Vrije Universiteit Brussels – Belgium
Faculty of Sciences

In collaboration with

Ecole des Mines de Nantes – France

Universidade Estadual do Norte Fluminense – Brazil

1999

Use of Web-based Three Tiers Architectures: Applying Separation of Concerns to the Modelization and Implementation of a Dynamic Internet Database Interface

A Thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
(Thesis research conducted in the EMOOSE exchange
project funded by the European Community)

By: Thomas Wallet

Promotor: Prof. Theo D'Hondt (Vrije Universiteit Brussels)
Co-Promotor: Prof. Cabral Lima (Universidade Estadual do Norte Fluminense)

USE OF WEB-BASED THREE TIERS ARCHITECTURES: APPLYING SEPARATION OF CONCERNS TO THE MODELIZATION AND IMPLEMENTATION OF A DYNAMIC INTERNET DATABASE INTERFACE

Abstract

With the exponential increasing of the web-users number, web-based three-tiers applications are an appropriate way for companies to provide through the Internet some services to a wide number of users. When conceiving such applications it is important to realize an appropriate choice between the numerous technologies, tools and techniques existing for such development.

This thesis draws an analysis of some possibilities to realize the design of web-based three-tiers applications as well as some technologies and tools that can be used for implementing such applications. It focuses particularly on the way to program with separation of concerns, and specifically by using Aspect Oriented Programming (AOP) and the AspectJ tool developed by Xerox Parc Corporation, Palo Alto, California.

We applied the results of this analysis to the development of a web-based three-tiers application called RECINTERNET, which is a web-based dynamic database interface realized for a Brazilian federal company called DATAPREV. We carried out the modelization and implementation of this application following two approaches: a conventional object-oriented one and an aspect-oriented one.

In the context of the aspect-oriented approach, we created a way to realize and systematically organize aspect-oriented design for web-based three-tiers applications. Our approach is based on a step by step decomposition process and results to some aspects design tables, that are used to structure and visualize interactions between the different entities of an aspect-oriented design.

Finally we illustrate the benefits of our support for aspect-oriented design in the concrete case of the RECINTERNET application design and implementation. We draw then a comparison between the object-oriented and the aspect-oriented approaches in the concrete RECINTERNET case.

Keywords

Web-based Three Tiers Architectures, Dynamic Database Interface, Object Oriented Design and Implementation, Separation of Concerns, Aspect Oriented Design and Implementation, AspectJ, Aspects Design Methodology and Representation.

ACKNOWLEDGMENTS

This thesis could not have been developed without the help, the remarks, the pieces of advice and the critics of several persons. These persons all know how grateful I am for every little thing they did to help me carrying out this work, but I would like to name few of them that were particularly important for me during this thesis:

My first thanks go to Paulo Ramos and Claudio Passos, from GRAAL, who welcomed me in their working team and always did their best to provide me the appropriate help during all the time I spent in DATAPREV. They have been a precious advantage in the development of the RECINTERNET project for their knowledge and for their suggestions and critics about my work.

I have a particular thought for L.G. Kyal. Having L.G by my side day by day has been more than a great support for this thesis. With smiles, warm support and funny words L.G. has been like a sun ray in this thesis. L.G. can count on much more acknowledgments than these few words.

As the coordinator of the RECINTERNET project, Dr. Emmanuel Passos helped me a lot to choose the best options in order to develop a coherent system for the RECINTERNET project, even given the time constraints. By his brief but pertinent remarks he efficiently supported me and directed my work in the good way to finally reach concrete results representing the most part of the RECINTERNET project development.

My best thanks are for Dr. Cabral Lima, who has been the best support I could imagine to carry out this thesis. I am grateful for all he did to find and prepare such a collaboration between UENF and DATAPREV, for his close supervision during my period in DATAPREV and for his efficient support in the final part of my thesis in UENF. I could write several pages to enumerate all he did to welcome me in Brazil and to arrange the best conditions for my work, as well as his help in several work and extra-work tasks. In these few lines of acknowledgements I can just address him a particular *"Muito obrigado!"* that only badly reflects all the gratitude I feel.

I cannot finish these acknowledgments without a special thought for the friends that shared with me their apartment in Rio. During all this time in Rio they helped me in everyday life and made me discover and appreciate this particular Brazilian way of life. I thank them also for their great support and jokes about my difficult Portuguese learning. I also address particular wishes to my friends in Campos that welcomed me in their house and support my intensive finishing period in a very friendly way.

Finally I would like to thank the European Community, GRAAL and the Universidade Estadual do Norte Fluminense for their financial support in this EMOOSE thesis.

TABLE OF CONTENTS

I - CONTEXT

I.1) General context

It is important to explain the general context of this thesis since it is a bit particular. This thesis took place during six months in Rio de Janeiro city and Campos city, in the state of Rio de Janeiro in the south-east of Brazil. It stands both for a research master thesis of the European Master of Sciences in Object Oriented Software Engineering (EMOOSE) and for a final year industrial project of the engineering school Ecole des Mines de Nantes.

That is the reason why this thesis had to group in a same project a research aspect and an industrial aspect. To satisfy these constraints this thesis was defined as a research work applied to a concrete industrial case, in a collaboration between the *Universidade Estadual do Norte Fluminense (UENF),* which is a university of Rio de Janeiro State situated in Campos city, and *DATAPREV,* which is a federal company situated in Rio de Janeiro city.

Due to this collaboration, we had to deal with different requirements coming both from the "research" thesis specifications and from the company needs:

To put it in a nutshell, as a research master thesis this thesis was bound to provide a substantial analysis and synthesis of past and actual researches in a domain of object oriented software engineering, and then to carry out a research work to develop some new aspects in this domain.

In the same time, from the company standpoint, this work had to provide a concrete result in the given time. As this work was included in a project, the given objective had to be reached efficiently in order that this work could be easily and concretely use by the project team.

These two kind of requirements sometimes go in two distinct directions, and so a particular balance had to be done during all the time of this thesis, in order to be able to present an appropriate and effective work for both aspects.

I.2) Working context

I.2.1) Universidade Estadual do Norte Fluminense

The research development of this thesis was done under the responsibility of the *UENF: Universidade Estadual do Norte Fluminense (State University of North Fluminense),* in Campos, under advising of Pr. Cabral Lima, head of the *SCC: Setor de Computação Cientifica (Scientific Computation Department).* This department is mainly specialized in Artificial Intelligence and Software Engineering (see for example [Lim95], [Lim97] and [RLKS98]).

I.2.2) The company

The "industrial" part of this thesis took place inside a project included in a partnership between GRAAL, UNISYS and DATAPREV. GRAAL is a company specialized in software engineering and artificial intelligence. GRAAL works jointly with UNISYS (a multinational involved in software and hardware developments) in different projects contracted by DATAPREV, which is a federal company managing all the data related to Brazilian social welfare system. The industrial aspect of this thesis was realized for GRAAL and was supervised by Dr. Emmanuel Passos (chairman of GRAAL) and some research features by Pr. Cabral Lima (from UENF), as he is also scientific consultant for this company.

I.2.2.1) DATAPREV

DATAPREV[1], *Empresa de Processamento de Dados da Previdência Social* (Company of Social Welfare Data Processing) is a federal company of the *Ministério da Previdência e Assistência Social* (Brazilian Ministry of Social Welfare) existing (in its current structure) since 1974.

Having its headquarters in Rio de Janeiro city, and two more data processing centers (Rio de Janeiro and São Paolo), DATAPREV also counts state offices in all Brazilian states. Currently, the company counts around 3,800 employees.

DATAPREV is responsible for the management of all Brazilian social welfare data. It includes particularly the computer development of Brazilian social welfare institutions, the processing of all the calculus concerning social benefits (temporary retirements, health insurance, etc...), the processing of the main part of the pay rolls of the country (reaching 17 millions of beneficiaries per month) and the collecting of social contributions of around 5 millions people and 3 millions companies.

Rio de Janeiro headquarters are situated in a recent 13-floors building where are working around 1,300 employees. An important part of the work realized in DATAPREV is organized in projects which are frequently delegated partially or totally to others companies.

I.2.2.2) UNISYS and GRAAL

UNISYS[2] is a multinational present in more than 100 countries around the world in various activity domains of computer science such as hardware, software, electronic business, databases, communications. Created in 1986 from the merge of Sperry Gyroscope Co. and Remington Typewriter Co., UNISYS is now providing computer science related services to tens of thousands of customers worldwide, including well-known companies such as Microsoft, Intel, Compaq, Oracle, SAP, etc...

GRAAL is a company of Rio de Janeiro city specialized in software engineering. Founded in 1990, this company counts 15 employees and is led by Dr. Emmanuel Passos. GRAAL works for different partners (such as EMBRATEL, Brazilian telecommunication company or DATAPREV) in software engineering projects dealing with artificial intelligence, networks, database systems. The advisor of this thesis, Pr. Cabral LIMA, is in charge of GRAAL activities scientific bases through research projects developed in software engineering and artificial intelligence domains.

UNISYS and GRAAL work jointly in projects for DATAPREV since 1997 These two companies provide together specific services for DATAPREV in the domains of software engineering, database systems and artificial intelligence.

This thesis took place in the context of a project of this triangular partnership, called *RECSINWIN*. This project was started in 1997, and it consists in developing a software package that will be used by DATAPREV employees to access efficiently Brazilian social welfare data of important databases (see Section I.3 for more details about RECSINWIN). The main software package was at the moment of this thesis in its evaluation and testing stage.

This project was developed by a team of GRAAL working in DATAPREV. This team is composed of three people: Claudio Passos, Paulo Ramos and Dr. Emmanuel Passos (head of the project).

During this thesis we worked in a project called *RECINTERNET*, which is a project developed in parallel to RECSINWIN. This project aims to develop a web-interface that enables web-users to compose dynamically requests to a large database of DATAPREV and to visualize their results (see Section I.3 and I.4 for more details about RECSINWIN and RECINTERNET).

[1] Online information about DATAPREV can be found in DATAPREV Home Page, at the following URL: http://www.dataprev.gov.br/
[2] Online information about UNISYS can be found in UNISYS Home Page, at the following URL: http://www.unisys.com/

I.2.3) Thesis organization

I.2.3.1) Organization

This thesis was entirely supervised by Pr. Cabral Lima in his quality of responsible of the *Setor de Computação Cientifica* (Scientific Computation Department) of the UENF and also for his involvement in GRAAL as responsible of GRAAL activities scientific bases.

The first part of this thesis was developed in DATAPREV since the RECINTERNET project required a good knowledge of the RECSINWIN software package and its environment. Being working with the RECSINWIN team in DATAPREV, we were then able to understand the different aspects of the system that would be used also for the RECINTERNET project. Given the security restrictions imposed by DATAPREV polices, we had to be present in DATAPREV in order to understand and use the system that had been the support of our work.

During this part, the RECINTERNET implementation work of this thesis was led by Dr. Emmanuel Passos (chairman of GRAAL). This work was inserted in a research process that we also developed in DATAPREV, closely followed by Pr. Cabral Lima by the way of meetings and weekly reports.

The last part of this thesis was carried out at UENF. There were developed the finalization of the research aspects of this thesis as well as the formalization of this thesis report. This last part was achieved through a close collaboration with Pr. Cabral Lima.

Finally through this particular organization it has been possible to give the required industrial dimension to this thesis – as it stands also for a final year industrial project in the Ecole des Mines de Nantes – and in the same time to include beneficially a concrete industrial application case into the research process of this master thesis.

I.2.3.2) Working conditions

For the comprehension of the general context of this thesis it seems necessary to quote few important aspects of the particular working conditions encountered during the implementation part of this thesis in DATAPREV.

Given the well-known economic problems faced by Brazil in this decade, it is a fact that most of the Brazilian public companies suffer from infra-structures problems. On one hand Brazilian government invests a lot in young researchers formation through national and international educational scholarships at several formation levels. But in the other hand it is generally assumed that Brazilian government has to rely on foreign infra-structures for their industrial park. This can be partially explained by the lack of national companies able to work with new technologies and by the omnipresent of United States influence, particularly in their computer material monopoly.

As a result, even being the 10th economic world power, Brazil depends of United States willing for its own technologic progress, and then, even in the case of a public Brazilian company dealing daily with billions of crucial computerized data as DATAPREV, used technologies are sometimes exceeded since many years. Thus, in the working part in DATAPREV of this thesis, we had to face relatively important material problems, such as hardware and software inadequacy or continuously disturbed slow networks connections.

In addition to these difficulties, appeared some problems in information collection about a database and its access system. The RECINTERNET project was based on an important database grouping a high number of data concerning social welfare in all the Brazilian states. As this database is considered as confidential, it was arduous to obtain pieces of information or help in order to be able to use it in the project. Moreover developing the implementation work of this thesis in a federal company with strong security polices involved also other problems going from long delays to strict refusal to obtain required information or service.

To close this small parenthesis it is important to note that even when working conditions were difficult, the entire project in DATAPREV was very interesting. Applying a research work to a concrete whole industrial project dealing with multiple aspects of software engineering such as re-engineering, design, implementation or evaluation was very stimulating. Moreover Pr. Cabral Lima had always been giving pieces of advise for solving the encountered difficulties and he provided an adequate working environment for the final part of this thesis in UENF.

I.2.3.3) Thesis report structure

This thesis report is divided into 5 parts:

- In Section I we have introduced the general context of this thesis. We will now present RECSINWIN, which was the starting point of our work. Then comes an explanation of the objective of the RECINTERNET project and of the research interests of this thesis.
- In Section II we present a state of the art of the different technologies and techniques usable for web-based three-tiers applications such as RECINTERNET, both for design and implementation stages.
- In Section III we explain how to do and how to represent an aspect-oriented design a web-based three-tiers application. Then we draw a comparison between aspect-oriented design and conventional object-oriented design of web-based three-tiers applications.
- In Section IV we extend these two approaches with the RECINTERNET implementation. We explain first how we implemented RECINTERNET out of the conventional object-oriented design presented in the previous part, and then give some guidelines to realize the implementation of the aspect-oriented approach design.
- In Section V we present the conclusions and perspectives of our thesis, for the work we achieved for GRAAL and DATAPREV, for the comparison of the object-oriented and the aspect-oriented approaches and for the original approach we propose for aspect-oriented design organization and representation.

This report contains the following appendix:

- Appendix A: RECSINWIN graphical relational model
- Appendix B: The *Node as a Navigational View* Hypermedia System Pattern
- Appendix C: Navigational Framework Classes Description
- Appendix D: RECINTERNET Interface Description
- Appendix E: Woven code of the Dynamic Node Load aspect

Lists of tables, figures and references can be found at the end of this report.

I.3) RECSINWIN: composing dynamically requests to SINTESE database

The RECINTERNET project that we had to carry out for the company is based on an existing project: RECSINWIN. Both of these projects central goal is the development of a software package. It seems then important to have a brief presentation of the RECSINWIN software in order to understand the context of our work in the RECINTERNET project.

I.3.1) RECSINWIN project

RECSINWIN is a software developed by GRAAL for DATAPREV. This project began in 1997, and is now in its final stage of tests and modifications.

RECSINWIN means ***REC**uperação e Tratamento de Séries do **SIN**TESE para **WIN**dows* (recuperation and treatment of SINTESE series for Windows operating system). SINTESE is a system used by DATAPREV to manage a huge number of data (20 GBytes) about Brazilian social welfare. SINTESE groups different sub-systems providing different functionalities (RECSINWIN: recuperation of SINTESE series, ADMSINWIN: system maintaining the integrity of the different databases used).

RECSINWIN has been developed for all the people (more than 3,000) working in DATAPREV that need to access some data of the SINTESE database. Before using RECSINWIN, these users were only able to retrieve some data from the database with some complicated and obscure (syntactically) command lines directly addressed to the database host via the intranet of the company. The visualization of the obtained data was something perilous and absolutely not adapted. That is the reason why an efficient software package like RECINTERNET was necessary to provide user a simple requests composition mechanism as well as a clear visualization of the results.

- ***Series in SINTESE database:***

In the SINTESE database, data are organized in series (succession of data). Each serie is defined in a unit, with a time unit and one to three "space" units. The "space" term is not used for spatial coordinates, but as a way to express that the series can be expressed in different dimensions.

Entities in SINTESE database tables are represented as coded mnemonics, and finding the corresponding names requires a difficult table-to-table complicated research. For a better understanding of what are SINTESE series, we present an imaginary simplified example of a serie:

The *Companies_Benefits* serie represents the benefits of governmental companies in dollars (the unit), for different months (the time unit), for different states of Brazil (the first "space" unit), for different companies size ranges (the second "space" unit), for different sectors of activity (the third "space" unit).

There can be many combinations for visualizing this serie. Let's illustrate this with two of them:

(in USD)	**January 1996**	**February 1996**	**April 1997**
Less than 1,000 employees	12,546,000	11,425,000	11,200,000
Between 1,000 and 2,000	25,123,000	30,222,000	28,565,000
More than 2,000	11,256,899	10,556,000	13,255,000

Table 1 **-** *Companies_Benefits serie visualization by month and company size*

The Table 1 shows the *Companies_Benefits* serie organized by months and by company size ranges.

(i ı USD)	**January 1996**		**February 1996**	
	Sector Administration	Sector Communication	Sector Administration	Sector Communication
São Paolo	2,005,000	3,658,899	1,985,322	2,123,300
Rio de Janeiro	1,789,000	2,562,000	1,562,333	3,005,000
Minas Gerais	989,000	1,525,300	1,250,025	1,502,000

Table 2 - Companies_Benefits serie visualization by month, sector and state

The Table 2 shows the *Companies_Benefits* serie organized by months, by activity sectors and by states.

- ***Description of the RECSINWIN software package***

RECSINWIN software package provides an object-oriented interface to compose the visualization of SINTESE series. This RECSINWIN interface work in Windows environment. Each user needs to install the software package on his machine, and then he can use it to compose requests (one by one) to the SINTESE database.

Through a navigation in different screens, the user chooses some series and define the way to visualize them. This request is then transmitted to the SINTESE host (the system that manages connections to the huge SINTESE database), the obtained result is received by the application and then the user can visualize it and he may modify some parts of it, or compose it with other ones.

RECSINWIN also provides a particular form to compose the visualization of SINTESE series: natural language. Using this mode, the user can write in his natural language his request (for example: *How much benefits did the state companies in the communication sector in all the months of the year 1997?).* The systems interprets it using different filters and dictionaries, proposes some corrections in case of badly-formed requests, and then the user can visualize the results as in screens navigation mode.

I.3.2) RECSINWIN model

I.3.2.1) Introduction

Today many companies are still using some specific software packages for different parts of their activities, and it is common that these software packages were developed in old procedural languages such as C, Fortran, Cobol, Ada, and designed with standards specific to each company. As new needs and technologies emerge, these software packages need to evolve to provide some more appropriate services. Considering the high costs of software developing, it often seems more benefic to reuse the existing old software as a base for a new software rather than starting the development of a new one from scratch.

However reusing this kind of software is generally a difficult task since they have not always been developed with the preoccupation of being reusable or when it is the case different things happened in the software development cycle that complicate reusability. Unfortunately whenever software design is a crucial condition for reuse, numerous are the cases where there is no explicit model used for the software implementation, or only for parts of it, or there are models in a non-adapted form for reuse. Often standards are adopted for models descriptions within a same company, but these standards become inadequate. Code evolution without models updates, missing documentation, code developed by different persons interfering with the models are just few examples of other possible perturbations that affect the reusability of a software and its models.

The reuse of the RECSINWIN software for the RECINTERNET project is a concrete example of these reuse difficulties. Even if the RECSINWIN project is a recent one, its modeling was described through models standards of DATAPREV. These standards are becoming old and provide only incomplete description of the

RECSINWIN model. In fact they are specifically conceived to provide close directives for implementation, and do not provide sufficient abstraction to be efficiently reused or modified.

In this context it was difficult to base the RECSINWIN reuse only on these inappropriate design models. In the case of the RECSINWIN project, most of the design models only exist as a “live knowledge”, that is to say that design models partially exist “in the heads” of the members of the project team even if they are not defined formally in written or electronic design models. So finally the reuse of the RECSINWIN design for the RECINTERNET project was achieved through a complex mix of information from RECSINWIN formal models, from RECSINWIN code and from project members information.

We will now describe the RECSINWIN design model. Basically this model is organized in two parts: a description of the database systems and a description of the navigation. We add in this section a little description of the architecture used, because we think it is important to understand the model.

I.3.2.2) Architecture description

The Figure 1 presents the global architecture of RECSINWIN.

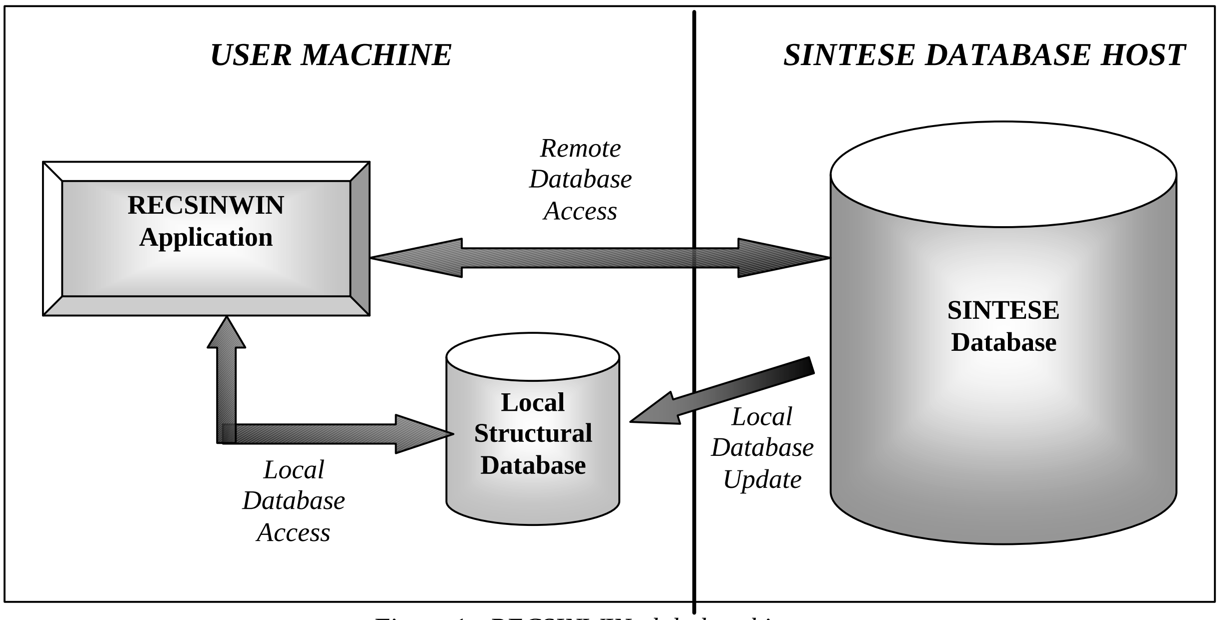


Figure 1 - RECSINWIN global architecture

The Local Structural Database is a structural copy of the SINTESE main database. It means that this database stores the same series than in the SINTESE database, but not in the same way: only the structure of the series are stored, and not all the entries of the series. In the case of the *Companies_Benefits* serie for instance, we have different dimensions (month, sector, state, company size) and entries (expressed in USD) corresponding to the values of the serie in the different dimensions crossing. For example we have one entry (12,236,000 USD) corresponding to the crossing of a time element (January 96), an element of the first dimension (companies of less than 1,000 employees), an element of the second dimension (administration sector) and an element of the third dimension (Rio de Janeiro State). In the SINTESE database for each serie are stored all the dimensions, all the dimensions elements and all the entries for all the dimensions crossings. In the Local Structural Database will be stored for each serie only its dimensions and dimensions units, but not all the entries.

The advantage of using a Local Structural Database is in efficiency. When composing a request through the RECSINWIN application, data about the selected series are necessary. It could be possible to access the SINTESE database each time, but it would be very inefficient (this access is done through a network, and this database can be accessed by other users in the same time). Then to provide the data necessary to compose the final request, RECSINWIN has this local structural database, and access it easily.

There is absolutely no modifications of the data of the local structural database since it is only used to retrieve data. So there is no need for checking if the SINTESE database is coherent to the entries of the local structural database. On the other hand the local structural database must follow the entries modifications of the SINTESE database since new entries are sometimes added to it (this database is used in several applications of DATAPREV). This updating is achieved through an application called ADMSINWIN[3], which automatically updates the local structural database according to the modifications appeared in the SINTESE database.

By using this local structural database all the composition of the request corresponding to the way of visualizing some series is done without accessing the SINTESE database host. The final request is submitted through the network to the SINTESE database host only when it has been totally defined locally. The SINTESE database host proceeds it and returns the results to the RECSINWIN application. Then the results are shown for the user.

I.3.2.3) Databases design model

RECSINWIN is based on the SINTESE database. This database and the way to access it was existing before the RECSINWIN project. Moreover this database is also part of other applications used in DATAPREV for different purposes. Then for the RECSINWIN project it was not possible to use another database or to use the SINTESE in another way.

A modeling of the structure of this database has been realized for the RECINTERNET project, since it was necessary in order to be able to create the local structural database and to use it efficiently. The model used is called *Modelo Relacional Gráfico* (graphical relational model). This model can be found in Appendix A (RECSINWIN graphical relational model).

This model describes all the tables used to store the structure of series, as well as the way the different entities of these tables are related. It can be seen as a kind of relation-entity model of the series storage. This model contains also some specific information that are hard-coded in the RECSINWIN application about these database tables. Finally this model contains also some textual descriptions of the storage formats of each entity stored in each table of the database.

This model describes only the internal structure of the database, and does not provide a view about the entities of the application. Here we just have the model of the low level corresponding to the tables structure. It would have been interesting to have another model representing the different objects (stored in the database tables) used by the application. This should have provide a greater level of abstraction that is needed for a clear understanding or reuse.

I.3.2.4) Design model describing the navigation

RECSINWIN design is also described through a navigational model. Navigation in RECSINWIN application is modeled as following: navigation is split in basic entities that are screens. A new screen is defined each time a user action makes appear or disappear an element on the screen.

Based on this screens division, the navigation is modeled as following: there is a kind of graph where every node is a screen, and every edge between two nodes is a navigation between these two nodes. Navigational interaction (such as button clicks) are also described in this model.

[3] When the RECSINWIN application is started, the user can choose to update or not the local structural database. In the update case, the ADMSINWIN application is automatically started and updates the local structural database with the modifications which have happened in the SINTESE database.

We do not present all the schemes of this navigational model. Here the idea is just to show the model behavior. The Figure 2 (adapted from [RSW99]) presents a translation of a part of this navigational model.

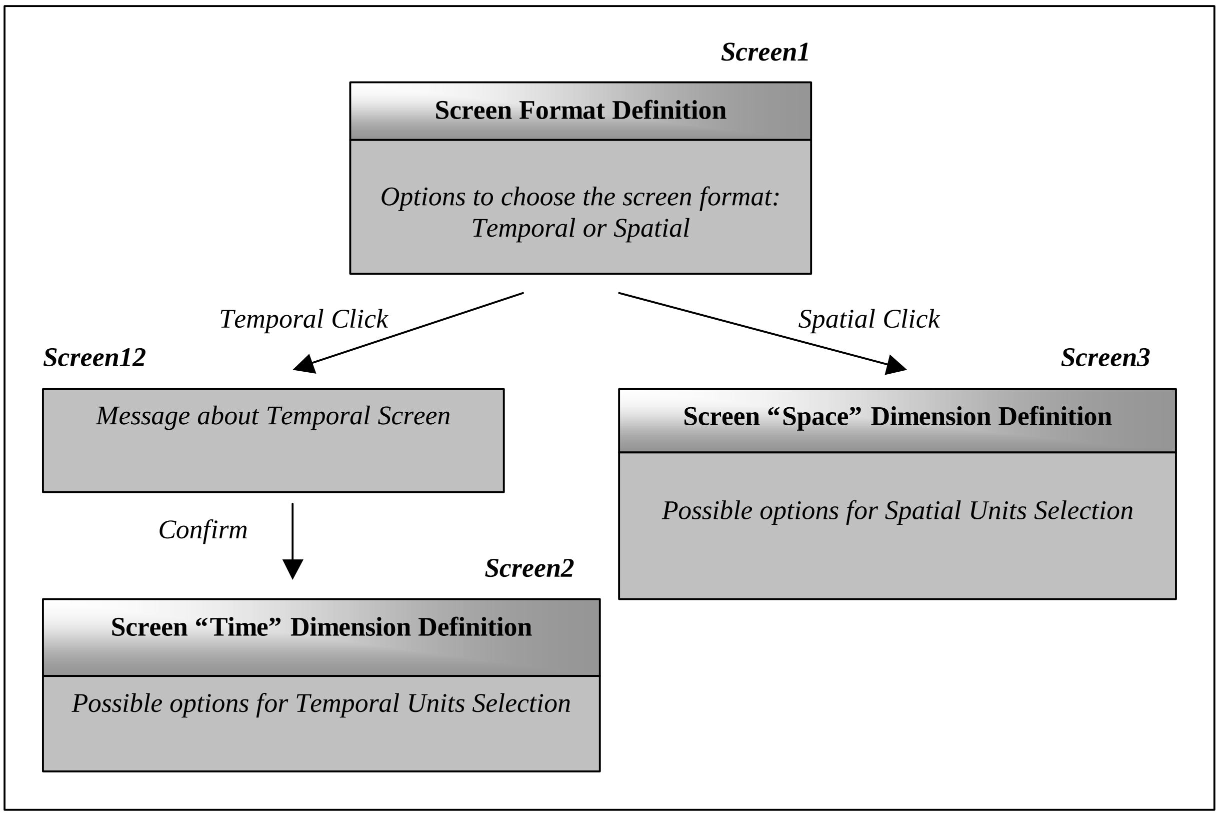

Screen1
Screen Format Definition
Options to choose the screen format: Temporal or Spatial
Temporal Click
Spatial Click
Screen12
Screen3
Message about Temporal Screen
Screen "Space" Dimension Definition
Possible options for Spatial Units Selection
Confirm
Screen2
Screen "Time" Dimension Definition
Possible options for Temporal Units Selection

Figure 2 - Translated extract of the RECSINWIN navigational model

In addition to these navigational model, there are some descriptions of all the procedures corresponding to the actions performed by the user during the navigation. Navigation is defined as a tree where a branch corresponds to a choice realized by the user. Then the model uses a decomposition in functions and sub-functions representing the path to reach a given place in this navigational tree. Procedures description are then organized into these functions and sub-functions.

In this hierarchical sub-function decomposition, each procedure resulting from user interaction is described in pseudo-code. Without explaining the details of this particular procedure, we give an example of the way these procedure descriptions are achieved in this navigational model. The Figure 3 (extracted from [RSW99]) shows a translated description of the procedure called when the user chooses to adopt the "temporal" composition way.

To conclude this section about RECSINWIN model, it can be said that the two design models used (databases model and navigational model) have probably provide the appropriate support for RECSINWIN since they describe very closely the technical way it must be implemented. However these models are expressed into specific formats (standards of DATAPREV) that do not provide abstraction mechanisms to express efficiently entities of object oriented design models. This lack of abstraction provides a close view onto the RECSINWIN implementation code but involves a difficult global understanding of the different objects used in this object-oriented application.

Function: **SERIES RETRIEVING. USING SCREENS**
Sub-function: **BuildingScreen. Confirm. BuildingTemporalScreen. <u>TimeDefinition</u>**

```
Proc:    Open RSCREEN2:

         Do I = 1 a 10;          *Activate possible time options
                                 for selected series*

               If UTEMP.Count[I] = CountSeries,
                     Activate corresponding option in RSCREEN2;
               end;
         end;

         Desactivate Confirm button;
         Show and activate Exit button, place icon and name;

Proc_end;
```

Figure 3 - *Translated extract of RECSINWIN procedures descriptions*

Moreover this lack of more “abstract” design models is a crucial handicap for future modifications, maintenance or reuse of RECSINWIN. Finally as software reuse is mainly based on design models, the reuse of RECSINWIN for the RECINTERNET project has been strongly restricted by the abstraction poverty of its design models.

I.3.3) RECSINWIN implementation

RECSINWIN has been a starting point for the RECINTERNET project. However RECSINWIN concrete implementation details do not present a crucial interest in this thesis, since its implementation has not been used in the RECINTERNET project that takes place in a totally different implementation context: internet programming with a different architecture from RECSINWIN. That is the reason why we only explain in few words the main characteristics of the RECSINWIN implementation:

RECSINWIN is an object-oriented application programmed for Windows environments. This application has been developed with DELPHI 1.0, and not with a more recent version, because some users run it with Windows 3.1, and more recent versions are not compatible. The DELPHI code of RECSINWIN counts around 28,000 lines of code.

The deployment of this application installs locally the application, the local structural database, its access elements, and also the necessary elements to access the remote SINTESE database.

Concretely the RECSINWIN application provides the user different functionalities. The main one is screen navigation to compose a request to the SINTESE database. A similar request definition is possible through sentences in natural languages. Additional functionalities are provided, such as local structural database updating, a contextual help, saving and composing results possibilities.

I.4) RECINTERNET: A Dynamic Web-based Interface

The RECINTERNET project consists in developing a web-based application that enable users to compose dynamically requests to the SINTESE database and to visualize their results. We were entirely responsible for this project, and the major part of it has been started and carried out during this thesis. The work realized during this thesis led to the development of the most part of the RECINTERNET project, which will be finalized by the other members of the RECSINWIN/RECINTERNET team. The following sections explains briefly the starting point of this project as well as the given objectives.

