and UML, but present more advanced characteristics: they present a pattern catalog which can be applied to the different diagrams to modify both the model and the final implementation. Anyway, as in the case of OOHDM, they are not suitable for domains in which there are no identifiable classes or relationships.

The *Hera* methodology [29] is inspired by RMM and includes several features in order to provide a better support for an automated design which can be applied to adaptive Web applications. Like in RMM, there is a conceptual model expressed in an Entity-Relationship manner. Consequently it is not applicable in those applications where RMM is not suitable.

The *Fundamental Open Hypertext Model (FOHM)* [30] defines a common data model and a set of related operations that are applicable to the three domains of Open Hypermedia Systems [31]. Open hypermedia is concerned with producing a

protocol which allows components of heterogeneous hypermedia systems to communicate with each other. Although the FOHM scope is limited to this involvement, it presents an interesting characterization of the hypertext state using an *abstract machine*. The Pipe model uses a similar characterization of the hypertext state, but without worries about hypertext interoperability.

2.2. *Process models*

The need for a design model is just a part of the problem. A constant concern in software engineering is the presence of a process model that structures the software development process. Despite this fact, there are few process models specifically designed for hypermedia development [32]. In particular, we can find those provided by Fraternali [4], Nanard [5] and Ginige [16]. Other authors like Lowe [33] introduce reference process models that are used to evaluate the process models.

Although there are valuable process models in *generic* software engineering (e.g. [34], [35]) in addition to the reasons explained in the introduction that justify the need for specific hypermedia modeling techniques, there are other causes that lead to the specialization of the generic process models [13].

The process model presented by Fraternali [4] collects the key ideas of several process models. It is formed by two development loops. The first on *conceptualization-prototyping*, and the second is focused on *design-development*. This process model is the one chosen in our approach where the Pipe model directs the conceptualization stage. This (more specific) process model is called *Plumbing*, and also shows the influence of the Ginige-Lowe process model [16] (which in turn is very similar to Fraternali's process). We have chosen Fraternali's approach instead of Nanard's [5], another iterative process model, because, in our opinion, Nanard's process is very dependent on the implementation techniques. We present Fraternali's process model in the description of the Plumbing process model.

Regarding prototyping, the ideal case is to generate prototypes automatically by using the structures provided by the hypermedia-oriented model. This is only possible if the hypermedia-oriented model provides a default browsing semantics (as Pipe does). By specializing the Plumbing process model with XML and Java prototyping techniques, we obtain *PlumbingXJ*, a more specific process model, closer to a methodology than to a classic process model.

In our process models, there are no specific design-development techniques. We think that nowadays it is not reasonable to provide a closed solution to these stages, and approaches like [36] are more realistic.

2.3. *Markup languages*

Today, the use of markup languages is a common approach for developing a hypermedia application. In the approach presented in this paper, XML is the formalism offered to authors to define customized languages appropriate for describing the structure of the static contents and relationships of its application domain. In our

approach the navigational schema of such applications is also described using an XML-based language. This is possible because XML, as a tool for defining languages, can provide authors with all the flexibility and expressiveness needed to describe the structure of most application domains. If the structure of the domain is not the main issue, an HTML-like DTD can be defined. On the contrary, if the elements in the domain have some structure that is not suitable to be described using a relational schema (i.e. educational or literary work) an XML-based language can be the most appropriate formalism to structure such a domain [37]. In this section we briefly review some of the most relevant markup languages related to the hypermedia domain, in order to contrast our approach.

The *HyperText Markup Language (HTML)* [38] used for Web development is one of the most widely used markup languages in history. Our XML solution in PlumbingXJ benefits from the expressive power and flexibility of XML-based languages and at the same time tries to maintain the HTML simplicity in order to achieve a greater applicability.

In contrast to this approach, the set of *Architectural Forms* of *Hypermedia/ Time-based Structuring Language (HyTime)* [39] was a powerful way to introduce hypermedia structure via markup languages, but never had the success that it deserved. The lack of a *clear* browsing semantics in the scheduling and presentation modules, and the huge size of HyTime handicapped its success. The Pipe browsing semantics and the limited complexity of the XML representations try to prevent these drawbacks.

The greatest shortcoming of the *Synchronized Multimedia Integration Language* (SMIL) [40], a *multimedia* markup language, is the need to define the links of the application within the description of the user interface of the application. This characteristic could lead to reusability problems with the contents and their navigational interpretation.

The use of XML made in PlumbingXJ is similar to WebML [22], which is based on the work of [41]. WebML defines a markup language that generates hypermedia applications using XML descriptions of relational hypermedia applications. The main differences between WebML and the PlumbingXJ approach derive from the different application domains where they are applied.

Finally other approaches like *XML User Interface Markup Language* (XUL) [42] and *User Interface Markup Language* (UIML) [43] are closer to the GUI design of a generic software application, and require a serious effort to be customized in hypermedia development.

3. The Pipe Model

The Pipe model [44] formalizes our vision of a hypermedia application as a two-layer structure, where the contents and their semantic relationships [18] are mapped to an abstract representation of the GUI by means of the so-called *canalization functions*. The Pipe model characterizes a hypermedia application by means of

five components:

— *CG* is the *Contents Graph*. It models the content elements and their relationships in the application with total independence of its navigational treatment.
— *NS* is the *Navigational Schema*. It models the elements of the GUI and the navigational relationships established between them.
— *CF* are the *Canalization Functions*. They relate the contents graph and the navigational schema.
— *BS* is the *Browsing Semantics*. It provides a browsing semantics to the general structures obtained when relating a contents graph and a navigational schema via canalization functions.
— *PS* is the *Presentation Semantics*. It is similar to the *BS*, but includes a function that provides information on the style characteristics (position, font, color, etc.).

Consequently, according to the Pipe model, a hypermedia application is characterized by a tuple $\langle CG, NS, CF \rangle$, whilst the *BS* and *PS* are common to all the applications. We will see in detail the components of the model, which have (slightly) evolved since [44]. We omit the specific formalization of the model and provide the associated graphical notation instead[b].

3.1. *Contents Graph (CG)*

The *Contents Graph*, *CG*, is the representation of the content elements and their relationships (or links) in a hypermedia application. Let C and L be the *set of all the contents* and *all the links* that can appear in a hypermedia application. Then *CG* is a tuple $\langle C_A, L_A \rangle$, where:

— $C_A \subseteq C$ is the set of *Contents* of the application. These contents include the *subordinate processes*, small applications that are executed inside the hypermedia applications, but without affecting navigational behavior.
— $L_A \subseteq L$ is the set of *Links* (or relationships) of the application. This set represents all the intrinsic semantic relationships between contents relevant to the application. First we will define it for the static case, and later we will extend the definition to the dynamic case.

3.1.1. *Static linking*

Let $\mathbb{N}$ and H be the set of *Natural numbers*, and the *set of all the anchors* that can appear in a hypermedia application. Then in (1) we define the *Static links*, L^0, of a hypermedia application.

$$L^0 \subseteq C^0 \times H^0 \times \mathbb{N} \times C^0 \times H^0 \tag{1}$$

[b]The complete formal presentation of the Pipe model would require the inclusion of a mathematical apparatus not absolutely necessary for its understanding. Therefore some expressions may appear simplified.

where

— $C^0 \subseteq C$ is the *set of static contents* of the hypermedia application.
— $H^0 \subseteq H$ is the *set of anchors* of the hypermedia application.

A tuple $(s, sa, n, d, da) \in L^0$ represents a *source content* (s), a *source anchor* (sa), a *link number* (n), a *destination content* (d) and a *destination anchor* (da). The link number serves to characterize n-ary links. Thus a binary link between the source content A and the destination contents B and C can be represented using the links $(A, toBC, 1, B, total)$, $(A, toBC, 2, C, total)$, *total* being a distinguished anchor that represents the whole content. This link number is needed in order to define the formal browsing semantics because the selection of anchor $toBC$ in content A activates all the links $(A, toBC, \_, \_, \_)$.

The previous definition of set L^0 is too generic. Ultimately, this set must denote a binary relationship between pairs of source content and anchor with the destination content and anchor. To obtain a simpler characterization of the browsing semantics, and to permit the characterization of dynamic contents, we are going to split the links into their source part (content, anchor and link number) and their destination part (content and anchor). In this way, set L^0 can be expressed as a relationship. This relationship is characterized by the *relationship function* r, which in the static case is described in (2) by the expression[c] r^0.

$$r^0 : C^0 \times H^0 \times \mathbb{N} \to C^0 \times H^0 \tag{2}$$

In (3) we use r^0 to define the set of links of the application.

$$L^0 = \{(s, sa, n, r^0(s, sa, n))|(s, sa, n) \in C^0 \times H^0 \times \mathbb{N}\} \tag{3}$$

In this way the linking between contents can be characterized as pairs of source content and anchor with a link number $((s, sa, n))$, and a destination content and anchor $(r^0(s, sa, n))$. Note that in Pipe, the property of being *source* or *destination* anchor is not intrinsic to the anchor definition, but is provided by the *link definition* that assigns this role. Our approach is similar to those presented in Labyrinth [7] and Dexter [8].

If the application only has static components, then $C_A = C^0$, and $L_A = L^0$.

3.1.2. *Example. Static case*

Galatea is an educational hypermedia application for the comprehension of Romance languages [45]. The *contextual dictionary* is one of Galatea's modules and is going to be our case study throughout the paper. In this contextual dictionary the learner can select any sentence from a text to check the contextual meaning of any word in that sentence, including the relationships with surrounding words.

[c]The relationship function r is defined in both the static and dynamic case. The expression r^0 characterizes this function in the static case.

