

HyperFrame - A Framework for Hypermedia Authoring

S. Crespo✉, M. F. Fontoura*, C. J. P. Lucena*, D. Schwabe*

* Pontificia Universidade Católica do Rio de Janeiro - Departamento de Informática

✉Universidade do Vale do Rio dos Sinos - Unisinos RS

[crespo, mafe, lucena, schwabe]@inf.puc-rio.br

Abstract:

This paper summarizes our current research about an open framework layered development architecture. The objective of our research program is to capture design principles for adaptable, flexible, and composable open software systems in particular domains. Furthermore, these open systems are presented in terms of frameworks that allow us to define the basic elements or kernel of a number of related applications associated with a domain, and then reuse these elements or kernel in the development of members of this application family. This research concentrates on the methodology, the tools and techniques, and the theoretical foundation associated with the development of such architecture. One of the domains for which this architecture will be instantiated and built is hypermedia authoring. The specification of this case study is the main goal of this paper.

We will describe the general architecture for framework construction and then instantiate it for the case of HyperFrame, a hypermedia-authoring framework. The framework can be seen as a process-centered software engineering environment that uses the OOHDM method as the development process.

Keywords: Frameworks, object-oriented design, hypermedia, OOHDM, software-engineering environments, processes.

1. A General Open Framework Development Architecture

A framework is defined as a generic software system for an application domain, which provides a reusable semi-finished software architecture that allows both single building blocks, and the design of (sub) systems to be reused. Furthermore, it is composed of a collection of collaborating and extensible classes, conceived to work together and represents a generic subsystem that can be instantiated [Alencar, Cowan, Fontoura, Lucena & Nelson 97].

A framework consists of a collection of abstract and concrete classes and the interface between them, and can be seen as defining a high-level language with which applications in a domain are instantiated (or created) through inheritance and composition. The reuse of a framework in a particular application involves providing a set of refinements, and integrating them into the framework's generic architecture.

We now present a general open architecture for framework development [Alencar, Cowan, Fontoura, Lucena & Staa 97]. The architecture is divided into several layers. Each layer is responsible for a different required functionality. Figure 1 illustrates this approach, and the following subsections explain each one of the layers.

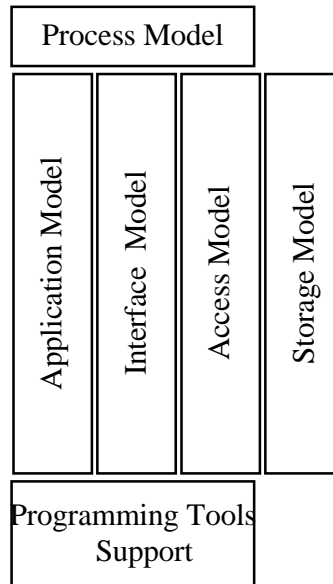


Figure1: A Layered Architecture

1.1 Storage Layer

The Storage Layer is responsible for the persistence of all the framework data. It may use any kind of storage mechanism, such as flat files or databases systems.

1.2 Access Layer

This layer is the responsible for providing access to the data. It has all the functions that manipulates the data and is a client of the storage layer. Examples of access methods that may be used within this layer are the SQL-like access, the sequential access, and the hypertextual access.

1.3 Interface Layer

This layer is responsible for the manipulation and transformation of the data obtained by the access layer so that the other higher level layers can use it.

1.4 Application Layer

This layer is responsible for the application specific function and behavior.

1.5 Programming Tools Support Layer

This layer provides all the programming support for the construction of the framework perspectives and extensions. It may be divided in two parts: a programming part and an auxiliary part. The programming part uses any programming language, such Lua [Jerusalimschy, Figueiredo & Celes 96], Java, Visual Basic, or C++. The auxiliary part consists of auxiliary tools such as libraries (e.g., object-oriented libraries) or toolkits (e.g., user interface toolkits).

1.6 Process Support Layer

The process support layer provides all the process control functions needed by the each instance of the framework. This layer must give support to process programming, control, and analysis. The interface of the final environment (generated by the framework) may also be controlled by through process execution.

2. HyperFrame Specification

HyperFrame is a robust framework that gives support for large-scale hypermedia systems authoring. As main characteristics of this framework we have:

1. Support for multiple authoring languages: it will allow that each user to be able to use the language of his/her preference. Examples of these languages can be HTML, SGML, and C++;
2. Support for multiple user interfaces: final users will also define the interface diagrams;
3. Development method and process control: a process-enacting engine will control the framework. HyperFrame will support cooperative work, where each framework user will be performing a role in the process program. The process language used by the enactor will match all the requirements of hypermedia development processes. As a development methodology we will use the Object Oriented Hypermedia Design Method (OOHDM) [Schwabe, Rossi & Barbosa 96; Rossi 96], described next.