I.4.1) Specifications

The objectives of the RECINTERNET application are to provide to any web-user through an internet navigation the possibility to compose dynamically requests to the SINTESE database and to visualize their results. This application must then provide to any web-user of the world an easy and efficient to navigate into different screens to define step by step the characteristics of his own request and to visualize its results, in the same way than the RECSINWIN application provided a local solution to compose a request and visualize its results.

Concretely the RECINTERNET application should run in any web-browser without needing installing anything special on the user machine, and it should also be possible to have several users in the same time, from different places. Though these improvements RECINTERNET is open to all the employees of DATAPREV that need to access the SINTESE database, but it is also open to any anonymous user that wants any information about Brazilian social welfare.

RECSINWIN has been the starting point of the RECINTERNET project since both systems are used in the same functionality of providing a user a way of dynamically compose requests and visualize their results. However we had to develop a totally new approach for the RECINTERNET project since web-based applications require specific architectures, internet-specific way of programming, users access control and security, functionality modifications, etc... In this context an important reflection was necessary to conceive the RECINTERNET project, and an important work of design modeling was necessary to provide the appropriate support for an efficient implementation.

Then most part of the functionalities of the RECSINWIN application had to be realized in the RECINTERNET application, or adapted to an internet way of programming. RECINTERNET had then to provide a way to compose a request through screen navigation and different choices of the user in each screen. It means that the user has the possibility to organize the lines, the columns and the sub-columns of the result sheet that presents his selected series. The results of such requests had to be visualized and possibly downloaded in a file. Contextual help screens had to be available at any moment of the navigation. However for a first prototype of RECINTERNET few functionalities were not taken into consideration. They will only be inserted one by one to future versions of RECINTERNET. This is the case of the possibility to express requests in natural language in RECSINWIN, or manipulate and combine requests results.

Finally as there were several similarities between RECSINWIN and RECINTERNET functionalities, we tried to reuse some design models of RECSINWIN when it was possible. However for RECINTERNET we had to carry out an important and specific work of design modeling to efficiently support the implementation of this system based on the Internet.

I.4.2) DATAPREV requirements for the RECINTERNET project

Given the short period (four months) where we worked on the RECINTERNET project inside DATAPREV, it was not possible to carry out the whole project. Thus, considering strictly the company standpoint, the main objectives of our thesis were defined as following:

- ***Modelization***

Based on the RECSINWIN experience, develop a design model for RECINTERNET that provide an appropriate support for implementation, reuse or modifications in future versions of RECINTERNET. That is the reason why we had to choose and to define design models in appropriate formats that would cover the different aspects of the RECINTERNET project: architecture, communication, database, navigation...

- ***Implementation***

RECINTERNET implementation had to be realized using the best appropriate technologies existing in the different domains covered: internet navigation, three-tiers applications, communication protocol, database access... Given the time restrictions we were asked to go as far as possible in implementation, and to leave to the project team complete parts of implementation and guidelines for uncompleted parts so that the other members of the RECSINWIN/RECINTERNET team could easily finalize the project.

- ***Evaluation***

All the completed parts of our RECINTERNET prototype implementation had to be evaluated carefully, in order to leave to the project team tested and finished implementation parts that could be efficiently used to finalize the project.

To put it in a nutshell, from the company "industrial" standpoint, we had to carry out all the designing stage of the RECINTERNET project, to browse the last existing technologies in the different aspects of the project to apply them in an implementation (or parts of it) and finally to test all we had implemented.

In spite of the time restrictions, the work realized during this thesis provided appropriate design models and completed tested parts of implementation. The project was then left in an advanced phase that would provide to the three persons of the team the necessary elements to implement efficiently the missing parts of the application.

As we will see it in the next part, these company requirements were closely linked to the research carried out during this thesis.

I.5) Research objectives

The main strength of the work realized during this thesis was that we had to carry out a research work that could be directly applied to a concrete industrial case. This constraint was due to the fact that this thesis stands in the same time for a Master of Science thesis and for a final year industrial project of the Ecole des Mines de Nantes. Given this context, the research realized during this thesis was organized into three aspects:

First we analyzed the existing technologies and techniques suitable for design and implementation of web-based three-tiers applications with large databases such as RECINTERNET. This study was concretely applied to the RECINTERNET case by choosing the best techniques and technologies for its development.

In a second step we studied the way to program web-based three-tiers applications with separation of concerns, and particularly with Aspect Oriented Programming (AOP). For this purpose we developed our own approach to aspect-oriented design for such applications.

The third aspect of this research work was dedicated to apply this approach to realize separation of concerns in web-based three-tiers applications development and to study the differences with a conventional object-oriented approach. So we drew a comparison between an object-oriented approach and an aspect-oriented one for the design and implementation of the RECINTERNET project.

I.5.1) State of the art – Technologies and Techniques for developing Web-based Three-tiers Applications

Our first research objective was then to realize a analysis of the different technologies and techniques applicable in the RECINTERNET project. Given the exponential development of public interest to the internet media, there is an explosion of the number of web-based applications since they provide an appropriate and easy way for companies to provide their own services to a wide public.

Given this context, new techniques and technologies for web-based three tiers applications are permanently created or evolving in order to provide a better support to the clients or companies needs for their web-based applications. That is the reason why we decided to draw a state of the art of the different technologies and techniques (related to web development or not) that can be applied in a project such as RECINTERNET.

In terms of design stage, we will identify different techniques to efficiently program separation of concerns in object oriented paradigm, to realize navigational design of dynamic interfaces or to choose between different kind of architectures.

In terms of implementation stage, our study will cover domains such as internet programming solutions, remote communication technologies, databases access, but also applying some of the special techniques described in the design part.

For both design and implementation, we have applied the appropriate studied techniques and technologies to the RECINTERNET project. We present in details this state of the art and its application to the RECINTERNET project in the Section II.

I.5.2) Developing aspect-oriented web-based three-tiers applications

This part of the research carried out during this thesis is focused on a particular technique that can be beneficially used in projects such as RECINTERNET: *Separation of Concerns*, and particularly *Aspect Oriented Programming (AOP).*

AOP is a particularly active research domain of software engineering which provides an efficient solution to program separation of concerns. Concrete solutions for implementing applications with AOP are developed such as the solution for Java provided by the Xerox Parc Corporation: *AspectJ* [AJ]. However, as a new emerging research domain, designing and programming with aspects processes need to be developed in order to become really applicable to large scale applications.

In this context we focused our research on the way to realize aspect-oriented design, particularly for web-based three-tiers applications. We tried to develop an efficient way to organize an aspect-oriented design for such applications, and to develop techniques to provide appropriate support for aspect-oriented implementation.

We present our approach for aspect-oriented design of web-based three-tiers applications in Section III.

I.5.3) Comparison of object-oriented and aspect-oriented approaches for the RECINTERNET development

We divided this comparison into two parts: design and implementation.

First we realized a conventional object-oriented design for the RECINTERNET application. Then, applying our methodology and representation for designing web-based three-tiers applications with aspects, we realized an aspect-oriented design for the same application. This provides a concrete illustration of our approach of aspect-oriented design and “real-world” example of its possibilities. Finally we compared the object-oriented approach and the aspect-oriented one. These different parts of the design comparison can be found in Section III.

Secondly we focused our work on the implementation of these two designs. We implemented most parts of the conventional object-oriented design for RECINTERNET. For time constraints it was not possible to implement the aspect-oriented approach. According to the company objectives for the RECINTERNET project, we only gave guidelines and examples of how to realize such an aspect-oriented implementation. The RECINTERNET implementation with the two approaches is explained in Section IV.

To conclude this section about the context of this thesis, it can be said that mixing in a single project a research work and a concrete industrial application was a very interesting challenge, even if it imposed many constraints to be able to complete the both aspects in the given time.

II - State of the Art - Developing Web-based Three-tiers Applications

In this section we draw a state of the art of the existing techniques and technologies that can be interesting for design and implementation of web-based three-tiers applications such as RECINTERNET.

II.1) Introduction

II.1.1) Web-based Three-tiers applications

The *"web-based three-tiers application"* term is generally used to reference a system involving three tiers that communicate through the internet. The three tiers involved in such applications are generally *clients* (that request services), *servers* (that provide services) and *shared resources* (printers, databases, modems or high powered processors).

Given the exponential development of the internet media and its wide public, companies see a particularly interesting and easy way to reach a large public with web-based three-tiers applications. The fast development of numerous applications of this kind also involves the emergence of many new techniques and technologies. Specific requirements for each of the three tiers have an important impact on the chosen technologies. As a result there are many possible variations in the way to develop such applications, based on different techniques and technologies.

The involved technologies cover different areas at different levels of the application development cycle. At design level, such applications require a particularly elaborated and robust design since they deal with three tiers having each their own distinct functionality and collaborating with each other in specific. It is then important to choose appropriate and efficient design methodologies and techniques.

At implementation level, developers of such applications must pay a particular attention to choose appropriate technologies between the wide range of new or permanently evolving technologies applicable. Roughly all the recent technologies dealing with internet programming, internet communication protocols or database systems have to be carefully studied when developing such applications.

Finally developing a web-based three-tiers application nowadays requires first an important research of the appropriate techniques and technologies in the case of the application.

II.1.2) Technologies and techniques applicable to RECINTERNET

The RECINTERNET project, as a web-based three-tiers application, provides an interesting reference to settle a state of the art of the technologies and techniques applicable in such applications. In the context of this thesis, we analyzed several techniques and technologies related to web-based three-tiers architectures. Anyway the objective of the state of the art realized during this thesis was to browse the best existing techniques and technologies applicable in the RECINTERNET project, as well as justifying the choice of the most appropriate ones.

For time constraints, the covered domains choice in this state of the art were directed by the RECINTERNET specifications. As this project deals with complex concerns such as distribution, shared resources, connections control, it was important to have the possibility to program efficiently with *separation of concerns.* In an application involving three tiers distributed on remote locations, it was also important to study carefully the *architectural alternatives.* As a dynamic interface using the internet to compose database requests, RECINTERNET needed to use some appropriate technique to design *internet navigation.* In the same time, all the *technologies available for the three tiers implementation* had to be studied, as well as the existing solutions for *communication between these three tiers.*

To put it in a nutshell, this state of the art will then be organized as follow:

- Separation of concerns design and implementation (Section II.2)
- Architectural alternatives (Section II.3)
- Internet navigation design (Section II.4)
- Client, server and database technologies (Section II.5)
- Communication between the different tiers (Section II.6)

After each topic in this state of the art we will justify which technologies and techniques we used for the design and the implementation in the RECINTERNET project, providing in this way a concrete illustration of the applicability of such a state of the art.

II.2) Programming Separation of Concerns

II.2.1) Separation of concerns

Software application complexity is permanently increasing. Nowadays many applications deal with complex concerns such as distribution, concurrency, real-time constraints, debugging facilities, security, communication strategies, persistence, error checking, memory management, historization...

Actual programming languages only provide component abstraction mechanisms. [KLM+97] defines a component as an entity that “can be cleanly encapsulated in a generalized procedure (i.e. object, method, procedure). By cleanly, we mean well-localized, and easily accessed and composed as necessary”. Unfortunately, classical components abstractions cannot support clear encapsulation of such complex concerns into programming entities. That is the reason why using some of these concerns in an application increases consequently code complexity and by the way code size, understandability, modularity, maintainability and reusability.

In fact the main problem comes from the fact that, in actual programming languages, the code corresponding to a concern is cross-cutting the code of the different components of the system. Let’s illustrate this with an example of book locators extracted from [LK97]. We just present the main ideas of separations of concerns through a small parts of this example.

The Figure 4 presents mainly the Java code of two methods (`register` and `unregister`) of the `BookLocator` class. This class is the main class of a system providing book location services, and these two methods are used respectively to add a new book in a location and to remove an existing book. The code presented focuses only on the implementation of these two methods. The particular code of the `Book` and `Location` classes, as well as the code of the other methods of the system will not be given here, for space reasons.

Now, let’s suppose that this system can be accessed concurrently, i. e. the `books[]` and `locations[]` variables are critical resources. Then synchronization mechanisms must be added to the initial system. A way to do it is presented in Figure 5. The idea is to add two methods (`beforeWrite` and `afterWrite`), that will be synchronized and will organize a queue for the different entities trying to access the critical resources, based on two counting variables (`activeReaders` and `activeWriters`). The concurrency concern added to the initial code involves code modifications in different places scattered throughout the code (the added code is underlined in this figure).

```
public class BookLocator
{
   // book[i] is in locations[i]
   private Book books[];
   private Location locations[];
   private int nbooks = 0;

   // ...

   public void register (Book b, Location l)
   throws LocatorFull {
      if (nbooks > books.length)
         throw new LocatorFull();
      else {
         // Just put it at the end
         books[nbbooks] = b;
         locations[nbbooks++] = 1;
      }
   }
```

```
   public void unregister (Book b) {
      Book abook = books[0]; int i = 0;
      while (i < nbooks &&
            abook.get_isbn()!= b.get_isbn())
         abook = books[++i];
      if (i == nbooks) return;
      // Simply shift down the rest
      while (i < nbooks – 1) {
         books[i] = books[i+1];
         locations[i] = locations[++i];
      }
      --nbooks;
   }

   // ...
}

// ...
```

Figure 4 - Book locators (1): simple code example

```
public class BookLocator
{
   // book[i] is in locations[i]
   private Book books[];
   private Location locations[];
   private int nbooks = 0;
   protected int activeReaders = 0;
   protected int activeWriters = 0;
   // ...
   public void register (Book b, Location l)
   throws LocatorFull {
      beforeWrite();
      if (nbooks > books.length) {
         afterWrite();
         throw new LocatorFull();
      }
      else {
         // Just put it at the end
         books[nbbooks] = b;
         locations[nbbooks++] = 1;
         afterWrite();
      }
   }
   public void unregister (Book b) {
      Book abook = books[0]; int i = 0;
      beforeWrite();
      while (i < nbooks &&
            abook.get_isbn()!= b.get_isbn())
         abook = books[++i];
      if (i == nbooks) {
         afterWrite();
         return;
```

```
      }
      // Simply shift down the rest
      while (i < nbooks – 1) {
         books[i] = books[i+1];
         locations[i] = locations[++i];
      }
      --nbooks;
      afterWrite();
   }

   // ...

   protected synchronized void beforeWrite()
   {
      while (activeWriters > 0 ||
            activeReaders > 0)
         try
         {
            wait();
         }
         catch (InterruptedException e) {}
      ++activeWriters;
   }

   protected synchronized void afterWrite()
   {
      --activeWriters;
      notifyAll();
   }
   // ...
}
// ...
```

Figure 5 - Book locators (2): example with concurrency concern

This example illustrates the fact that there are complex concerns (such as the concurrency one in this example) that are tangling the basic functionality code of a program.

In [HL95] is explained the main idea of the separation of concerns paradigm: different concerns should be expressed in different modules, clearly separated from basic functionality components code. In the previous example, applying separation of concerns would mean to have in a module (or file), the initial code of the Figure 4, and in a separated module the concurrency concern. Then we would have a clear distinction between components (classical object-oriented entities such as the initial code of the example) and concerns, but also a clear distinction between different concerns (each concern expressed in one module). The way to express the different concerns and the way they are related is specific to each technique used to realize separation of concerns. To give an example, we express the concurrency concern of the book locators example in the D language, described in [LK97], which is a language used in the aspect-oriented programming (see Section II.2.2.5) way of solving separation of concern. This language is used to express concurrency and distribution concerns.

The Figure 6 presents the concurrency concern of the book locators example in the D language. The `Blcoord` concern expresses all the changes seen in the Figure 5. Basically the `selfexclusive` mechanism is used to define methods that can only be executed at most by one thread at a time. The `mutexclusive` mechanism is used to define a set of methods that mutually exclude each other. In this example we see that expressing this concern this way provide a clear separation of concern, and also a modular program less complex and long than the same program expressed without separation of concern.

```
coordinator Blcoord : BookLocator {
  selfexclusive{register, unregister};
  mutexclusive{register, unregister, locate};
}
```

Figure 6 - Book locators (3): example of the concurrency concern expressed in the D language

After this modular decomposition of a system into different components and concerns, [HL95] specifies that mechanisms must be defined to compose these components and concerns into an executable or compilable program.

Finally the separation of concerns paradigm enables to express separately the different concerns and components of a system, providing then a better understanding through a well-defined structure, a decreased complexity and easier reusability, maintainability and modificability.

II.2.2) Techniques for separating concerns from components code

Different techniques are developed to realize separation of concerns. These techniques provide ways to express clearly concerns in separate modules from components code and also they solve the problem of composing these concerns and components to create efficient systems. The following sections present the most advanced ones.

II.2.2.1) Meta level programming [SW96], [Kai98]

Meta-classes are used in different programming languages to express the behavior of classes, in the same way that classes express the behavior of objects. Reflexive languages such as Java or Smalltalk provide users the possibility to access or/and modify meta-classes, and then it is possible to affect the behavior of classes.

Meta Object Protocols (MOP) reify in the chosen language itself the mechanisms of the language. It means that MOPs describes in the chosen language some protocols used for example for compiling, executing or debugging programs. They are based on some reifications of entities used in these protocols, such as representation of message passing, representation of debugging stack.

Some strongly reflexive programming languages (such as Smalltalk), or extensions of languages (such as *Metaxa*, for Java) provide programmers the possibility of modifying MOPs. It is then possible, for example, to modify the corresponding MOP which is responsible to solve the methods lookup (within an inheritance classes hierarchy) realized at run-time. Thus modifying MOPs provides an important power to the programmer, that can modify strongly the initial language in order to support his own execution mechanisms.

In this context modification of MOP provide the appropriate support to realize language modifications in order to be able to program with separation of concerns. In [SW96] the authors explain that MOP can be used to separate concerns from basic functionality components. Components are implemented at basic level whereas concerns are implemented at meta-level. In this way we have a clear separation of concerns. They can be expressed in meta-classes and then mapped to components code of basic level through modified MOPs. This can be done for example by modifying MOP so that message passing are trapped and execution modifications are realized instead of the initial message execution. In [Kai98], Kai Böllert proposes a solution where MOPs are modified so that components instances classes are replaced by classes that have been changed to insert modifications expressed in separated concerns.

The main interest is the strong flexibility for realizing separation of concerns in the most appropriate way. MOPs modifications provide strong possibilities for the programmer to develop systems where concerns can be separated and expressed in the most appropriate from, and then composed with components of basic level as needed.

The restrictions of using meta-level programming to realize separation of concerns are that it requires the use of strongly reflexive programming languages or reflexive extensions of programming languages in order to be able to modify MOPs and meta-classes. Moreover reaching the meta-level at run-time and executing specific manipulations involving meta-level entities slows down consequently the execution speed of a program. In this case specific tricky optimizations are needed to decrease the cost of meta-level calls.

Multiple experiments have been carried out on this possibility, such as the Smalltalk system [Kai98] of Kai Böllert, to separate synchronization and tracing concerns from components code expressed at classes level. Actually this system is used to support aspect oriented programming (see Section II.2.2.5), but it uses the mechanisms described before (modifying MOP) to perform this objective. As in this case, meta-level programming is often used to implement frameworks to support the other approaches for separation of concerns.

II.2.2.2) Composition filters [AT98], [Ber94], [Cza98]

The composition filter approach is based on applying filters on incoming messages and outgoing messages of an object. It is then an extension of the classical object model with *messages filters* and related mechanisms.

As it can be seen on the Figure 7(adapted from [Cza98]), modifications have been done to the classical object model. In the composition filter approach, an object is composed of an *inner object* (that can be seen as a conventional object of the classical object model) and an *interface layer.* This *interface layer* contains a given number of *input filters* and *output filters.* These filters are used to intercept messages incoming to the object or outgoing from the object. *Incoming filters* are applied one by one to incoming messages. *Output filters* are applied also one by one to the outgoing messages. A filter contains conditions to apply to the message to filter and action to realize for each case defined by these conditions. For example depending on the conditions of the filter being applied to a message (incoming or outgoing) this message can be rejected, normally evaluated, delegated to *external objects* referenced by the object, delegated to *internal objects* of the object, it can be added to a queue or it can start other specific action.

Filters are defined as first class objects and then can be manipulated easily. Composition filter implementation is based on using meta-objects to reify messages and message passing. Filters will then be able to manipulate these message representations. Filter classes are used to express different filtering behaviors. They can be sub-classed to define a more specific behavior. Pre-defined filters are provided: *delegation filters* for delegating messages, *wait filters* for buffering messages and *error filters* for throwing exceptions.

The composition filter approach provide the appropriate support for realizing separation of concerns. Basic functionality components can be expressed in conventional objects and cross-cutting concerns can be implemented with composition filters. Filter classes can be expressed for a given concern and then applied to different objects. We then have a clear separation between basic components code and concerns code.

This approach is particularly appropriate for implementing concerns related to synchronization constraints, real-time constraints, error checking as well as other concerns that rely on message passing filtering.

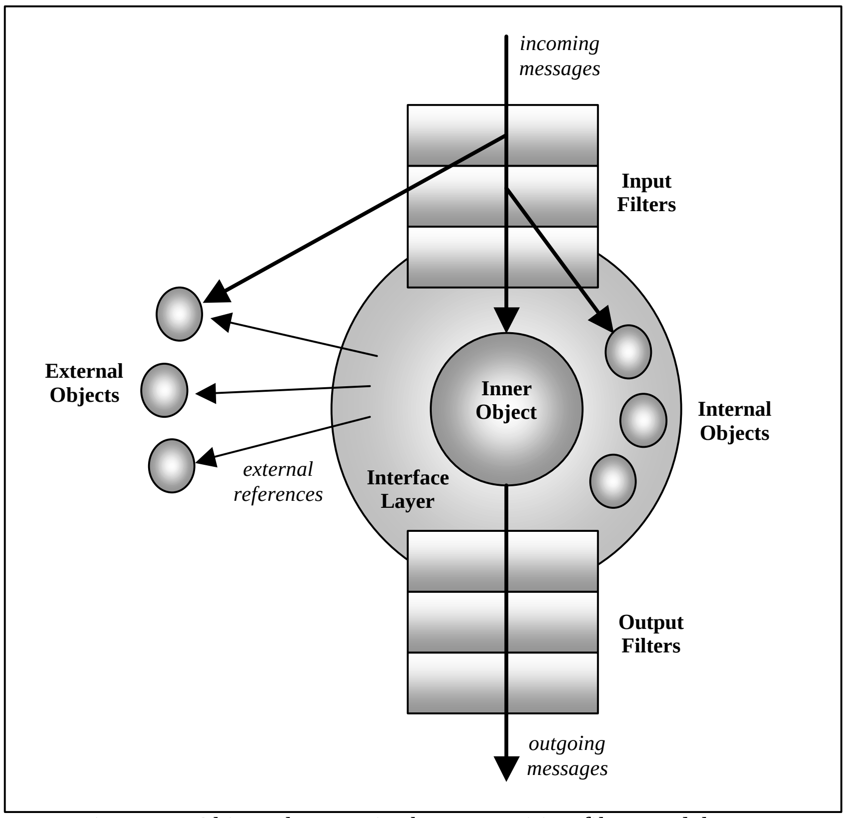


Figure 7 - Object elements in the composition filter model

However such an approach have its limits for concerns expressivity, since the filters defined for a concern have to be applied to all the involved objects one by one. Even with filter classes, cross-cutting implementation has to be realized for each concern. Moreover this approach request language extension with Meta Object Protocol (MOP) modifications, and access to meta-level will be needed at run-time, which can limit execution speed.

The composition filter model has been implemented as extensions of C++ and Smalltalk languages. These extensions are based on meta-objects representation of filters, so that can they be manipulated at run-time. A prototype for supporting composition filters in the CORBA middleware communication mechanisms has been developed but is not yet available.

II.2.2.3) Subject-oriented programming [OHBS94], [OKK+96], [AW99]

Subject-oriented programming proposes a way to encapsulate in subjects object oriented systems or sub-systems. Subjects are subjective views on a part of a system. They can be composed to obtain larger subjects.

A subject defines a class hierarchy and its operations. It represents a subjective view of a domain. Inside a subject, a set of classes is described as in the conventional object-oriented paradigm. Different subjects can be views of the same domain. The Figure 8 (adapted from [AW99]) shows an example of two subjects *(Renting* and *Driving License Management)*. Some of their items are common between the two subjects (`Car` and `Driver`/`Renter`).

Subjects can be composed to form larger subjects. Composition is automatically realized according to composition rules specified by the programmer. These rules specify which classes should be mapped, as well as the way to map them. The programmer defines in fact classes correspondences, methods correspondences, variables correspondences and methods combinations. Combination of two methods (from two composing subjects) can be done by using only one of them is the composed subject, or by appending one method after the other. Subject composition can affect code or bytecode, depending on the supporting framework used.

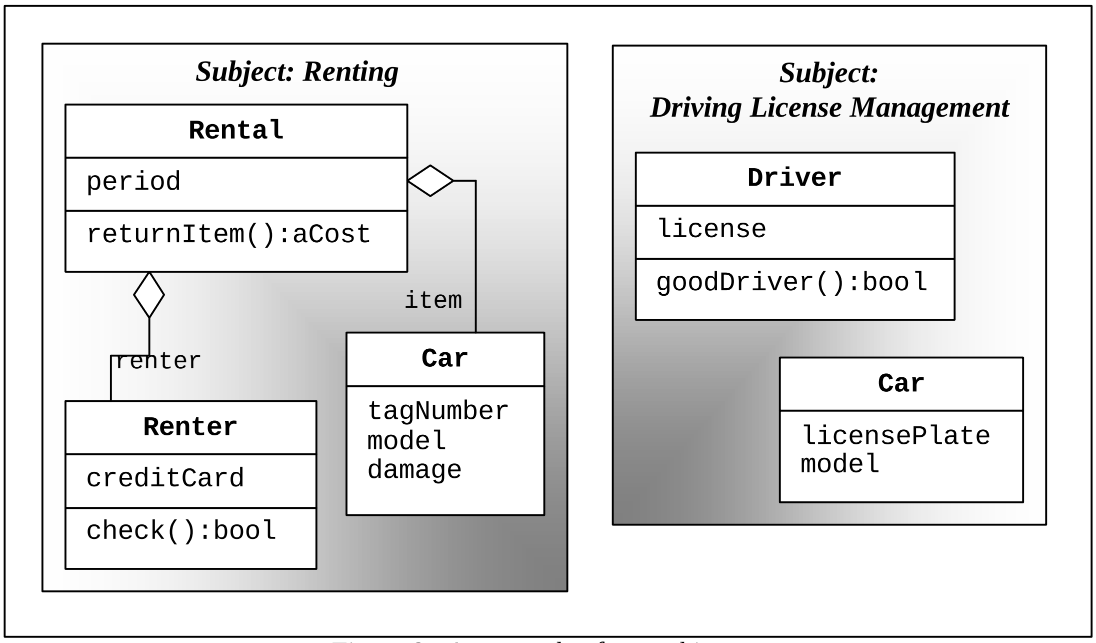


Figure 8 - An example of two subjects

The Figure 9 (adapted from [AW99]) shows an example of a subject *(Car renting)* resulting from the combination of the two subjects described in Figure 8. Elements combined from initial elements of the two composing subjects are distinguished in italic.

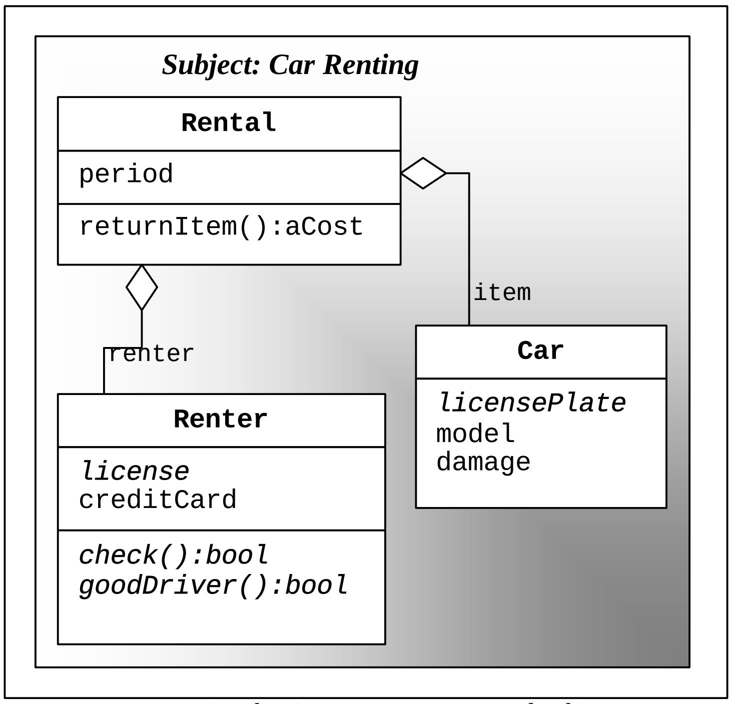


Figure 9 - The Car Renting composed subject

Subject oriented programming provide a good support for separation of concerns. Each concern can be expressed in a separated subject. A concern will be defined in a subject with a set of methods and variables of different components. The components expressed in such a subject will be only the components involved by the concern, and moreover will be expressed only the part of these components affected by this concern. Composing a concern with some components will be achieved through the subjects composition mechanism. Then an entire program will be obtained by composing several subjects describing each a concern and several subjects describing a part of the components of the system.

An experimental support of subject-oriented programming was developed for C++ (as an extension to IBM Visual Age for C++ 4.0). Some prototypes for Java and Smalltalk also exist but are not yet available.

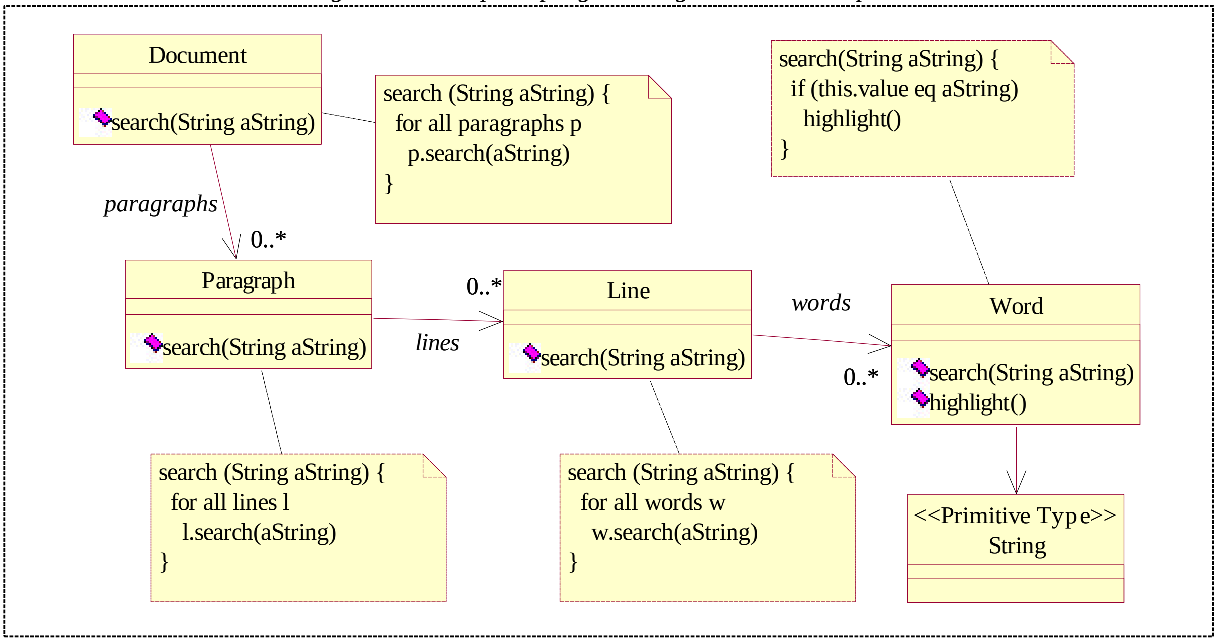
The subject oriented approach is a particularly interesting support for separation of concerns in distributed development contexts: different concerns can be developed independently and then an unanticipated composition of different concerns and components can be realized. The composition mechanisms are really flexible. However the main drawback of the subject oriented programming approach is that it is not yet based on real-world systems experiments.

II.2.2.4) Adaptive Programming [Lie92], [LO97]

Adaptive Programming basic aim is to enable developers to express some particular classes behavior in addition to object-oriented structure description such as class diagrams.

Traversals are defined in [LO97] as navigations through a group of related objects with the purpose of accomplishing a task. Object oriented programs tend to use many “small” methods used to perform little computation but mainly to pass information to another method of another object until a method realizes the “real” computation. The Figure 10, adapted from [LL96], describes a class structure containing a traversal for word searching: when searching a word in a document, the research is first transmitted to all the paragraphs, that will then transmit it to all of their lines, that will transmit it to all of their words.

Figure 10 - Adaptive programming traversal example



When classes structures are modified, maintaining these traversals are a difficult task. In most cases the maintaining of such traversals consists in modifying little pieces of code and can be done straightforward, but it takes however a lot of time in large class structures.

The idea of Adaptive Programming (AP) is to use a language to describes succinctly traversals instead of developing them as numerous methods in class structures. A traversal is then described with the origin class of the traversal (in our example `Document`) as well as the destination class (`Word`) and with the appropriate code to execute for the destination objects (the body of the method `search()` in the class `Word`). The class structure is described without any of the methods involved in the traversal.

A composition mechanism will enable to obtain an executable system with traversals fully executed. An important advantage of adaptive programming is that when modifying the initial class structure, when inserting new classes involved in traversals, it is not necessary to redefine anything concerning these traversals. The new classes are inserted in the traversal execution mechanism automatically, and the traversals will be executed completely with these classes.