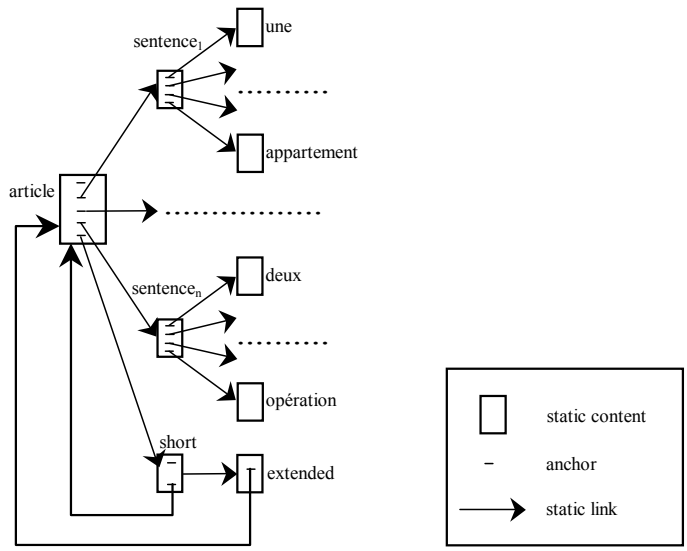

Fig. 1. Static contents graph for the article from the contextual dictionary.

In addition, the text is also linked to a short description of the text. This short description is linked to a detailed description. Both descriptions are linked to the original text. For the sake of conciseness, the example is focussed on a text from the dictionary, specifically, an article about a fire in an apartment. Figure 1 depicts the contents graph for the example.

Although this contents graph was manually generated, our idea is to use the CASE support that we are developing, called *PlumbingMatic*, to generate it together with other Pipe structures. This information is used in the prototyping phase, and it is reused in the design and development stages. Regarding the contents graph, the envisioned PlumbingMatic tool takes care of the definition of the nodes, anchors and links from a diagrammatic point of view. Moreover it facilitates the identification of these nodes in the XML representation of the contents of the application via the *overmarkup* technique, which is explained in Sec. 5.1 and Sec. 5.2.

Note that during the use of Pipe in conceptualization, the true nature of the contents, or the actual contents themselves, are not relevant. The contents graph only characterizes their existence, and the presence of several anchors for defining links between them.

3.1.3. *Dynamic linking and relationships*

The use of the relationship function r in the definition of the set of links of the application, L_A, simplifies the definition of the browsing semantics by hiding the real nature of the content link (static or dynamic). With static links we will be able to provide an *extensional* definition of function r. With dynamic links we will

need to provide an *intensional* definition of function r in terms of the *generation function* g, whose signature is defined in (4).

$$g : C \times H \times \mathbb{N} \to C \times H \times 2^{H \times \mathbb{N}} \times 2^{H} \tag{4}$$

This function acts as an interface that dynamically builds the destination content and anchor, and two sets with all the source and destination anchors required by the generated content. In this way $g(s, sa, n) = (d, da, GAS, GAD)$ represents the *generated destination content* (d), *generated destination anchor* (da), *generated anchors used as source* (GAS) and *generated anchors used as destination* (GAD) after the activation of the link with source (s, sa, n). The distinction between GAS and GAD is necessary for a new application of function g on the generated content d.

We now use the generation function to extend the definition of the relationship function for covering both the static and dynamic cases. This extension is shown in (5) (Π_i represents the projection on the ith coordinate in a tuple, i.e. Π_{12} represents the first and second coordinates in a tuple).

$$\begin{aligned} r : C \times H \times \mathbb{N} &\to C \times H \\ (s, sa, n) &\to r^0(s, sa, n), \text{ if } (s, sa, n) \in C^0 \times H^0 \times \mathbb{N} \\ (s, sa, n) &\to \Pi_{12} g(s, sa, n), \text{otherwise.} \end{aligned} \tag{5}$$

In this way, function r acts as a *black box* that hides the nature of the content links (static or dynamic) from the browsing semantics. For static links function r will have an extensional definition (in terms of function r^0). For dynamic links function r will have an intensional definition (in terms of function g). There are several functions and sets that must be defined in order to provide Pipe with its complete characteristics, but their description is outside the scope of this paper.

Thus, if we have an application with static and dynamic contents, (6) defines the contents and links in the application.

$$\begin{aligned} C_A &= C^0 \cup genCont(g) \\ L_A &= L^0 \cup genLink(g) \end{aligned} \tag{6}$$

The expressions $genCont(g)$ and $genLink(g)$ denote the contents and links generated by the generation function g, respectively. Again we have omitted its complete characterization.

3.1.4. *Example. Dynamic case*

In Galatea, dynamic content and anchors are generated when a student looks up a term in the *general dictionary*. Galatea's general dictionary contains all the words and expressions in the static contextual dictionary of each text. The definition of every word is obtained by filtering the definitions and grammatical descriptions of the words that are used in similar frame sentences with the same semantic or

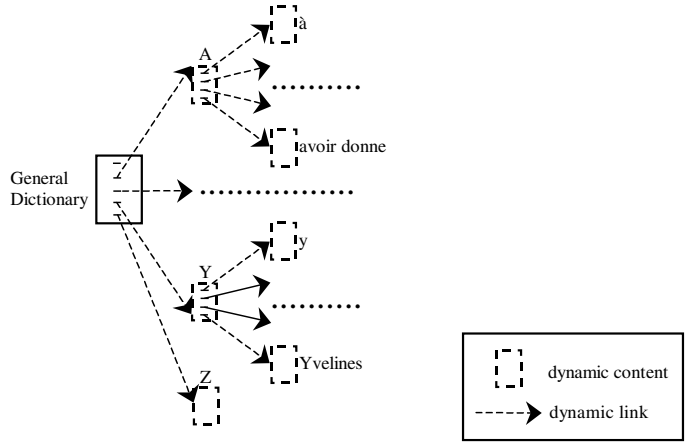

Fig. 2. Dynamic contents for the Galatea General Dictionary.

functional value. The first page of the general dictionary has an index (in this case static) with all the letters of the alphabet. Every letter is linked to a page that contains all the words beginning with the selected letter in the static index. As we have explained, these pages with the words beginning with the same letter are dynamically generated by processing and filtering all the static contextual dictionaries associated with texts that have been incorporated into the Galatea application up to the moment. Finally, the general meaning of a word appears when such a word is selected. Figure 2 depicts the contents graph for the general dictionary.

The Pipe model only defines the signature of the generation function, and every concrete model for a concrete application must provide the definition of this function (in our case, the function that looks up and filters the contextual dictionary of texts). In this way, the generation function acts as an interface in the object-oriented sense: only the description of the behavior is provided, while the concrete behavior of the function is defined in every case. A formalism for defining this function is not provided by the Pipe model. Standard mechanisms such as UML activity and interaction diagrams [46] can be used. This approach is similar to the one proposed in [7], but because the Pipe model is used at the conceptualization stage, it is no longer necessary to provide the explicit function code.

The management (or naming) of dynamic contents in the Pipe model is made via some selected functions that deal with the dynamically generated information. Basically, these functions describe the contents and links generated after several interactions, in the same way as the term[d] $s_n = s_{n-1} + s_{n-2}$ characterizes the n^{th} number of the Fibonacci recurrence although the actual value of s_n is unknown. The full description of the Pipe model and these functions are outside the scope of this paper.

[d]For $n \geq 2$.

Note also that notwithstanding the existence of several texts, a non relational approach was chosen. The need for a well-defined educational approach subjacent to the whole application made it necessary to include a formalism that is able to structure and define the educational contents, thus making the conceptualization and design in terms of E-R diagrams not very suitable. Indeed, at the implementation level, an XML DTD structures the educational contents. Specifically, Pipe captures the semantic relationships among the contents of Galatea without worrying about their inner structure (an XML structure, in fact).

3.2. *Navigational Schema NS*

The *Navigational Schema NS* is an abstraction of the main elements of the application GUI (i.e. screens, panes, buttons) and the navigational paths established between and by them. It is formalized as a tuple $\langle N, A \rangle$ where,

— N is the set of *Nodes* of the application. This set represents the elements of the user interface.
— A is the set of *Arcs* of the application. This set represents the structural and timing relationships established by and between the elements of the user interface.

The navigation is not exactly the same as the interface, but in the end, the navigation has to be represented in a user interface. For example, OOHDM uses a navigation chart to describe the behavior of the nodes when a link is traversed, Amsterdam uses the notion of context for defining the behavior of the components when a link is traversed, the Trellis model uses the simultaneously active states in a Petri net for characterizing the behavior of the hypertext when it is being browsed. In the end, the simultaneous contents that are being displayed and the navigational transformations that occur while traversing links have to be represented in a graphical user interface. Precisely, the Pipe navigational schema defines the minimum elements of a user interface, and the canalization functions relate contents with their navigational access. This is very similar to the OOHDM navigation chart. The main difference is that in OOHDM a context schema and context classes are needed because the access between objects may vary depending on the relationships and the classes accessed, and in Pipe the only existent relationship is the navigational relationship between elements.

3.2.1. *The set of nodes N*

The *Nodes N* of the navigational schema represent the elements of the user interface (whilst in other approaches the term node is used to name the members of the set C_A). In (7) we define this set.

$$N = N_x \cup N_c \cup N_a \tag{7}$$

The *nexus nodes* (N_x) represent the "windows" of the applications. They work as glue for the *container nodes* (N_c). The container nodes represent the "panes"

(concrete or virtual, in the *context* sense of the Amsterdam model [9]) inside the "windows". Container nodes work as the content holders (i.e. text, images, subordinate processes, etc.) of the application. Finally, there are *nexus activator nodes* (N_a) that represent "buttons" inside "windows".