2.1 OOHDM

Independently of the particular life cycle model used, there seems to be a growing consensus about the kind of activities that must be performed, in terms of the sequence of processes and products involved in the development of an application. In this respect, the process of building hypermedia application is not intrinsically different from the one used when building conventional applications.

Domain or conceptual modeling is aimed at understanding the problem domain and building adequate models of it, while design deals with abstractions in the software universe and is biased towards maximizing modularity and reuse. The design model is application-independent in the sense that though it may take into account some implementation setting, a particular implementation environment does not condition it. Finally, during the implementation, design abstractions are converted into concrete implementation artifacts, i.e. those existing in a real implementation environment.

The OOHDM [Schwabe, Rossi & Barbosa 96] considers the hypermedia application development process as composed of four activities that are performed in a mix of iterative and incremental styles of development, in each step a model is built or enriched. The four activities are:

1. Conceptual Design;
2. Navigational Design;
3. Abstract Interface Design;
4. Implementation.

During Conceptual Design, a model of the application domain is built using well know object-oriented modeling principles [Rumbaugh et. al. 91] augmented with some primitives such as attributes perspectives and sub-systems. Conceptual classes may be built using aggregation and generalization hierarchies. The product of this step is a class and instance scheme built out of sub-systems, classes, and relationships. Whereas the conceptual model is built without concern for particular kinds of users and tasks, the Navigational Design phase produces a set of models that is built taking into account the types of intended users, and the set of tasks they will perform using the application. As a consequence, in OOHDM one application is seen as a navigational view over the conceptual model. The navigational structure of a hypermedia application uses (Navigational) classes, called nodes, which reflect the chosen view over the application domain. The navigation space is defined in terms of navigational contexts, essentially, sets of navigational nodes which are induced (in different ways depending on the type of class) from navigational classes such as nodes and links. During Navigational Design we also define the way in which navigation will proceed by specifying transformations in the navigational space, i.e. the set of accessibly navigational objects at a given time.

The third phase, Abstract Interface Design, is concerned with making the navigational objects perceivable by the users, so an interface model is built. In particular, it defines the way in which different navigational objects will look like, which interface objects will activate navigation, the way in which multimedia interface objects will be synchronized and which interface transformations will take place.

Finally, by mapping the navigational and abstract interface models into concrete objects, i.e. those available in the chosen implementation environment, the author produces the actual hypermedia system to be run.

3. Formalizing the Process

In this section we will formalize the OOHDM process, specifying the tasks, actors, and procedures involved in each phase of the process. By doing this we will define the tools needed in each step of the process. These tools will compose the HyperFrame kernel, defining the Application, Interface, Access, and Storage layers structure.

3.1 Conceptual Design

During this task a conceptual diagram is developed. This diagram represents the objects and relationships in the application domain. In the OOHDM this diagram is constructed over classes, relationships, and subsystems. Class and relationship cards, similar to the ones defined CRC [Wirfs-Brock et. al. 90], are used to help the documentation process. These cards assist the project decisions, during the whole process.

3.1.1 Actors

This phase will involve two different kinds of actors. One, the Domain Engineer, is specialized in the application domain. The other, Application Designer, is specialized in capturing the Domain Engineer knowledge and representing it by OOHDM diagrams.

3.1.2 Pre-conditions

The actors involved in this phase must be familiarized with object-oriented concepts, such as: objects, attributes, classes, relationships (aggregation and inheritance), subsystems and domain knowledge.

3.1.3 Artifacts produced

A design diagram representing the semantic modeling of the application domain is produced. This diagram is very similar to the ones specified in others OOADM (object-oriented analysis and design methods), such as OMT. The next phase, navigational design, will map the classes specified in these diagrams into hypermedia nodes and the relationships into links.

3.2 *Navigational Design*

In this phase the navigational contexts, navigation class(nodes) supported by the application are defined. The actors involved in this process will use the conceptual diagrams defined in the previous phase to define the application navigational structure. They are responsible for providing different navigational options, or different views of the hypertextual information.

3.2.1 Actors

The same kind of actors involved in the previous phase. Now the Domain Engineer is the one responsible for defining the various navigational options, which will be modeled through navigational contexts by the Application Designer or Context Designer.

3.2.2 Pre-conditions

The semantic modeling of the application domain, definition of user profiles and tasks defined in the previous phase, must be well known by the actors involved in this phase.