As we have a separated language for expressing traversals, it is possible to realize separation of concerns with adaptive programming. Basic functionalities will be expressed in class structures and separated concerns will be expressed in distinct traversals description.

Different software have been developed to support the concepts of adaptive programming. Demeter/C++ provides an AP support in C++, and Demeter/Java in Java.

II.2.2.5) Aspect Oriented Programming [KLM+97], [Kic98], [MLTK97]

Aspect Oriented Programming main idea is to decompose any object-oriented system in components and aspects. Components are basically conventional object-oriented entities encapsulating functionalities. Aspects can be seen as expression of concerns that cross-cut this components implementation and result in concerns scattered throughout the components code.

Aspect Oriented Programming abstracts the implementation of concerns from components by expressing them into separated modules. Aspects modules will then be written in one (or more) specific aspect language. The two main parts of an aspect are *join points* and *modifications. Join points* identify the places in the components code that will be affected by the aspect, and modifications describes which code modifications will be realized at these places. The Figure 11 illustrates this aspects/components decomposition of a system. The edges represent join points expressed in the aspect code.

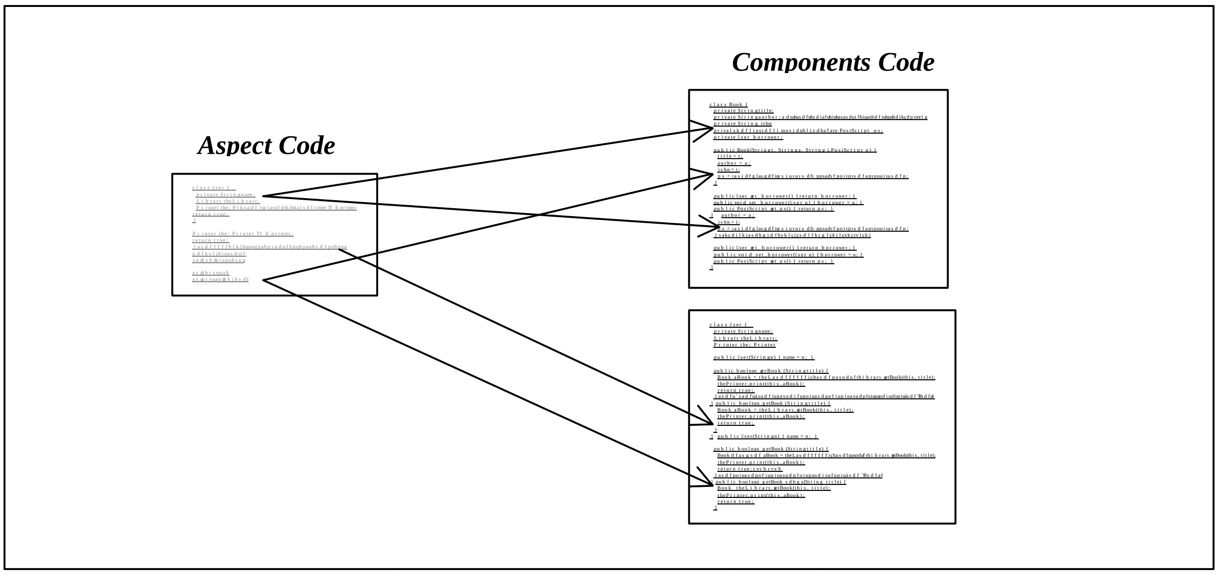


Figure 11 - Aspects/components interaction

Aspect Oriented Programming provides a mechanism called composition to obtain executable programs out of this aspects/components modules decomposition. The idea is to apply automatically the modifications described in aspects to components code. This process of merging components and aspects is called *weaving*. The *weaver* is a kind of pre-processor that identifies *join points* between aspects and components, and apply the transformations described in the aspects to the components code. The result is a woven code including into components the modifications due to aspects. This code can then be classically compiled to obtain executable programs.

Separation of concerns is then supported by the fact that features cross-cutting components code can be expressed easily in separated modules that are aspects. An aspect will group in a separated module all the code related to a given concern as well as the way to apply them to components code.

Different experiments are developed to support AOP. The main one is AspectJ [AJ], developed by Xerox Parc Corporation (Palo Alto, California) and provides a general-purpose aspect language a weaver to apply them to Java code. Other experiments are carried out to propose frameworks to support AOP, such as the Smalltalk framework of Kai Böllert [Kai98], or *TyRuBA*, the meta logic programming system supporting AOP of Kris De Volder [DVDH99].

II.2.3) Applying separation of concerns to RECINTERNET

As a web-based three-tiers application, RECINTERNET deals with different complex concerns such as distribution, different communication strategies, connections control, concurrency... It seems then particularly interesting to apply separation of concerns techniques to this project. Developing RECINTERNET with separation of concerns techniques will then be consequently benefic in terms of code understandability, easy evolution, reusability and complexity decreasing.

As we have just seen in the previous part, many techniques are being developed to support separation of concerns programming. As this domain is quite new in software engineering research, most of these techniques are still in definition stages and permanently evolving. It seems then difficult to apply them to real-world applications development, such as RECINTERNET.

However Xerox Corporation has developed an efficient practical solution to support aspect oriented programming with Java: AspectJ [AJ]. This tool provides a language to express different kinds of aspects and to merge them with Java code. The resulting code is usable as any Java code and has the advantage to give detailed explanations about the way composition between aspects and components has been realized. Plugging in or out aspects to components code is practically simple thanks to the AspectJ "pre-processor" that composes the output code. It seems then particularly benefic to develop RECINTERNET with AOP, and to implement it with AspectJ.

As a technique based on components implementation, AOP tools like AspectJ do not provide any support for designing systems with aspects. Aspect-oriented design is an active domain of software engineering, but time is needed before it reaches maturation and provides efficient techniques and tools for supporting AOP at design level. In this context it was particularly important to factor out a way to realize aspect-oriented design that could be applied to RECINTERNET. We present this point and also give further explanations about AOP and AspectJ in Section III.

II.3) Architecture alternatives

In web based applications, the internet is used to realize a communication between remote machines. Particularly, in the case of an application like RECINTERNET, a user runs a client application to retrieve data from a remote database. This kind of scenario is widely used in applications known as client/server applications.

In client/server applications based on a database access, three parts are generally defined in software architecture: the *User Interface,* the *Business Logic* and the *Data Management. User Interface* contains the support for presentation and user interaction. *Business Logic* groups the processing of the different requests addressed by the user, and *Data Management* supports the access to the database.

Different software architectures are possible for dividing these three parts. The following sections present the existing ones: *two-tiers, three-tiers* and *multi-tiers architectures.*

II.3.1) Two-Tiers Architecture [GR96], [C/S], [Hun98]

The two-tiers architecture involves two computers: a client machine and a server machine. These two remote tiers of the application communicate through the internet. The computing client talks directly to the server with no other intervening processes. There can be multiple clients using the same server.

In the case of a database application, the client communicates with a data server running a DataBase Management System (DBMS) that support the database access. The Figure 12 shows the two-tiers architecture.

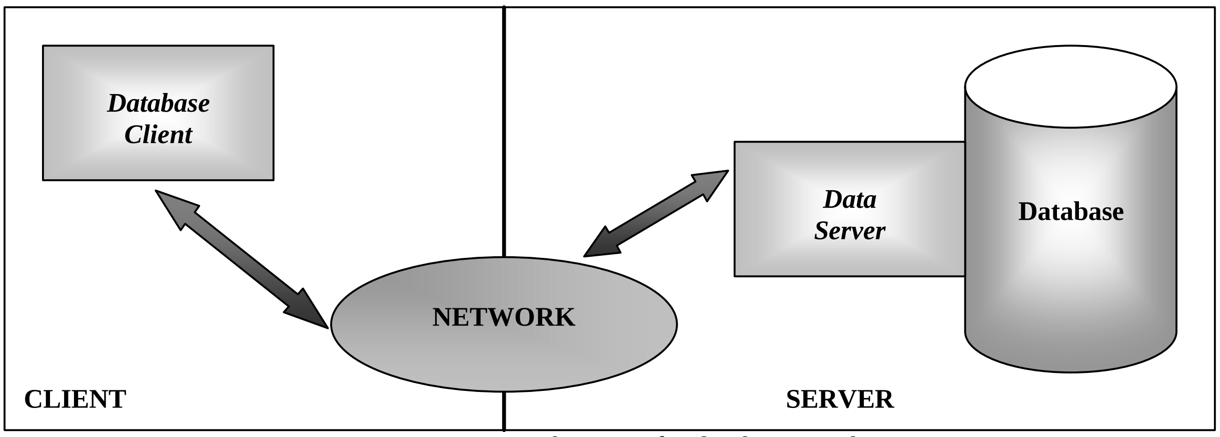


Figure 12 - Two-tiers architecture for databases applications

The three parts of the software (*user interface, business logic* and *data management)* are divided between these two tiers:

- the client tier application covers the *user interface* tasks and a part of the *business logic* tasks
- the server tier application covers the other part of the *business logic* tasks and the *data management* tasks

In fact the client application enables the user interaction and composes the corresponding requests to the database (generally in a SQL-like language), and then transmit these requests to the server tier. When receiving the results from the server, the data returned can be manipulated by the client application for further sub-selection, business modeling, reporting, visualization, etc...

The server application runs the DataBase Management System (DBMS) in order to processes the clients requests to retrieve the appropriate data from the database and to support database updating and integrity checking tasks.

The communication between the client and the server is assured through the internet, and can be realized by different ways (see Section II.6).

The two-tiers architecture is a simple architecture that is perfectly suitable for many applications. Its main advantages are that a two-tiers architecture software is developed faster than more complex architecture software. This kind of architecture is perfectly suitable for applications destined to a few number of users and where there is no need of elaborated processing between the user interactions and the requests transmission to the server.

However this kind of architecture is often not enough robust for elaborated database applications. First of all performance deteriorates with high number of users, because clients connect directly to the server. Secondly, the client application must handle all the processing corresponding to SQL preparation and SQL exploitation, and it can compromise flexibility since each client application need to use the correct requests corresponding to the server requests processing management. Finally two tiers architecture are not suitable for elaborated applications when important processing tasks are required between user interaction and requests transmission to the server, since client applications would then be complex and maybe difficult to deploy.

II.3.2) Three-Tiers Architecture [Hun98], [GR96], [C/S]

Three-tiers architectures insert one middle-tier between the client and the database server. We will then have a client application running on a machine and communicating through the internet with one server application running one another machine. This server will communicate with a database server, potentially running on another machine. This three-tiers architecture is depicted in the Figure 13.

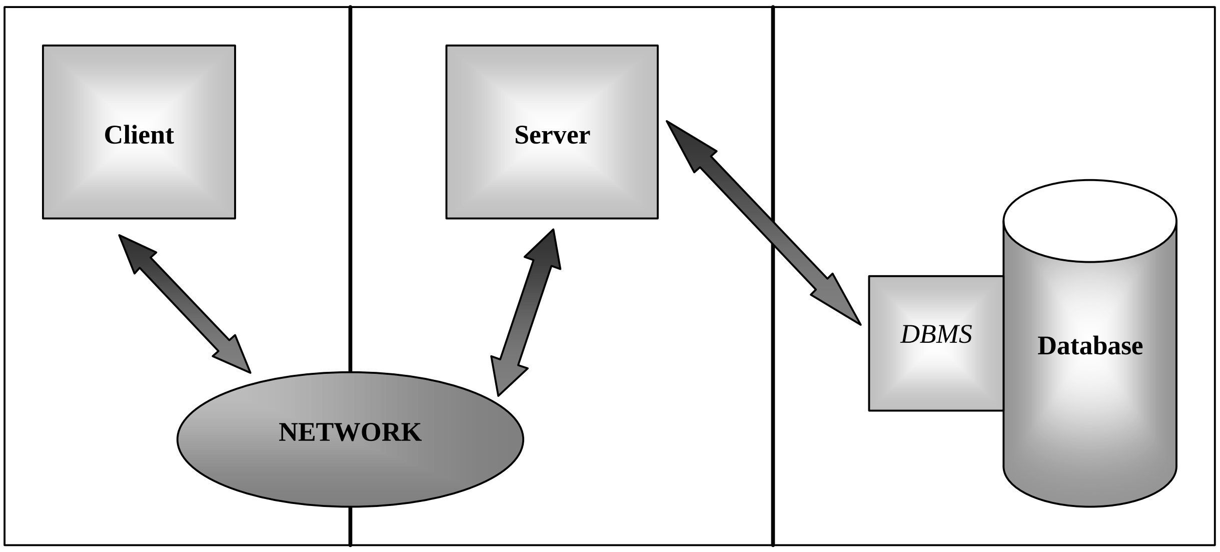


Figure 13 - Three-tiers architecture for database applications

The middle-tier in three-tiers architecture (that we call *server)* enables to totally separate the *business logic* part of the application form the client tier. In a three-tiers application, the different tasks are cleanly divided as following between the different tiers:

- The client tier covers the tasks related to *user interface*
- The server tier covers the tasks related to *business logic*
- The data server covers the tasks related to *data management*

Concretely the client application supports the user interaction and transmits the resulting demands to the server. The client application is then more specific than in a two tiers application since it does not realize processing (or only few) when the user interacts, but just transmit the demand to the server.

The server application manages the demands coming from the different clients and realizes the appropriate processing corresponding to each of them. The server is also responsible to provide to the clients the data they asked. Then the server code the corresponding SQL queries and transmit them to the data server and collect the answers, in order to be able (eventually after processing the results in different ways) to send the appropriate answer to the client. The client is the totally freed by the different tasks related to database access.

The data server is just running a DataBase Management System (DBMS) and is able to receive the SQL queries of the server. It processes them and return the appropriate results.

In a three-tiers architecture there are two kinds of communication. The first one is the communication between the client and the server. This communication is achieved through the internet, potentially with one of the solutions described in Section II.6.1 (HTTP – HyperText Transfer Protocol, TCP/IP – Transmission Control Protocol/Internet Protocol, Java RMI – Remote Method Invocation, CORBA – Common Object Request Broker Architecture, etc...).

The communication between the server and the data server can be done remotely or not. Here are used some standards for database connections that are concretely achieved through different drivers for the different kinds of databases used (see Section II.6.2).

Three tiers architectures provide many advantages compared to two tiers architecture. First of all the clear functional separation between the three tiers can clarify the development process and the modularity, maintainability and flexibility of the entire system. The use of the middle-tier enables also to have a robust application since this tier can organize queuing mechanisms. It also increases considerably the efficiency of he entire system since the client can process other tasks once he had transmit its demand to the server. This one will be in charge of realizing it and then will re-contact the client with the result. Moreover the client is “lighter” than in two-tiers architecture since it does not deal with anything specific with database (for example the client never uses SQL, it just sends parameters to the server that creates the appropriate SQL requests). Finally introducing a middle-tier enables to process specific computations to react to user interaction, and not only access a database.

All these improvements compared to two-tiers architecture are unfortunately paid by the increased needs for more network traffic management (two connections instead of only one with two tiers), and even if they increase robustness of the application, specific developments for client connections (such as queuing systems).

II.3.3) Multi-Tiers Architecture [Hun98], [GR96], [C/S]

Multi-tiers architectures are a particular case of three-tiers architectures: the *business logic* tasks of the middle-tier can be realized by more than one server, and with connection to more than one database. Then we have a n-tiers architecture with multiple (n) servers and/or multiple database servers with their own DBMS. It can distribute clients demands across multiple servers and can access data in multiple databases. The Figure 14 - Multi-tiers architecture for database applications presents this architecture.

This architecture derived from three-tiers architecture is used when several databases must be accessed, or when there must be different servers in order to deal with different kind of requests from clients. It is particularly interesting to use this kind of architecture to distribute the clients requests between multiple servers in order to divide the work between the different CPU of the servers (load-balancing).

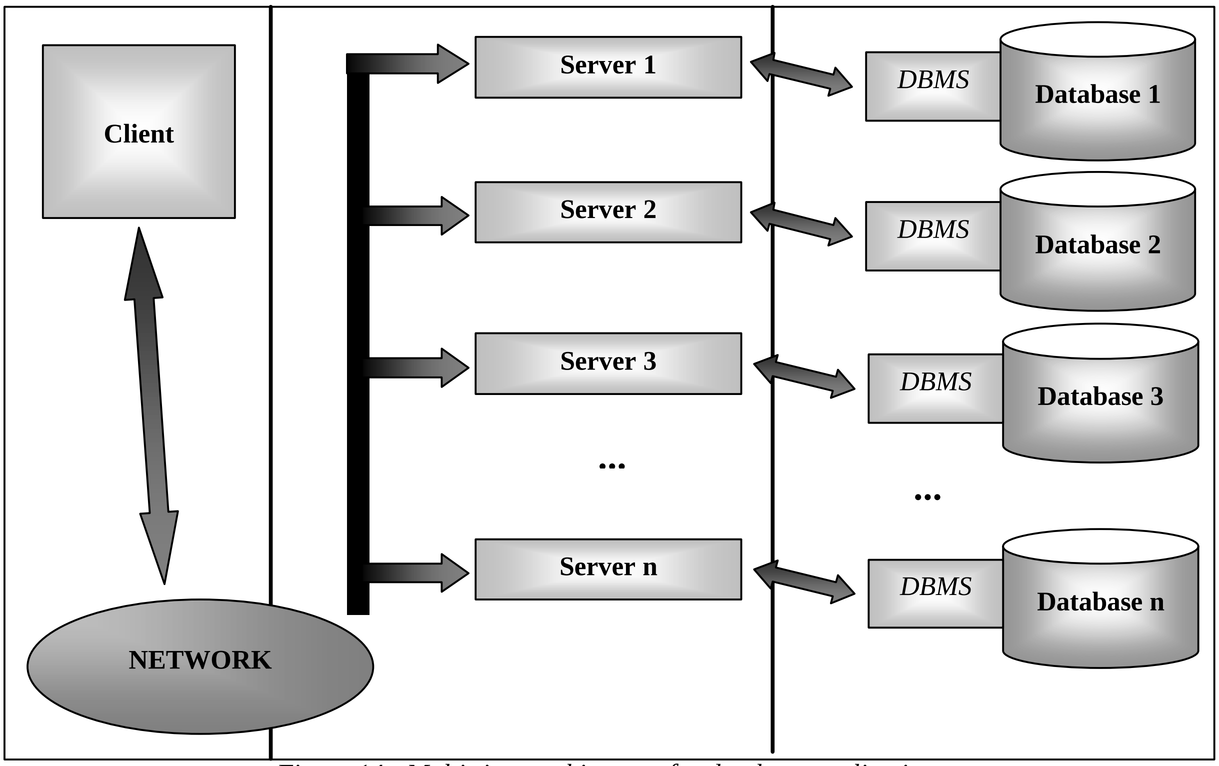


Figure 14 - Multi-tiers architecture for database applications

- ***Server Local Structural Database***

We present in this section a specific case of multi tiers architectures. This improvement is interesting only in certain conditions. The idea is to use a second database that is a structural copy of the classical main database. This second database will be located on the middle-tier host. Both databases will be accessed by a unique server. This special multi-tiers architecture is illustrated in the Figure 15.

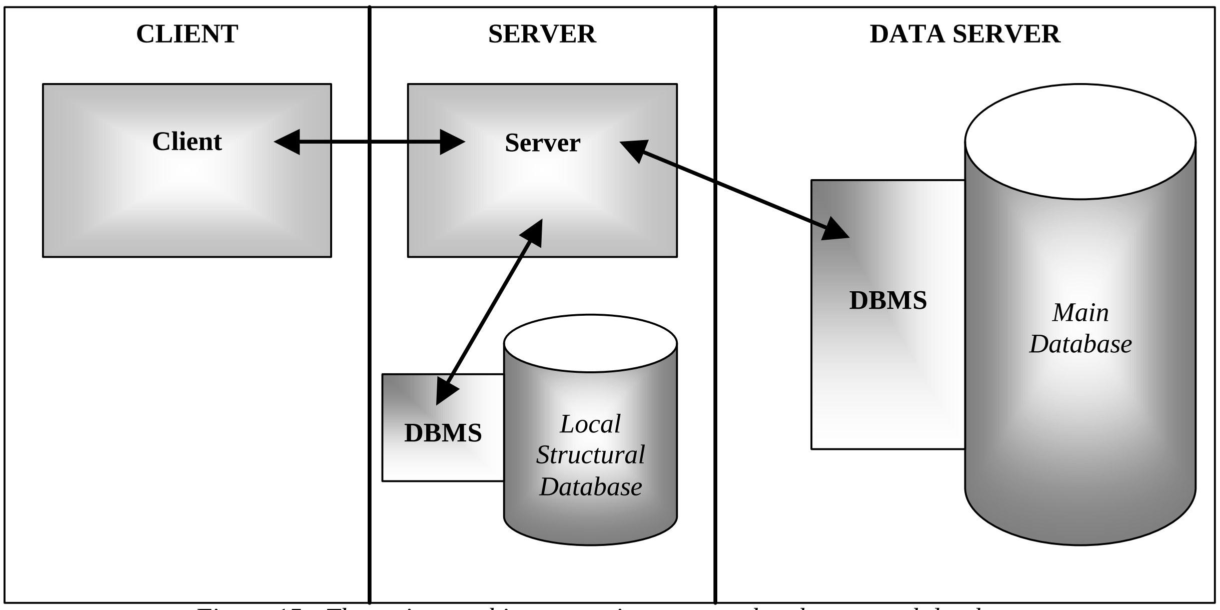


Figure 15 - Three-tiers architecture using a server local structural database

The Server Local Structural Database is a “structural” copy of the main database. It means that this database is stored only the structure of the main database, and not all the entries. As an example, we can imagine that in the main database there is a table containing a succession of entries called Birth representing respectively the number of girls and the number of boys born during each of the last 20 years in Brazil. The information copied in the server local structural database will only be the name of this succession of entries, the two sexes and the 20 years, but not the numbers of each case.

The aim of this second database is a question of efficiency. The mechanism to retrieve data from the main database is then the following: the client application transmit demands to the server application. This one accesses only the server local structural database to provide the data requested by the client, except when the data entries of the main database are needed (then the server connects to the main database to submit its request).

The advantages of this improved architecture is the efficiency of the system. First because we decrease the number of connections to the main database (this is particularly interesting when this main database is located on another host than the server), replacing them by local connection. Moreover the local database is smaller than the main one, so request proceeding in this database is faster than in the main one.

This approach is particularly interesting when the main database is a huge one that can be “structurally copied”, and when this database suffers from multiple access from many users and different applications. It is of course interesting if parts of the users requests do not require the data entries only located in the main database.

This approach is subject to several restrictions and can be applied only in specific conditions: as there are two databases that are supposed to share the same information, the coherence between the two must be maintained. Concretely it means updating the local structural database when the main one is modified and vice et versa. When this updating cannot be done dynamically, a careful use must be defined in order to have a coherent system. Also it needs to install a database on the server host, with a DataBase Management System (DBMS).

II.3.4) RECINTERNET architecture

In the RECINTERNET case, it seems unsuitable to adopt a two-tiers architecture. First because the interaction of the users require some processing before being translated as database queries. It would mean including this processing in the client application, what would increase its complexity and decrease its flexible deployability. Moreover, the connection to the SINTESE database (where are all the data requested by the client) can only be achieved through particular connections (see Section II.6.2.2), and it is complex to encapsulate them in a client application distributed through the internet.

The three-tiers architecture seems particularly better in the RECINTERNET case. It enables to encapsulate the functionality processing related to user interaction in an application separated from the client application. It also provides the possibility to program efficiently connection strategies (such as clients demands queuing, database access control...). Finally all the code related to database connection will be totally separated from the client application.

In the case of RECINTERNET, we will use an ameliorated multi-tiers architecture with one server and two databases: the main one and a server local structural database[4]. The RECINTERNET project fulfills all the specific conditions for using such an architecture. First, the data entries of the main database are only necessary when the user submits its final request, because the user can use only the structural information contained in the structural database to compose this final request. Secondly it must be remarked that the RECINTERNET system is only used to consult data from the SINTESE database, and that it never modifies any data of this database. So there is no need to check the coherence of the SINTESE database with the local structural one since this last one is never modified. In the other way, the SINTESE database is only updated rarely, and the updates are planed and realized at once. Moreover a system (called ADMSINWIN) exists to start an automatic update of these kind of structural copies of the SINTESE database. It seems then possible to use this ameliorated three-tiers architecture since the coherence between the two databases can be guaranteed.

[4] We should call this architecture multi-tiers architecture, but as it is just a variation of the three-tiers architecture, we will use the term three-tiers architecture in this report.

II.4) Internet Navigation Design

Hypermedia applications are applications were users can navigate in a system to define the way they want to organize or "visualize" some multimedia data (such as images, text, video, sounds, etc...). Hypermedia applications design shares many points with navigational applications such as RECINTERNET, were a user navigates to organize and visualize some data coming from a database. In this context it is interesting to have a look on the design methodologies dealing with navigation that were developed for hypermedia applications.

Designers of navigational applications have to face different difficulties such as combination of different ways to navigate within the same application, appropriate representation of data, navigation efficiency, etc... All these problems should be solved in a systematic and modular way, through a design methodology separating the addressed concerns in distinct design activities, each expressed at the proper stage and at the proper level of abstraction.

In this context were developed hypermedia design methodologies such as RMM – Relationship Management Design Methodology [ISB95], HDM – Hypermedia Design Model [GSP93], or OOHDM – Object Oriented Hypermedia Design Method [SRB96]. These methodologies aim to separate hypermedia application design in distinct stages dealing each with a particular view on the system. This design stage decomposition enables iterative and incremental development life cycle.

As we will explain it in this part, some principles of the OOHDM methodology can be easily and beneficially applied to navigational applications such as RECINTERNET. That is the reason why we give an overview of this method specific to hypermedia applications and also to a hypermedia design patterns system that can be applied to navigational applications.

II.4.1) OOHDM [SRB96], [SR98]

The Object Oriented Hypermedia Design Method (OOHDM) is a model-based approach that provide ways to realize a robust and efficient design for large scale hypermedia applications. The idea of OOHDM is to use classification, aggregation and generalization/specialization mechanisms in an object oriented framework to allow a concise description of complex information items and in the same time to allow the specification of complex navigation patterns and interface transformations.

Hypermedia applications are then built in four distinct steps that allow an incremental development process: *conceptual design, navigational design, abstract interface design* and *implementation.* Each step is focused on a particular design concern, and leads to the building of an object-oriented model or implementation. Table 3, extracted from [SR98], summarizes the different steps, products, mechanisms and design concerns addressed in OOHDM.

- In the C*onceptual Design* step, a conceptual model of the application domain is built using well-known object-oriented modeling (as OMT or UML) and relation-entities principles. The idea is to represent as neutrally as possible the domain semantic, with no concern for the types of users and tasks. Finally the conceptual schema resulting of this design step will be a model created thanks to sub-systems, classes and relationships of the domain entities that will be used in the application.

- In the *Navigational Design* step, the hypermedia application is defined in terms of navigational structure. This navigational structure deals with entities such as navigational contexts, which are induced from navigation classes such as nodes, links, indexes... This navigational structure takes into account the types of users and their tasks. Nodes represent logical navigational entities ("windows", "screen", "view") involving conceptual classes defined during domain analysis. The idea is that different navigational models can be built out of the same conceptual model, as different views of the same domain. Finally in this step is defined the navigational semantic in terms of nodes and links.

Activities	Products	Formalisms	Mechanisms	Design Concerns
Conceptual Design	Classes, sub-systems, relationships, attribute perspectives	Object-Oriented Modeling constructs; Design Patterns	Classification, aggregation, generalization and specialization	Model the semantics of the application domain
Navigational Design	Nodes, links, Access structures, Navigational contexts, navigational transformations	Object-Oriented Views; Object-Oriented State charts; Context Classes; Design Patterns; User Centered Scenarios	Classification, Aggregation, generalization and specialization.	Takes into account user profile and task. Emphasis on Cognitive aspects. Build the Navigational Structure of the Application
Abstract Interface Design	Abstract interface Objects, Responses to external events, Interface transformations	Abstract Data Views; Configuration Diagrams; ADV-Charts; Design Patterns	Mapping between navigation and perceptible objects	Model perceptible Objects, Implementing chosen metaphors. Describe interface for navigational objects. Define lay-out of interface objects
I nplementation	Running application	Those supported by the target environment	Those provided by the target environment	Performance, Completeness

Table 3 - OOHDM development steps

- In the *Abstract Interface Design* step is built an abstract interface model defining perceptible objects (e.g. a picture, a city map, a text, etc...) in terms of interface classes. Interface classes are defined as aggregations of primitives components (such as text fields and buttons) and can also be composed of interface classes. These classes are used to give an perceptible appearance to the navigational objects defined in the previous step. Events handling and links between the different interfaces and the navigational objects corresponding define the interface behavior.

- In the *Implementation* step interface objects are mapped to concrete programming language objects. A same interface model can be integrated into different kind of hypermedia applications such as web-sites, client/server with databases, multimedia guided tours...

This division in four step used in OOHDM is really suitable for huge hypermedia application development. It provides a way to proceed a incremental development process since each of the four steps used provides a model covering a clearly separated aspect of the design process. During each step a set of object-oriented models describing particular design concerns are built from previous iterations.

Moreover the separation of conceptual, navigational and interface design allows to concentrate on different concerns one by one. It results in modular and reusable designs, encapsulated in a general methodological framework where designing experience is represented into different modules dealing each with a particular design concern (conceptual, navigational and interface).

II.4.2) OOHDM design patterns for web-based applications [LRS98], [RSG97]

Design Patterns have known an increasing success in software engineering since they provide a way to record design experience and to simplify consequently software development through reuse of patterns which are known as being appropriate design solutions for recurrent problems of software design. In [GHJV94] are named, explained and evaluated the most famous important and recurrent designs in software systems.

Sometimes it is possible to structure simple patterns in order to develop a pattern language, that is to say a set of patterns that are often used together in a given application domain. One major idea of the OOHDM method is that it is possible to define design patterns specially applicable for hypermedia application design.

In this context OOHDM defines a set of simple design patterns that address usual concerns of hypermedia applications. These specific design patterns are grouped in a pattern language that is basically suitable for the OOHDM method. These patterns are divided in three groups serving the model decomposition of the OOHDM method: *Patterns for Hypermedia Systems*, *Navigational Design Patterns* and *Interface Patterns.*

- *Patterns for Hypermedia Systems*

This group of patterns presents the bases of the OOHDM decomposition. Here are defined which mechanisms can be used to differentiate conceptual model, navigational model and interface model. These patterns can be used to build hypermedia applications, or to extend conventional applications with hypermedia functionality. The initial patterns of this group were: *Node as a navigational view, Link as a relationship view, Anchor, Navigation strategy, Navigation observer, Node class – link class,* and *Wrapper node.*

- *Navigational Design Patterns*

This group of patterns present the way to design navigational mechanisms. Patterns of this group can be used for organizing the navigational structure of a hypermedia application in a clear and efficient way. The initial patterns of this group were: *Node as a single unit, Node creation method, Link creation method, Navigational context* and *Active reference.*

- *Interface Patterns*

This group of patterns provides solutions to interface design problems. The given patterns are environment independent but can be used by hypermedia GUI (Graphical User Interface) designers to efficiently organize graphical interfaces. The initial patterns of this group were: *Information on demand, Information-interaction decoupling, Information-interaction coupling, Behavioral grouping, Behavior anticipation* and *Process feed-back.*

As an example, the detailed description (from [RSG97]) of the *Hypermedia System Pattern* named *Node as a Navigational View* is given in Annex B. For complete definitions of all the OOHDM patterns, we refer to [LRS98] and [RSG97].

To put it in a nutshell, OOHDM defines a pattern language (composed of several related patterns applicable at different design level) and integrates it at the different steps of its design models decomposition (conceptual design, navigational design and interface design).

OOHDM patterns are particularly interesting since they provide appropriate and proven solutions in navigational and interface design. Moreover they can be used also in other contexts that pure hypermedia navigation, because they address problems of domains (navigation structure, interface organization) present in many software development (GUI design, navigation, etc...).

II.4.3) Applying OOHDM to RECINTERNET

RECINTERNET is incompletely concerned by the OOHDM methodology since it only deals with navigation and visualization of literal data, and not multimedia items such as images, video, sounds, etc... Moreover the different navigational units (screens or nodes) that will be used in RECINTERNET navigation are not purely views on the conceptual entities as defined in OOHDM, but more logical divisions for requests composition.

However many concepts of OOHDM are totally suitable in the RECINTERNET case. The main characteristic of RECINTERNET is to be a dynamic interface for the internet. It means that users navigate to compose as wanted their request. Then OOHDM navigation principles are absolutely adaptable to this navigation. And even if the RECINTERNET interface does not require multimedia items "visualization", its graphical aspect (nodes components appearance, window organization, look, etc...) is important, and requires an appropriate design methodology such as OOHDM.

That is the reason why we decided to apply the design decomposition proposed in the OOHDM methodology. The RECINTERNET design will be then separated into a conceptual model (where we represent the data on which is based the application), a navigational model (where will be defined the navigational structure and mechanisms) and a interface model (where will be defined the graphical appearance of the application), independent from the programming environment chosen. This decomposition will be particularly interesting for addressing separately the different parts of the design cycle, and will provide robust and appropriate models for a simple implementation of this complex application.

In this context several patterns of the OOHDM patterns language can be used in the RECINTERNET design. Many of the patterns of each of the three groups *(Hypermedia Systems Patterns, Navigational Design Patterns* and *Interface Patterns)* can be used to simplify the design process and to provide a robust and proven solution to the RECINTERNET application.