3.2.2. *The set of arcs A*

The elements of this set play two different roles in the abstraction of a GUI: (i) they indicate the basic elements (panes and buttons) that compose a window of the GUI, and (ii) they show the navigational paths that the user can follow between these elements. Formally, let Y, Y_s and $\mathbb{R}$ be $Y = \{connection, path\}$ types of arcs between nodes, $Y_s = \{sConnection, sPath\}$ types of synchronization arcs between nodes, and $\mathbb{R}$ the set of *Real numbers*. Then in (8) we define the *arcs of the navigational schema A*, which represent the structural and timing connections between the elements of the navigational schema (i.e. the nodes).

$$A \subseteq (N \times N \times Y) \cup (N \times N \times Y_s \times \mathbb{R}) \tag{8}$$

Y *connection*s represent structural relationships between nexus and container nodes; Y *path*s represent navigational paths between nexus and container nodes; Y_s *sConnection*s and *sPath*s represent *connection*s and *path*s that are time activated, with $\mathbb{R}$ characterizing the time information attached. Note that these sets can be understood as a typed graph, which we denominate *extended graph*. Figure 3 shows the allowed relationships between the elements of the navigational schema using these arcs.

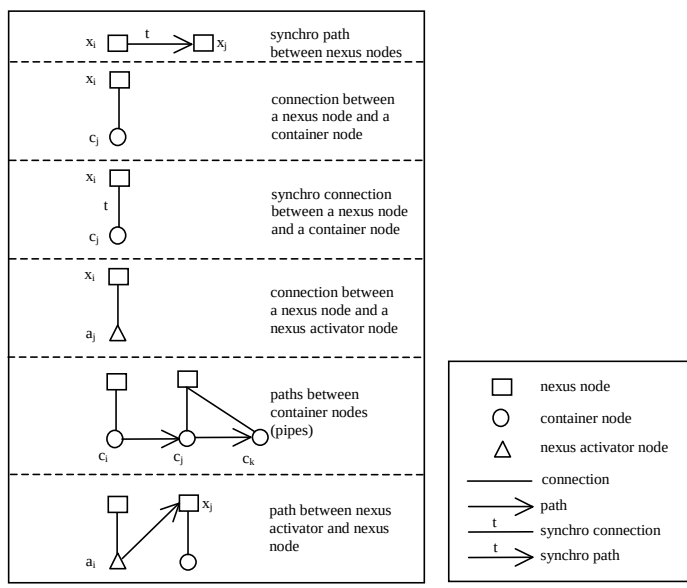

Fig. 3. Permissible relationships between nodes in the navigational schema.

The set of navigational *path*s between container nodes is called *Pipes*, denoted by P and expressed in (9).

$$P = \{(c_i, c_j, path) \in A | c_i \in N_c, c_j \in N_c\} \tag{9}$$

We call them pipes because they are responsible for *canalizing* (*interpreting*) the links (semantic relationships) between contents (i.e. the elements of the set L_A) into the navigational level (i.e. in terms of the set A).

Regarding synchronization, the Pipe approach is similar to that presented in [11], where the basic structural and piping information is decorated with timing information (hence the presence of *sConnection*s and *sPath*s). Moreover, the timing information can be calculated in terms of elements of the navigational schema using four functions that synchronize the playing of the contents assigned to a container node based on other container nodes. We have depicted the functions in (10).

$$\begin{aligned} atBeginning: &\quad N_c \to \mathbb{R} \\ atEnd: &\quad N_c \to \mathbb{R} \\ afterBeginning: &\; N_c \times \mathbb{R} \to \mathbb{R} \\ beforeEnd: &\; N_c \times \mathbb{R} \to \mathbb{R} \end{aligned} \tag{10}$$

These functions permit the playing of a container or nexus node *at the beginning* of the playing of (the content assigned to) a container node, *at the end* of the playing of a container node, an amount of time *after the beginning* of a container node, and an amount of time *before the end* of playing a container node. Note that the expressive power of this approach is similar to the one provided in [9].

3.2.3. *Example*

In the example of the contextual dictionary, the article, sentences and words appear in a window (the nexus node x_1), and the descriptions of the article appear in another window (the nexus node x_2). The first window (x_1) has three panes (the container nodes $c_{1.1}, c_{1.2}$ and $c_{1.3}$) and two buttons that activate two different buttons (the nexus activators $a_{1.1}$ and $a_{1.2}$). The first pane ($c_{1.1}$) has a navigational path (or pipe) with the second one ($c_{1.2}$), and the second pane has another navigational path with the last one ($c_{1.3}$). The second window has two panes (the container nodes $c_{2.1}$ and $c_{2.2}$) and two buttons (the nexus activators $a_{2.1}$ and $a_{2.2}$). Moreover the first pane of the first window ($c_{1.1}$) has a navigational path with the first pane of the second window ($c_{2.1}$), and both panes in the second window ($c_{2.1}$ and $c_{2.2}$) have a navigational path with the first pane of the first window ($c_{1.1}$). Finally, the first pane of the second window ($c_{2.1}$) has a navigational path with the second pane of the second window ($c_{2.2}$). Figure 4 depicts this navigational schema, and Fig. 5 depicts the actual user interface.

Again this schema has been manually constructed, but the PlumbingMatic tool will support its development. Note that in the hypermedia-oriented models studied,

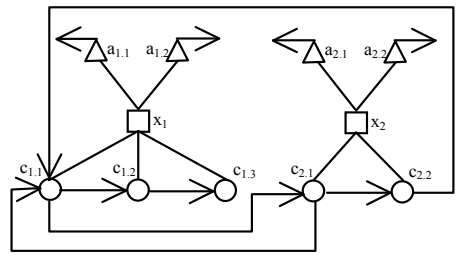

Fig. 4. Navigational schema for the Galatea Contextual Dictionary.

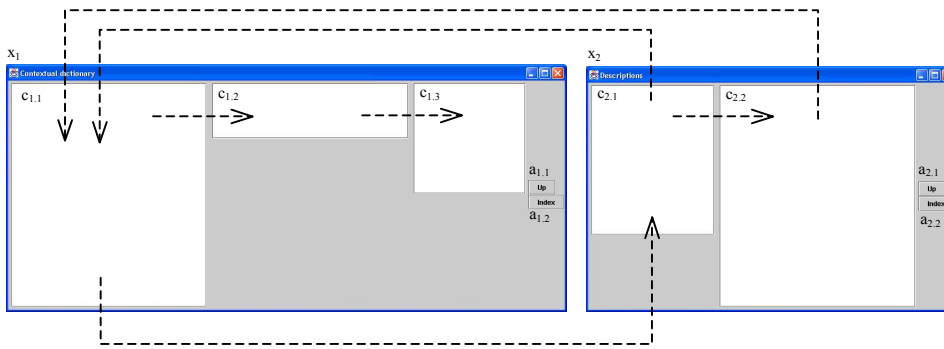

Fig. 5. User interface for the navigational schema for the Galatea Contextual Dictionary. Dashed arrows represent the pipes.

only Labyrinth and OOHDM consider an explicit representation of the GUI. In these cases they are design notations without a formal browsing semantics. Our approach is similar to WebML, but in Pipe, the graphs-based notation is more closely related to the browsing semantics than in WebML. Therefore Pipe's navigational schema allows for the definition of the browsing semantics using the graph structure, whilst WebML needs an external formalism (statecharts, which in turn, is a graph) to provide its browsing semantics.

3.3. *Canalization Functions CF*

These functions are the third and fundamental element of Pipe, and provide it with most of its flexibility. Using canalization functions the same contents graph can be mapped (adapted) to different navigational schemas, and the same navigational schema can be used with different contents graphs. Formally, the *Canalization Functions CF* relate the navigational schema with the contents graph assigned to it. It is a tuple $\langle d, l, p \rangle$ where,

— d is the *content assignation function*. This function relates the contents of the application with the navigational schema (i.e. the GUI).
— l is the *canalization function*. This function interprets the semantic relationships

between contents (content links) into the navigational schema. In this way, the semantic relationships can be considered or ignored at the navigational level.

— p is the *presentation function*. This function provides specific presentation information (e.g. color, font, etc.).

3.3.1. *Content assignation function*

This function assigns a default content to every container node (*null* value, if there is no default content), and the set of contents that is going to appear inside this node in the same way as [11]. In (11) the signature of the *content assignation function* d is presented.

$$d : N_c \rightarrow (C^0 \cup \{null\}) \times 2^C \tag{11}$$

Note that only static contents (or the *null* value that represents no content) can be assigned as default content, whilst the rest of the contents that appear in a container node can be static or dynamic.

3.3.2. *Canalization function*

This function assigns a pipe to a set of content links, capturing the idea of navigational interpretation of content links. In (12) we can find the signature of the *canalization function* l.

$$l : P \rightarrow 2^L \tag{12}$$

Function l presents several restrictions to guarantee the consistent assignation of content links into a navigational level. These restrictions are extremely important when applying browsing semantics. For the sake of conciseness, the details have been omitted. Regarding the graphical representation of the canalization functions, colors and patterns can be used.

3.3.3. *Presentation function*

This function assigns presentation specifications to nodes and contents. It is extensively used by the presentation semantics. In (13) the signature of the *presentation function* p for both nodes and contents is given.

$$\begin{aligned} p &: N \rightarrow PS \\ p &: C \rightarrow CPS \end{aligned} \tag{13}$$

PS is the set of *Presentation Specifications* that can be assigned to a specific node, while CPS is the set of *Content Presentation Specifications* that can be assigned to a specific content.

3.3.4. *Example*

Continuing with the Galatea static example, and as previously mentioned, the navigational schema is composed of two windows (x_1, x_2) with several panes ($c_{1.1}$, $c_{1.2}$, $c_{1.3}$, $c_{2.1}$ and $c_{2.2}$) and nexus activators ($a_{1.1}, a_{1.2}, a_{2.1}$ and $a_{2.2}$). The content assignation function states that the content *article* is the default content (that is, the content shown when the window is activated) for the first pane ($c_{1.1}$) of the first window. The sentences selected from this *article* will appear in the second pane ($c_{1.2}$). Hence the canalization of these content links by the pipe between $c_{1.1}$ and $c_{1.2}$. The contextual meaning of every word selected from a sentence will appear in the third pane ($c_{1.3}$). Hence the canalization of these content links by the pipe between $c_{1.2}$ and $c_{1.3}$. When the user selects the short description of the article, it appears in the first pane of the second window ($c_{2.1}$). Hence the canalization of the content link by the pipe between $c_{1.1}$ and $c_{2.1}$. When the user selects the extended description, it appears in the second pane of the second window ($c_{2.2}$). Hence the canalization of the content link by the pipe that connects ($c_{2.1}$) and ($c_{2.2}$). Finally if the user selects some of the back links from the descriptions to the article, the article appears in the first pane of the first window ($c_{1.1}$). Hence these returning content links are canalized by the pipes that connect the panes in the second window ($c_{2.1}$, $c_{2.2}$) with the first pane in the first window ($c_{1.1}$). Nexus activator nodes provide access to other windows that we are not going to consider for the moment. Figure 6 shows the relationships between the navigational schema and the contents graph (omitting some information).