3.2.3 Artifacts produced

Navigational class diagrams containing the whole navigational structure. This structure is represented in terms of navigational nodes, links, access structures, context definitions, and navigational transformations.

3.3 *Abstract Interface Design*

A critical step in the development of hypermedia applications is the interface specification. The specification of a formal model that maximizes the dialog independence and permits that components be reused is very important.

The possible metaphors and their static and dynamic properties and relationships define the abstract interface model.

3.3.1 Actors

The actors involved in the previous phases and an Interface Designer.

3.3.2 Pre-conditions

It is necessary to know:

- The way the various users will deal with the application objects;
- The interface objects;
- The relationship between the objects and the events that controls the navigational process;
- Synchronization of objects such as audio and video;
- The changes in the interface produced by external events;
- Abstract data views (ADV);
- Abstract data objects (ADO);
- The ADVChart notation;
- Configuration diagrams.

3.3.3 Artifacts produced

In this phase the following diagrams will be produced:

- Configuration diagrams, that represents the communication between the various objects;
- ADVCharts, that represents the dynamic aspects of the interface;
- ADVs are defined to present one navigational classes and contexts;
- ADVs are also defined for others kinds of objects, such as buttons and menus;
- Diagrams containing the static relationship between ADVs and ADOs;
- The specification of ADVCharts for each ADV, allowing a graphical visualization of the dynamic aspects of the application.

3.4 *Implementation*

This phase is a consequence of the previous phases. If all the previous steps were well defined this phase is developed is very straightforward. This phase is the one responsible for selecting the development plataform, languages, and supporting environments. A lot of emphasis must be devoted in the validation and test steps.

3.4.1 Actors

The actors involved in this phase are software engineers.

3.4.2 Pre-conditions

All the documents developed in the previous phases.

3.4.3 Artifacts produced

The final documentation and the application are the artifacts produced in this phase.

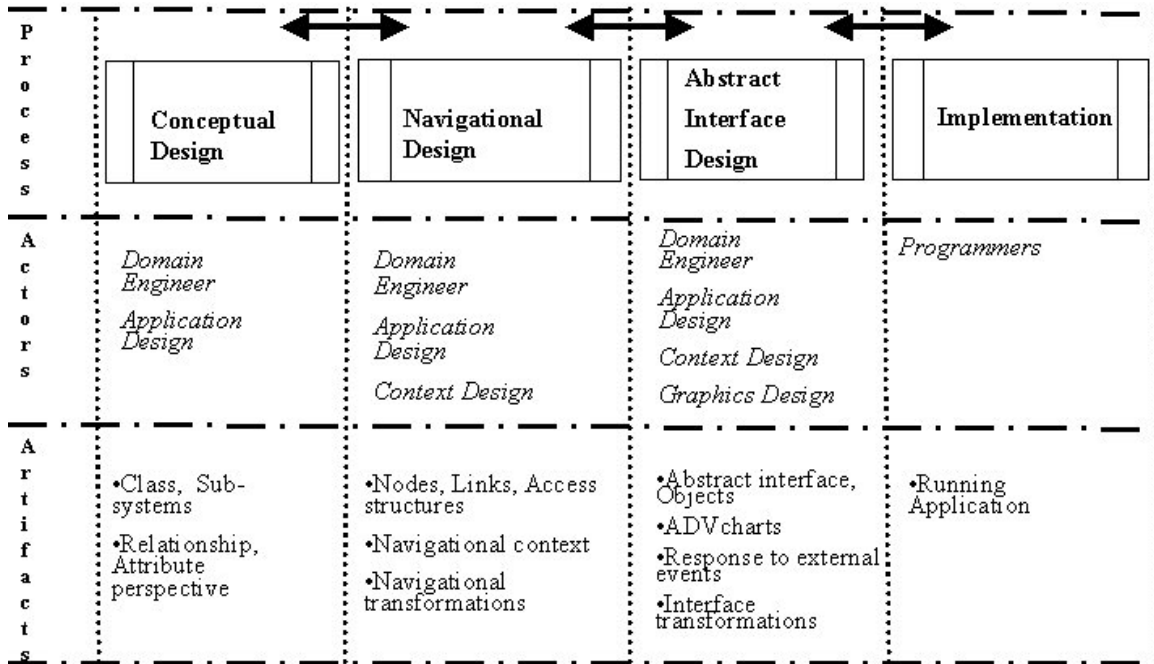


Figure 2: OOHDM Process Formalization

4. The Kernel Definition

By analyzing the process formalization we can define the tools needed by each step of the development process. The first two phases, Conceptual Design and Navigational Design, are basically centered in designing diagrams. Being so, our framework must provide a powerful diagram editor. The HyperFrame requirements specify that the user will be able to configure the diagram interface and working notation. This editor must support graphical and textual descriptions and will be defined as a meta-editor, which can be customized for each one of the steps of the method.