II.5) *Some technologies for the three tiers*

Many applications are based, as this is the case for RECINTERNET, on a three-tiers architecture (client/server/shared resources). As we have already explained it clients are characterized by the fact that they are requesting services, servers by the fact that they are providing services, and shared resources by the fact that they are used by servers to provide services.

In the case of RECINTERNET, clients are applications ran by users and using the internet to communicate with a server. This server is situated on another machine and manage the requests of the clients by accessing databases.

As the number of systems of this type is increasing consequently and quickly, many new related technologies are also emerging. Anyway few of them are efficiently usable in the case of applications like RECINTERNET. In this section we will present the main technical solutions existing for implementing each of the three tiers of applications like RECINTERNET[5].

II.5.1) *Client*

The following sections present the best existing technologies for programming efficiently a client application that uses the internet to communicate with a server.

II.5.1.1) *HTML* [Html1], [Html2]

HTML (HyperText Markup Language) was developed at CERN by Tim Berners-Lee, who is now Director of the World Wide Web Consortium (W3C) at MIT's Laboratory for Computer Science. HTML descends from SGML (Standard Generalized Markup Language), the ISO standard language for text. SGML is in widespread use by the US Government and the publishing industry for representing documents.

HTML is not a complete programming language but a simple markup language. An HTML document is an ASCII text with markups (embedded instructions) that affect text display. It is based on tags (such as `<UL>`, `</B>`) to specify text, hyper links or components format.

Basically a web browser will fetch the HTML document by it's name (that can be a URL for example), interpret the HTML and display the document. This process can also involve additional HTML documents fetching or special areas display that can accept user inputs or involve other HTML documents fetching. Then an HTML application can be seen as a collection of related web pages managed by a single HTTP (HyperText Transfer Protocol is the TCP/IP protocol that defines the interaction of WWW clients and servers) server.

Through this language it is possible to visualize elaborated components such as lists, buttons, tables... It is then possible to handle users interaction such as button click, item selection in a list, etc... Successions of HTML documents can be accessed through hyper links navigation. In the section II.6.1.1 we will describe how HTML clients can communicate with servers to handle elaborated users interaction.

The main advantage of programming a client in HTML is platform independence. HTML is supported by all web browsers since HTML documents displaying is their basic feature. This enable to reduce consequently the important proportion of developers resources used for developing and maintaining versions of their products for the different hardware/software platform combinations.

On the other hand programming a client purely in HTML is not really practical. HTML language is limited in its computational power. Moreover it only provides a set of tags but no possibility of abstraction in data

[5] Most of the technologies depicted in this part are based on new or always evolving features of programming languages or communication systems. Therefore this part of this state of the art is mainly based on recent descriptions, often only available online on internet, and not formally described in books or papers. Anyway the corresponding references will be provided as precisely as possible.

structures or procedures. Programming in HTML requires then a very low level programming and a laborious repetition of code modules that leads to complicated high size code files.

Sections II.5.1.3 and II.5.1.4 show that HTML documents are in general used as a support for other kinds of client programming.

II.5.1.2) Application

Another solution to program a client using the internet is a basic application. It means that the client will be an application developed in any programming language and ran by the user on a machine.

The chosen programming language can be any programming language that provide possibilities to develop communication with a server through the internet. The permanent increasing of internet activities led to the development of internet programming facilities in many programming languages such as internet components for Delphi 4 [Can98], Distributed Smalltalk [Stk], programming Ada WWW applets [Ada], etc...

In the domain of programming languages targeted at Internet applications, Java is today viewed as the most appropriate one. Particularly adapted for internet communication (for example through Java Remote Method Invocation, described in Section II.6.1.3) Java offers important GUI (Graphical User Interface) libraries for client programming.

Without entering into details specific to each characteristics of each programming language, in the case of client programming with internet communication, the main problems of most programming languages (except Java) is platform dependence and immature or complex internet communication facilities (since they are languages not explicitly developed for internet applications). From this point of view, the use of Java brings platform independence since Java programs code is compiled in bytecode that can be run in every Java Virtual Machine.

Anyway, in any case an important disadvantage of using an application developed in a programming language as client is the fact that it requires a specific deployment in each machine where a client is run.

II.5.1.3) ActiveX [ActX1], [ActX2]

Used extensively across the Internet computing environment, ActiveX controls are being employed as site-enhancing objects, aids for application development, and standalone programs.

We can define ActiveX as a set of technologies that allows developers to build objects that can interact with one another in networked environment. These objects are self-contained and become independent from the programming language in which they had been created (Java, C++ and Visual Basic support ActiveX programming [ActX1]). ActiveX is built out of two Microsoft technologies:

- DCOM (Distributed Component Object Model) provides the low-level, object-binding mechanism that lets objects communicate with each other (locally and remotely). Like Java Remote Method Invocation (RMI), DCOM provides transparent message-passing between objects located in different machines with different operating systems. [ActX1]

- OLE (Object Linking and Embedding) uses DCOM to provide high-level application services, such as linking and embedding, to let users create compound documents. Built on the foundation of Component Object Model (COM, the predecessor to DCOM), OLE is optimized for end-user usability and integration of desktop applications. [ActX1]

ActiveX controls are the basic elements of this ActiveX technology. They describe themselves through a binary description file that lists the properties, methods and events that should be exposed. They can be embedded in web sites and respond interactively to events, providing high level functionality.

In this way ActiveX can be used to program a client communicating with a server through the internet. ActiveX components will be embedded in HTML documents that can be viewed with web browsers. When these HTML documents will be fetched from a HTTP server, the ActiveX controls will be downloaded and saved in the user hard drive and then executed to compute and manipulate data or communicate with other controls.

Programming a client with ActiveX provide a high level interaction functionality and also the possibility to compose easily specific components (ActiveX controls) to build an entire application. An important advantages of this technologies is that once the different components of the application have been downloaded, they can be use very efficiently (in terms of execution speed).

The main drawbacks are security risks and portability. There is no limitation to actions realized by ActiveX controls. They can use functionality of the machine on which they are run, and then security issues are absolutely not assured. ActiveX system only uses "cryptographic" techniques to make sure that an application comes from its supposed point of origin, and that it has not been altered by any party along the Internet's string of host-to-host connections. From the portability standpoint, the ActiveX binaries are only runnable in Windows environments. This is a particularly important drawback of this technology.

II.5.1.4) Java Applet [DD97a], [FC97], [Applet]

Java applets are Java programs intended not to be run on their own, but rather to be embedded inside another application. The Java bytecode corresponding to the applet can be loaded from a remote machine in order to run it on any Java Virtual Machine accepting applets.

Java applet code is compiled into Java bytecode. This bytecode can be embedded in a HTML page (thanks to the `<APPLET>` HTML tag). In this case the bytecode files are stored with the HTML document file on the HTTP server. When a web browser will load this HTML page from the HTTP server, the classes described in the applet bytecode will be loaded in the Java Virtual Machine of the web browser. Web browser will then start the applet execution in its Java Virtual Machine. When new classes are required for the applet execution, they are loaded automatically from the HTTP server. In this way we obtain an application easily deployable in any web browser.

Applets are implemented in Java and must respect standard methods needed for any web browser to run them. The Applet class provides a standard interface between applets and their environment, in order to manage user interaction, communication with other applets, information providing for the web browsers. Java GUI libraries (such as Abstract Window Toolkit or SWING) are then available to develop through applets really efficient web applications.

Security aspects are assured by the fact that applet execution is limited to the web browser Java Virtual Machine. Web browsers running applets must ensure some security restrictions in applet execution:

- Applets cannot read or write in local files
- Applets cannot start a communication session with another machine that the one from where they have been downloaded
- Applets cannot call executable applications on the user machine

Finally applets provide a good way to program client using the internet since they can be easily downloaded through HTML documents and they provide the functionalities of the Java language.

The main advantages of programming client with Java applets are platforms independence and easy deployment. As applets bytecode are interpreted in web browsers Java Virtual Machine they can be executed on any platform with any operating system only with a web browser recognizing the Java language version used. Easy deployment is provided through embedding in HTML documents (as simply as images). Applet way provide then easy application loading through classical HTTP downloading process.

A drawback in applet approach is that the applet bytecode must be downloaded each time the HTML page containing it is reload in the browser since there is no caching for applets loading in web browser. This mechanism is sometimes long and so initialization of an applet can take time.

II.5.1.5) Choice of a client technology for RECINTERNET

One crucial issue in the choice of the technology for RECINTERNET client programming was easy portability. It means being able for a new user to run the client whatever its platform and operating environment could be, and through an easy deployment.

This easy deployment constraint excludes the possibility of using a basic application for the client side, since application requires an installation process. Moreover evolutions of clients as applications require re-distribution to all the users and re-installation for each new version. Non Java applications also involve important work for providing versions for different platforms.

The concept of giving a user the possibility to run the client application only with a web browser on any platform with any operating system is particularly convenient for the RECINTERNET project. ActiveX is then excluded because of its restriction to Windows environments.

Applets provide more programming potentiality than HTML. They seem then particularly adapted also for communication with the server as we will describe it with Java RMI in Section II.6.1.3. They provide also the GUI facilities of Java and can really be used with the most used web browsers[6] (HotJava, Internet Explorer or Netscape).

RECINTERNET will then use a Java applet for its client side. It will then be possible to use Java programming facilities and library for creating clients that can interact in appropriate way with users and communicate efficiently with server, keeping in the same time an easy deployment mechanism for any platform or operating system.

II.5.2) Server

Before presenting the different ways to program a server, we need to open a special parenthesis about HTTP servers. In fact we have seen in Section II.5.1 that client applications can be embedded in HTML documents, and then downloaded through the HTTP protocol. This requires in the server part a HTTP server. This server will be contacted through a given URL and then provide to the different users the HTML documents and embedded applications requested. All the necessary files (HTML pages, images or applications embedded in these HTML pages) will be stored on this HTTP server, and then downloaded from it when necessary.

In addition to this HTTP server, another server will be needed for communication. The basic functionality of this server is to provide services to clients. Often these services require access to databases. That is the reason why the choice of the technology used for a server depends on the technologies chosen for clients (Section II.5.1) and databases (Section II.5.3), as well as the technologies used for client/server communication (Section II.6.1) and for server/database communication (Section II.6.2).

We will present in the following sections the best existing technologies to program a communication server that can efficiently communicate with clients and database and realize required computations.

II.5.2.1) CGI and Scripts [Cgi], [ND98], [Wil95]

CGI (Common Gateway Interface) provides a way to manage clients/servers application through the web. The CGI mechanism enables through HTTP protocol clients to request server services. In the current section we will only present the server side used with CGI: how can be programmed servers with CGI scripts.

[6] At the moment where this thesis took place, the version 1.2 of JDK was not supported by all the recent versions of the commonly used web browsers. It was then possible to fix this problem simply by downloading and installing the version 1.2 of Java Runtime Environment (http://java.sun.com/products/jdk/1.2/jre/) on these web browsers.

A CGI script is a program that is stored on the HTTP server and executed on this server in response to a request from a user (Section II.6.1.1 provides more details). A CGI script file is written in a programming language and can be compiled to run on this server (in languages such as C, C++, Visual Basic, Ada...[ND98]) or interpreted on this server (in languages such as Perl, JCL: Job Control Language, JavaScript...).

Clients can then start the execution of CGI scripts, and also provide these scripts some parameters through different mechanisms (see Section II.6.1.1). In this way, for each new request of a client, a new process executing the appropriate script is started on the server with the given parameters. If needed a HTML page can be returned to the client through the classical HTTP protocol (for example to show some results, or to say that the action was correctly executed).

It seems then that CGI scripts could be appropriate to program server side of applications such as RECINTERNET. However there are some important disadvantages in using CGI. First each action (like a button click, an item selection, etc...) of the user must be related to a script on the server, even if the corresponding action (provoked by this user interaction) is just a minimal one that should not request the server work. Then all the computational processes corresponding to all the users interaction are done on the server CPU. When dealing with an important number of clients, this can lead to performances worst than with other systems than CGI.

Secondly an important disadvantage comes from the fact that CGI transactions are absolutely stateless. It means that once a user has requested a service through CGI to a server, the server totally “forgets” everything about this user. If the server needs to do some computations related to each client that requests it, then if the same user requests several times the server, the server will have to redo these computations each time.

II.5.2.2) Applications

Another alternative for programming the server side of an application like RECINTERNET is to use an application running on the server machine. The server application will be developed in a given programming language and then ran on the server machine[7]. Its functions will be to communicate with clients (receive requests and return answers), to eventually compute some actions necessary for the service requested, and to communicate with databases to proceed some requests in these databases and receive the appropriate results. The interface used in this server can be very basic since it should require only few configurations and only compute actions for services requested by clients.

Developing the server side as an application requires a programming language satisfying some particular constraints. The first of them is an appropriate support to communicate through the web with the clients. Moreover the clients can be developed in a programming language, and then a particular solution should be found.

In this context the CORBA standard proposes a solution enabling communication between remote applications developed in heterogeneous languages (Java, C, C++, Smalltalk...), as it will be described in Section II.6.1.4. Thanks to CORBA, it is then possible to program in a transparent way the server application communicating with the clients one. The Java alternative to CORBA is Java RMI (described in Section II.6.1.3) that provides the same facilities of remote communication between a Java client and a Java server. It is then possible to choose to program the server application in a programming language providing some appropriate and easy solutions for communication with the client. The Section II.6.1 about client/server communication will bring more elements for this choice.

So a particular attention must be paid to the ability to support a multi-clients system on a same server. Since the server can be requested by many clients in the same time, the chosen programming language must provide an efficient way to execute a new process for each new client request, in other words it must support a safe multithreading mechanism. As described in [GmG95], multithreading is supported in most of the languages through added libraries for threads programming. A critical point is then thread safety, i.e. being sure that locks

[7] If the clients are Java applets downloaded from a HTTP server, it is necessary that the communication application server (to which are addressed the requests from clients) is located on the same machine than the HTTP server. In fact this restriction comes from the security restrictions of applets, that cannot open communication sessions with another host than the HTTP server from where they were downloaded.

can always be freed, even in case of threads abnormal termination. The Java Virtual Machine and the Java language are particularly adapted for multi-threads support through a sophisticated set of synchronization primitives and threads class, as well as a robust run-time threads management.

At last another point is crucial for the programming language choice: communication with the database. The chosen language must provide good support for database accessing. The main way to retrieve some data from a database is achieved with some SQL-like requests. In this context it is important to have an efficient gateway between the programming language used and these SQL-like mechanisms. A good point is that however numerous are the different kinds of databases, industrial standards are adopted to simplify programming, such as JDBC (Java DataBase Connectivity), ODBC (Open DataBase Connectivity) or OLE DB (Object Linking and Embedding DataBase). Then specific drivers are developed and available for most of the existing databases in many programming language (C++, C, Java). When choosing the programming language for the server application, it is important to know what are the facilities with the databases that will be used. More details about server/database connection are in Section II.6.2.

To conclude this section about applications, we could say that different programming languages provide some elaborated and robust support for the important aspects required when developing the server (communication with the client or the database, multi-client management). The choice of one language in particular depends of course of the technologies used for both databases and clients, but also of the specific constraints of the system.

II.5.2.3) Servlets [Servl1], [Servl2], [Zeig99]

Java Servlets provide through the simple and flexible *Servlet API* interesting features for request/response-oriented web programming. The Servlet API group the necessary technologies for such systems server side programming. Servlets can be seen as a considerable improvement in CGI-like web programming since they provide web developers particularly appropriate and adapted functionalities in server programming.

The main functionality of servlets is to provide a way for servers to respond to clients requests. A servlet is basically an executable Java code that will be run on the server when a client requests it, like CGI scripts. A servlet is described in a Java class that is loaded, instantiated and initialized in the executing environment of the server. The server is then able to call the different methods of this object when needed. The easy loading and unloading of a servlet on the HTTP server is supported through the basic Java methods used for servlets (***`init()`*** and ***`destroy()`***).

A client will then download some HTML documents from the HTTP server (the different servlets are running also on this server). When the user will interact through some elements of the HTML documents (such as clicking a button, selecting an item, navigating through a hyper link...), this interaction is transmitted to the HTTP server to create a request thread. This thread is given as a parameter of the appropriate method to the appropriate servlet. The servlet computes then the appropriate actions in the server execution environment and can return as a result a new HTML page to the client. In this way we can obtain dynamic answers to client interaction.

When receiving a request thread from a client (through the HTTP server), the servlet can know different information such as the different parameters corresponding to the request in the HTML document of the user, the name of the remote host where the client application is executed and the name of the server that received the request. The request is also transmitted to the servlet with an output “reference” that will be used to provide in the appropriate format the possible answer of the servlet to the client, such as a new HTML document.

It has to be noticed that unlike CGI scripts where a new script is executed for each client request, only one servlet is run on the server and will be in charge of responding to all the client requests, that will be queued and passes as requests thread to the servlet methods. A servlet can then support multiple requests concurrently and control the different clients access.

Being explicitly written in Java, servlets can use many features of the Java language. It is possible to develop servlets communicating together, for example to forward requests to different servlets in a load balancing system. It is also possible to use all the facilities of Java to access databases. Written in Java, servlets can also use

RMI or CORBA facilities to communicate with clients through the internet. As all Java code, servlets are platform independent and easily deployable.

A important advantage is also the fact that all the basic functionality used for servlets are included in Java classes and interfaces providing all needed to deploy efficiently servlets on a server. It is then easy to subclass these classes and interfaces in order to create more specific functionality to servlets.

Anyway the main restriction to servlets is that, even providing many programming facilities and potentialities, they are still limited to a CGI-like web programming. It means that servlets methods are executed each time a client submit a request through a classical HTTP communication mechanism (such as activating a component of a HTML document, or navigating to a new HTML document), but only for this kind of request. If the interaction of the user is embedded into a Java Applet for example, servlets methods can only be called through other communication protocols, such as Java RMI. This is possible but then the interest of using servlets is considerably decreased.

Finally, given this restriction, servlets provide an easy way to enhance consequently servers, thanks to their particularly simple and flexible use and thanks to the fact that they provide a way to use all the powerful features of Java.

II.5.2.4) Choice of a server technology for RECINTERNET

The main function of the RECINTERNET server is to be able to receive requests from the RECINTERNET clients, and thanks to connections and requests to the two databases (the local structural one and the SINTESE one), to return the required answered. Because of that the technology choice for the server depends a lot on the technology used for the client side (Java Applet) and on the technology used for client/server communication (see Section II.6.1).

CGI technology seems too restricted because of its “stateless” quality, and mainly because of the necessity of having a script executed on the server for each simple interaction of the user. It would be unnecessarily costly (in term of remote connections client/server) to access the server and execute a script on it each time the user interacts on the client application.

An application seems a good solution for the RECINTERNET server, and particularly a Java application since it presents all the facilities needed in the RECINTERNET case: appropriate communication protocol with the clients, appropriate communication protocol with the database and efficient multithreading support for multi-clients support. Appropriate developing environments are also available to program efficiently such a server application.

As explained before, servlets provide important server programming facilities but are basically conceived to respond to clients requests transmitted through the classical HTTP communication protocol. It seems them not particularly interesting in the RECINTERNET case to use servlets, because it would mean restrict the client interaction to interaction in HTML documents, and not in embedded applications such as Java applets.

An interesting alternative should have been to realize the server using both servlets and a classical Java server application, where the used servlets would have been able to communicate with the server application. Some controls on users connections or some load-balancing between several servlets should have been implemented through the facilities brought by the servlets way of programming, and also to use the Java applications facilities for communicating with clients. In the given time we focused our work only in developing the server side of RECINTERNET as a Java application

II.5.3) Database

Many companies that decide to realize three-tiers applications through the web where the third tier is a database generally reuse their own database, and do not construct a new database from scratch for this new application.

This way of doing avoids two important tasks necessary to create a database from scratch. First designing the database, which is a non-trivial task, and secondly re-computerize all the entries of the existing database, a potentially very fastidious task.

This is the case in the RECINTERNET project. As explained in Section II.3.4, two databases will be used. The first one is the local structural database. This database is located on the same machine than the server, and contains all the information necessary to compose the final request of the user. The second one is the SINTESE database, located on the DATAPREV intranet. This database will be remotely accessed by the server to get the results of the final request submitted by users.

The local structural database was designed for the RECSINWIN project, and can be reused for the RECINTERNET project since in both cases this database is used for the same kind of requests.

The SINTESE database is a very huge one, used since a long time by several distinct applications. The RECINTERNET system is basically designed for accessing this database, and it cannot be modified since its size is too important and since it is also used by other applications.

II.6) Communication between the three tiers

Web based three-tiers applications use a server tier in order to provide a layer between a client and a third tier (databases, printers, files, high-powered processors...). Requests of clients are processed by the server using the third tier resources (in this case databases). Consequently appropriate solutions have to be found in order to support communication between clients and server, and between server and databases.

II.6.1) Client/Server communication

The following sections present the main existing technologies to support communication between client and server in web-based three-tiers applications.

II.6.1.1) HTTP communication protocol [ND98], [Wil95], [Servl1]

HyperText Transfer Protocol (HTTP) is used to support communication through the internet related to HTML documents. From a client to a server or reciprocally, this protocol is used to transmit different kind of files (such as HTML pages, images, mails...) or parameters.

Thanks to a web browser, the user requests a document through a URL (Uniform Resource Locator). The corresponding HTTP server receives this request and will send to the client web browser the appropriate document (that is stored on the HTTP server) in a MIME (Multipurpose Internet Mail Extension) format[8]. The document is received by the web browser and displayed for the client.

This first mechanism is achieved through the GET method of the HTTP protocol. This method is sent by the web browser to ask for a document. When using this method, the browser must specify different parameters such as the name of the requested file, the version of HTTP used, the MIME format it will accept in return and different characteristics describing itself. The HTTP server receives this GET request and sends to the user the code representing the MIME document with different parameters such as information on the server, the protocol used, the document.

Another mechanism exists for files transfers. It is achieved through the POST method of the HTTP protocol. HTML proposes some tags to include in HTML pages some components to enable user interaction (like text fields, text areas, buttons, radio buttons, etc..). This enable to present to the user some forms he can fill (for example some different text fields to enter personal information: name, age, city, etc...). When the form is complete, it is submitted by the client and then the POST method is used. As the GET method this method is sent to the HTTP server with some parameters, and amongst them are included the different values entered in the form. The server receives it and passes it to some server programs (like CGI scripts, servlets) that can treat it in order to compute an action and eventually returns a MIME document to the user, generally a HTML page.

These two mechanisms (POST and GET) are the base of any HTTP transfers through the internet. With the description of these mechanisms we can note that the HTTP communication protocol is particularly conceived for a certain type of client and server.

Using HTTP is particularly suitable for users interacting in a HTML document and its components. Appropriate servers for this kind of HTTP communication are basically CGI-like servers, where the server can address the requests received through GET and POST methods to some given programs (or scripts) running independently on the server, like CGI scripts or Java servlets.

The advantages of such a protocol are mainly that it is universally used by all the internet community, since it is one of the basic communication protocol standard on which is based the internet. The fact that the communication with the HTTP protocol is efficient and rapid is also a great advantage for using HTTP between a

[8] The MIME format groups many formats used in web transfer such as ASCII and non ASCII text files, GIF, JPEG image files, MPEG video files, WAV audio files, etc...

client and a server. Finally HTTP is also interesting since it is a protocol that does not suffer any restrictions for firewalls crossing.

The critical limitation of this protocol is that it is by definition limited to its two classical communication mechanisms (GET and POST). These two mechanisms are very efficient but do not provide adequate support for more elaborated exchanges between entities of two remote applications. Finally HTTP is an efficient communication protocol, but limited by its too low-level communication mechanisms.

II.6.1.2) Socket-based communication [DD97b], [MAB+98]

Another alternative for the client/server communication is the use of sockets. Different programming languages provide sockets support for networking (C++, Visual Basic, Java...). Sockets exchanges are based on the TCP/IP (Transmission Control Protocol/Internet Protocol) suite of protocol and provide a way to transfer data between two parties through streams connection.

TCP/IP is the suite of protocols that defines the Internet. It is named as TCP/IP for its two most important protocols TCP and IP. Originally designed for the UNIX operating system, TCP/IP software is now available for exchanges between many heterogeneous environments. TCP and IP are some layers used in the multi-layers systems used to achieve connection between remote machines. TCP/IP protocols provide "low-level" functions needed for networked applications.

As in Java, most programming languages supporting sockets provide sockets manipulation through I/O streams: a program can read from a socket or write to a socket as if it was a file. A socket represents a connection between two processes, potentially on remote machines. The protocol used for the transmissions in these connections is TCP/IP.

There are principally two kinds of sockets used: stream sockets and datagram sockets. Stream sockets are “connection-oriented” sockets since they provide a connection between two processes through a continuous stream. It can be compared to a phone call protocol, where once the two parts have establish the communication, the connection is maintained until the end. This kind of connection is realized through the TCP protocol.

The second kind of sockets are datagrams sockets. Datagrams sockets are “connectionless” since exchanges are realized through packets sends and receptions with no continuous connection between the two processes. It can be compared to the mail post service protocol, where letters are sent from to an address. This kind of connection is realized through the UDP (User Datagram Protocol) protocol, which is one of the TCP/IP suite of protocols.

Programming languages generally provide some classes (*java.net* package in Java) or libraries (*netinet* and *socket* in C++) grouping the needed elements to realize applications using sockets. The most common methods used in sockets management are for sockets creation, socket allocation, socket connection, waiting for a socket connection, accepting a socket connection, stream send and receive, closing a socket.

Then programming client/server communication with sockets is suitable for both client and server applications (or embedded applications such as Java applets or ActiveX controls) developed into a programming languages supporting it.

The advantages of using sockets for client/server connection is that as they are based on TCP/IP protocols, they can be used between applications running on heterogeneous environment. Moreover the good sockets support in Java and C++ for example enable to program efficiently connections between remote applications, but also to really control all the details of these connections since sockets provide a certain “low-level” communication control.

This “low-level” characteristic of sockets management is also a disadvantage since programming real exchanges between applications requires complex and elaborated communication scenario. In fact the sockets layer is generally covered by a high level layer that enable to program communications between applications more transparently.

II.6.1.3) Java RMI [Jdk], [Rmi]

One particularly interesting alternative for remote communication between applications is the Java RMI (Remote Method Invocation). Java RMI is used to interlink objects that are distributed throughout a network and physically reside on different machines (and running in different Java Virtual Machine). This is realized through a support for call of remote methods. From the application's point of view, a remote method and a local method are invoked in the same manner using the same semantics. RMI takes care of the details at a lower level.

In the distributed object model used in RMI, the methods of a class that can be called remotely are described in a RMI interface[9]. This interface lists the remotely accessible methods signatures. A remote object is then known by other objects as a Java interface. Concretely, the class of this object must implement all the methods that are described in the RMI interface. Through this interface mechanism, calls to local or remote objects methods are programmed with the same syntax.

Java programs with RMI use a specific RMI compiler *(rmic,* which is part of the Java Development Kit). For each class implementing RMI interfaces, this compiler generates two files: a *stub* and a *skeleton.* The stub resides on the client machine and the skeleton resides on the server machine. The stub and skeleton are comprised of Java code that provides the necessary link between potentially remote objects, and use object serialization to marshal and unmarshal parameters needed when realizing a remote invocation.

When executing a Java program using RMI, when a client object invokes a server method, the Java Virtual Machine looks at the stub to do type checking (since the class defined within the stub is an image of the server class). The request is then routed to the skeleton on the server, which calls the appropriate method on the server object. This method is executed in the server Java Virtual Machine. To put it in other words, the stub acts as a proxy to the skeleton and the skeleton is a proxy to the actual remote method. The Figure 16 illustrates this mechanism.

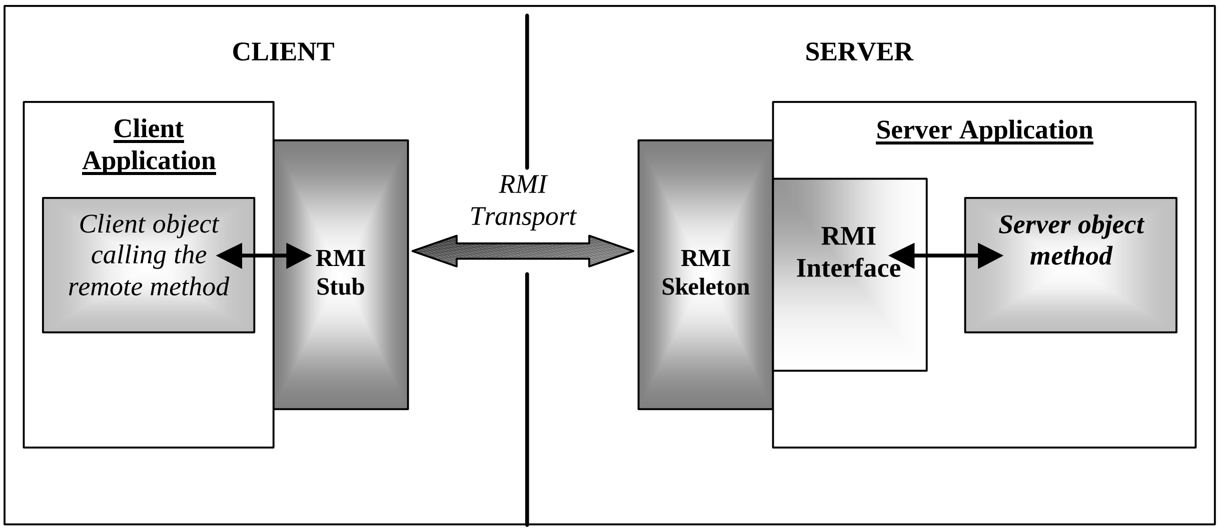


Figure 16 - RMI transport in distributed applications

Java RMI package provides different services. To only name the main ones, there is the *bootstrap naming service,* that is used to provide the concrete reference of server remote objects out of their given name[10], and there is also a *security manager service,* that define a default security policy

[9] Actually a class can implement several RMI interfaces when it is needed for providing distinct sets of methods remotely accessible.
[10] This lookup is achieved in the RMI registry that is managed (creation, update, destruction) on the server side. Server objects are referred in this registry by a name associated to the concrete reference that can be used for RMI transport.

Java RMI is particularly suitable for client/server communication when both client and server are programmed as Java applications. The main advantage of using Java RMI is that it enable to work in Java environments in client and server side. Moreover the communication between the two sides is realized transparently since once the references of remote objects are known, local or remote calls are coded exactly in the same way. There are no restriction on the type of objects that can be transferred, and there is no special coding before sending or after receiving a remote method invocation. The Java packages defining Java RMI provide the basic functionality for remote communication and can be easily extended to more specific policies.

In heterogeneous environments, the fact to have to use Java on both client and server sides can be a disadvantage (for example when parts of a system are already existing in another programming language). Moreover using Java RMI in web browser embedded applications is not possible in all web browsers. It requires to install some plug-in onto the web browser in order to obtain the compatibility. Finally Java RMI communication can be refused by certain firewalls.

II.6.1.4) CORBA [Omg], [GGM97], [ACW98]

The Common Object Request Broker Architecture (CORBA) has been defined by the Object Management Group (OMG)[11] in its first version in 1991. Its aim is to define a standard answer to the need for interoperability among the rapidly proliferating number of hardware and software products available today. CORBA allows heterogeneous applications to communicate through a network within a specific implementation of an Object Request Broker (ORB).

Using the ORB middleware, an object of a client application can invoke transparently a method of an object of the server application, even when these two applications are running on different machines. In fact the ORB intercepts all the method calls and is in charge of finding the appropriate server object, passing to it the corresponding parameters and starting the appropriate method to return the result to the client.

As in Java RMI, methods that can be invoked remotely are described in interfaces. For CORBA these interfaces are defined in the Interface Declaration Language (IDL). This language, specified by the OMG, is independent from programming languages but offers a vocabulary common to the different existing programming languages. The interfaces described in IDL can then be mapped to the programming language used in the application. These interfaces are compiled and a stub (client side) and a skeleton (server side) are created. These stub and skeleton will be used when a remote method is called in order to transfer this call to the appropriate server object through the ORB (or through several ORBs linked together).

It is then possible to program distributed applications where different parts are developed in heterogeneous environments, even with different programming languages. The client object calling a remote method does not have to be aware of where the server object is located, its programming language, its operating system, or any other system aspects that are not part of an object's interface. Through the mapping of IDL with the most popular programming languages (such as Java, C++, C, Smalltalk...), and through the management of all the aspects of the distribution by the ORB, it is then possible to program remote clients and server in a totally transparent way. CORBA is then particularly suitable when programming client/server applications in heterogeneous environments and different programming languages.

The Figure 17 presents the general CORBA architecture used for communication between distributed applications.

[11] The Object Management Group (OMG) was founded as a consortium in April 1989 to promote the adoption of industrial standards for managing distributed objects. The OMG groups several important companies such as 3Com, Canon, Sun Microsystems, Unisys, Hewlett-Packard...