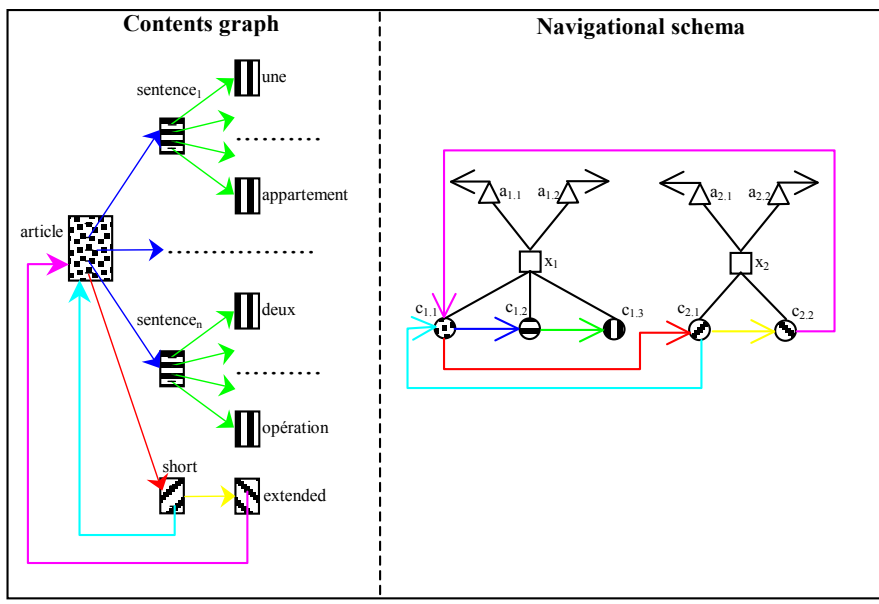

Fig. 6. Relationships between the navigational schema and the contents graph.

Note how the colors and patterns denote the canalization. The patterns denote the assignation of contents to container nodes, while colors provide information about the canalization of links by pipes. The figure was manually built, but again, the PlumbingMatic tool will be able to construct both graphs and to relate them. Note that in this example we have chosen the static dictionary, but the use of the general dictionary has the same complexity[e] due to the definition of the Pipe browsing semantics.

None of the hypermedia-oriented models previously analyzed provide a functional characterization of the relationships between contents/links with its navigational representation, explicitly characterized by means of an abstract GUI. This is one of the most relevant characteristics of the Pipe model, and hence its name. Only WebML presents a similar approach, but it is restricted to relational domains.

3.4. *Browsing semantics*

The *Browsing Semantics BS* represents the dynamic appearance of the application, according to interaction with the user. It is a tuple $\langle a, f \rangle$ where

— a is the *activation function*. This function provides information on the application behavior after the activation of a nexus node (i.e. a window).
— f is the *link activation function*. This function basically provides information on the application behavior after the activation of a content link.

These functions act on nodes and anchors providing the state of the application, which is described by the following sets.

— $Actives \subseteq V =_{\text{definition}} N \cup (N \times \mathbb{R})$. This set presents the set of active nodes with their associated timing information.
— $Show \subseteq S =_{\text{definition}} (N_c \times \{C \cup \{null\}\}) \cup (N_c \times \{C \cup \{null\}\} \times \mathbb{R})$. This set represents the contents that every container node shows (or plays) with their associated timing information. *null* represents no content.

In (14) we can find the signature of function a.

$$a : N_x \rightarrow 2^V \times 2^S \tag{14}$$

The details regarding the definition of functions a and f are omitted, preferring to give an idea of their functionality instead. Function a describes the behavior of the system after the activation of a nexus node (a window). All the nodes connected to the activated nexus node are trigged according to their attached timing information. Moreover, the container nodes show their default content.

The signature of function f appears in (15).

$$f : ((C \times H) \cup \{\perp\}) \times N \times 2^V \times 2^S \rightarrow 2^V \times 2^S \tag{15}$$

[e]Once some auxiliary functions have been introduced.

This function acts on a content and an anchor inside the content that is being displayed in a container node, or on a navigational path (symbol $\perp$ represents nexus activator and time activated paths) in a nexus node, changing the state of the application (i.e. the sets *Actives* and *Show*). The definition of the function takes into account four main cases.

a. Function f is activated by the selection of an anchor that activates a content link.
b. Function f is activated by the triggering of a synchro path established between two nexus nodes.
c. Function f is activated by the selection of a nexus activator node belonging to the navigational schema.
d. Other situations.

Consequently, we define function f according to the enumerated cases.

a. Function f is activated by the selection of an anchor that activates a content link. Then, the source is in a container node, and the destination contents are in other container nodes. There are two options depending on whether the source and container nodes are not connected to the same nexus node (i.e. they are panes of different windows) or they are connected to the same nexus node (i.e. they are panes of the same window).
a.1. The destination content nodes are in another nexus node. In this case, the nexus node where the link originated is deactivated. Then the nexus node connected to the container nodes, which are the destination of the pipes, is activated. Next, all the container nodes show their default content, except those nodes that are the destination of the pipes. These destination nodes show the content that is the end of every content link. Due to the characterization of static and dynamic linking via the relationship function r the actual nature of the link (static or dynamic) is hidden from the browsing semantics.
a.2. The destination content nodes are in the same nexus node. In this case, every destination container node of a pipe that canalizes an activated link shows the content (static or dynamic) accessed by the link and it stops showing the previous content.
b. Function f is activated by the triggering of a synchro path established between two nexus nodes (note that there is no source anchor, and the element $\perp$ is used instead). In this case, the destination nexus node is activated according to the timing information attached, and the source nexus node is deactivated.
c. Function f is activated by the selection of a nexus activator node belonging to the navigational schema (note that again, in this case, there is no source anchor, and the element $\perp$ is used instead). In this case, the destination nexus node is activated, and the nexus node where the source nexus activator node is placed is deactivated.
d. In other situations, function f acts as the identity function.

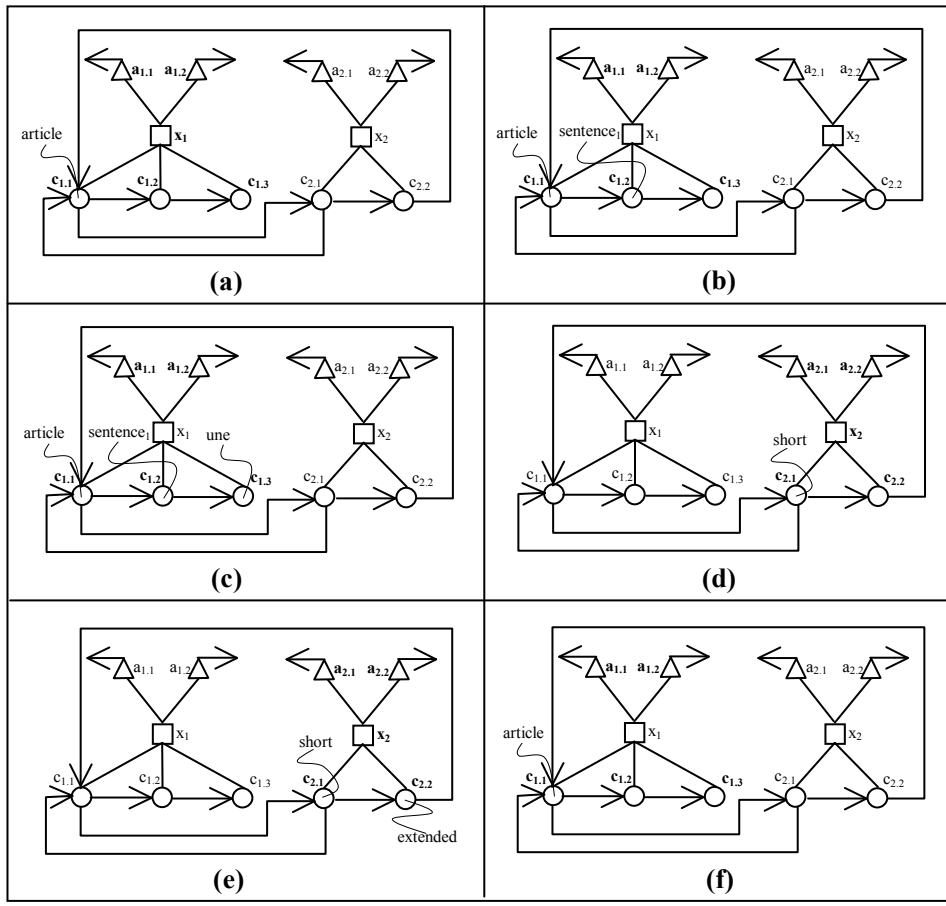

Fig. 7. An example of the browsing semantics.

We are aware that this browsing semantics (and its mathematical representation) is too complicated to be clearly understood by a *broad audience*, but this is not our main aim. The Pipe browsing semantics is used to define a tool that can handle a hypermedia application by using Pipe structures. In other words, this browsing semantics is the kernel of a CASE tool for the development of hypermedia applications.

Concerning graphical notation, the activated nodes are represented by a dot inside the node or by putting its name in bold letters (as in this paper).