The framework's meta-editor must guarantee the persistence of all the objects being manipulated. This will define the structure of the Access, Interface and Persistence layers, described in figure 1. Another feature of the editor is the use of direct manipulation for the creation of new diagrams and entities.

In the Abstract Interface Design phase we must be able to define interface templates. These templates will specify the design and behavior of the interface. In this case we will need to use a interface construction tool that has the notions of ADVs and ADOs embedded in its kernel. This will be a very important tool of the framework's kernel.

This tool, called ADV based interface editor, will give support for defining the behavior and the relationships between the objects in the application abstract interface.

In the implementation phase we will need an interface that will allow us to specify the application's implementation characteristics, such as language used, target platform, etc.

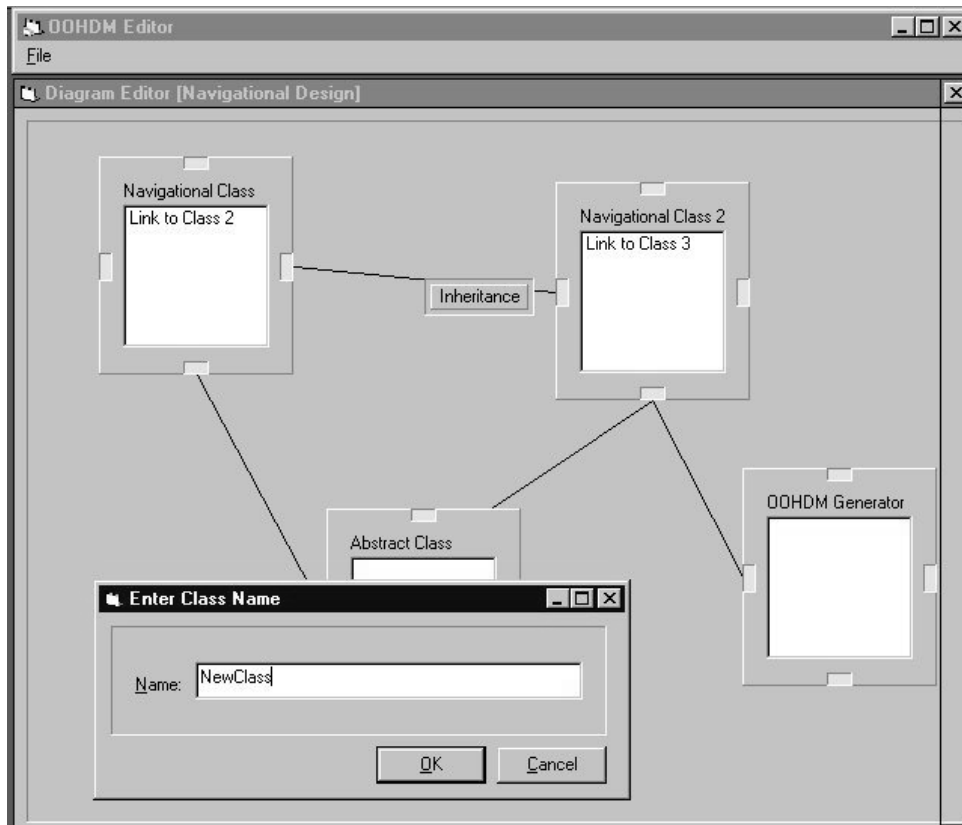


Figure 3: A Meta-editor Prototype

Basically our framework will be composed of three integrated tools: a meta-editor, a ADV based interface editor, and an application generator tool, that will generate the final application according to the implementation characteristics specified.

5. Conclusions and Future Work

We are now working on a more formal definition of each one of the tools that compose the framework kernel. We already have a very good specification of the meta-editor and the ADV based interface editor and are currently working on the specification of the application generator tool.

Figure 3 illustrates a navigational diagram constructed in a meta-editor prototype that is already developed. This prototype will be adapted to the HyperFrame requirements and integrated in the framework kernel.

The Storage and Access layers are also already implemented. The storage mechanism is based on a relational database and an API that defines the functions and structures for accessing the navigational objects defines the access. This API is implemented in C++ and can be used as CGI scripts.

Our first approach will be the use of a fixed architecture for generating Web applications. We will use CGI-Lua [Lua 97] for specifying the dynamic aspects of the application. The ADV based interface editor will generate HTML templates that will be “combined” with the navigational objects to generate the final application. This architecture will be validated in the development of Web-based Courses using HyperFrame.