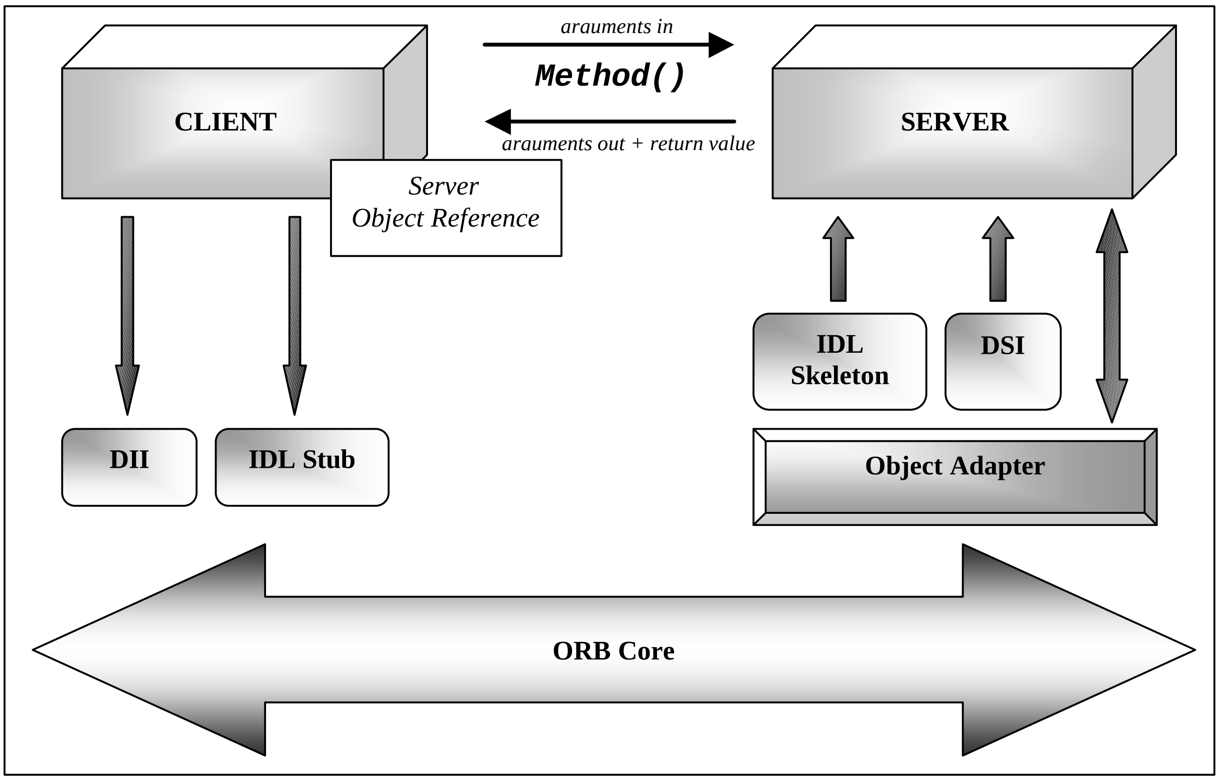


Figure 17 - CORBA ORB architecture

The Dynamic Invocation Interface (DII) provides an alternative to the classical requests through the IDL stubs. It enables a client to dynamically addresses requests without requiring IDL interface-specific stubs to be linked in.

The Dynamic Skeleton Interface (DSI) is the server side's analogue to the client side's DII. The DSI allows an ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of the object it is implementing.

The Object Adapter assists the ORB with delivering requests to the object and with activating the object.

The main advantages of using CORBA for client/server programming are the fact that it provides robust standard industrial technologies for distributed applications which are divided into modules potentially in different environments and developed in different programming languages. The remote access is then assured in a transparent way for the user, thanks to different technologies and services (such as the *Naming Service*, to find the reference of an object from its name, the *Trading Service*, to find the objects based on their properties) of the ORB.

The main limitation for using CORBA is that its complex deployment. Using CORBA requires the installation and configuration of a middleware (the ORB), and involves some specific configurations before using the remote invocations transparently. In the case of clients and server applications using the same language, it seems better to use another technology that can be deployed and programmed more easily. As the Java RMI technology, using CORBA for communication between distributed applications can be limited by certain firewalls.

II.6.1.5) Others

In this section we will just give a brief overview of other existing ways of realizing client/server communication. These technologies do not provide the adequate support for applications such as RECINTERNET. For further information on these technologies, consult the given references.

- ***COM/DCOM*** [Dcom]

The Distributed Component Object Model (DCOM) is a Microsoft protocol that enables components of distributed applications to communicate over a network. DCOM uses the Microsoft Component Object Model (COM) and can be used over different network transport protocol such as HTTP to achieve communication between active components (such as Java applets and ActiveX) embedded in applications. The crucial restriction of this technology is that it can be used only for Windows environment.

- ***RPC*** [Tan89]

Remote Procedure Call (RPC) systems provide a way to realize remote calls in procedural languages. Using client and server *stub* as proxy, a remote procedure can be called across a network. All the management of the communication through sockets and TCP/IP protocols are hidden to the user. As this kind of systems is designed for procedural languages (such as C, FORTRAN, Ada), it seems not interesting in object-oriented applications. RPC can be seen as the ancestor of object oriented equivalent such as Java RMI or CORBA.

II.6.1.6) Choice of the client/server communication technology for RECINTERNET

In the RECINTERNET case, an elaborated technology for client/server communication must be chosen since the RECINTERNET interface must provide important interaction possibilities to the user. It seems also that the best way to implement the client and the server side is an application, or an embedded application into HTML documents (as explained in Section II.5.1 and Section II.5.2). As a result the chosen technology for client/server communication must be adapted to this kind of solutions.

The HTTP protocol is limited to basic communication mechanisms, only supporting basic interaction of a limited set of components used in HTML programming. That is the reason why it seems laborious to use it in the RECINTERNET case. It would mean coding all the client side in HTML, with only a few set of components allowed in HTML documents, and also to program the server side only with CGI scripts or Servlets. Adopting HTTP seems then too constraining for the client/server communication.

Using a socket-based system for this communication seems also not adequate. Sockets programming involves a “low-level” programming where remote call needs decomposition in different stages such as treatment before and after the communication in order to able to transfer messages in packets or into streams. As there are some more elaborated solutions that add a layer to this socket mechanism and provide transparent remote call, the socket way was not chosen for RECINTERNET.

Java RMI and CORBA seem to provide an appropriate solution since they enable to program remote method call inside applications in a transparent way. Both of these technologies provide a flexible and robust solution to distributed application. But Java RMI is particularly easily usable and deployable when both client and server are developed in Java. On the contrary the CORBA system is more dedicated to cases where programming languages used in client and server are different, because it requires the deployment of a set of CORBA components that seems a bit exaggerated when some features of the shared programming language can be used more simply. That is the reason why we chose Java RMI to realize the client/server communication of RECINTERNET.

II.6.2) Server/Database communication

This section will be focused on three-tiers applications were the third tier is a database. We will then present the best existing solutions for accessing a relational database through SQL (Structured Query Language) queries expressed at programming language level.

However, as we have seen in Section II.3, a good alternative for three tiers architectures (specifically in the case of RECINTERNET) is to use two databases. One will be located on the same host than the server, and will be a structural copy of the main database. The main database contains all the data entries and can be located on another machine.

In the case of RECINTERNET, the communication protocol between the server and the main database (SINTESE database) is particular and specific. That is the reason why we will separate this analysis about server/database communication into two parts: one for the access to a "normal" database (using SQL queries), that can be applied to the server/local structural database communication in RECINTERNET, and one to describe the particular communication between the server and the SINTESE database in RECINTERNET.

II.6.2.1) Server/ "normal" database communication

The feature depicted in that section is the integration of the server program with the DataBase Management System (DBMS) of the local structural database. It means describing the technologies available for making the link between the programming language used in the server and the SQL system that concretely access the entries of the database.

Given the high number of different relational database products (to only name few of the main vendors[12], there are Oracle, IBM, Sybase, SAS, Borland, etc...), standards had to be adopted in order to provide some common programming-level interface for communicating with databases. Drivers specific to each kind of relational databases have to be developed in order to support the connectivity mechanism described in these common interface standards. The two most popular standards are ODBC, the Microsoft's Open Database Connectivity standard, and JDBC, the Java DataBase Connectivity standard.

In this context, "connectivity vendors" take really care of providing for each kind of relational database the appropriate drivers in order to be compliant with the main standards.

- ***ODBC*** [Odbc]

The Microsoft's Open DataBase Connectivity standard is an API (Application Programming Interface) widely accepted for database access. It as been adopted as an industrial standard to provide a common interface for connectivity with relational databases at programming language level. ODBC can be used for almost any PC database and with different programming languages (such as C, C++, Visual Basic, Smalltalk, Java).

ODBC is based on the specifications for databases APIs (called CLI: Call-Level Interface) defined by ISO and X/Open. ODBC uses SQL for expressing relational queries for database access. It then provides a way to simply call SQL queries on a database inside programs written in most of the programming languages.

ODBC link with DataBase Management Systems (DBMS) is achieved through drivers specific to each kind of databases and to the platform running the application using ODBC. These drivers are installed on the machine running the application and provides the underlying layers for processing the SQL requests express through the ODBC interface. Since ODBC is the most used connectivity interface standard, all the relational database products vendors have developed their specific drivers for ODBC, and generally they can be freely downloaded from the internet.

[12] The main vendors are enumerated in: *Integrating Databases with Java via JDBC.* Rawn Shah. Javaworl 1996 http://www.javaworld.com/javaworld/jw-05-1996/jw-05-shah.html

Moreover many programming environments (such as JBuilder for Java, Borland C++) have an adapted support for ODBC since they provide some ODBC components that group all the ODBC functionality in simple units responsible of the different aspects of the database access (such as database connection, database tables, statement for supporting the SQL query, etc...).

The main advantage of ODBC is that as it has been adopted as an industrial standard widely used, all the database products vendors provide the different supports to be compliant with it. Unfortunately, the main disadvantage of ODBC is that it involve deployment restrictions since the drivers used with it are platform dependent.

- ***JDBC*** [Jdk]

Java DataBase Connectivity (JDBC) standard is as ODBC an API for providing a common interface at programming level for database access. Its specificity is to be usable only with the Java language. Given the Java "fever" in application development, the JDBC standard is becoming more and more widely accepted by the industrial community. It is also "the" standard for all Java applications using database access, and it would be particularly stupid to not adopt it in a Java application.

JDBC is similar in concept to ODBC since it is also based on the X/Open Call-Level Interface (CLI) specifications for Structured Query Language (SQL). It also provide a simple way to include database access in Java code by a specific support at language level for SQL queries. The *java.sql* package provides the necessary classes for the basic functionalities required for database access.

As with ODBC, the link between the language interface provided by JDBC and the DataBase Management System (DBMS) is supported by drivers specific to each kind of relational databases. A main distinction between ODBC and JDBC is that these drivers are not platform dependent. They are independent from the chosen platform and moreover are loaded dynamically at application runtime. In fact drivers for JDBC are represented as Java classes that are dynamically loaded by the standard *DriverManager* class of Java at runtime. The fact to have JDBC drivers written all in Java involve true platform independence and easy deployment. Moreover the Java Deployment Kit (JDK) provides a set of common JDBC drivers for the most used kind of databases that can be chosen and used dynamically.

An important feature of JDBC is that JDK proposes some bridges between JDBC and ODBC, in order to be able to use ODBC drivers with Java.

The basic scenario used in Java programs to read from a database is the following:

- The driver corresponding to the used relational database is loaded dynamically as a Java *diver class*
- A *connection* object is created to represent the connection to the database (defined through a *database URL* address, that can be local or remote)
- A *statement* object is created. This statement will use a *SQL String* to execute the SQL query
- The result of the processing of the SQL query in the database is loaded into a *result set* object
- This *result set* object can be exploited as wanted with different methods *(next, getString...)*
- The *result set, statement* and *connection* objects are closed

There are also different functionalities for modifying the database.

Finally the JDBC standard is very useful and simple to use within Java programs, mainly because a set of Java classes groups all the required functionalities. The main advantage of using JDBC is the fact to be able to load dynamically platform independent drivers, and mainly that JDBC is "THE" reference for accessing databases with Java.

▪ ***RECINTERNET case***

The strategy for the RECINTERNET case was to choose a standard API for the connectivity with the local structural database, so that it would be easy and simple to include database access code in the server application code.

The major parameter in this choice was the fact that the server side of RECINTERNET was developed in Java. Then using the JDBC standard was the logical choice for realizing the connection to this database. JDBC was then a good choice because it was possible to use the existing Java classes to insert easily in the code databases access. Moreover it was possible to insert Java variables directly in the SQL strings used for database queries.

The local structural database developed and used for the RECSINWIN project was reused for the RECINTERNET project. This database is a *Interbase* database of approximately 22 MBytes, working with the DataBase Management System (DBMS) *InterServer*. All-Java drivers for JDBC/Interbase are available for free[13], and provided all the needed functionalities for the database connection.

The choice of JDBC for the RECINTERNET project simplified also the database access programming thanks to many reusable examples and to the good documentation provided by Oracle[14] freely.

II.6.2.2) The particular case of server/SINTESE database communication

It would have been particularly interesting to also use a standard like JDBC to realize the communication between the RECINTERNET server and the main database containing all the entries (the SINTESE database). Unfortunately this was not possible because the SINTESE database is inserted into the SINTESE system that uses a particular DataBase Management System (DBMS) that is not based on SQL queries.

As explained before it was not possible to modify the SINTESE system and so we were obliged to adapt the RECINTERNET project to this system. We will now describe briefly the specificity of the communication with this SINTESE database, and the way we used it in the RECINTERNET project.

The SINTESE database is a DMSII database of approximately 20 GBytes, located in the CTRJ (Centro de Tratamento de Informações do Rio de Janeiro – *Information Treatment Center of Rio de Janeiro),* on a high-powered storing computer. This computer is part on the intranet where are connected the different machines of DATAPREV headquarters and other centers.

To access this database, a user must first connect to the SINTESE system, through a TCP/IP protocol (LCW gateway). He can then use the functionality of the system through a terminal. In order to retrieve data form the SINTESE database, he must use a particular syntax which is very different from the SQL syntax since it was specifically defined for the access to this database. The way to retrieve some entries of the database is to combine some command lines describing the way to organize the lines and columns of the table showing the resulting serie. The Figure 18 illustrates this specific syntax:

```
- LI ESP ESTADO ( * REGIAO SE )
- M $SUB.CONTRI ANO 97 98 99 POSTO FA3056
- M $DOLAR.C MES 0199 0299 0399 0499 0599 0699
```

Figure 18 - Example of the SINTESE request syntax

These three command lines represent a request to the SINTESE database. In the first line of this example is specified that the lines of the results sheet will be all the states of the region *Sudeste.* The second line specifies that in the columns will be the serie referenced as *Sub.Contri* for the years 1997, 1998 and 1999, for the *Posto*

[13] Drivers downloadable on the Interbase Site: http://www.interbase.com
[14] JDBC documentation and examples: http://www.oracle.com/java/codesamples/jdbc/index.html

referenced as *FA3056*. The third line specifies that in other columns of the same sheet will be the serie referenced as *Dolar.C* for the months 01/1999 to 06/1999.

Once he has submitted its command lines combination, the user can visualize the result table on his terminal screen. Thanks to a special SINTESE command, the user can retrieve the result as an ASCII text file.

In order to be able to access the SINTESE database with the RECINTERNET server, we needed some specific measures. First of all the connection to the remote SINTESE system will be done thanks to a specific Java API that simulates the terminal connection using the TCP/IP protocol.

Then we needed to use the specific syntax to express queries to the SINTESE database. In fact in the RECINTERNET system, the user compose dynamically a final request through a navigation and accesses to the server and the local structural database. This final request is structurally described by the different navigational choices done by the user. The different parameters of this request are chosen by the user in the different navigation screens. Finally there are only few different structural request-types, that need to be filled in with the specific choices of the user. So we just needed to express the structure of these request-types in the SINTESE request language, and then complete them with the SINTESE code of the different parameters.

Finally when the user has composed his final request, the RECINTERNET server passes the corresponding command lines structure completed with the user parameters to the SINTESE simulated terminal, calls the special command to receive the ASCII results file, and then parses it to create an exploitable Java object that will be sent to the client.

By this mechanism we can then use the specific SINTESE system to access the SINTESE database from the RECINTERNET server.

II.7) Conclusions

To conclude this analysis of the main technologies and techniques usable in the design and implementation of the RECINTERNET project, we will sum up the different choices we did for RECINTERNET.

We will use some concepts and patterns of the Object Oriented Hypermedia Design Method (OOHDM) since it enables to adopt robust and proven solutions to recurrent navigational applications design problems. The design decomposition in *conceptual design, navigational design* and *interface design* seems particularly suitable to address in different steps the different concerns of the application design. Using the set of *hypermedia design patterns* of OOHDM will also be very benefic since it will enable to organize the different mechanisms used in RECINTERNET in coherent and efficient ways.

Our study of the architectural possibilities for an application like RECINTERNET highlighted that three-tiers architectures are particularly suitable for web-based applications involving shared resources such as databases. In the case of RECINTERNET we will use a variation of three-tiers architectures. A server local structural database will be used to increase requests preparation efficiency since only final requests will be addressed to the main SINTESE database.

Then we chose different technologies for the three-tiers of this dynamic internet database interface and the communication between the three-tiers. The client part of RECINTERNET will be programmed in a Java applet, embedded in a HTML page downloadable with any web-browser. This is the best solution we found to provide an easy way for users to reach an efficient application from any web-browser. The server part of RECINTERNET will be programmed as a Java application running on the server host. It seems the appropriate solution to implement the different functionalities of RECINTERNET and to support simple communication protocol with the client and simple databases access. Client/server communication will then be realized with Java RMI since it provides a transparent way to realize method calls between remote objects of different Java applications. Structural database access from the server application will be achieved through the JDBC standard and the corresponding drivers, that enable to construct SQL queries and exploit queries results into Java code. The access to the SINTESE database will be done through the simulation of a terminal of the SINTESE DataBase Management System (DBMS) with a remote connection by TCP/IP protocol. The constraint of using this existing database and its associated DBMS is then included into the RECINTERNET project.

The Figure 19.presents a summary of the different technologies used in the RECINTERNET project for the three tiers and for the communication between them.

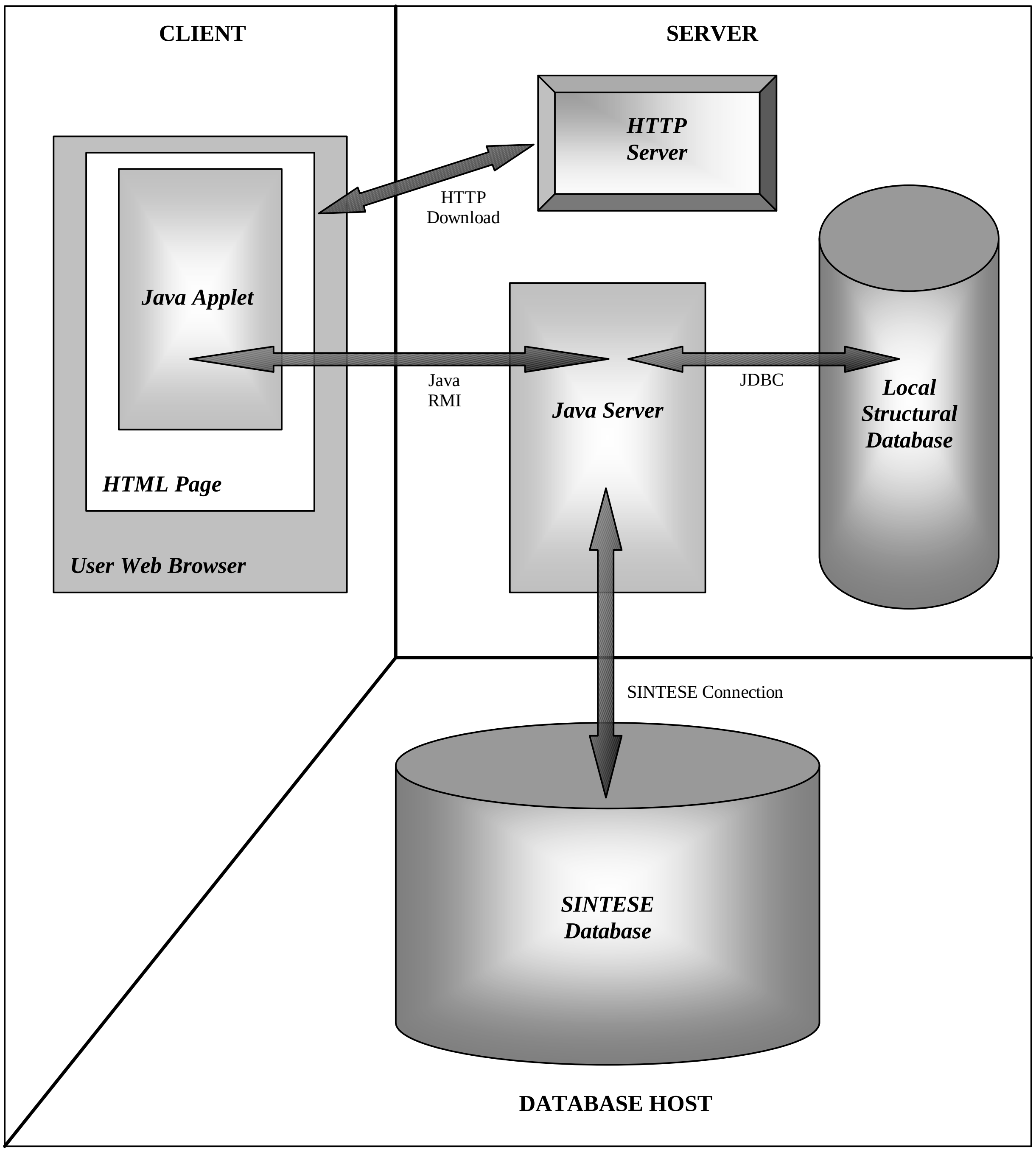


Figure 19 **-** *Summary of the chosen technologies for RECINTERNET*

Finally we will also program the whole system with separation of concerns. It seems particularly suitable to enable an efficient and flexible programming for different concerns used in RECINTERNET such as connection control, user interaction, synchronization... Out of the different techniques existing to support separation of concerns, Aspect-Oriented Programming (AOP) provides the appropriate mechanisms and technology to efficiently express complex concerns cross-cutting the code into separated modules. By using the AspectJ tool it is possible to implement aspects that will be automatically woven with Java components code.

The following sections of this thesis will present the way we developed the RECINTERNET project with the different techniques and technologies. Our research is particularly focused on one of these techniques: AOP. We will present in Section III our approach for designing with aspects for web-based three-tiers applications. We will illustrate this way of programming in a comparison between a conventional object-oriented design and an aspect-oriented one for the RECINTERNET case in Section III, and present the way to implement these two solutions in Section IV.

III - Applying Separation of Concerns in Web-based Three-tiers Applications Design

III.1) Introduction

In this introduction we will go further in the descriptions of separation of concerns and Aspect Oriented Programming (AOP) given in the Section II.2. We will also state the interest of using these techniques for web-based three-tiers applications and present the objectives of their application at the design level of an application such as RECINTERNET.

III.1.1) Separation of concerns

As we have seen it in Section II.2.1, the separation of concerns paradigm aims to define programs in terms of components and concerns. Components are entities that can be encapsulated in traditional object oriented structures to express the basic functionality of a program, and concerns are more complex entities that cannot be encapsulated in components structures because they are cross-cutting the basic functionality code. The idea of separation of concerns is to have separated modules for each new concern.

Now we explain in details the composition/decomposition mechanisms involved by this way of programming:

- ***Decomposition***

The aim is to decompose a system into different concerns and components. The decomposition in components is done in the classical way of dividing a system into objects. Then concerns are defined for cross-cutting entities, for example for concurrency control, remote communication strategy, real-time constraints, security. Concretely the idea is to define in separate modules all the modifications that will be done to the components code for one concern.

Each different technique for realizing separation of concerns have its own mechanism for identifying and expressing concerns, but all of them try to provide a way to express a set of modifications to the basic functionality code into separated modules. Each concern is closely related to the design and eventually to the implementation of the components of the system. The way to express a concern is also specific to the technique chosen. It can be specific aspects languages (in aspect oriented programming), meta-level code (in meta-programming), behavioral specifications (in adaptive programming), or special entities such as filters classes (in composition filters), or subjects (in subject oriented programming). As this is a quite new and active research domain, there is not definitive and precise methodology to realize decomposition.

- ***Composition***

The different concerns and components resulting from the decomposition are expressed in different ways and cannot be compiled or ran like that. That is the reason why the different techniques for separation of concerns try to define a composition mechanism in order to obtain a compilable or executable system out of the concerns and components as they are expressed.

These composition mechanisms are also specific to each technique and can work either at source code level, either at compiled code level, or at both levels. In aspect oriented programming composition is realized by an *aspect weaver* taking code as input and producing code as output. In subject oriented programming, automatic composition of code or bytecode of some *subjects* are realized with *composition rules*. Meta-programming way is to compose concerns and components by modifying *meta object protocols*. Composition filters use a modified way to evaluate message passing in order to add the concerns functionality. Adaptive programming generates code in classes involved in *traversal strategies.* The idea is too realize automatically this composition for the user.

By these mechanisms the separation of concerns paradigm provides important improvements to the classical object oriented software development. For many kinds of applications it provides ways to reduce program complexity and to ameliorate flexibility, reusability, maintainability and evolutivity.

III.1.2) Aspect Oriented Programming

Aspect Oriented Programming (AOP) is the base of the work presented in Section III and IV about applying separation of concerns in web-based three-tiers applications design and implementation. That is the reason why we extend now the definition given in Section II.2.2.5 with further details.

III.1.2.1) Concepts

There are "AOP-specific" terms that are independent from the concrete solution chosen to support AOP. We will define these terms in this part. A system and its implementation can be decomposed into components and aspects.

- ***Component***

"A component can be cleanly **encapsulated in a generalized procedure** (i.e. object, method, procedure, API). By cleanly, we mean well-localised, and easily accessed and composed as necessary. Components tend to be units of a system's functional decomposition, such as image filters, bank accounts, service providers, GUI widget." [KLM+97]

- ***Aspect***

"An aspect cannot be cleanly encapsulated in a generalised procedure. Aspects tend not to be units of the system's functional decomposition, but rather to be **properties** that affect the performance or semantics of the components in systemic ways. Example of aspects include memory access patterns and synchronisation of concurrent objects." [KLM+97]

- ***Decomposition***

Decomposition in AOP is the definition of a system in terms of components and aspects. Aspects must be cleanly separated from components, and aspects must be separated from each other. At design level the choice is done between what will be expressed in components, and which aspects will be used. Aspects are then expressed in a specific language (or in distinct specific languages[15]) that enable to define efficiently concerns in relation with involved components already defined in the classical programming language used[16]. Aspects are always based on components implementation.

- ***Granularity***

An important characteristic in AOP is the granularity of the aspects. It means on what level of granularity do the aspects affect the components code. In object-oriented languages, aspects can reach methods and variables of a class. It means that generally, aspects are allowed to add new methods and variables to a class, but also to wrap an existing method of a class with further code, that will be executed before or after the normal body of the method. Anyway this generally assumed granularity level can be different depending on the aspects languages used.

[15] For example, in the *D Framework* [LK97], two aspects languages are used: *Ridl,* to express remote access strategies and *Cool*, to express threads coordination.

[16] Even if the main research work on AOP is applied to object-oriented languages, AOP principles are also valid for other paradigms, as specified in [KLM+97]. For example aspects can be defined on procedural systems, or aspects can even be used to express entities cross-cutting aspects.

- ***Join point***

Join points are used to identify the different locations of the components code that aspects will affect. Expressing aspects is dealing with two issues: expressing modifications that will be applied to components and specifying where these modifications will take place. More details about this decoupling can be found in [Beu99]. The last of these two feature is in fact expressing join points. Aspects languages provide specific semantics for defining join points, and some join points abstractions to express for example "all the methods of a class"[17].

- ***Weaving***

The aim of AOP is to provide an automatic mechanism of composition. Aspects and components will be composed to create a usable system (by usable, we mean a system that can be compiled or directly run). It means that the aspects will be taken one by one and applied to the components code. This mechanism is assured by an *aspect weaver,* which is a kind of pre-processor that takes as input components and aspects expressed in given languages to produce as output a woven code. This woven code represents the components plus the different aspects modifications added.

III.1.2.2) AspectJ

AspectJ [AJ] is an general-purpose aspect-oriented extension to Java developed by the Xerox Parc Corporation in Palo Alto. AspectJ provides an aspect language that enables to express aspects applicable to classical Java code. As explained in Section II.2, AspectJ is the most advanced solution for efficiently programming with aspects. Moreover it works with Java, that we were using for the RECINTERNET project, and is easily usable. We worked then with the version *0.3.0 beta 3 release,* that can be downloaded freely from the AspectJ Home Page [AJ].

More details will be presented about concrete implementation with AspectJ in Section IV, but it is interesting to present in this section few characteristics of AspectJ that will influence the way to design with aspects.

The Figure 20 presents the global mechanism of the AspectJ weaver. Components Java code and aspects code (expressed according to the AspectJ semantic) are combined statically by the AspectJ weaver (that can be seen as a pre-processor). The output is some woven Java code that can be compiled as classical Java code to produce executable Java bytecode[18]. It has to be noticed that the input aspects and components files are not modified by the weaver. The output is created in distinct files.

The interest of such a mechanism is the flexibility obtained for plugging in or out aspects. Plugging in a new aspect is done by weaving it with the components code. Plugging out an aspect is done simply by taking the initial component code that has been left unmodified by the previous weaving.

Without entering in the details of the AspectJ weaver, we can describe the weaving mechanism as following: the weaver takes the aspect files one by one, then analyses the aspect to solve the join points considering the components code. Taking the solved join points one by one, it then writes the corresponding modifications in the different output files with some comments to describe the realized modifications. Weaving errors can stop the weaving process (such as incorrect join points), and they are then reported by the aspect weaver.

[17] With AspectJ [AJ], it is possible for example to express "all the public methods of the Java class `Point`" with this syntax: `public * Point.*(..)` (for more details about the syntax, we refer to [AJPrimer]).

[18] Actually it is possible to directly obtain the Java bytecode corresponding to the Java woven code compiled. An option of the AspectJ weaver enables to weave the aspects and components and directly call the classical Java compiler to provide to the user the final woven bytecode.

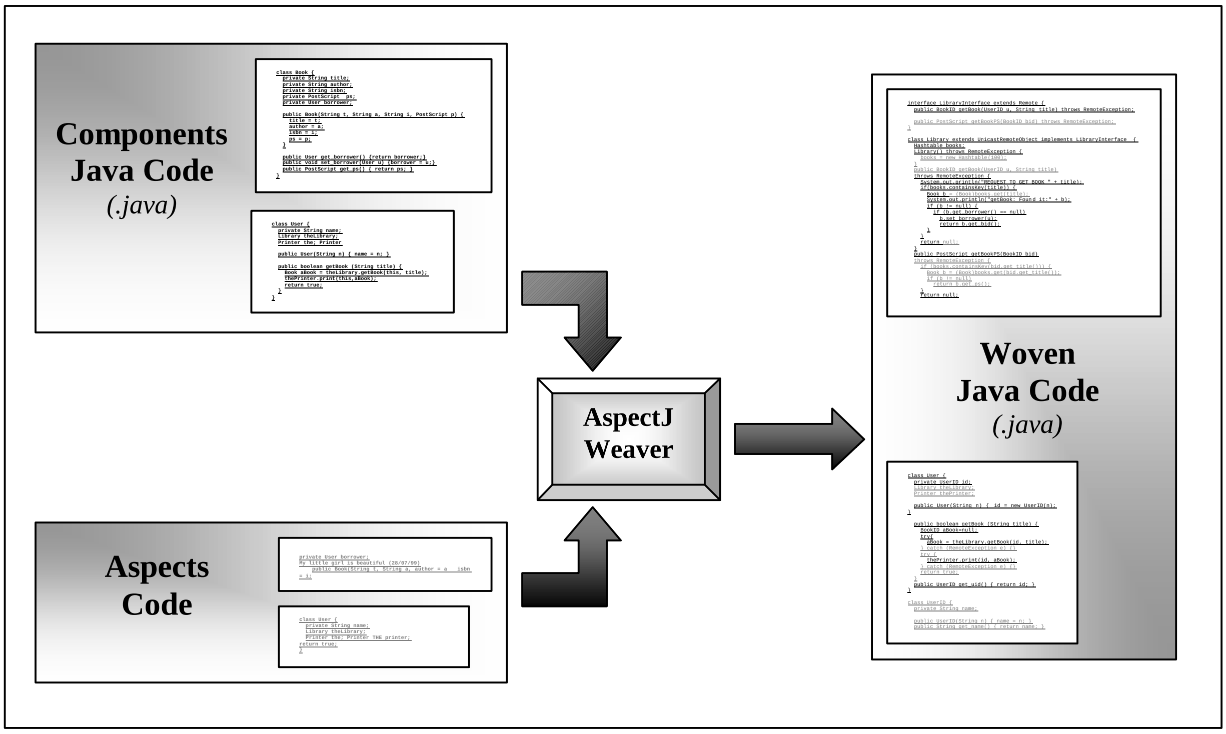


Figure 20 - AspectJ weaving mechanism

Finally in AspectJ we can see the expression of aspects as directives for the weaver to insert pieces of code into components code. The following section will give more details about the way to express these directives and what can be reached with these directives.

III.1.2.3) Expressing aspects with AspectJ

This section is mainly based on the explanations given by the AspectJ team in [AJPrimer]. In this section we will speak of aspects in the specific context of AspectJ. It is important to clearly understand what can be expressed with aspects, in order to be able to design aspects effectively realizable with AspectJ.