3.4.1. *Example*

Let us look at a trace of execution as described by BS. Figure 7 shows the behavior described by the BS. In (a) we begin the navigation by activating the nexus node x_1. Then the container node $c_{1.1}$ shows its default content, the *article*. In (b) we

$Actives_1 = \{x_1, c_{1.1}, c_{1.2}, c_{1.3}, a_{1.1}, a_{1.2}\}$ $Show_1 = \{(c_{1.1}, article), (c_{1.2}, null), (c_{1.3}, null)\}$ **(a)**	$Actives_2 = Actives_1$ $Show_2 = \{(c_{1.1}, article), (c_{1.2}, sentence_1), (c_{1.3}, null)\}$ **(b)**
$Actives_3 = Actives_2$ $Show_3 = \{(c_{1.1}, text), (c_{1.2}, sentence_1), (c_{1.3}, une)\}$ **(c)**	$Actives_4 = \{x_2, c_{2.1}, c_{2.2}, a_{2.1}, a_{2.2}\}$ $Show_4 = \{(c_{2.1}, short), (c_{2.2}, null)\}$ **(d)**
$Actives_5 = Actives_4$ $Show_5 = \{(c_{2.1}, short), (c_{2.2}, extended)\}$ **(e)**	$Actives_6 = \{x_1, c_{1.1}, c_{1.2}, c_{1.3}, a_{1.1}, a_{1.2}\}$ $Show_6 = \{(c_{1.1}, article), (c_{1.2}, null), (c_{1.3}, null)\}$ **(f)**

Fig. 8. State of the application in every case.

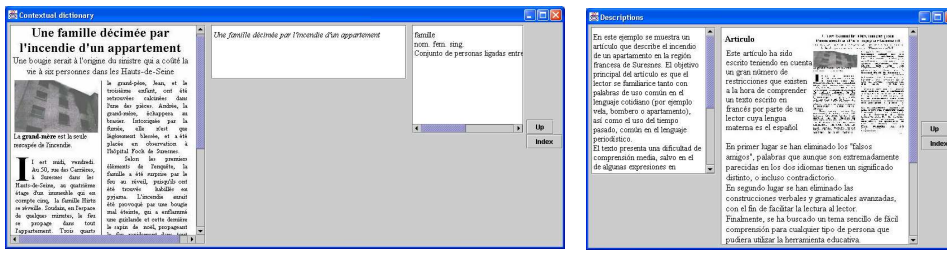

Fig. 9. Navigation in the Galatea Contextual Dictionary. In this example these two windows are never simultaneously shown.

can see what happens if the user selects the first sentence from the article. As the pipe between $c_{1.1}$ and $c_{1.2}$ canalizes such links, the sentence appears in node $c_{1.2}$ (case **a.2** of the BS). The same behavior occurs if the user selects the first word in the sentence, but using the pipe between $c_{1.2}$ and $c_{1.3}$ (c). If the user selects the link between the article and the short description, this description appears in the first pane of the second window because the content link is canalized by the pipe between $c_{1.1}$ and $c_{2.1}$ as (d) depicts (case **a.1** of the BS). (e) depicts the situation when the link between descriptions is traversed (case **a.2** of the BS). Finally (f) depicts the situation after the link between the extended description and the article is traversed (case **a.1** of the BS). Note that traversing the link between the short description and the article produces the same result.

The canalization functions are responsible for interpreting the content links at a navigational level. In the previous example, case (f) is obtained because the content link between the extended description and the article is canalized by the pipe that connects the nodes $c_{2.2}$ and $c_{1.1}$. Otherwise, the link is ignored because case **d** of the link activation function is selected.

The state of the application in every case is described in Fig. 8.

Finally, Fig. 9 depicts the two windows of the example.

From a conceptual viewpoint this semantics is very similar to that presented in FOHM, whilst graphically it is closer to those provided by the HAM or Trellis approach. Note that the Pipe model does not support the Model-View-Controller architecture feature supported by models such as OOHDM. In this way a

default built-in browsing semantics can be provided with less effort (like in HDM or WebML). This browsing semantics is the key for the development of prototypes, and in our opinion, the possibility of obtaining early prototypes at the conceptualization-prototyping stage is more important than being able to capture all the characteristics of the hypermedia applications. As Fraternali states [4], a prototype is typically built prior to design and with a simplified architecture, e.g. as a set of manually implanted pages containing samples of the application content, which emulate the desired appearance and behavior. This is the reason for initially choosing a two-layer architecture. Of course, in design more advanced architectures than the simplified architectures used in conceptualization-prototyping can be used.

According to the *default* browsing semantics, every time that a window is left, all the computational processes that are running inside it are stopped. In turn, subordinate processes implement a Java interface with a `stop()` method that stops any computational activity that the subordinate process may be running. A more advanced browsing semantics could be provided, but this would complicate its definition.

3.5. *Presentation Semantics PS*

The *Presentation Semantics* PS is similar to the browsing semantics, but functions a and f include presentational information via function p. In (16), the modifications in set S, which includes this information, are shown.

$$S = (N_c \times \{C \cup \{null\}\} \times PS \times CPS) \cup (N_c \times \{C \cup \{null\}\} \times \mathbb{R} \times PS \times CPS) \tag{16}$$

Note that this minimum variation in this set represents very little change in the browsing semantics, and introduces specific information about the style in the application.

4. Plumbing Process Model

In this section we describe the *Plumbing* process model, an evolution of the process model proposed by Fraternali [4] with some influences of the model proposed by Ginige and Lowe [16].

4.1. *Fraternali/Ginige-Lowe process model*

Figure 10 shows the Fraternali/Ginige-Lowe process model, an evolutive model with two loops.

The first loop begins with a *requirements analysis*, where developers determine the mission of the software application identifying prospective users and defining the nature of its information base. Later, in *conceptualization* the application is represented through a set of abstract models (Pipe in the case of Plumbing) that

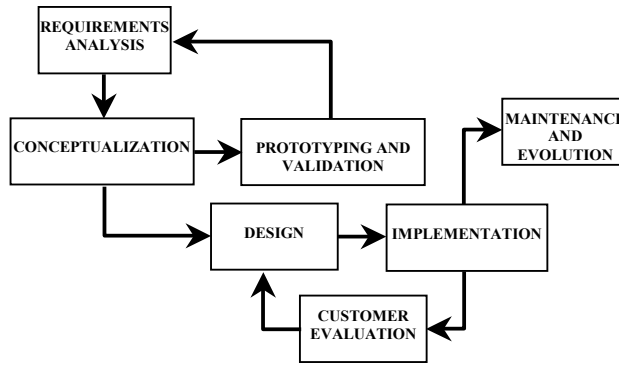

Fig. 10. Fraternali/Ginige-Lowe process model.

characterize the main features of the application: structure, navigation and presentation. In the hypermedia context, conceptualization differs from the same activity in the *classic* analysis, since the main interest is to identify the contents, links and navigational schema, instead of the specific software requirements and their software representation. In *prototyping and validation* the user evaluates reduced versions of the application using a prototype which simplifies the architecture of the solution. The user's evaluation provides valuable information prior to design. In our opinion, in the case of hypermedia applications, the prototyping has more relevance than in traditional software development because the complexity and relevance of the contents and the GUI are greater, and need to be evaluated in detail by the user.

At the *design* stage, developers transform the abstract schemas provided by the conceptualization phase into low level representations, closer to the demands of the implementation. Also, at this point developers determine the architecture of the solution, for example using patterns like those provided by [36]. Developers build the application and provide the contents at the *implementation* stage. Naturally, the contents must be represented according to the selected architecture. Finally, in *evolution and maintenance* developers modify the application in response to changes in software requirements, and solve the bugs and problems that might eventually appear in the application. These changes may force modifications in the structure, navigation or presentation of the application, and for that reason an iterative enacting of the process model is the best way to manage them.

4.2. *Plumbing*

This section analyzes the *Plumbing* process model [44] a specialization of Fraternali's model where Pipe is used to guide the conceptualization and prototyping stages. As previously mentioned, Plumbing is not concerned with the design and development stages. Figure 11 depicts the conceptualization and prototyping phases in Plumbing.

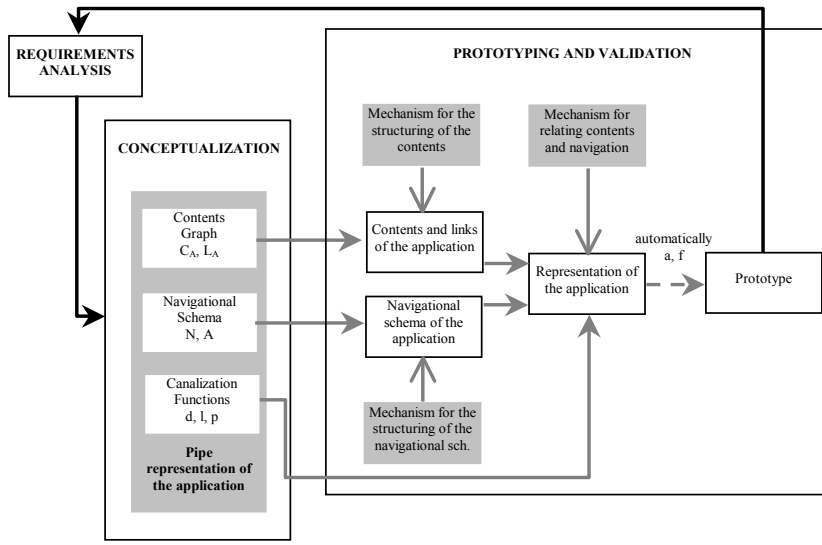

Fig. 11. Conceptualization and prototyping in the Plumbing process model.

4.2.1. *Requirements analysis and conceptualization*

The requirements analysis phase does not differ from Fraternali's, but in Plumbing the conceptualization stage is totally directed by Pipe. In traditional software applications, the requirements analysis produces a *software requirements specification* that guides the design of the application. In hypermedia applications this specification must facilitate the design without compromising it, and the conceptualization stage offers the solution to this problem. This stage needs mechanisms with the capacity to express the contents, links and navigational schema of a hypermedia application in an abstract way. In our approach, Pipe provides all the mechanisms for representing this information.