We think that the specification and development of this tool will help us to validate and refine our framework construction approach and will also be important in the improvement of our understanding of the development method OOHDM.

We are now working on the specification of hypermedia design patterns [Garrido, Rossi & Schwabe 97], which will be very important in the definition of our framework architecture.

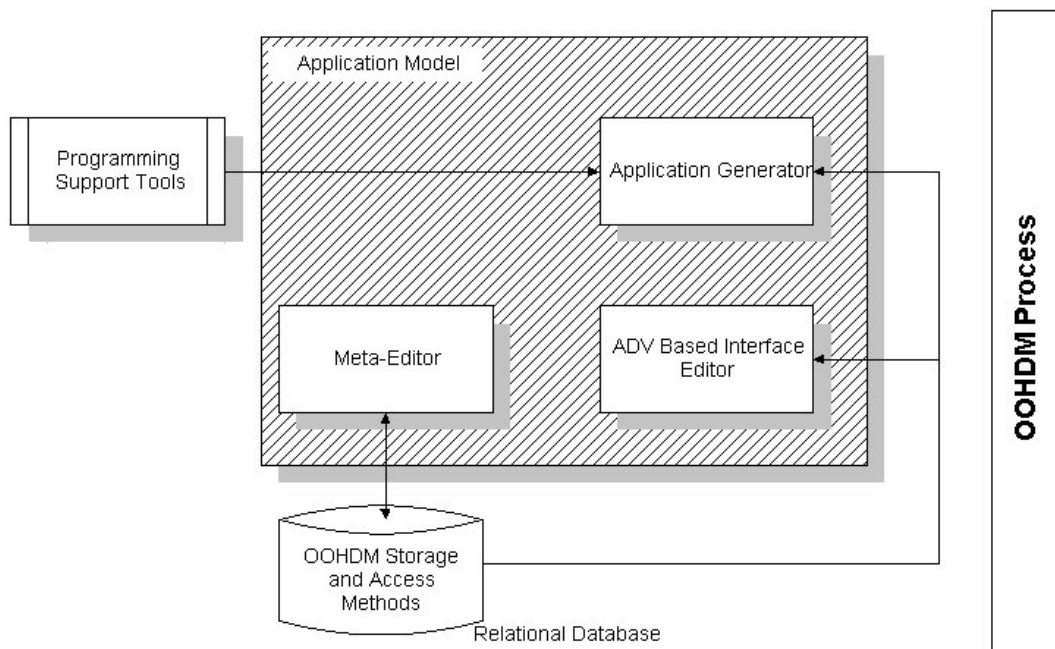


Figure 4: The HyperFrame Architecture

6. References

[Alencar, Cowan, Fontoura, Lucena & Nelson 97] P. Alencar, D. Cowan, M. Fontoura, C. Lucena & T. Nelson 97: "A Framework Development Approach Based on Viewpoints". Technical Report, Computer Systems Group, University of Waterloo, 1997.

[Alencar, Cowan, Fontoura, Lucena & Staa 97] P. Alencar, D. Cowan, M. Fontoura, C. Lucena & A. Staa 97: "Frameworks in Software Engineering: Methods and Tools". Technical Report, Computer Systems Group, University of Waterloo, 1997.

[Garrido, Rossi & Schwabe 97] A. Garrido, G. Rossi and D. Schwabe: "Pattern Systems for Hypermedia", to appear, 1997.

[Ierusalimschy, Figueiredo & Celes 96] R. Ierusalimschy, L. H. de Figueiredo and W. Celes F. "Lua - an Extensible Extension Language". *Software: Practice & Experience* 26(6), 635-652, 1996.

[Lua 97] <http://www.tecgraf.puc-rio.br/scripts/cgilua/manuais/cgilua/cgilua.html>

[Rossi 96] Rossi, G., "Uma Metodologia para o Projeto de Aplicações Hiperfídia", Tese de Doutorado, Departamento de Informática, PUC-Rio, 1996.

[Rumbaugh et. al. 91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: "Object-Oriented Modeling and Design", Prentice Hall, Englewood Cliffs, NJ, 1991.

[Schwabe, Rossi & Barbosa 96] Schwabe, G. Rossi and S. Barbosa: "Systematic Hypermedia Design with OOHDM". *Proceedings of the ACM International Conference on Hypertext (Hypertext'96)*, Washington, March 1996.

[Wirfs-Brock et. al. 90] R. Wirfs-Brock, L. Wiener and R. Wilkerson: "Designing Object-Oriented Software", Prentice Hall, 1990.