The idea of AspectJ is that aspect declarations are similar to Java class declarations:

```
aspect MyAspect {
   ...
}
```

The different members declared in an aspect can be variables, constructors and methods (as in Java class declarations), but also *introduce weaves* and *advice weaves. Weaves* designate where code will be inserted (through *designators)* and what Java code to insert. We will now define these different terms involved in aspect declarations:

- ***Designators***

Designators are used to define the aspects join points. They point to one or more methods or constructors of one or more class. For example:

```
public void Point.setX(int x)
```

points to the method `setX` method of the class `Point` that takes an integer as parameter and has a `void` return. The idea is the same for constructors. It is also possible to use packages in class reference.

It is possible to use specific characters. The “`*`” is used to say that anything can replace it. It can be used for package name, class name, return type, constructor name or method name. The “`..`” is used to specify that the parameters of a method (or a constructor) can be anything. The “`!`” character enables to express some conditions on the modifiers. For example:

```
public !abstract void MyPackage.*.set(..)
```

refers to all the public `set` methods (with any kind of parameters) of all the classes of the package `MyPackage` which are not abstract.

With this syntax it is then possible to refer to multiple methods or constructors in the same time.

- ***Introduce weaves***

The *introduce weaves* are used to insert in one or more classes variables, methods or constructors. The place where they will be inserted is defined by one or more designators. The body (or initializer) of the *introduce weave* will be the body (or initializer) of the methods or constructors (or variables) introduced. For example:

```
introduce public int Point.getX() { return x; }
```

will introduce the method `getX` with the appropriate body in the class `Point`.

And, for the variable *introduce weave*, for example:

```
introduce private Color Point.color = new Color(0,0,0);
```

will introduce the variable `color` and its initializer to the class `Point`.

It is also possible to define a weave that introduce several methods:

```
introduce public String Point.getName()
                  public static String Line.getName()
                  protected String Square.getName() {
     return name;
  }
```

- ***Advise weaves***

Advise weaves are used to insert code into methods or constructors. As *introduce weaves* they use designators to point to the wanted methods or constructors.

The *before advise weave* will insert some code before the body of the initial method. For example:

```
advise public void Point.setX(int _x) {
          static before { if (!assertX(_x)) return; }
   }
```

means that the given body will be executed just before the `setX` body, each time the `setX` method will be executed.

The *after advise weave* is the same but will be added at the end of a method body.

The *catch advise weave* is used to add a “try-catch” wrapping the entire body of a method. For example:

```
advise public void Point.setX(int _x) {
     static catch (Exception e) { System.out.println(e); throw e; }
  }
```

will add the given behavior to execute when exception are raised in the `setX` method of the class `Point`.

The *finally advise weave* is used to add a “finally” wrapping the entire body of a method body. For example:

```
advise public void Point.setX(int _x) {
     static finally { releaseResources(); }
  }
```

will add the given behavior to execute at the end of the `setX` method of the class `Point`, even if exceptions are raised.

As for introduce weaves, an single *advise weave* body can be expressed for several *designators.*

- ***Aspects variables, constructors and methods***

As for any Java class, an aspect declaration can contain some variables, constructors and methods. The syntax is the one used in Java classes. These aspect variables, constructors and methods can be called or used from within this aspect. For example:

```
aspect MyAspect {
     private static int countPointInstances = 0;  // The aspect variable

     advise Point(..) {
           static after { countPointInstances++; }
     }
}
```

will increment the `countPointInstances` variable of the aspect `MyAspect` after each time a constructor of the class `Point` is executed.

- ***Specific variables***

Specific variables can be used in aspects. They are used to hold references solved at run-time:

- `thisObject:` this variable holds the reference to the current object (equivalent to the “`this`” in Java).
- `thisJoinPoint.methodName`: holds the name of the current method on which the current advise weave is being executed.
- `thisJoinPoint.className`: holds the name of the current class containing the method on which the current advise weave is being executed.

- `thisResult`: holds the return value of the method on which the current advise weave is being executed, if any.

- ***Remarks***

In the latest versions of AspectJ[19], some new features related to non-static weaves have been added. It is possible to instanciate aspects in the same way than classes instantiation (by calling an aspect constructor). Aspect instances can then be associated with objects (being referenced by an instance variable of the object) and objects to aspects (through an aspect *domain* that holds references to associated objects).

To support these features, AspectJ provides some new specific variables and methods, in order to be able to "manipulate" these associations. It is then possible to have references to aspects from objects and automatic references to objects from aspects (through their domain). These new features enable some "non-static" exploitation of aspects and components weaving. We will not provide further information about these features since we decided to not use them for our design with aspects. In fact they are quite recent in the AspectJ development and should evolve consequently in the near future. Moreover most of these new features can be obtained with the "classical" AspectJ features.

Another point is the particular use of Java interfaces. Normally a Java interface only defines methods signatures that must be defined in classes implementing this interface. In AspectJ, a join point referencing a method of an interface will in fact reference all the implementations of this method in the classes implementing this interface. This enable to factor out some modifications on methods of several classes.

For further details about this aspect language, we refer to [AJPrimer].

Now that we have explained more in details AOP and AspectJ, it is possible to understand what can exactly be expressed in aspects, and the way it will affect the components code. Finally AspectJ provide an appropriate tool for programming with aspects as well as a powerful language that provides a wide range of possible actions on components code.

III.1.3) Objectives

In the following sections we will present the different stages of our research process. The principal idea was to apply separation of concerns (and particularly aspect-oriented programming) in web-based three-tiers applications development.

Aspect oriented programming is an emerging way of programming that provides many improvements in application development. Unfortunately as a new technique, a clear methodology to be efficient and to take all the benefits of it is missing.

Moreover aspect oriented programming, and particularly aspect oriented programming with AspectJ, is a technique mainly based on implementation stage. Aspects are defined relying on basic components implementation. However inserting aspects at design level is a motivating challenge since it provides important advantages such as composition conflicts solving or better reuse to name only few.

That is the reason why we will present in the following sections how we did to design a web-based three-tiers application with aspect. We will compare this approach with a classical approach (without aspects). This comparison will take place in the concrete case of the RECINTERNET project.

We will present in Section III.2 what are the motivations, the difficulties and a technique to realize aspect oriented design in web-based three-tiers applications. The idea is then to draw a comparison between a design without and with aspects for this kind of applications, and particularly in the concrete case of the RECINTERNET applications. For this purpose we will present in Section III.3 the design without aspects that we realized for RECINTERNET, and in Section III.4 a design with aspects for the same application. The Section

[19] Previous to the version *0.3.0 beta 3 release.*

III.5 will then draw a comparison and these two approaches in this concrete case and we will give the conclusions of this research for the design level in Section III.6.

III.2) Designing web-based three-tiers application with aspects

Web-based three-tiers applications design is a particularly complex stage of their development cycle since this kind of systems involve distinct machines with different technologies for each part of the system. Complex concerns are then scattered throughout the code of the different components of the system. An aspect oriented design of such systems is then particularly interesting.

III.2.1) Motivations for designing web-based three-tiers applications with aspects

III.2.1.1) Aspect-oriented applications (AOP)

Using AOP in any kind of applications development provides several advantages. Most of them are highly related, but we can separate the main advantages of using AOP as following:

- ***Modularity***

With AOP it is possible to express aspects in separated modules from components code. Moreover it is possible to express different aspects in separated modules. Then an important advantage is that an application will be programmed in distinct modules, each of them corresponding to a clearly defined concern. This modularity is a important advantage during design stage, where it is possible to concentrate on one concern at a time, but also at implementation stage, where it is easier to implement concerns one by one than all together in a same module.

- ***Size***

As we have seen with AspectJ, it is possible to express very succinctly in one aspect some modifications affecting several components code locations in the same time. For this reason, code duplication is avoided, and it is possible to express in few lines of an aspect some modifications corresponding to a high number of components code lines. For this reason using aspects in applications development generally decreases programs size.

- ***Complexity***

The modularity provided by AOP is an important factor of complexity decreasing. Understanding an aspect oriented program is easier than understanding a conventional one. Even if the program is divided into several modules (corresponding to the different aspects and components), as each concern is addressed distinctly from the others, it is easier to better realize the impact of each concern on the code than if this concern was scattered throughout the code and mixed with several other concerns. Moreover aspect oriented programs small size increases their understandability.

- ***Flexibility***

Aspect oriented programs are particularly flexible since, as in AspectJ, it is possible to plug-in or plug-out aspects easily. The aspect weaver support these two mechanisms in such a way that replacing an aspect by another one is a simple operation involving just a static re-weaving and a compilation of the woven code. By this way it is possible to obtain easily distinct programs with modules addressing different issues out of a basic program structure by the way of plugging-in or out aspects. It can then be realized without modifying all the code, even if these different issues involve entities tangling the basic functionality code in many places.

- ***Maintainability***

With AspectJ, the executable Java bytecode is obtained by the compilation of the woven Java code. For this reason, aspect-oriented programs errors are difficult to track since they refer to the woven code and not to the initial aspects and components Java code modules. However the woven Java code is carefully commented by the weaver in order to describe clearly from where is coming the code. Moreover, since separated concerns are addressed in distinct modules, it is possible to maintain separately parts of a system corresponding to distinct concerns. Locating the involved modules is then a simple task.

- ***Reusability***

Reusability issues are particularly well addressed with AOP. Reusing an existing aspect-oriented system can be seen as a matter of plugging-in new aspects and plugging-out non necessary old aspects. Moreover understanding the reused system is a crucial part of reuse, and this is consequently simplified thanks to systems complexity and size decreasing due to AOP. Finally the modular design of aspect oriented systems is an important advantage for their reuse.

III.2.1.2) Aspect-oriented web-based three-tiers applications

Involving multiple tiers and multiple technologies, web-based three-tiers applications deal with several complex concerns that can beneficially be expressed through aspects. This makes the aspect oriented approach particularly interesting for their development cycle. Here is a non-exhaustive enumeration of some particular features of web-based three-tiers applications that are consequently interesting to develop with aspects.

- ***Remote communication***

Web-based applications rely on remote communication between a client and a server. As stated in Section II.6.1, solutions like Java RMI can be adopted for this communication. However programming this remote communication concretely cross-cuts the functionality components code (with *stub* and *skeleton* management code, remote interfaces code, remote exceptions handling and so on). It is then particularly powerful and brief to express this remote communication feature in aspects.

- ***User interaction***

As explained in Section II.5.1, internet provides different possibilities to support applications dedicated to user interaction handling. In the Java applet solution, user interaction is managed through events handling. The web browser react to events such as mouse actions (click, moves...), windows events (close, minimize...) and so on, and these events must be converted into the appropriate method call to realize the needed actions. It can be much simple to implement this user interaction scheme with aspects.

- ***Server functionality***

In web-based three-tiers applications, the middle-tier (the server) can implement different functionalities. The fact to deal with multiple users require some multi threading policies that are involving elements in different parts of the server code, and then can be encapsulated efficiently in aspects. Moreover in a multi-tiers application with multiple servers, the different elements related to the load balancing feature cross-cut the basic components of the server application. For this concern it is also interesting to use AOP.

- ***Database access***

In the same way, the code related to database access (database connection, SQL strings creation, results exploitation...) can involve many entities of the server code, and also several features which are specific to the kind of databases used in the application. Expressing this code in aspects can avoid code tangled throughout components code but also reduce efficiently code complexity by expressing in distinct modules specific concerns of different kind of databases access.

III.2.1.3) Motivations for defining aspects at design level

As explained in [Aop]: *"Like objects, aspects are intended to be used in both design and implementation. During design the concept of aspect facilitates thinking about cross-cutting concerns as well-defined entities. During implementation, aspect-oriented programming languages make it possible to program directly in terms of design aspects, just as object-oriented languages have made it possible to program directly in terms of design objects"*.

Then, even if aspects are based on components implementation, define them at design level can enhance in many ways the development cycle.

First of all, defining aspects at design level will help designers to separate systems in different modules related to distinct concerns. This mechanism will enable to address one concern at a time and to design it almost independently from the other concerns. It means that designing the aspect encapsulating all the elements of a concern will be easier than designing these elements into the entire system without aspects since everything related to this concern will be expressed in only one module, and almost independently from the other concerns.

Another advantage of defining aspects at design level is that it is then possible to define some pluggable aspects. In fact aspects can also be seen as pluggable modules that can be added or removed to a system to support different functionality.

For instance we can imagine a system where some clients access as a remote server. A strategy could be that if the connection is not realized after a given time limit, an error message is displayed to inform the client that the connection is not possible. Another strategy could be that when this time limit is reached, the client request is automatically transferred to another server. These strategies could be expressed each in one aspect. Then choosing a server policy could be done by plugging-in the wanted aspect and eventually plugging out the previous one. These alternatives are chosen before the code is compiled, and it can be very useful. These kind of pluggable functionalities must be defined at design level, since they affect consequently the whole system implementation.

An important decision in developing aspect-oriented systems is choosing what to put in aspects and what not. The boundary is often difficult to define at implementation level. It is then important to have defined during the design of the system which concerns will be addressed in aspects, and then the different elements involved in the given concerns can be extracted from the basic functionality components code to be expressed in aspects. Finally the implementation will have to follow closely what the design expressed and then aspects will be defined more clearly and in a homogeneous way since they respect the aspect decomposition policy defined during design.

An aspect is based on components implementation. It will be woven to components code to add some code into these components, but there can be some composition conflicts. For instance if two aspects define some code addition on the same join point, the result can be different from excepted since the pieces of code added by the two aspects can be incompatible. Expressing clearly the join points of aspects and the modifications involved at design level can enable to detect these kind of composition conflicts.

Finally defining aspects at design level enable to efficiently organize design in separated concerns. It will result in systems where each feature has been clearly and separately defined. Implementing such a design will then be facilitate and composition conflicts can be avoided. Finally a clear structure will be provided for design models. Reusing these design models will then be easier since they will be more structured and more understandable through a clear separation of concerns.

III.2.2) Difficulties of aspect design

III.2.2.1) An emerging programming paradigm

Separation of concern is one of the most active research domain of software engineering. But as a new paradigm, related techniques are quite new. AOP is one of these emerging techniques to realize separation of concern. Thus AOP is in a maturation phase where it evolves a lot and where it is not yet an efficient and generally adopted way of programming. Many improvements and propositions have to be done before an efficient way to design with aspects is defined and accepted as standard.

In this context Xerox Parc Corporation decided to develop AspectJ. As a beta version, AspectJ is evolving permanently and new features are added in frequent new releases. It evolves to provide new features requested by its users in an efficient way and to solve the problems of the previous versions. Important modifications also involve different ways to design a system, in order to be able to implement it with AspectJ.

As an example, the version 0.1.0 of AspectJ was supporting two aspect languages to provide only mechanisms for concurrency control (COOL) and remote communication (RIDL). For more details on these languages, see [LK97]. The version 0.2.0 aims to support general-purpose cross-cutting mechanisms. Consequently using one version instead of another modifies not only the way to implement a system, but also the way to design it.

Moreover many frameworks for AOP are using some specific-purpose aspect languages, and then use a specific methodology for decomposing a system into aspects and components. It is then difficult to speak of a global methodology to design systems with general-purpose aspect languages frameworks.

Currently there is not a standard methodology to design with aspects. This is a domain of software engineering that will require maturation and experiments in order to be able to abstract some efficient way to design with aspects. In Sections III.2.3 and III.4 we explain a way to realize an aspect-oriented design and to represent it for web-based three-tiers applications and we illustrate it with the concrete design of RECINTERNET.

III.2.2.2) Decomposition challenge

The AOP decomposition mechanism – separating a systems into different components and aspects – must be defined at design level for the reasons we explained in Section III.2.1.3. However this is a difficult challenge for which no standard methods are defined.

The first difficulty is in choosing what should be expressed in aspects and what should not. Web-based three-tiers applications have been and can be totally designed and implemented in a conventional object oriented way. In an degenerated view we can also imagine such an application where any entity that cross-cut just a bit the components structure is expressed in an aspect. Then we would have a proliferation of aspects with a very poor components structure. It seems that an intermediate solution between these two extremities should be found in general. The pending question is then: until what point should we put cross-cutting entities into aspects? Obviously the answer depends on the kind of application. We try to answer this question in the case of web-based three-tiers applications in the following parts.

Another difficulty is that aspects are based on components implementation. At design level components are not yet implemented and then it can be a hard task to clearly define aspects at this moment of the development cycle. Join points are a particularly important feature in aspects expression. It is then important to define as clearly as possible these join points between aspects and components, even at design level. We try to explain the way to express these join points in an appropriate way at design level for web-based three-tiers applications in the following parts.

III.2.3) How to design web-based three-tiers applications with aspects

In this section we describe a step by step methodology to support aspect-oriented design in web-based three-tiers applications by proposing some guidelines to structure the aspects/components decomposition, as well as a clear way to define aspects/components interactions. We finally propose a table representation for aspects/components decomposition design.

This methodology and representation will be used and illustrated in details in Section III.4 in the concrete case of RECINTERNET. In fact we propose a way to design based on a step by step decomposition and characterization that we created for the RECINTERNET case.

III.2.3.1) Step by step aspects design

The process we propose to design a system with aspects is divided in different steps organized in four parts. It has to be noticed that it is difficult to strictly follow this method step by step since many steps are closely involved, and sometimes defining a step will modify another step previously defined and then all the intermediate steps will have to be re-done. The basic idea of this methodology is to decompose the designed system by characterizing the different elements of its two dimensions (the aspects dimension and the components dimension), and to clearly define the interactions between these two dimensions.

⊗ **POINT I: ASPECTS IDENTIFICATION**

⊕ ***Step I-1: Aspects Areas Identification***

In this first step are defined the different characteristics of a given system that address some concerns that will surely cross-cut components implementation and that could be beneficially expressed with aspects. These aspects areas are then named and a brief description of the kind of aspects of such areas is defined. In Section III.2.1.2 we have named some of these areas that can be used in web-based three-tiers applications aspect design *(shared resources, user interaction, remote communication, server functionalities, database access...).*

In the case of a web-based three-tiers application, we can pick up for example the following aspect area: *shared resources.* We can describe it briefly the potential aspects of this area as following: *the aspects of this area will express all the code modifications related to methods synchronization or shared variables access synchronization.*

⊕ ***Step I-2: Aspects Identification***

In this step, taking one by one the previously described aspects areas, we will try to point out all the aspects of the area. We will then identify precisely what each of these aspects is supposed to provide. This point is the answer to the question: *what is this aspect concretely concerned with?* This should be explicitly described for each of the identified aspects. This description is not directly concerned with components, but address the principal objectives of an aspect. For example we could have an aspect *Database Access Synchronization* which could be described by: *realize the synchronization of all the threads requiring access to the database.*

⊕ ***Step I-3: Aspect Type***

We propose two characterizations for aspects: *type* and *plugging constraints.*

- *Type:* an aspect is either a *required* or an *optional* one. A *required aspect* will be woven to the components code to produce an executable code. The original components code cannot be run without being woven with such an aspect. An *optional aspect* represents an option that can be plugged or unplugged to the system. The system can work without the plugging of such an aspect. Weaving such an aspect to the system can add a particular feature to the system.

- *Plugging constraints:* this characterization of aspects enables to specify the way aspects or groups of aspects can be plugged. We have identified two kinds of plugging constraints, but it is not an exhaustive list. The first one is that an aspect can be used only when another aspect (or several others aspects) are also plugged on the system. The second one is that only one aspect out of a group of aspects can be plugged on a system

As an example we can have two *optional aspects* of cannot be plugged together*:* the first one will be used to specify that a user will receive an error message after a time-limit for the connection to the server, and the second one that the request of the user will be re-directed to another server after the connection time-limit.

So in this step we will describe for each aspect its type *(required* or *optional),* and if it is involved in *plugging constraints,* we will specify the different constraints and other aspects involved.

⊕ ***Aspects Definition Representation***

We can use a table to summarize the results of this first part. As an example we take a table used in the aspects design of the client application of RECINTERNET from Section III.4.2.

	User Interaction Support	Navigation Support
	Aspects related to user interaction and direct consequences of user interaction	*Aspects related to navigation actions or tasks required for navigation*
	Events Handling	Dynamic Node Loading
	Defining components interactibility and linking interaction events to the appropriate methods	*Searching or creating required nodes when navigating*
	Required	Required
	No	No

⊗ **POINT II: COMPONENTS DESIGN**

Designing the components of the system is done in the conventional object oriented way. Based on the application specifications, the system is decomposed into functional entities (classes and objects). The specific point due to AOP is that the designer must be careful to definitively not address in this functional decomposition of the system any feature related to some concerns that have been defined to be treated in aspects. It is important to take one by one the identified aspects to check that there is nothing related to them in the components model.

⊗ **POINT III: ASPECT CHARACTERIZATION**

At this point of our step by step decomposition, we have designed the components and also identified and briefly defined the aspects that will be used. It is now possible to characterize these aspects:

⊕ ***Step III-1: Aspect description applied to components***

Based on the aspect description realized in *Step I-2: Aspect Identification,* and using the components designed, we can refine the aspect description by precisely expressing the modifications involved. In this step we try to apply the first description realized to the designed components. For example if in a web-based three-tiers application a database is accessed through a given set of methods, the *Database Access Synchronization* aspect could be redefined as: *all the methods accessing the database X will be synchronized using a lock on which they will be queued.*

⊕ ***Step III-2: Involved components***

With this description and with the components model, it is possible to track for each aspect the involved components. This point is the answer to the question: *what are the components involved that are affected (i.e. will require some modifications) by something in the description of this aspect?*

⊕ ***Step III-3: Join points Definition***

In this step we will explicitly name the join points for each aspect and for each component involved. We will then name them one by one, without generic naming (such as the possibility to use wildcards in AspectJ[20]). We will use the Java signatures of methods, constructors and variables to define join points. In a case where the join point does not exist in the component (for example when adding a variable to a class), the join point will be the new member name (for example `Point.shadow` in the case of adding the variable `shadow` to the class `Point`). This step is the answer to the question: *Where will each component involved with an aspect be modified?*

Taking one by one the join points previously defined, we will define their type. This point is the answer to the question: *what kind of modifications will this aspect do on the given join point?* There can be several modifications for a same join point, and in this case we will do as if there were two distinct join points. There are pre-defined modification types (that are strongly related to the aspect language used). We give these modification types as a base for aspects expressed with AspectJ: *variable adding, constructor adding, method adding, before wrapping, after wrapping, catch wrapping, finally wrapping* in reference to the aspect expression possibilities of AspectJ described in Section III.1.2.3.

⊕ ***Join Points Representation***

We can then use a table to summarize the definitions of this part. As an example we take such a table from the aspects design of the client application of RECINTERNET, presented in Section III.4.2.

`RINode3`	`new()`	*For adding listeners to the “interactable”components created in this constructor*
`Transformation`	`void action(String name)`	*For defining the* `targetNode` *variable accessed in this method*
`NavigTransformation`	`void specificAction(String name)`	*Idem*
`IntraTransformation`	`void specificAction(String name)`	*Idem*
`ExtraTransformation`	`void specificAction(String name)`	*Idem*
`RIPrevNextTransformation`	`void specificAction(String name)`	*Idem*

⊗ **POINT IV: ASPECT MEMBERS DEFINITION**

⊕ ***Step IV-1: Aspect Members Identification***

For each aspect described before we will try to group all the join points that will be treated in the same way by the aspect. It means if the same piece of code will be added at the end of several methods, these methods can be grouped. Join points can be grouped only if they have the same type. A join point included in two distinct modifications should be seen as two join points. We call the obtained group aspect members, because they will represent the different parts in the aspect implementation.

[20] Like `* Point.*(..)` to refer to all the methods of class `Point`.

We also try to express the join points of an aspect member through generic expression using the particular characters of AspectJ syntax, as for example: `public !abstract * set(..), * *.get(String name)`, which means all the non abstract public `set` methods of any class and all the `get` methods of any class taking a `String` as parameter.

For each aspect member we will also explain what modifications will be realized on the corresponding join points. This step has the objective to provide enough information for the implementation of the given aspect. It answers to the question: *What modification should be implemented for this group of join points?* For example, in the case of a debugging aspect that is used to print a message each time a variable of the class `Point` is modified (through a mutator), we could have the following description for the *post-method wrapping* on the join points `Point.setX(..)` and `Point.setY(..)`: *print the following message on console window: "a variable of the class Point has been modified".*

Finally we also specify the type of each aspect member. This type is the type of all the join points of an aspect member. It is one of the types given in *Step III-3: Join Points Definition.*

⊕ ***Step IV-2: Aspectual Members***

With AspectJ it is possible to have variables, constructors and methods inside an aspect. They can be called from within the different members of the aspect and should be specified in the design since they can be important in the realized modifications to components code. We call them aspectual members

These aspectual members should also be clearly identified. We will use the same way than for components join points to identify them. The "component" will be the aspect concerned, and the "join point" will be the signature of the given aspectual member.

Aspectual member type will be specified. It can be one of the following: *aspect variable, aspect constructor* and *aspect method.* We also need to give a brief definition of these aspectual members.

⊕ ***Aspectual Members Representation***

As we have represented join points, we can represent aspectual members in a table. The following table, extracted from Section II.4.2, represents three aspectual members (methods):

`Events_Handling_RINode3`	`Boolean dispatch(Event evt)`	*Dispatch an event to the appropriate method of the node 3*
`Dynamic_Node_Load`	`Node createNode(String name)`	*Create a new node out of its name*
	`Node getNode(String name)`	*Call the* `getNode` *method of the context. If the node is not found, call the* `createNode` *method*

III.2.3.2) Aspects design table

Based on this two-dimensions (components and aspects dimensions) step by step decomposition we propose a way to represent aspects at design level. The idea is to represent the characterization of each aspect in a table. This table contains different parts representing the different elements addressed in the previously explained decomposition. Some parts of the tables can be extended (specifying more information about some given points) or collapsed (to not use some information that could not be useful in the design).

The Table 4 presents a simple example of such a table in the case of a debugging aspect. In this example all the table parts are entirely shown.

		Debugging[a]		▨
		Aspects dealing with debugging stuff		▨
		`VarAccessShowing`[b]		▨
		Show all the accesses to the variables of the classes `Point` ***And*** `Circle`		▨
		Optional Aspect – No Plugging Constraints		▨
		Read member[c]	Write member	▨
		Display a message telling that a variable has been read	*Display a message telling that a variable has been modified*	▨
		After wrapping	After wrapping	▨
C	▨	No[d]	No	▨
`Point`[h]	`GetX()`[e]	▨	[g]	
	`GetY()`	▨		
	`SetX(int x)`		▨	
	`setY(int y)`		▨	
Circle	`getCtr()`	▨		
	`getRay()`	▨		
	`setCtr(int c)`		▨	
	`setRay(int r)`		▨	

Table 4 - Aspects design table: simple example

In this table, we present how the `VarAccessShowing` aspect can be described at design level. The following notes explain some specific points. This example is used to present the main idea of the aspects design table.

(a) When a new aspects area is added in a table, the table is extended horizontally with a new column.

(b) When a new aspect is added into an aspects area, a new sub-column is added into the column corresponding to this area.

(c) The names given to the aspect members are just used to distinguish them. These names are not used in AspectJ implementation.

(d) These cells are filled in when some syntax of AspectJ (using '`*`' or '`..`') can be used to define several join points of the corresponding member in one expression.

(e) When a new join point must be added for a component, a new sub-line is created in the appropriate component line. One join point can be involved in several aspects members. Here we do not show the entire signature of the different join points (for space reason).

(f) A colored cell means that the corresponding join point is involved in the corresponding aspect member.

(g) A non colored cell means that the corresponding join point is not involved in the corresponding aspect member.

(h) When a new component must be added, a new line is created.

III.2.3.3) A specific use of the aspects design table: detecting composition conflicts

The aspects design table can be used to track some kind of composition conflicts. A composition conflict appear when the weaving of an aspect will provoke some errors or some interference with already implemented code. There can be composition conflicts between aspects and components or between different aspects. In this part we will focus only on aspects composition conflicts. This kind of conflicts will appear when weaving an aspect is defining some modifications on components code that will interfere with some modifications defined in another aspect (or in other aspects). The composition will then create some problems that will maybe be specified by the weaver, but maybe the problems generated will not be detect and then will create some problems at execution.

These conflict generally arise because two distinct aspects can be developed independently, only based on the components implementation. Many distinct composition conflict types can exist. Some of them are very subtle, or specific to each application. Anyway some of them can be detected and then corrected at design level. The aspects design table provide a simple visual way to track some types of conflict.

The conflicts addressed in this visualization with the aspects design table are some conflicts due to the fact that some join points can be involves in different aspect members (potentially from different aspects). So all the join points that are present in several aspects members are potentially conflict sources. With the aspects design table we can simply identify such join points. These are all the join points represented by the lines where several cells are colored. The Table 5 shows an imaginary example where the join point `Point.init()` is used by two distinct aspect members. The line corresponding to this join points is colored in two cells, corresponding to the two columns of the involved aspect members. This potential composition conflict source is then easily visualized thanks to the aspects design table[21].

		Resources Management			▨
		A Resource management		All resources creation	▨
		Release Resource A	Reset Resource A	Creation	▨
		After wrapping	Before wrapping	Method Adding	▨
C ▨	▨	`Point.init(..)`	No	No	G ▨
`Point`	`init()`	▨		▨	
	`init(int i)`	▨			
	`getX(int x)`		▨		
	`setX(int x)`		▨		

Table 5 - Detecting potential composition conflicts

All the join points involved in several aspect members are potentially sources of composition conflicts and should be carefully checked at design level. However, there are cases where we can precise the composition conflict type, depending on the modification type of the different members involving the same join point. The Table 6 shows these composition conflict types. In this table are presented what kind of conflicts can happen due to the modification types of two aspect members involving the same join point. The colored cells define cases that cannot happen because of the syntax used for join points (for example it is not possible to use a join point expression that will reference both a variable and a constructor. The explanations in the cells present possible

[21] In this case there is effectively a composition conflict since the aspect member *Creation* objective is to define a new method `init()` in the class `Point`. Either the method already exist in the component implementation and then it cannot be redefined (this is a component-aspect composition conflict), either it does not exist yet, and in this case the aspect member *Creation* must be woven to the components code before the aspect member *Release Resource A* (that makes some modifications on this method), unless there will be a composition conflict.

causes for possible composition conflicts. It has to be noticed that here are only presented potential conflicts sources directly involved by two modifications types mix. The possible conflicts due to the composition of a single aspect member with some components are not included in this table.

	Aspect method	Aspect constructor	Aspect variable	Finally wrapping	Catch wrapping	After wrapping	Before wrapping	Method adding	Constructor adding	Variable adding
Variable adding	▨	▨	▨	▨	▨	▨	▨	▨	▨	Duplicate definition
Constructor adding	▨	▨	▨	▨	▨	▨	▨	▨	Duplicate definition	
	▨	▨	▨	Weaving order problem (a)	Weaving order problem (a)	Weaving order problem (a)	Weaving order problem (a)	Duplicate definition		
Before wrapping	▨	▨	▨	Compatible types	Compatible types	Compatible types	Compatible types			
After wrapping	▨	▨	▨	Compatible types	Compatible types	Compatible types				
	▨	▨	▨	Weaving order Problem (b)	Duplicate catch for the same error type					
Finally wrapping	▨	▨	▨	Incompatible types (c)						
Aspect variable	▨	▨	Duplicate definition							
Aspect constructor	▨	Duplicate definition								
Aspect method	Duplicate definition									

Table 6 - Modification types compatibility

(a) It happens when an aspect member references a method that has been added to a component by another aspect member. In this case the aspect member defining the *method adding* should be woven first.

(b) A modification type conflict can occur if there is a *finally wrapping* aspect member related to a *catch wrapping* aspect member and if this *catch wrapping* aspect member was not woven before the *finally wrapping* aspect member.

(c) Adding two “finally” clauses around the same method without any “catch” clause between is not possible.

It is then possible to easily visualize some possible sources of composition conflicts due to a delicate mix of two or more aspect members dealing with common join points. Moreover as the aspects design table is based on a very simple principle (decomposition of a system in two dimensions: a dimension for aspects and a dimension for components), it does not seem complicated to implement a little program that will identify the join points of the table that can be sources of composition conflicts[22]. Anyway there are composition conflicts that are subtle that require a careful track to be detected.

III.2.3.4) Conclusions

We have presented in this section the some guidelines to realize aspect-oriented design as well as a simple way of representing this design. Our step by step design process decomposition provide an efficient way to structure and organize aspects and components.

Important characteristics of this decomposition can be represented in aspects design table. The decomposition is then shown in two dimensions (aspects and components dimensions). Each entity of both dimensions is described and carefully identified. The aspects design tables provide a simple way to concretely

[22] In a software package such as Microsoft Excel, a macro can easily be defined for that. It will localize the join points involved in several aspect members (they can be colored in a specific color), and according to the Table 6 it can show some messages about potential composition conflict reasons.

visualize aspects/components interactions. An important point is that these interaction can be visualized in both ways: from a given component it is possible to track all the concerns (aspects) in which it is involved, and from a given aspect it is possible to find all the involved components. This is a powerful issue of our proposition since a clear identification of aspects/components interactions at design level provide possibilities to efficiently separate concerns without having problems in the final composition after implementation.

Aspects design tables can provide a good support for detecting simple composition conflicts between aspects. Identification of shared join points between distinct aspects is an important step in solving composition conflicts. This can be done very easily with the aspects design tables.

Finally we have proposed some guidelines and a way of representing aspects at design level. We illustrate their utility and present further use examples in the concrete case of the aspect-oriented design of RECINTERNET in Section III.4.