According to Plumbing, at the conceptualization stage developers must identify the contents graph of the application, that is, they must determine the sets of contents C_A and links L_A of the specific application.

Once the contents and their relationships are defined, developers must specify the navigational schema of the application, that is, the sets of nodes N and arcs A. After specifying the navigational schema, developers must relate the contents graph with the navigational schema by defining the canalization functions and, optionally, the presentation function (functions d, l and p, respectively). They can choose to define these functions at the same stage as the navigational schema, or at an independent stage. To obtain a more general approach, Fig. 11 shows them as independent activities.

At the end of the conceptualization stage we have a complete Pipe representation of the application. It is important to note that although we have given a formal flavor to the Pipe model, we realize that a manual specification of the sets

and functions $(C_A, L_A, N, A, d, l, p)$ is a tedious task. For that reason we are building the PlumbingMatic CASE tool to reduce this problem. In conceptualization, PlumbingMatic must provide a view for the definition of the nodes and links of the contents graph as well as another view for the definition of the navigational schema. Using both graphical views of the application, canalization functions could be specified making *drag and drop* of contents and links.

4.2.2. *Prototyping and validation*

At this stage the user must validate the application that is being developed. The idea is to reuse the Pipe representation defined in the previous phase. Therefore, several languages or formalisms capable of expressing all the Pipe structures are needed. The first one must be capable of structuring the contents graph of the application, and the second one its navigational schema. With these two formalisms available, all the contents, links and abstracts features of the application GUI can be described. Finally, a third formalism must be used to relate both aspects of the application (as canalization functions do), obtaining the description of the whole application.

This description of the application must be capable of being processed by a program that implements the Pipe browsing semantics, providing a prototype of the final application. This program should be integrated into a CASE tool able to generate the navigational schema and its relationships with the contents graph. Of course, the formalisms used must be *in tune* with the design and development phase, or must be flexible enough to be employed in a wide range of domains. In any case, a first *ultra*-fast prototype can be deployed using the Pipe representation of the application. In this prototype the structural information provided by the contents graph (name of the content, number of anchors and content links) can be used as an abstract representation of the actual contents of the application. This helps in defining the hypermedia structure of the contents and their navigational behavior.

Users evaluate the prototype developed at the final stage, and the loop iterates if changes are required. The presence of the Pipe representation and the prototyping engine facilitates the iteration until the prototype needs no further changes.

For the prototyping phase, the Plumbing process model does not set any specific structuring language or formalism, and consequently does not imply any technology in the generation of prototypes. Although this is the true nature of a process model, we are going to set up some mechanisms and languages to make the Plumbing process model more functional.

5. PlumbingXJ Process Model

PlumbingXJ [44] is a specialization of Plumbing where XML is the selected mechanism for structuring the contents graph and the navigational schema of the

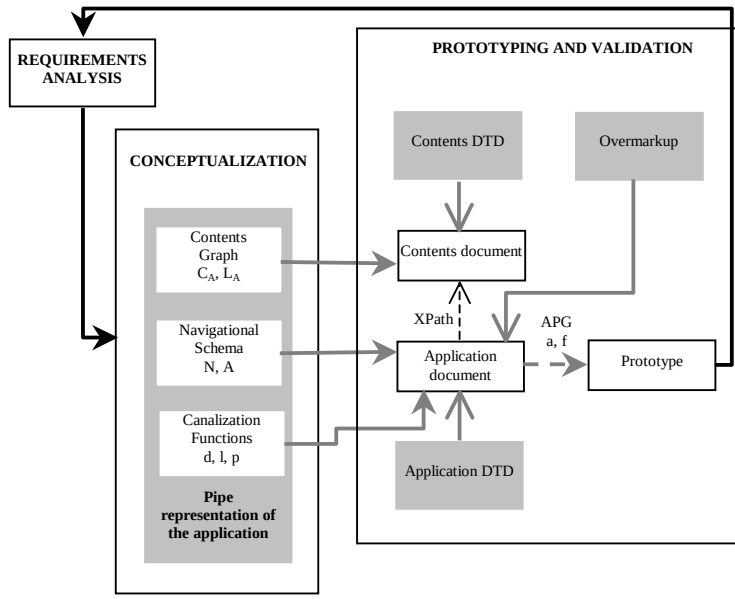

Fig. 12. Conceptualization and prototyping in PlumbingXJ process model.

application, and where Java is the selected mechanism for defining the subordinated process and for defining the tool that automatically generates the prototypes (hence the *XJ*). Figure 12 describes the conceptualization and prototyping stages in PlumbingXJ.

As this figure depicts, the only changes between Plumbing and PlumbingXJ appear at the prototyping and validation stage. Now XML defined languages are used to describe the Pipe structures. There are two reasons for choosing this language: today XML is accepted as a universal interchange format [47], [48], and it provides a great structuring capacity [49]. Both characteristics make XML a perfect choice for the characterization of hypermedia applications [37].

5.1. *Contents DTD*

The *contents DTD (Document Type Definition)* structures the contents and links of the hypermedia application. Due to the heterogeneous structure of the contents that can appear in a hypermedia application it is specific for every case. There are almost no restrictions on this type of DTD, although it provides a tree structure, and a (Pipe) graph structure must be obtained. The graph structure is built from the tree by using a few specific attributes (that basically define the anchors) and via the *overmarkup* technique described later.

The *contents document* is an instance of the contents DTD. It can include text and *non XML data* like images or subordinate processes. Subordinate processes implement a specific Java interface that allows the hypermedia applications to load

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE contents SYSTEM "cLEF.dtd">
<contents>
  <article image="c:\plumbingXJ\GAP\documents\fire.jpg">
    <anchor href="s1" coords=" 0,0,315,0,315,55,0,55,0,0"/>
    <anchor href="s2" coords=" 0,55,315,55,315,100,0,100,0,55"/>
    ..............................................
    <anchor idLink="l1" href="short" coords="0,600,150,600,150,630,0,650,0,600"/>
  </article>
  <sentences>
   <sentence id="s1">
     <word href="une1">Une </word>
     <word href="famille1">famille </word>
     <word href="decimee1">décimée </word>
     <word href="par1">par </word>
     <word href="l1">l'</word>
     <word href="incendie1">incendie </word>
     <word href="d1">d'</word>
     <word href="un1">un </word>
     <word href="appartement1">appartement</word>
   </sentence>
   ..............................................
```

Fig. 13. Fragment of the contents document for the Galatea contextual dictionary.

them dynamically when they are referenced. These subordinate processes can implement any functionality with the only constraint that they cannot interfere with the navigational behavior. Otherwise, the browsing semantics would be much more complicated. At present we are not able to manage the dynamic contents at the prototyping stage, and a static simulation is the only available solution. However we are working on a method, similar to the one provided for the subordinate processes.

Figure 13 illustrates part of the contents document for the Galatea contextual dictionary.

Note that the *article* identified in the Pipe contents graph (Fig. 1) is represented at the prototyping stage by a `jpg` file. This is irrelevant for the Pipe characterization. Moreover, we have chosen `href` attributes, which are similar to those of HTML, but this choice is not relevant to our approach. Indeed the inline inclusion of the anchors is an implementation decision that can be (effortlessly) replaced by external XLink [50] links.

Using XML representations in the definition of the contents graph introduces a problem: in the XML representation the nodes of the contents graph are not identified because they are embedded in the XML-tree structure. The identification of these nodes is made using XPath [51] expressions. These expressions *cut the tree*, obtaining the nodes of the contents graph. The PlumbingMatic tool can help in this task, allowing a graphical identification of these nodes and automatically defining the XPath expressions. In addition, this tool can help in the identification of the nodes, anchors and links of the contents graph in the XML document. In this way, the canalization functions are automatically defined because they have been defined at the conceptualization stage, and only the actual contents and links of the application need to be provided. As presented in the next section, this information is included in the description of the application GUI, where the actual elements

of this GUI reference the nodes of the contents graph using the XPath references. This is what we call *overmarkup*. Moreover, information about the contents link canalization is also included.

5.2. *Application DTD*

The *application DTD* structures the navigational schema of the hypermedia applications. It is unique for all types of applications and uses a closer GUI terminology instead of the Pipe vocabulary. This approach is similar to the one presented in XUL and UIML, but restricted to the hypermedia domain and the Pipe browsing semantics. Consequently there is no need to explicitly express the behavior of the application beyond the anchor selection, which is encoded in the Java engine that builds the prototypes.

The *application document* is an instance of this DTD and not only provides the elements of the GUI and their navigational relationships, but also encodes the canalization functions. The transition from the Pipe Navigational Schema to the application document is direct and can be directly accomplished by the PlumbingMatic tool. Notice that Fig. 4 depicts the windows, panes and buttons of the application, and this structure is directly representable in XML. The overmarkup technique and specific attributes are responsible for the encoding of the canalization.

A default content link canalization is made. A default pipe is defined for every container node. This pipe canalizes the content links that have this container node as source. If there is any other content link not assigned to this default pipe, its explicit canalization must be provided. The PlumbingMatic tool will permit the graphical selection of contents and links, and its *drag and drop* assignation to the navigational schema.

Figure 14 depicts a fragment of the application document for the Galatea example (bold text represents overmarkup). The `href` attributes in the contents document (Fig. 13) represent the destination of the links, and the information about the link canalization is included in the application document (Fig. 14) via `linksDestination` element (in italics). The attribute `pane` of this element characterizes the destination container node of the default pipe, whilst the attribute `pane` of the element `specific` denotes the destination container node of any link not canalized by the default pipe. In the example, in the first pane of the first window (`p1.1`), pane `p1.2` is the destination of the default pipe, and the content link that connects the article with its short description (link `l1`) is explicitly canalized by the pipe whose destination container node is `p2.1`. In this way, the canalization functions depicted in Fig. 6 are represented in XML format.

The Java application *Automatic Prototypes Generator (APG)* processes the application document and produces the desired prototype. In turn, Fig. 9 depicts the prototype generated using APG.

The navigational schema of the application is generated by the APG. This schema is a Java Swing [52] skeleton that supports the GUI of the application. The

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE application SYSTEM "application.dtd">
<application id="app4">
  <window id="w1">
    <name>window 1</name>
    <pane id="p1.1">
      <paneContent>
       <defaultContent>/contents/article</defaultContent>
      </paneContent>
      <linksDestination pane="p1.2">
        <specific idLink="l1" pane="p2.1"/>
      </linksDestination>
    </pane>
    <pane id="p1.2">
      <paneContent>
       <groupContent>/contents/sentences</groupContent>
      </paneContent>
      <linksDestination pane="p1.3"/>
    </pane>
    ..................................................
```

Fig. 14. Fragment of the application document for the Galatea contextual dictionary.

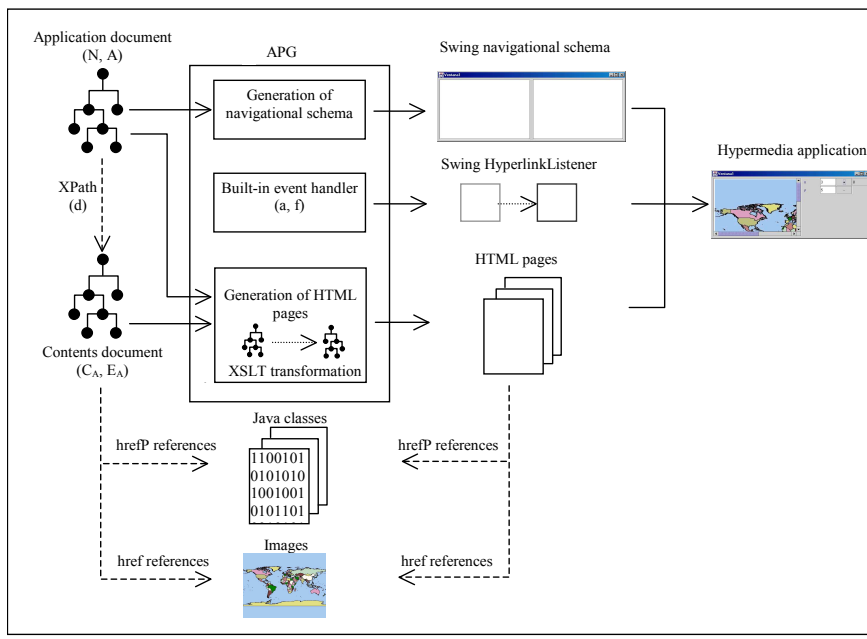

Fig. 15. Description of the APG operation.

contents that appear in this Java Swing skeleton are HTML pages generated by the APG from the contents document and the XPath expressions that appear in the application document. An XSLT transformation [53] produces the transition from XML to HTML. This transformation is always the same and uses the hypermedia structure encoded in the contents document via some selected attributes (e.g. `href` attribute). In this way the contents document can be structured in any desired way

because the XSLT only needs a few attributes to produce the HTML pages. This is similar to the concept of *Architectural Form* processing presented by *HyTime* [39]. Finally, the behavior of the application is provided by an event controller that implements Pipe browsing semantics via a Java Swing `HyperlinkListener`. APG processing is described in Fig. 15.

Naturally non XML data, like figures and subordinate processes, can be referenced from the contents document. In particular, subordinate processes are implemented by Java classes that are dynamically loaded by the application. The resulting HTML pages can contain everything that a Swing `JEditorPane` can display, which in Java 1.4.1 are HTML and RTF documents. Now the support of Web-datatypes is restricted to the support that the Swing `JEditorPane`s provide (that in Java 1.4.1 is HTML 3.2).

Moreover, the generated application can be an HTML-pure one, where there is no need for a Java running application. We have begun with the "Java solution" because it was clearer in the identification of the characteristics of the application represented by the Pipe structures.

The separation of layers in the application (like those applications that run in hypertext systems according to the Dexter model) allows for a fast generation of prototypes where changes in the navigational schema have no effect at the content level, or changes at the content level (maintaining the structure referenced by XPath expressions) have no effect on the navigational schema.

The construction of the navigational schema and its relationships with the contents (according to Pipe structures) can be done manually, or via a CASE tool (like PlumbingMatic). The PlumbingMatic tool can help in the assimilation of the XML contents and their relationships into the contents graph identified at the conceptualization stage. As previously mentioned, the overmarkup technique and direct assignation of anchors and links are responsible for this assimilation.

At the last stage, the customers evaluate the prototype, and when the hypermedia structure is stable enough, the design stage begins. The use of XML in the structuring of the document offers an easy transition to HTML (or another format) using XSLT transformations. This approach is preferable to the direct use of HTML in prototyping due to the flexibility and structuring power provided by XML [49].

Regarding design-implementation, PlumbingXJ does not fit any technology. For example, if Java Server Pages [54] are used, an adequate design formalism should be used and at the implementation level the XML documents produced in prototyping should be (totally or in part) reused.

5.3. *Galatea example*

There are no specific implementation techniques tied to PlumbingXJ, thus any method that uses the XML encoded information can be applied. For example, the work of Sierra [55] can be a *natural* implementation technique. This is not the case of Galatea because there was an original implementation designed for CD-ROM

Fig. 16. Web version of the Galatea contextual dictionary.

deployment. This version was based on markup technologies [45], and currently we are working on its Web translation. The first results can be found at [56]. Figure 16 shows the final application corresponding to the prototype automatically generated in Fig. 9.

The existence of the Pipe representation of the application, and the prototypes previously generated simplified the development of this application as the Fraternali process model had anticipated. Of course, there were final changes, but the main hypermedia structure remained unchanged.

Although the example included in this paper is not a web application, the expressive power of Pipe allows it to characterize a great number of Web sites. For example, Sun's Java website [54], which is mainly static, can be fully modelled using the Pipe approach. A moderately dynamic website like Apple's site [57] can also be modelled without problems. Highly dynamic websites like Amazon [58] can be modelled, but in this last case, the Pipe structures only identify the skeleton of the application in terms of the contents graph. The characterization of the dynamic components in Amazon must be represented in terms of the formalism used to describe the generation function (e.g. UML). This is because Amazon is nearer to a Computational Hypermedia Application [25] than to a non-Computational Hypermedia Application.

6. Conclusions

Nowadays, developers can choose between several fine hypermedia-oriented models to characterize the design of hypermedia applications. All these models present significant advantages and our work does not seek to replace the array of tools that designers have at their disposition, but rather to expand this array.

There are several distinctive features in the Pipe model. The first, and most important, is the flexibility and high level of abstraction provided by its two-graph representation of hypermedia applications and by the set of functions responsible for the mapping between both graphs. The second one is its ability to represent dynamic applications at this abstract level. In addition, Pipe is a model that has a browsing semantics that allows for the automatic generation of prototypes.

All these features refer to the expressive power of Pipe, however, one of the major advantages provided by this model is found in its use. Most hypermedia-oriented models are intended for design, but Pipe is intended for conceptualization, its goal being to assist in the design process rather than solving it. This is possible due to the abstraction level of the model, which enables an easy transition from model representation to final application.

Plumbing is the name of the process model that integrates the Pipe model in the conceptualization prototyping loop. The Plumbing process model is a specialization of Fraternali's well-known process model and it benefits from both Pipe and Fraternali's most relevant features. As has been repeatedly observed, during the development of hypermedia applications the major number of iterations appear at the conceptualization and prototyping stages, which are less expensive than the design and development stages. Moreover, in that first loop, interaction between customers and developers play a decisive role and during this stage customers can be isolated from implementation details. Furthermore, Plumbing is abstract enough to be open to different implementation instances.

PlumbingXJ is one of these instances where XML documents support the representation of Pipe graphs and functions, Java subordinate processes represent the non-hypermedia behavior of the application, and a Java application implements the Pipe browsing semantics. PlumbingXJ presents several distinctive features. In the first place the prototyping stage is directed by the Pipe structures provided at the conceptualization phase. In this way, Pipe features allow for the automatic generation of prototypes, and for independent changes in the navigational schema or in the structure and relations of the content elements of the application. In addition, the use of XML-based languages to structure the contents domain (via the contents DTD) provides a great readability level. XML syntax also provides a universal interchange format, and the provision of high-level application development techniques, easier to use than direct coding (but less flexible, on the other hand). Finally the combination of XML and Java present the advantage of platform independence.

At present, we are working on the APG finalization in order to deal with the full expressive power of Pipe. Solving how to include dynamic contents is our main goal. At this stage we are using implementation techniques similar to those used for the inclusion of subordinate processes.

We are also studying the instance of Plumbing where content elements or data are stored in a relational database. This implies the substitution of PlumbingXJ XPath expressions by SQL queries in the application document. We have to analyze the consequences of this substitution at formal and practical levels.

The APG is the kernel of the PlumbingMatic application. The existence of a CASE tool that integrates the Pipe structures and the automatic generation of prototypes will simplify the PlumbingXJ process model and will solve one of the principal shortcomings of hypermedia-oriented models.

Finally, we are applying PlumbingXJ to several applications that we are creating to test different ways of integrating our conceptualization and prototyping techniques with the process of design and development, our goal being to define a specific methodology combining both stages in a way similar to the combination of the Amsterdam model and the SMIL language.

Acknowledgements

The Spanish Committee of Science and Technology (TIC2001-1462 and TIC2002-04067-C03-02) has supported this work. We would also like to thank the anonymous reviewers for their very useful comments.