III.3) Web-based three-tiers application conventional object-oriented design – Application to the RECINTERNET case

In the following sections we draw a comparison between conventional object-oriented and aspect-oriented design. This comparison takes place in the context of web-based three-tiers applications, and is applied to the RECINTERNET concrete case. It is organized in three parts: the current section which presents the conventional object-oriented design of RECINTERNET, the Section III.4 which presents the same design but with an aspect-oriented approach, and finally the Section III.5 which draw a comparison of these two approaches.

So we begin in this section with the different parts of our object-oriented design for RECINTERNET. We present and explain the different design models we realized. Our design is separated into the navigational client application design and the entire system design including client, server and database.

III.3.1) Client application design

The design of the client side of RECINTERNET is realized according to some parts of the OOHDM methodology presented in Section II.4. This methodology divides the development cycle of hypermedia navigations into four incremental steps: *conceptual design*, *navigational design*, *interface design* and *implementation.* We apply partially this methodology for our design, and we also use some of the navigational design patterns described in the OOHDM methodology.

III.3.1.1) Conceptual model

In the OOHDM methodology, the first step of the development cycle is the definition of the conceptual model. It means representing in a model the different entities that are specific to the domain of the developed application. These specific entities can be defined in relational object oriented diagrams, in order to visualize easily the different relationships between the concerned entities.

We have decided to apply this *conceptual model* step of OOHDM in the RECINTERNET design. In the case of RECINTERNET, the entities specific to the application domain are all the entities stored in the databases. So we had to model the different entities stored in the databases. As explained before we reused exactly the databases of the RECSINWIN application (the local structural database as well as the SINTESE database). That is the reason why we reused the model that had been done for the RECSINWIN database. This model, that describes closely the tables structure of the local structural database, can be found in Appendix A.

In fact we reused this model to define different classes describing the different entries stored in the databases. Finally we defined an object oriented relational model, which is a UML (Unified Modeling Language) class diagram used to show the relationships between the classes describing the common behavior of the different domain-specific entities of the RECINTERNET application. The Figure 21 shows this conceptual model.

Without explaining all the details of this model, we will briefly present its main elements:

- `Serie`, `Group` and `Area`

Series are the basic entities of the SINTESE database (we refer to the previous definition of SINTESE series in Section I.3.1). The hierarchical organization used is the following: there are different interest areas, which can contain series or groups, where groups can then contain series. The idea of RECSINWIN and RECINTERNET is to compose a visualization of some entries of one or more series. Entries of a series are expressed in a unit (`Unit`).

- `Temporal_unit`

Series entries are organized in four dimensions. The first one is the temporal dimension. The entries of a serie are expressed for different temporal elements (for example different years, different months, etc...). Conversions between the different time units exist.

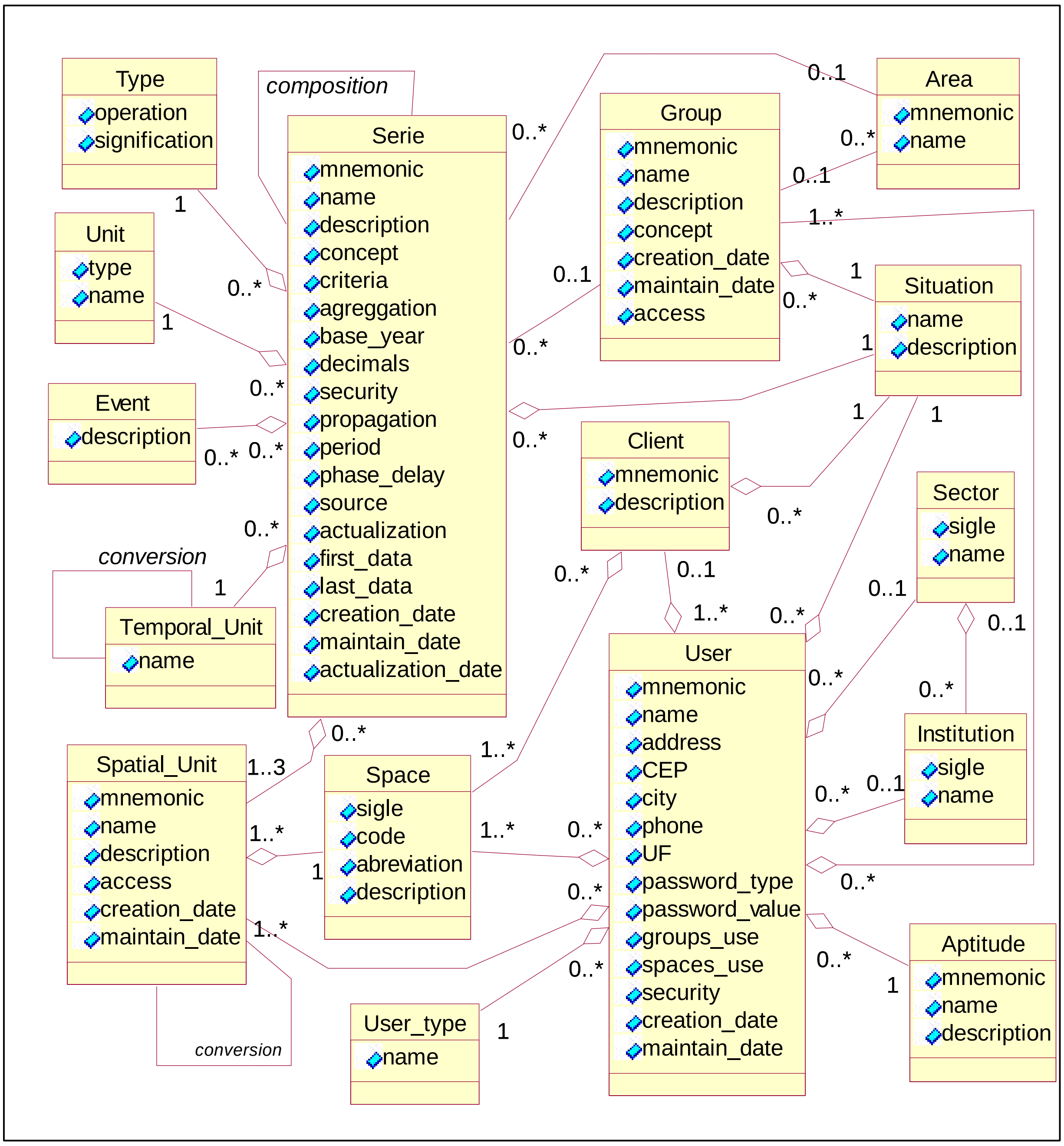


Figure 21 - RECINTERNET conceptual model

- `Space` and `Space_unit`

The three other dimensions of a serie are three “spatial” dimensions. The term spatial must not be taken literally, it is used to specify that a dimension can be expressed in three other dimensions than the temporal dimension. So the space represents the dimension (for example geographic dimension), and the space units are the different elements of a space (for example the different states of Brazil). Conversions can exist between the different space units.

- `User` and `Client`

Each user has some permissions to access some information of the database. His access on series groups, spaces, space units can be limited. A group of users having the same access permissions is called client.

III.3.1.2) Navigational model

In the OOHDM methodology, the *navigational model* is based on the entities of the *conceptual model.* A navigational node proposes a specific view on entities of the *conceptual node.* Some relationships between classes of the *conceptual model* can be abstracted in the *navigational model* as navigational links between nodes.

For RECINTERNET we do not realize the *navigational model* in the same way. A navigational node in RECINTERNET will not be a view on some entities of the *conceptual model* (as in OOHDM), but a logical unit proposing some choices in the organization for the visualization of entities of the conceptual model.

- ***Navigational Sequences***

The first task of the navigational design was to define the different ways to navigate in the RECINTERNET client application. This first step was adapted from the navigation in the RECSINWIN application.

The term *navigational sequence* describes the different navigation realized by a user from the moment he enters the application to the moment he visualize the results of his request. In Section I.3.2.4 we described how the *navigational sequences* of RECSINWIN are modeled. *Navigational sequences* in RECSINWIN and RECINTERNET are similar since both of them lead to the same result (the visualization of an organized view on a SINTESE serie). However, as RECINTERNET proposes this navigation in an “internet” way, adaptations had to be done. The idea was to not have more than 3 or 4 main nodes in a *navigational sequence,* in order to have a simple internet navigation. In the same time the different nodes had to be clear enough (i.e. not contain too many elements) to be easily understand by any user. As in RECSINWIN, a *navigational sequence* uses around 15 nodes, we had to re-structure the navigation. The Figure 22 presents a description of the *navigational sequences* in RECINTERNET.

- ***Navigational nodes***

Navigational nodes are perceivable navigational units. Navigation exists when the user goes from a node to another one. A node contains elements used to show information and navigate. An *intra node* is related to a *navigational node.* It is used to present more information about some elements of the *navigational node.* When navigating to an *intra node* from a *navigational node,* the *intra node* must be closed to allow the use of its *navigational node.* An *extra node* can be accessed from a *navigational node.* The *navigational node* and the *extra node* can be used then in parallel, independently. The typical use of *extra nodes* is for contextual help.

The RECINTERNET navigation is then organized as following:

- *Node 0 (Welcome)* presents some general information about DATAPREV, about RECINTERNET and how to use it. The user enters its login and password[23] and can then navigate to the *Node 1.*
- *Node 1 (Series Selection)* enables the user to select the series he would like to visualize. Once this choice realized, he can choose to organize the visualization *temporally* (the lines of the series visualization table will then be used for the temporal dimension) or *spatially* (the lines will be used for one of the “spatial” dimensions). The navigation is then processed either to *Node 211,* either to *Node 221.*
- *Node 211 (Lines Composition – Spatial way)* enables the user to select the “spatial” dimension he wants to use for the lines of the results table. He will then choose the elements to place in these lines. He can then navigate to *Node 212.*

[23] With no appropriate login/password, this user will only have some default access rights to SINTESE database.

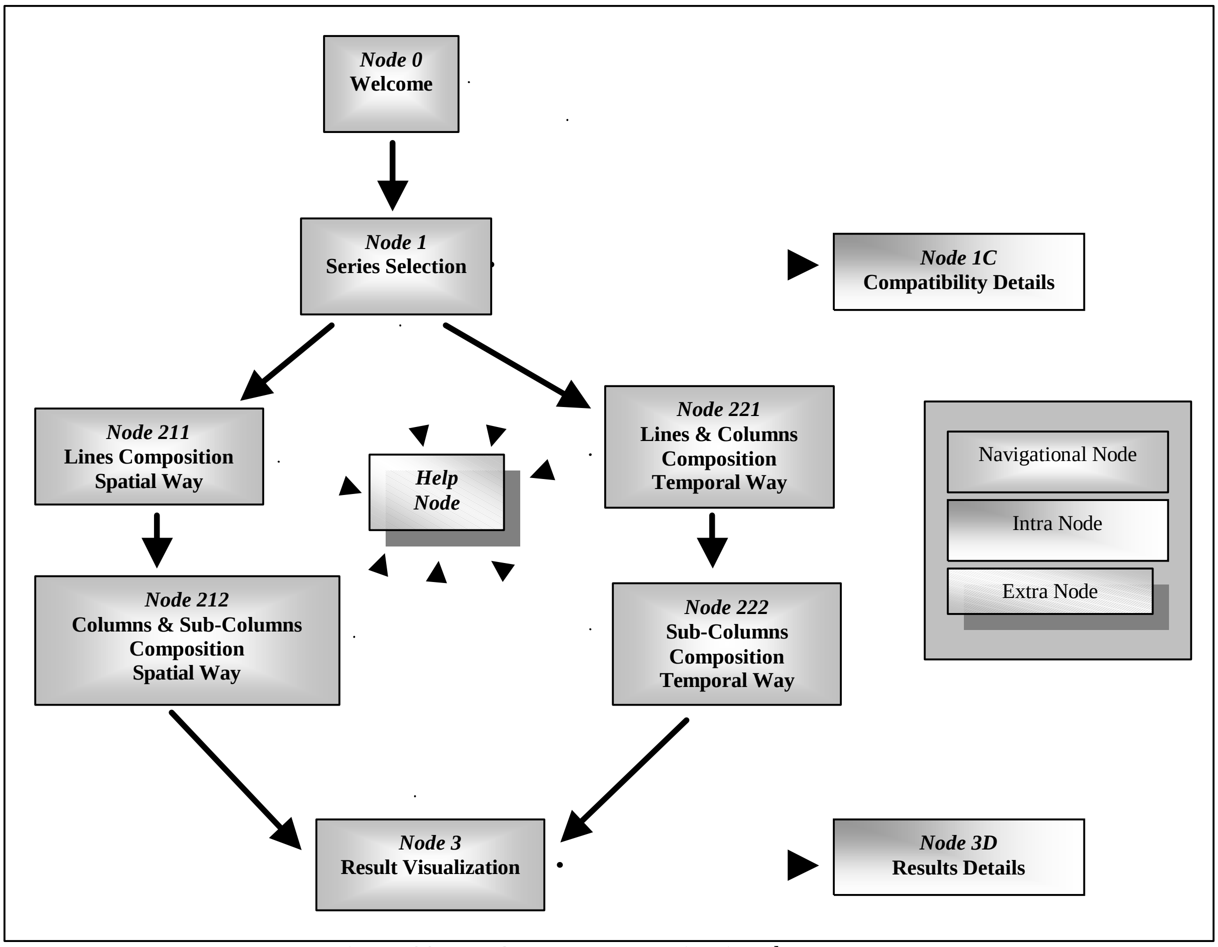


Figure 22 - RECINTERNET navigational sequences

- *Node 221 (Lines & Columns Composition – Temporal way)* enables the user to select a time unit and some temporal elements for the line of the results table. The user also specifies the columns of the results table (the columns will be different years). He can then navigate to *Node 222.*
- *Node 212 (Columns & Sub-Columns Composition – Spatial Way)* gives the possibility to organize the columns of the results table (columns will represent different temporal elements), and also the sub-columns (sub-columns will represent different elements of the two lasting “spatial” dimensions). The user can then navigate to *Node 3.*
- *Node 222 (Sub-Columns Composition – Temporal Way)* enables the user to select the “spatial” elements that will be used in the sub-columns of the results table. He can then navigate to *Node 3.*
- *Node 3 (Results Visualization)* presents to the user the table showing the selected series in the way corresponding to his composition.
- From all these nodes it is also possible to navigate to the *Help Node* which presents some contextual help about each node.
- From *Node 1* it is possible to navigate also to the intra node *Node 1C (Compatibility Details)* which presents some extra information about the compatibility between the selected series of the *Node 1.*
- From *Node 3* it is possible to navigate also to the intra node *Node 3D (Details)* which presents some extra information about the columns selected in the *Node 3.*

It has to be noticed also that for all the *navigational nodes*, some extra information (like localization in the navigational sequence) and some extra functionality (like re-starting the composition, exit the application, navigate backwards or forwards, navigate to help) are added through a *decorator*. A decorator is a kind of "mask" that is applied on a node to add some functionalities to this node. The advantage of using a decorator is to factor out some elements existing in different nodes.

- ***Navigational framework***

Based on these *navigational sequences* and *navigational nodes* definitions, we had to define a generic framework to support such navigations. For this task we applied some of the navigational design patterns described in the OOHDM methodology. The design patterns (or adaptations of them) we used for our navigational design are:

- ***Anchor:*** an anchor represents a link into a node and is responsible for this link activation. The link is then independent from any means to activate it.
- ***Navigation Strategy:*** proposes a solution to decouple links from the way their target are obtained.
- ***Navigation Observer:*** use of an History object (we will use the term *context)* to record the information about navigation.
- ***Node class:*** abstract the behavior of similar nodes. This class will be sub-classed to define the concrete nodes used in the application.
- ***Link class:*** abstract the behavior of similar nodes. This class will be sub-classed to define the concrete links used in the application.
- ***Wrapper node*** and ***Navigational context:*** the idea is to decorate nodes with some wrapper nodes (we will use the term *decorator)* providing some more interaction or information possibility to the node, eventually depending on the navigational context.
- ***Node as a single unit:*** a node encompass a self-contained "unit" of information that make sense for users performing a set of tasks in a given domain. All data that are relevant to this set of tasks should be included in this node.

For detailed descriptions of these design patterns, we refer to [LRS98] and [RSG97].

The *navigational framework* we designed is language independent and can be extended for several navigational applications requiring support for concepts such as decorators, navigational, intra and extra nodes, anchors, links, target and transformations or context. The Figure 23 presents this framework. A complete description of the elements of this class diagram is given in Appendix C.

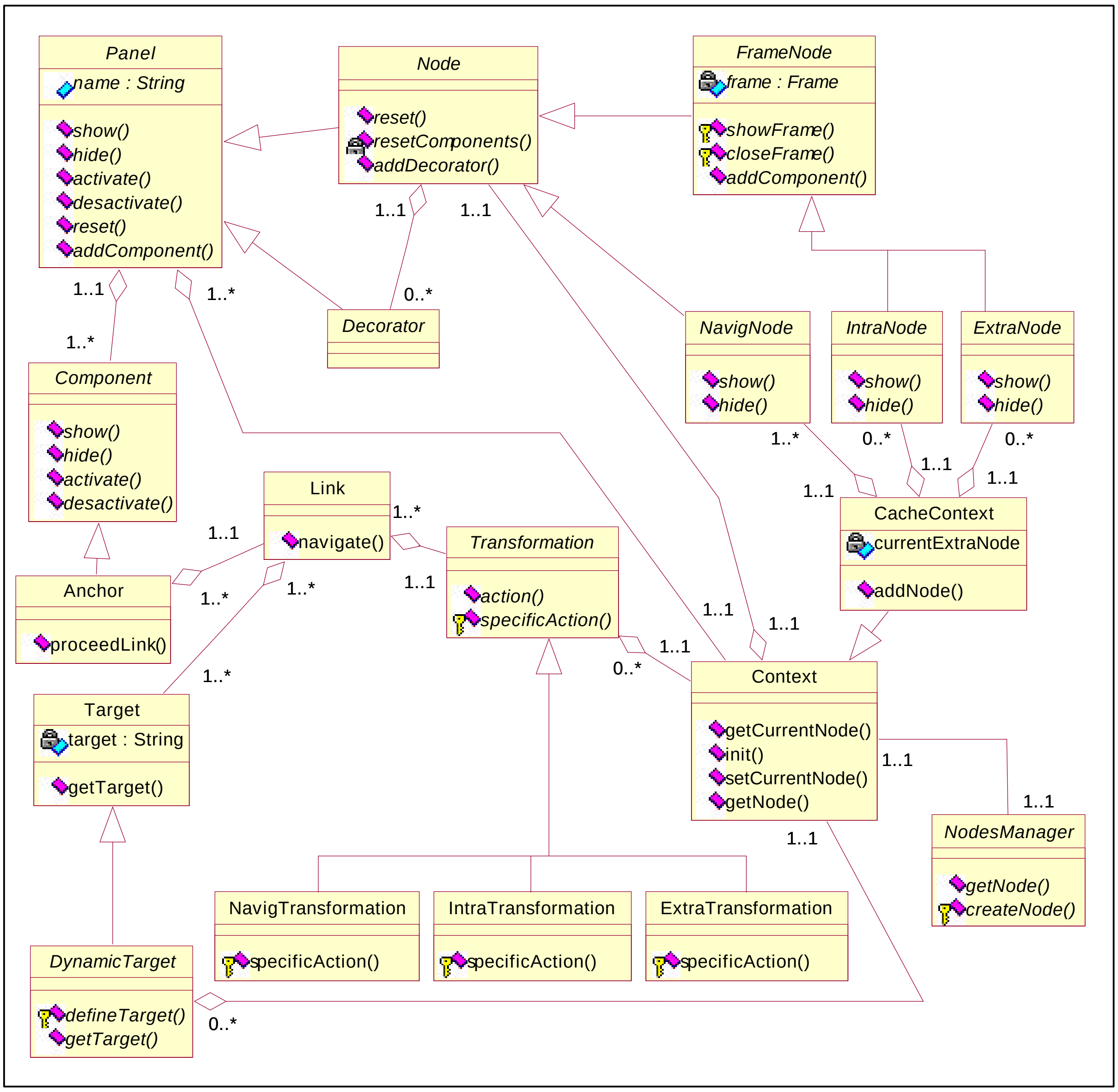


Figure 23 - Navigational framework class diagram

- ***Navigational model of RECINTERNET***

We extended the navigational framework described previously to support the navigation of the RECINTERNET client application. We present in the Figure 24 a class diagram of the RECINTERNET client application navigational elements. The prefix "RI" in the class name is for RECINTERNET. It shows that this class is specific to the RECINTERNET application. The classes without the "RI" prefix are part of the navigational framework presented in Figure 23. The different nodes and decorators will be described later.

It has to be noticed that we extended the classes `CacheContext`, `Transformation` and `DynamicTarget` in order to support forward (next) and backward (previous) navigation. It is possible to navigate this way between two navigational nodes without re-initializing the target node. The context keep a reference to know until which navigational node it is possible to navigate forward. The method `nextAllowed` check this reference to know if the forward navigation is allowed.

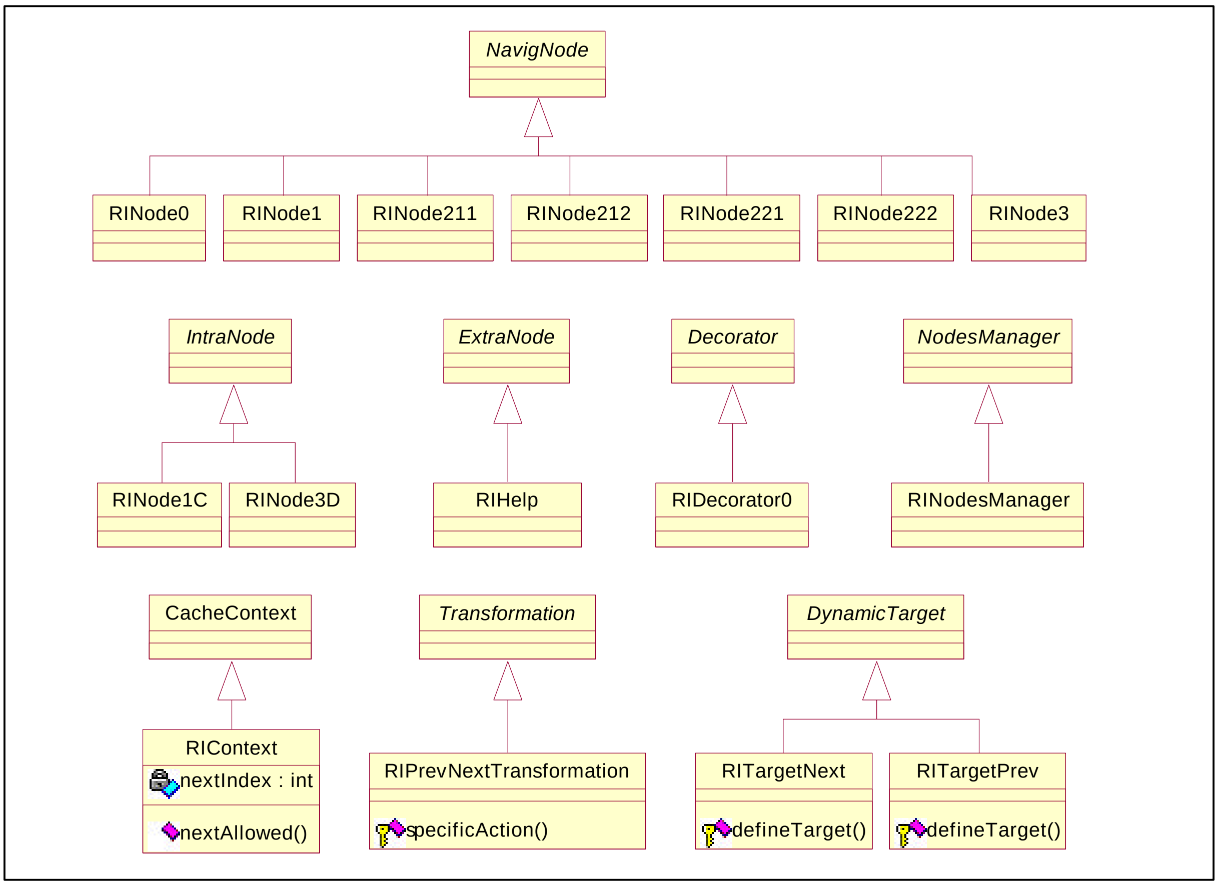


Figure 24 - Navigational framework extension for RECINTERNET

III.3.1.3) Interface model

The last step of the OOHDM hypermedia application design is the *interface design*. The aim of this step is to define all the elements of the graphical interface in classes, using the navigational model. The event handling and the organization of these elements will define the behavior of the interface.

In this part we use some *components types* that describe the kind of component (for example: button, text field, list, arborescent list, etc...). For the implementation these types will have to be mapped to the components existing in the libraries of the chosen programming language.

To define this graphical interface we designed each node and decorator. For that we defined which graphical components should each node (or decorator) contain. We defined then how would each node be organized (how to place the different components). For each component we described its characteristics and the way the user can interact with it (or not). We described also what should happen each time the user interact with a component. This description of the RECINTERNET interface can be found in Appendix D.

Based on this description we completed the design of the classes presented in Figure 24 to obtain the concrete elements that will be used in the interface. In the Figure 25 are presented these modifications. We do not present all the components of each node, but only the main characteristics of their anchor, as well as the different methods that will be used to handle user interactions[24].

[24] The concrete way these methods are called when the user interacts depends on the programming language as well as the technology used for the client application.

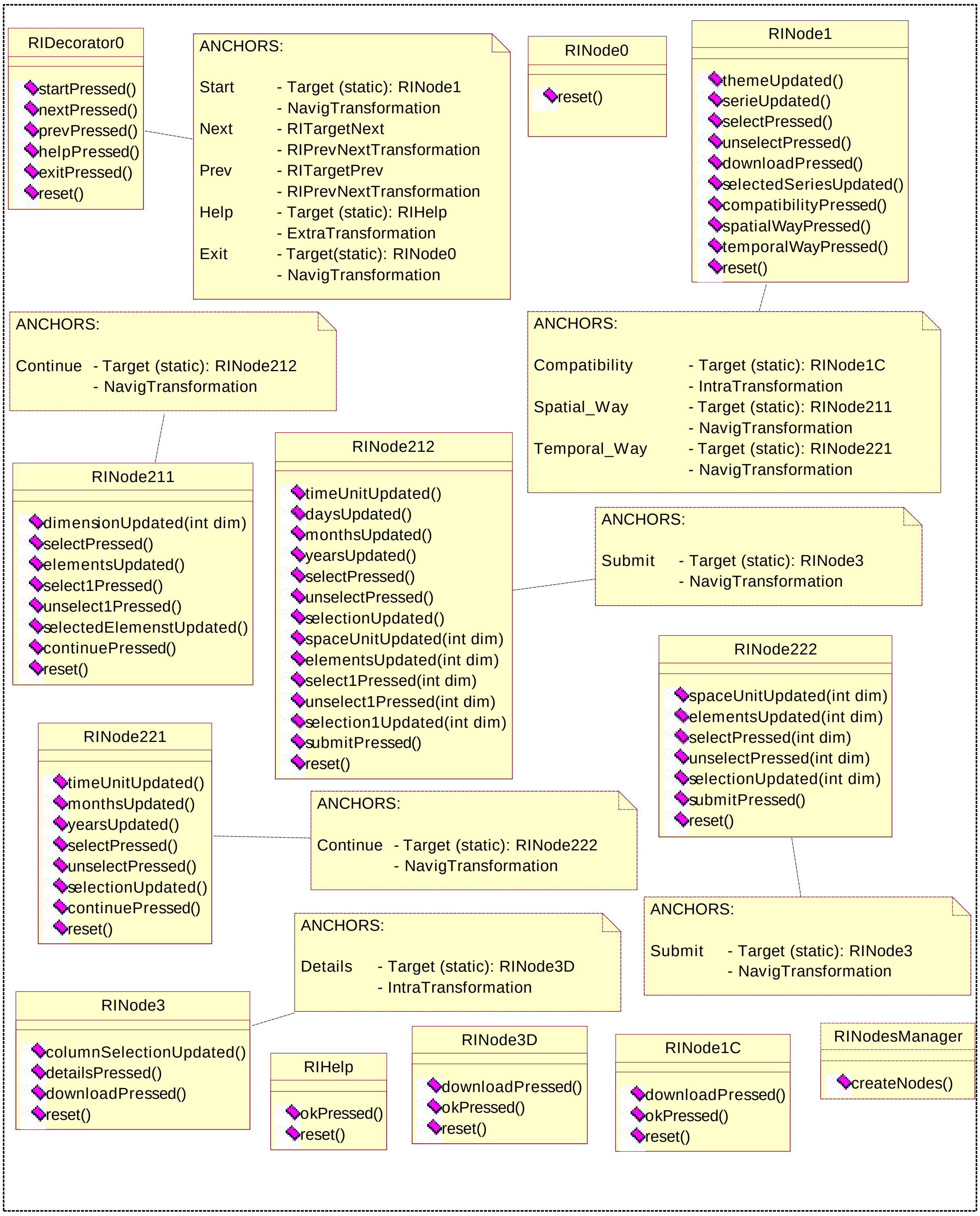


Figure 25 - RECINTERNET interface extension

Finally with this decomposition in three levels (conceptual, navigational and interface) for the client application design, we provide some design models that cover the different requirements of the concrete implementation of the client side of RECINTERNET. Moreover these models are basically language independent, and are also based on a navigational framework which is application independent. These points provide solid bases for future design reuse.

III.3.2) RECINTERNET entire system

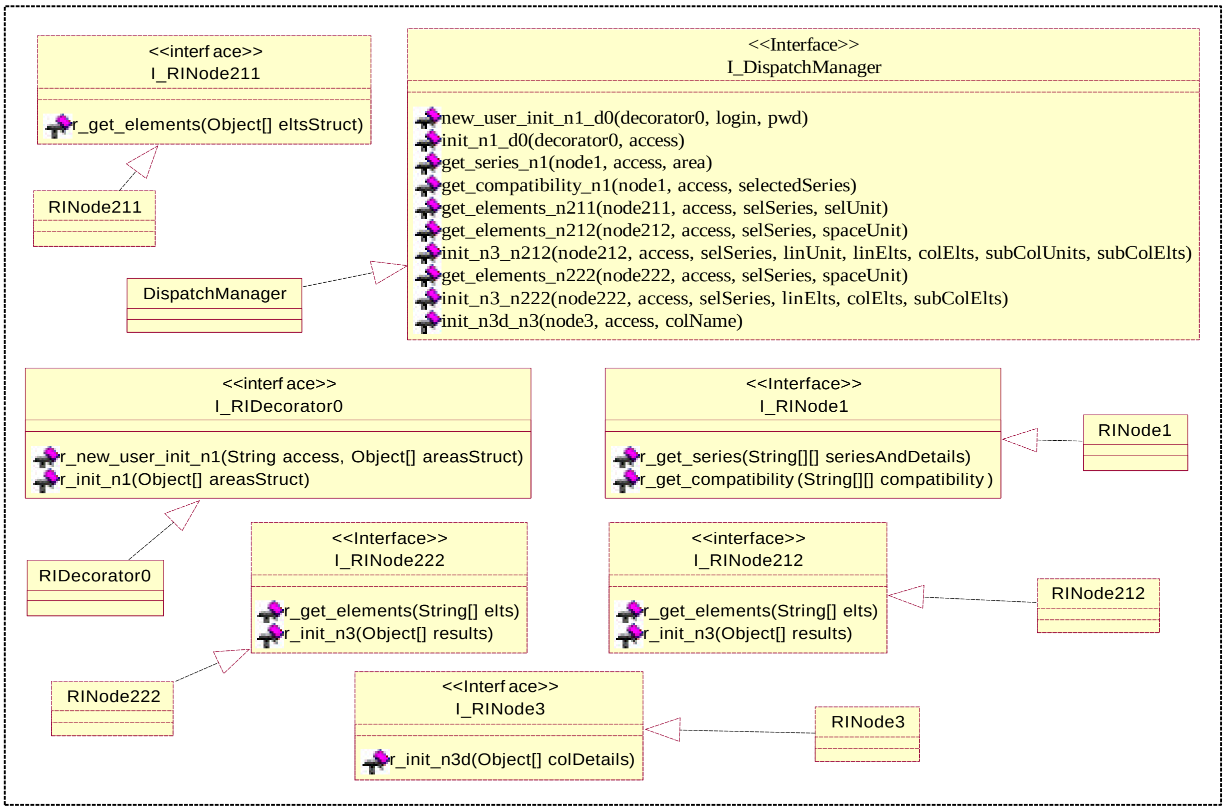
We have summarized in Section II.7 the different technologies used for RECINTERNET. The client application is implemented as a Java applet running in a web browser. The server is a Java application running on the server host. The client/server communication is realized through Java RMI (Remote Method Invocation). The server local structural database on the server host is accessed with SQL queries through JDBC (Java DataBase Connectivity). The SINTESE database on the SINTESE host is accessed with queries in SINTESE syntax through a local terminal simulation connected to the SINTESE host with a TCP/IP protocol.

However the design we propose for the entire RECINTERNET system is quite generic and could be used with other technologies. The specifications we followed for the design were: the client and the server are applications running on different hosts. The client/server communication is done through some API providing transparent remote references (such as Java RMI or CORBA). The access to the server local structural database is done with execution of SQL queries on a database connection. The access to the SINTESE database is done with SINTESE syntax on a simulation of a terminal providing queries execution possibilities.

We have described the client application design in the previous part. The following sections present the design for the other parts of the RECINTERNET system.