References

1. V. Bush, As We May Think, *The Atlantic Monthly* **176** (1945) 101–108.
2. F. Garzotto, P. Paolini and D. Schwabe, HDM: A model-based approach to hypertext application design, *ACM Transactions on Information Systems* **1** (1993) 1–26.
3. P. Díaz, I. Aedo and F. Panetsos, Labyrinth, an abstract model for hypermedia applications. Description of its static components. *Information Systems* **4** (1994) 33–45.
4. P. Fraternali, Tools and Approaches for Developing Data-Intensive Web Applications: A Survey, *ACM Computing Surveys* **3** (1999) 227–263.
5. J. Nanard and M. Nanard, Hypertext design environments and the hypertext design process, in *Communications of the ACM* **8** (1995) 49–56.
6. J. Rumbaugh, I. Jacobson and G. Booch, *The Unified Modeling Language. Reference Manual* (Addison-Wesley, 1999).
7. P. Díaz, I. Aedo and F. Panetsos, Modelling the dynamic behaviour of hypermedia applications, *IEEE Transactions on Software Engineering* **6** (2001) 550–572.
8. F. Halasz and M. Schwartz, The Dexter Hypertext Reference Model, *Communications of the ACM* **2** (1994) 30–39.
9. L. Hardman, D. C. A. Bulterman and G. van Rossum, The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter Model, *Communications of the ACM* **2** (1994) 50–62.
10. B. Campbell and J. M. Goodman, HAM: A general purpose hypertext abstract machine, *Communications of the ACM* **7** (1988) 856–861.
11. P. D. Stotts, R. Furuta, Petri-Net-Based Hypertext: Document Structure with Browsing Semantics, *ACM Transactions on Office Information Systems* **1** (1989) 3–29.
12. F. Tompa, A Data Model for Flexible Hypertext Database Systems, *ACM Transactions on Office Information Systems* **1** (1989) 85–100.
13. T. Isakowitz, E. A. Stohr and P. Balasubramanian, RMM: A methodology of structured hypermedia design, *Communications of the ACM* **8** (1995) 34–43.
14. D. Schwabe, G. Rossi and S. D. J. Barbosa, Systematic Hypermedia Application Design with OOHDM, in *Proc. 7th ACM Conference on Hypertext*, Washington D.C, (March 1996).
15. C. Barry and M. A. Lang, A Survey of Multimedia and Web Development Techniques and Methodology Usage, *IEEE Multimedia* **3** (2001) 52–60.

16. G. B. Wills, R. M. Crowder, I. Heath and W. Hall, *Industrial Hypermedia Design* (Southampton University, M98-2 4, 1998).
17. V. Balasubramanian, A. Bashian and D. Porcher, A Large-scale Hypermedia Application Using Document Management and Web Technologies, in *Proc. Hypertext 97*, Southhampton (April 1997).
18. M. Thüring, J. Haake and J. Hannemann, What's Eliza doing in the Chinese Room? Incoherent hyperdocuments and how to avoid them, in *Proc. Hypertext 91*, San Antonio (December 1991).
19. M. Thüring, J. Hannemann and J. M. Haake. Hypermedia and Coginition: Designing for Comprehension, *Comunications of the ACM* **8** (1995) 57–66.
20. N. Koch, A. Kraus and R. Hennicker, The Authoring Process of the UML-based Web Engineering Approach (Case study), in it Proc. First International Workshop on Web-Oriented Software Technology, Valencia (June 2001).
21. R. Bodner and M. Chignell, Dynamic Hypertext: Querying and Linking, *ACM Computing Surveys* **4** (1999).
22. S. Ceri, P. Fraternali and A. Bongio, Web Modeling Language (WebML): A Modeling Language for Designing Web Sites, in *Proc. www9 Conference*, Amsterdam (May 2000).
23. World Wide Web Consortium, *Extensible Markup Language (XML) 1.0.* (http://www.w3.org/TR/REC-xml, 1998).
24. P. Chen, The Entity-Relationship Approach: toward a unifed view of data, *ACM Transactions on Database Systems* **1** (1976).
25. G. Rossi, D. Schwabe and A. Garrido, Designing Computational Hypermedia Applications, *Journal of Digital Information* **4** (1999) 1–15.
26. O. M. F. De Troyer and C. J. Leune, WSDM: A user centered design method for Web sites, in *Proc. First International Workshop on Web-Oriented Software Technology*, Valencia (June 2001).
27. J. Gómez, C. Cachero and O. Pastor, Conceptual Modeling of Device-independent Web Applications, *IEEE Multimedia* **2** (2001) 26–39.
28. O. Pastor, S. Abrahao and J. Fons, An Object-Oriented Approach to Automate Web Applications Development. *Electronic Commerce and Web Technologies, International Conference, EC-Web 2001* Munich, Germany (September 2001).
29. F. Frasincar, G.-J. Houben and R. Vdovjak, Specification Framework for Engineering Adaptive Web Applications. *The Eleventh International World Wide Web Conference*, Honolulu, Hawaii (May 2002).
30. D. E. Millard, L. Moreau, H. C. Davis and S. Reich, FOHM: A Fundamental Open Hypertext Model of Investigating Interoperability between Hypertext Domains, in *Proc. 11th ACM Conference on Hypertext and Hypermedia*, San Antonio (may 2000) pp. 93–102.
31. D. Millard, H. Davis and L. Moreau, Standardizing Hypertext: Where Next for OHP, in *OHS/SC 2000*, eds. S. Reich and K.M. Anderson (Springer-Verlag, LNCS, Berlin Heidelberg, 2000) pp. 1–11.
32. L. Olsina, Functional View of the Hypermedia Process Model, in *Proc. 5th International Workshop on Engineering Hypertext Functionality at ICSE'98*, Kyoto (April 1998).
33. D. B. Lowe, A.J. Bucknell and R. G. Webby, Improving Hypermedia Development: A Reference Model-Based Process Assessment Method, in *Proc. 10th ACM Conference on Hypertext and Hypermedia*, Darmstadt (1999).
34. R. S. Pressman, *Software Engineering: A Practitioner's Approach, 5th Edition* (McGraw-Hill, 2001).

35. I. Jacobson, G. Booch and J. Rumbaugh, *The Unified Software Development Process* (Addison-Wesley, 1999).
36. G. Rossi, D. Schwabe and F. Lyardet, Designing Hypermedia Applications with Objects and Patterns, in *International Journal of Software Engineering and Knowledge Engineering* **6** (1999) 745–766.
37. A. Navarro, A. Fernández-Valmayor, B. Fernández-Manjón and J.L. Sierra, Using Analysis, Design and Development of Hypermedia Applications in the Educational Domain, in *Computers and Education: Towards an Interconnected Society*, eds. M. Ortega and J. Bravo (Kluwer Academic Publishers, The Netherlands, 2001) pp. 251–260.
38. World Wide Web Consortium, *HTML 4.01 Specification* (http://www.w3.org/TR/html4/, 1999).
39. International Standards Organization, *Hypermedia/Time-based Structuring Language (HyTime)* (ISO/IEC IS 10744.1992, 1992).
40. World Wide Web Consortium, *Synchronized Multimedia Integration Language (SMIL 2.0)* (http://www.w3.org/TR/smil20/, 2001).
41. P. Fraternali and P. Paolini, Model-Driven Development of Web Applications: the Autoweb System, *ACM Transactions on Information Systems* **4** (2000) 323–382.
42. N. Deakin, *XUL Tutorial. Version 11.17.02* (http://www.xulplanet.com/tutorials/xultu/, 2002).
43. Harmonia, Inc, *User Interface Markup Language (UIML) 3.0 Draft Specification. Document Version 02.12.2002.* (http://www.uiml.org/specs/index.htm, 2002).
44. A. Navarro, B. Fernández-Manjón, A Fernández-Valmayor and J. L. Sierra, Formal-Driven Conceptualization and Prototyping of Hypermedia Applications, in *Fundamentals Approaches to Software Engineering 2002, Proc. European Joint Conferences on Theory and Practice of Software 2002*, eds. R. D. Kutsche and H. Weber (Springer-Verlag, LNCS, Berlin, 2002), pp. 308–322.
45. A. Fernández-Valmayor, C. López-Alonso, B. Fernández-Manjón and A. Sere, Integrating an Interactive Learning Paradigm for Foreign Language Text Comprehension into a Flexible Hypermedia System, in *Proc. International Working Conference on Building University Electronic Educational Environments. IFIP WG3.2 and WG 3.6 Joint Conference*, California, (August 1999).
46. G. Booch, J. Rumbaugh and I. Jacobson, *The Unified Modeling Language User Guide* (Addison-Wesley, 1999).
47. M. Bryan, *Guidelines for Using XML for Electronic Data Interchange* (http://www.xmledi-group.rog/xmledigroup/guide.htm, 1998)
48. B. Peat and D. Webber, *Introducing XML/EDI* (http://www.xmledi-group.rog/xmledigroup/start.html, 1997)
49. C. M. Sperberg-McQueen and C. M. Goldstein, HTML to the Max A Manifesto for Adding SGML Intelligence to the World-Wide Web, in *Proc. 2nd World Wide Web Conference '94: Mosaic and the Web*, Chicago (October 1994).
50. World Wide Web Consortium, *XML Linking Language (XLink) Version 1.0.* (http://www.w3.org/TR/2001/REC-xlink-20010627, 2001).
51. World Wide Web Consortium, *XML Path Language (XPath). Version 1.0.* (http://www.w3.org/TR/xpath, 1999).
52. K. Walrath and M. Campione, *The JFC Swing Tutorial. A Guide to Constructing GUIs* (Addison-Wesley, 1998).
53. World Wide Web Consortium, *World Wide Web Consortium. XSL Transformations (XSLT). Version 1.0.* (http://www.w3.org/TR/xslt, 1999).
54. Sun Java website, http://java.sun.com

55. J. L. Sierra, A. Fernández-Valmayor, B. Fernández-Manjón and A. Navarro, Operationalizing Application Descriptions in DTC: Building Applications with Generalized Markup Technologies, in *Proc. 13th International Conference on Software Engineering and Knowledge Engineering*, Buenos Aires (June 2001), pp. 379–386.
56. Galatea website, http://www.simba.vsf.es/IndiceProyectos.htm
57. Apple USA website, http://www.apple.com
58. Amazon USA website, http://www.amazon.com