III.3.2.1) Client/Server communication

We designed the client/server communication for any middleware that provides a transparent way to realize remote method calls, by the way of remote invocations interfaces (such as RMI interfaces with Java or IDL interfaces with CORBA). We designed the different interfaces that will be necessary to support user interaction transmission to server as well as server answer. The requests will be submitted by the nodes objects of the client application to a unique dispatch manager object of the server application. The Figure 26 shows these interfaces.



<<interf ace>>
I_RINode211
r_get_elements(Object[] eltsStruct)
RINode211
<<Interface>>
I_DispatchManager
new_user_init_n1_d0(decorator0, login, pwd)
init_n1_d0(decorator0, access)
get_series_n1(node1, access, area)
get_compatibility_n1(node1, access, selectedSeries)
get_elements_n211(node211, access, selSeries, selUnit)
get_elements_n212(node212, access, selSeries, spaceUnit)
init_n3_n212(node212, access, selSeries, linUnit, linElts, colElts, subColUnits, subColElts)
get_elements_n222(node222, access, selSeries, spaceUnit)
init_n3_n222(node222, access, selSeries, linElts, colElts, subColElts)
init_n3d_n3(node3, access, colName)
DispatchManager
<<interf ace>>
I_RIDecorator0
r_new_user_init_n1(String access, Object[] areasStruct)
r_init_n1(Object[] areasStruct)
RIDecorator0
<<Interface>>
I_RINode1
r_get_series(String[][] seriesAndDetails)
r_get_compatibility (String[][] compatibility)
RINode1
<<Interface>>
I_RINode222
r_get_elements(String[] elts)
r_init_n3(Object[] results)
RINode222
<<interface>>
I_RINode212
r_get_elements(String[] elts)
r_init_n3(Object[] results)
RINode212
<<Interf ace>>
I_RINode3
r_init_n3d(Object[] colDetails)
RINode3

Figure 26 - RECINTERNET remote interfaces

III.3.2.2) Server

The server application receives the clients requests through its dispatch manager object. This one creates a new client thread (independent process running on the server) for each client new request. There is a client thread class for each possible request type, as shown on Figure 27.

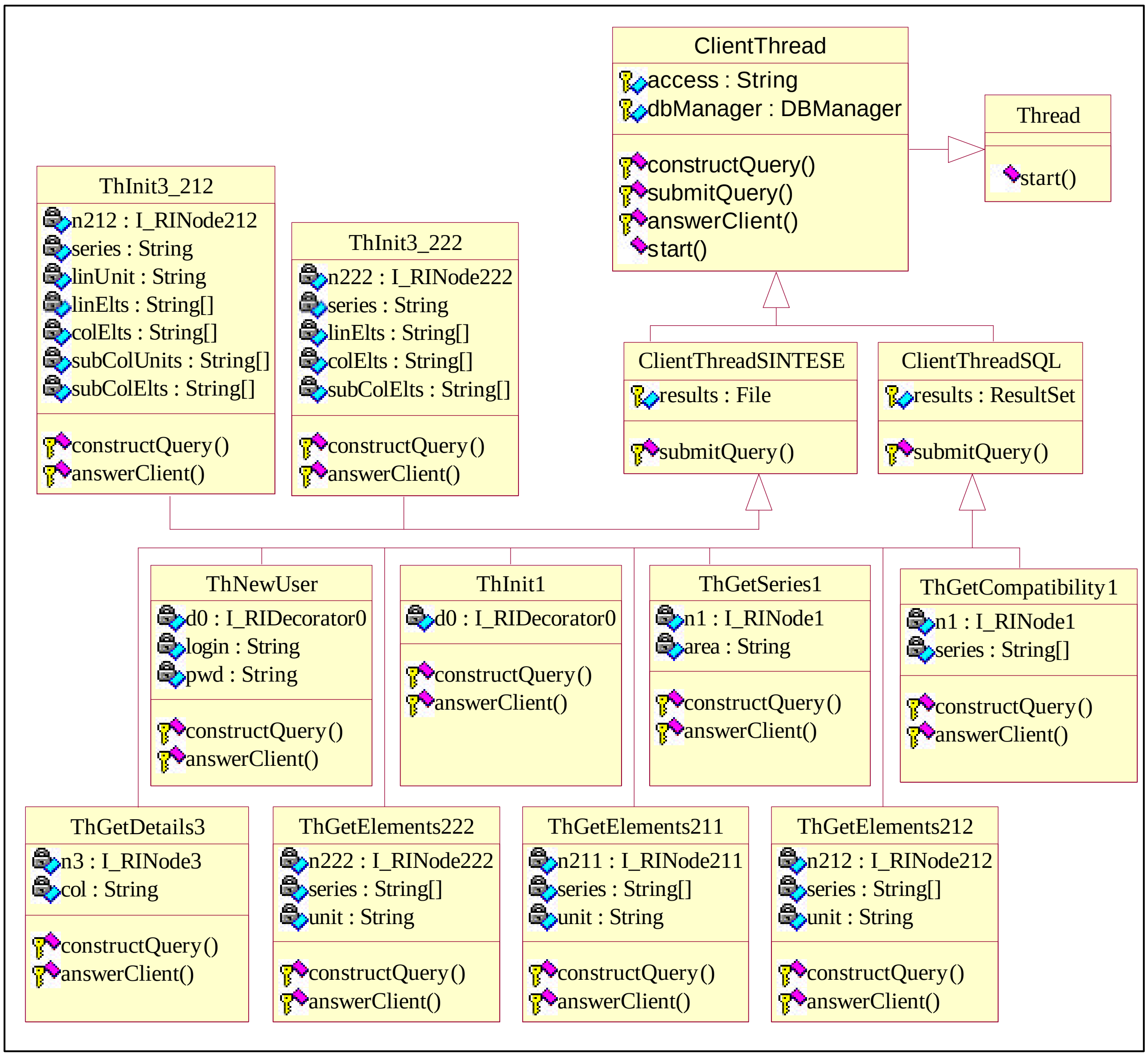


Figure 27 - Client threads class diagram

These client threads are used in the way presented in Figure 28. The server creates a new thread for each client request. This thread will be responsible for everything related to database queries. It means *query creation*: using pre-defined queries sentences (in SQL or in SINTESE syntax), entire queries are created by inserting the parameters given by the client into these pre-defined query sentences. The thread is also responsible for passing the query to a database manager (responsible for the database connection) and receives the appropriate result. The thread will then send back the result to the client object after having computed it (so that it could be directly usable by the client). It has to be noticed that the concurrent accesses to the databases are synchronized in the `DBManager` class, described in the next part.

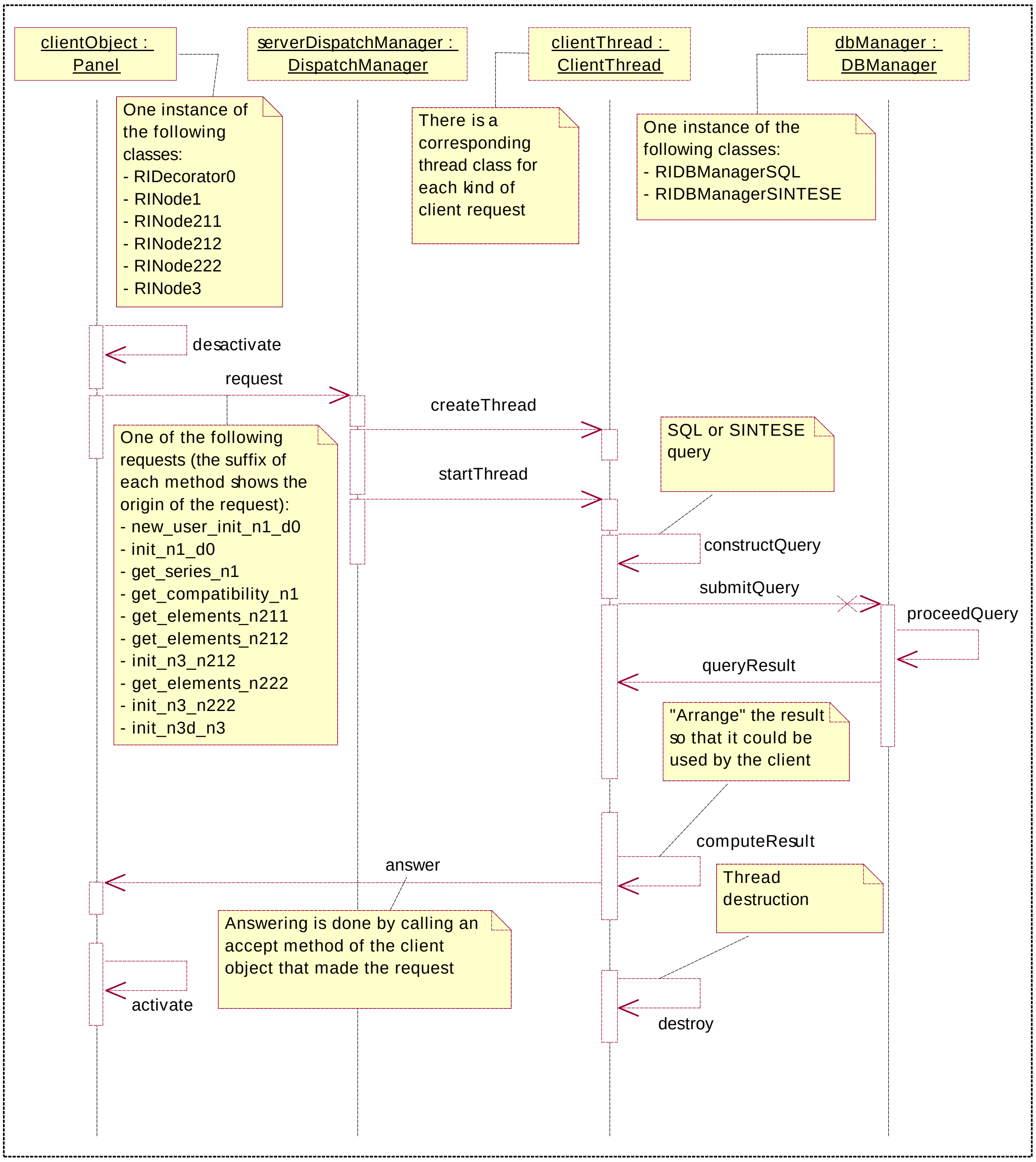


Figure 28 - RECINTERNET request scenario

III.3.2.3) Database access

We designed the databases management in order to concentrate all the related features in the class `DBManager`. This class synchronizes the different threads requesting database access. All the issues of database connection and queries processing are addressed by this class. This class will be sub-classed into classes specific to each kind of database access. The different methods of these classes will address the particular operations for the connection to their specific kind of database.

We designed two sub-classes (`RIDBManagerSQL` and `RIDBManagerSINTESE`) for managing respectively the access to the structural database and to the SINTESE database, as it can be seen in Figure 29.

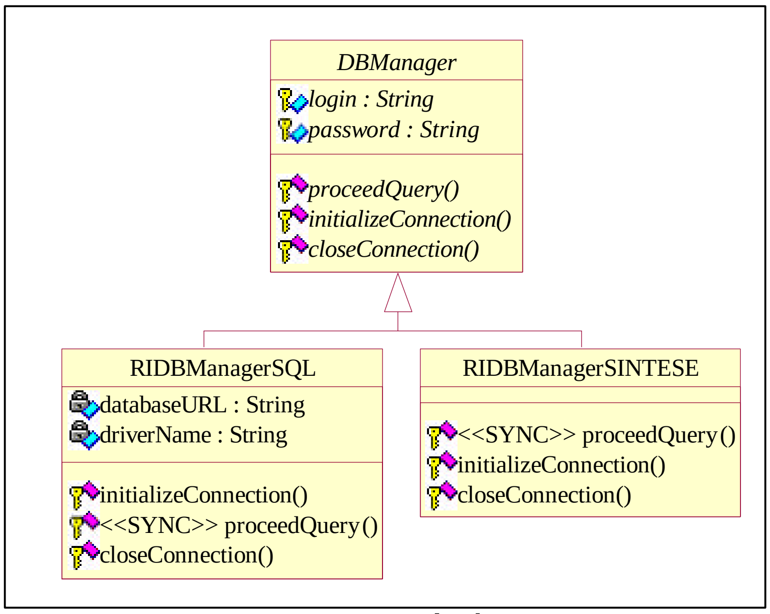


Figure 29 - RECINTERNET databases management

It has to be noticed that the results of a query processed by a database manager are returned in their basic form (SQL result set with JDBC for example, or a single stream with SINTESE), but after a cast to a generic class (in Java the class `Object`). These results will then be “arranged” by the corresponding threads.

III.3.3) Conclusions

We have presented in this section different design models that cover the different aspects of the RECINTERNET system. Following the OOHDM methodology and applying some of its design patterns for the client application, we have provided different models and description which are concrete and efficient supports for implementation, for each level of the application: *conceptual, navigational* and *interface.*

We have also specified design models describing the web-based three-tiers architecture of the RECINTERNET system, as well as the interactions between the three tiers: *client, server* and *database.* With UML diagrams we have defined the main characteristics of the different classes that will be implemented for the RECINTERNET system.

It has to be noticed that we proposed a language-independent framework providing sufficient abstraction to be reused in applications similar to RECINTERNET, basically web-based three-tiers applications. We specified this framework in order to provide an appropriate support for RECINTERNET implementation, but it seems possible to realize simply this extension for other applications.

To conclude this section about conventional object-oriented design of RECINTERNET, we can say that we have created different models covering the main needs of RECINTERNET implementation, but that also provide an appropriate support for future modifications of this system, as well as enough abstraction to reuse our framework or parts of it in different contexts.

In the following Section (III.4), we present the same design but with an aspect-oriented approach. These two design approaches are compared in Section III.5.

III.4) Aspect-oriented web-based three-tiers application design – Application to the RECINTERNET case

III.4.1) Introduction

We have explained in Section III.2 how separation of concerns and particularly aspect-oriented programming can be interesting for web-based three-tiers applications development, and especially at design level. We also explained in this section how an aspect-oriented design decomposition can be structured and represented with aspects design tables. We will now present the concrete case of an aspect-oriented design, in which we applied this decomposition methodology to the case of RECINTERNET.

We have divided the aspect-oriented RECINTERNET design into four parts: client application, server application, client/server communication and database. In each part we tried to identify cross-cutting concerns that could be expressed in aspects, mainly in the areas presented in Section III.2.1.2: *sharing resources*, *user interaction*, *server functionalities*, *remote communication* and *database access*. An important point is to express in aspects only features that improve design and implementation by being encapsulated in an aspect.

The following design is based on the possibilities of AspectJ since the aspects described were created to be composed with components by the AspectJ weaver. As AspectJ uses Java, we will often use some Java code in our design description. Moreover many parts of the design are very close to the conventional object-oriented design of RECINTERNET presented previously. We will refer to some parts of the conventional object-oriented models in order to avoid fastidious model descriptions repetition. Finally having previously designed the system in a conventional object oriented style helped us to identify better which concerns were cross-cutting consequently the code and should be beneficially expressed in aspects.

In the following sections we describe how we applied our aspects/components decomposition methodology (Section III.2.3) in each part of the RECINTERNET system. For the following explanations we will follow the three parts of this methodology: aspects identification, components design and aspects characterization. Once again it has to be noticed that these three parts are often overlapping each other and so their definition is often done in parallel. However we present them in this order. We will also used *aspects design tables* to represent the different entities of the RECINTERNET design.

III.4.2) Client application

For this section we will detail each step of the decomposition:

⊗ **POINT I – ASPECTS IDENTIFICATION**

⊕ ***Step I-1: Aspect areas identification***

The main features of the client application in RECINTERNET are: *user interaction support*, *graphical interface* and *navigation support*. *User interaction support* covers all the concerns dealing with user interaction events creation, handling and consequences. User interaction events mechanisms are involving many components such as different nodes of the navigational system. It seems to us that concerns of this area could beneficially expressed with aspects. Concerns addressed by the *graphical interface* feature are mainly visualization of data and interaction possibilities. It appears that these concerns can cleanly be encapsulated into object oriented structures, using GUI libraries such as Java AWT (Abstract Windowing Toolkit), and do not require specific concerns cross-cutting components implementation. Then we will not take the *graphical interface* feature as an aspects area. And finally the *navigation support* feature deals with concerns involving many components of the client application, such as nodes, anchors, links, context, etc... That is the reason why the concerns of the *navigation support* area should be better developed with aspects. So finally we will focus our aspects identification on the two aspects areas *user interaction support* and *navigation support.*

⊕ ***Step I-2: Aspects identification***

Taking one by one these two aspects areas, we have identified the following aspects:

⇒ *User interaction support aspects area:*
  ⇒ **Event handling** aspect: defining components interactability (register listeners associated to components which will create events for each user interaction) and associating interaction events (button click, selection, etc...) to the appropriate methods calls.

⇒ *Navigation support aspects area:*
  ⇒ **Dynamic node loading** aspect: searching existing nodes or creating non-existing ones when a client requests a navigation towards a node.

⊕ ***Step I-3: Aspects type***

⇒ *Event handling*
  ⇒ *Type:* required[25]
  ⇒ *Plugging constraints[26]:* no

⇒ *Dynamic node loading*
  ⇒ *Type:* required
  ⇒ *Plugging constraints:* no

⊕ ***Aspects Definition Representation***

	User Interaction Support	Navigation Support
	Aspects related to user interaction and direct consequences of user interaction	*Aspects related to navigation actions or tasks required for navigation*
	Events Handling	Dynamic Node Loading
	Defining components interactibility and linking interaction events to the appropriate methods	*Searching or creating required nodes when navigating*
	Required	Required
	No	No

⊗ **POINT II – COMPONENTS DESIGN**

We use the same design for the client application that in the conventional object-oriented approach. The different models of this client application can be seen in Section III.3.1. Few specifications for aspects support must however be explained:

• Event handling aspect: *defining component interactability* is achieved in Java by adding a listener to the wanted component. Once this listener will have been added to the component, an event will be created when this component is interacted. So, in the components design, the listeners adding must be totally not taken into account. *Linking interaction events to the appropriate methods* can be done in Java with some objects that receive

[25] As explained in Section III.2.3.1, *required* aspects need to be woven to components code to have a executable system., in distinction with *optional* aspects, which are not necessary to have an executable system.

[26] As explained in Section III.2.3.1, *plugging constraints* are used to express plugging dependencies between different aspects, such as *necessary aspects* to specify other aspects that are necessary in order to plug the current aspect, or *incompatible aspects* to define a group of aspects in which only one can be plugged to the system (like a set of options where only one can be chosen).

all the application events, test their origin and call the appropriate method. The components design must not take into account such components.

- Dynamic Node Loading: in the conventional approach, the `getNode(String name)` of the `CacheContext` class is used to look for a node. This method delegates the search to a `NodesManager` object that looks for the node in the cache context and if it does not find it, creates a new one. For the aspect oriented approach, we need to define these points differently since everything will be defined in the aspect. So there will be no more `NodesManager` class, and no more `getNode(String name)` in the `Context` class. We will just add a instance variable `targetNode` to the `Transformation` class. This variable will be used each time the target node must be called. The *Dynamic Node Loading* aspect will be responsible for what holds this variable. The Figure 30 presents the differences with the conventional model presented in Figure 23.

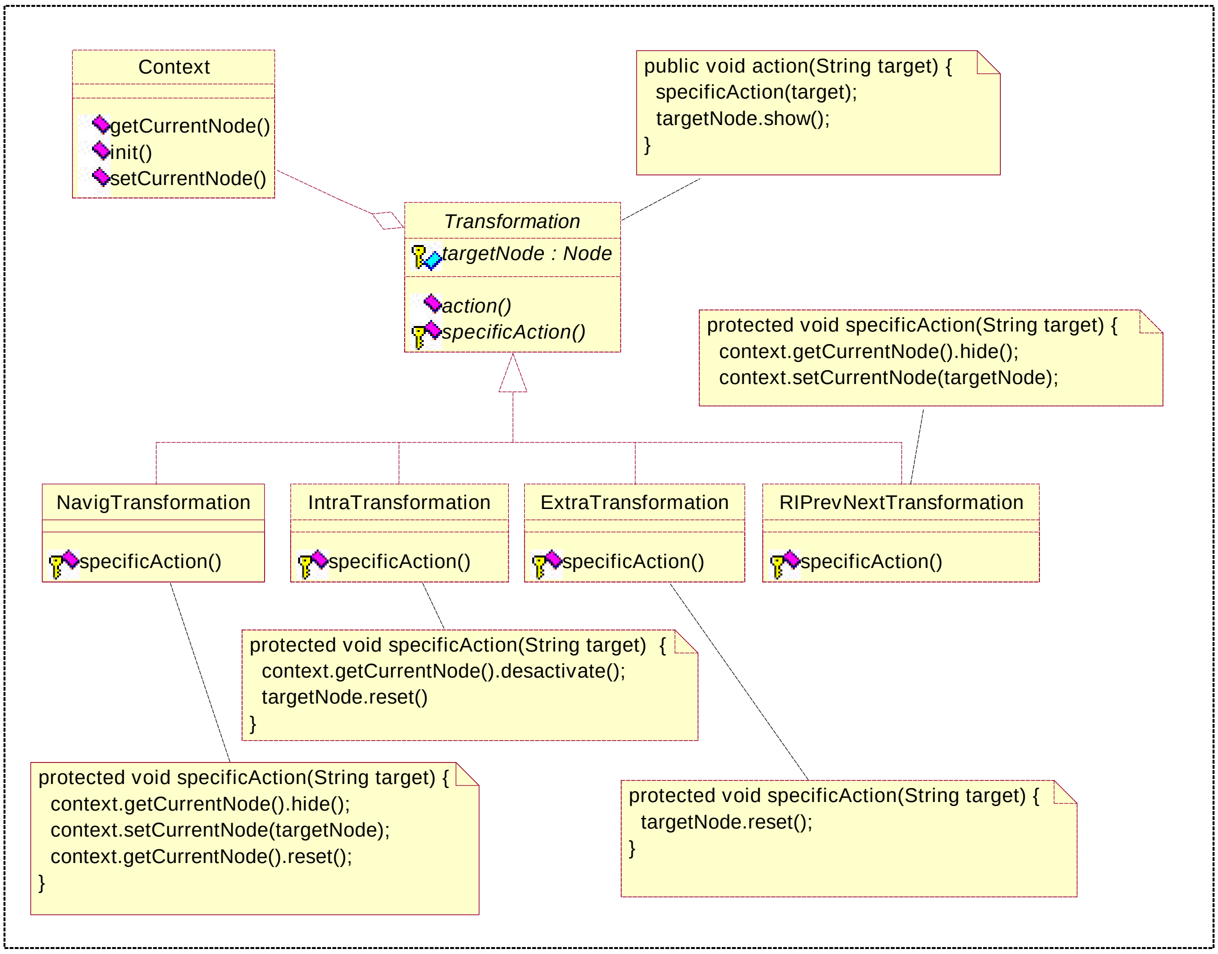

Context
getCurrentNode()
init()
setCurrentNode()
public void action(String target) {
specificAction(target);
targetNode.show();
}
Transformation
targetNode : Node
action()
specificAction()
protected void specificAction(String target) {
context.getCurrentNode().hide();
context.setCurrentNode(targetNode);
NavigTransformation
specificAction()
IntraTransformation
specificAction()
ExtraTransformation
specificAction()
RIPrevNextTransformation
specificAction()
protected void specificAction(String target) {
context.getCurrentNode().desactivate();
targetNode.reset()
}
protected void specificAction(String target) {
context.getCurrentNode().hide();
context.setCurrentNode(targetNode);
context.getCurrentNode().reset();
}
protected void specificAction(String target) {
targetNode.reset();
}

Figure 30 - Components involved in the Dynamic Node Loading aspect

⊗ POINT III - ASPECTS CHARACTERIZATION

⊕ *Step III-1: Aspect description applied to components*

⇒ *Events handling:* each method where a component that can be interacted is created and added to a node (or a decorator) will be wrapped to also add a listener to this component. This listener will define that when the component is interacted, a method of the aspect must be called with the created event. This aspect method will make the dispatch to the appropriate node (or decorator) method. For length reasons, we will define one aspect for each node (or decorator) events handling. We only describe the aspect for the `RINode3` class, for the other classes (`RIDecorator0`, `RINode1`, `RINode211`, `RINode212`, `RINode221`, `RINode222`, `RINode1C`, `RINode3D`, `RIHelp`) the decomposition principles are the same.

⇒ *Dynamic Node Load:* each method where the `targetNode` variable of the `Transformation` class is used will be wrapped so that this variable refers the correct node. This reference definition will be done through two methods, one for searching nodes in the context and one for creating non-existing nodes.

⊕ ***Step III-2: Involved components***

⇒ *Events Handling RINode3:*
⇒ `RINode3` (for listeners adding and for events/method linking)

⇒ *Dynamic Node Load:*
⇒ `Context` (for nodes search)
⇒ `Transformation` (for `targetNode` access)
⇒ `NavigTransformation` (for `targetNode` access)
⇒ `IntraTransformation` (for `targetNode` access)
⇒ `ExtraTransformation` (for `targetNode` access)
⇒ `RIPrevNextTransformation` (for `targetNode` access)

⊕ ***Step III-3: Join points definition***

⇒ *Events Handling RINode3:*
⇒ `RINode3.new()` – after wrapping

⇒ *Dynamic Node Load join points:*
⇒ `Node Context.getNode(String name)` – method adding
⇒ `void Transformation.action(String name)` – before wrapping
⇒ `void NavigTransformation.specifcAction(String name)` – before wrapping
⇒ `void IntraTransformation.specificAction(String name)` – before wrapping
⇒ `void ExtraTransformation.specificAction(String name)` – before wrapping
⇒ `void RIPrevNextTransformation.specificAction(String name)` – before wrapping

⊕ ***Join points representation***

`RINode3`	`new()`	*For adding listeners to the "interactable"components created in this constructor*
`Transformation`	`void action(String name)`	*For defining the* `targetNode` *variable accessed in this method*
`NavigTransformation`	`void specificAction(String name)`	*Idem*
`IntraTransformation`	`void specificAction(String name)`	*Idem*
`ExtraTransformation`	`void specificAction(String name)`	*Idem*
`RIPrevNextTransformation`	`void specificAction(String name)`	*Idem*

⊗ **POINT IV – ASPECTS MEMBERS DEFINITION**

⊕ ***Step IV-1: Aspect members identification***

⇒ *Events Handling RINode3:*
- ⇒ *Adding listeners member:*
  - ⇒ *Join points:* `RINode3.new(..)`
  - ⇒ *Type:* after wrapping
  - ⇒ *Description:* add the appropriate listener to all the components of this node that have to be interacted. This listener calls the methods `dispatch(Event evt)` of the aspect when reacting.

⇒ *Dynamic Node Load:*
- ⇒ *Target node access member:*
  - ⇒ *Join points:* `protected !abstract void *.specificAction(..), public void Transformation.action(..)`
  - ⇒ *Type:* before wrapping
  - ⇒ *Description:* call the aspect method to get a node out of its name
- ⇒ *Cache searching member:*
  - ⇒ *Join points:* Node `Concept.getNode(String name)`
  - ⇒ *Type:* method adding
  - ⇒ *Description:* in this method the three vectors (`navigNodes`, `extraNodes` and `intraNodes`) of the context are browsed to look for a node having the given name. The node is returned if found, `null` if not.

⊕ ***Step IV-1: Aspectual members***

⇒ *Events handling RINode3:*
- ⇒ *Events dispatching member:*
  - ⇒ *Identification:* `boolean Events_Handling_RINode3.dispatch(Event evt)`
  - ⇒ *Type:* aspect method
  - ⇒ *Description:* call the appropriate method of the class `RINode3` depending on the origin component of the event `evt`.

⇒ *Dynamic Node Load:*
- ⇒ *Node creation member:*
  - ⇒ *Identification:* `Node Dynamic_Node_Load.createNode(String name)`
  - ⇒ *Type:* aspect method
  - ⇒ *Description:* create a new node out of its name
- ⇒ *Searching Node member:*
  - ⇒ *Identification:* `Node Dynamic_Node_Load.getNode(String name)`
  - ⇒ *Type:* aspect method
  - ⇒ *Description:* call the `getNode` method of the context. The node is returned if found, and if not the result of the `createNode` method of the aspect will be returned.

⊕ ***Aspectual Members Representation***

`Events_Handling_RINode3`	`boolean dispatch(Event evt)`	*Dispatch an event to the appropriate method of the node 3*
`Dynamic_Node_Load`	`Node createNode(String name)`	*Create a new node out of its name*
	`Node getNode(String name)`	*Call the* `getNode` *method of the context. If the node is not found, call the* `createNode` *method*

⊕ ***Components/Aspects Interaction Representation***

		User Interaction		Navigation				
		Events Handling Node3		Dynamic Node Load				
		Adding listeners	Event dispatching	TargetNode access	Cache searching	Node creation	Searching node	]
		After wrapping	Aspect method	Before wrapping	Method adding	Aspect method	Aspect method	]
`RINode3`	`new()`	■						
`Context`	`getNode()`				■			
`Transformation`	`action()`			■				
`Navig Transformation`	`specificAction()`			■				
`Intra Transformation`	`specificAction()`			■				
`Extra Transformation`	`specificAction()`			■				
`RIPrevNext Transformation`	`specificAction()`			■				
Events Handling Node3	`dispatch()`		■					
Dynamic Node Load	`getNode()`						■	
	`createNode()`					■		

III.4.3) Client/Server communication

In the previous section we have described in details the aspects/components decomposition steps. For the following sections we will only present the main results of our step by step decomposition, highlighting only some important details.

⊗ **POINT I – ASPECTS IDENTIFICATION**

Client/server communication is based in RECINTERNET on Java RMI. The unique feature of this part of the system is *remote calls.* We consider it as an aspects area because it is a typical case of cross-cutting concern (components of the server and of the client).

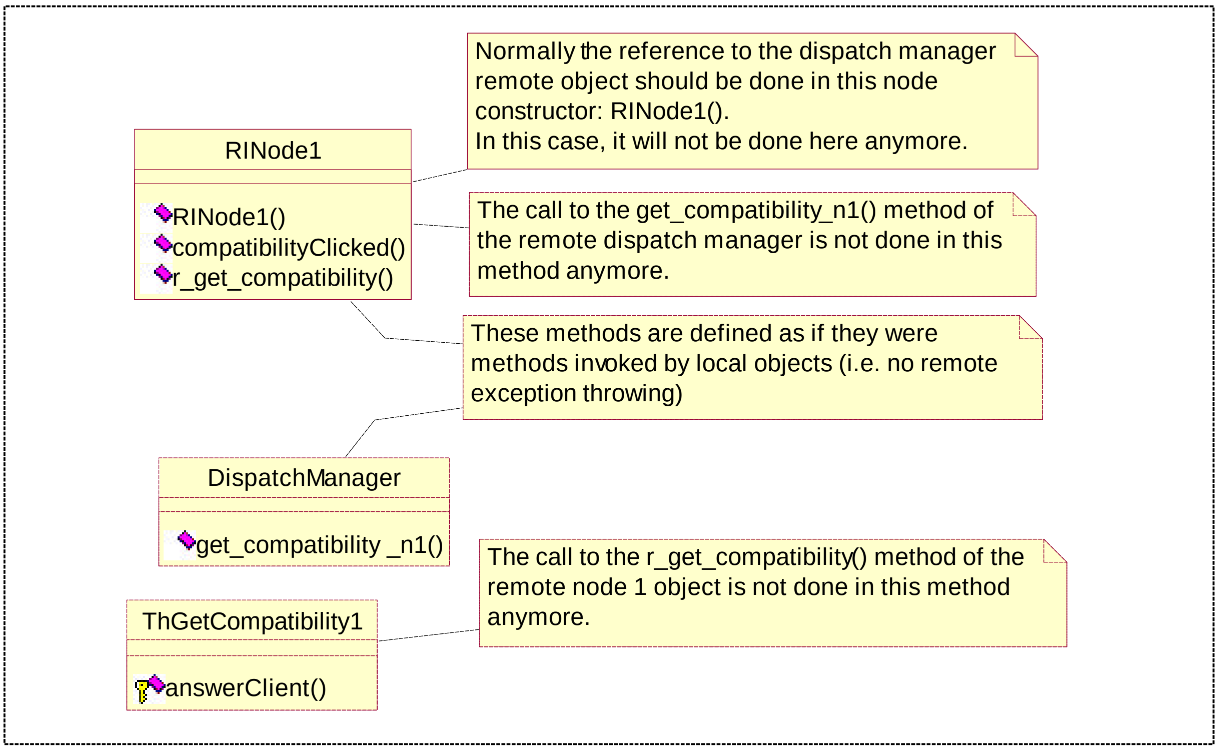
We identified one aspect in this area. We called it the **distribution** aspect. It deals with all the modifications due to the difference between a local method call from a remote method call. We used a specific feature for this aspect, inspired from the RIDL language of [LK97]. The idea is to pass as parameters to remote methods only parts of an object and not all the object. This enables to not have to pass entire objects (that can contain unnecessary variables) in remote methods call. This aspect is a very wide one since it involves all the communications between all the nodes (or decorators) and the server. Then we only present the mechanisms involved in the concrete case where the user click on the *compatibility* button in Node 1. These mechanisms can be applied to all the interactions requiring a communication with the server in the same way.

	Remote Calls
	Aspects related to method calls on remote objects
	Distribution
	Express all the modifications due to the fact that methods are called remotely and specify which objects parts will be passed to remote methods
	Required
	No

⊗ **POINT II – COMPONENTS DESIGN**

We use the same design for the client/server communication than in the conventional object-oriented approach. The different models of this client application were presented in Section III.3.1. However there are some modifications due to the *Distribution* aspect.

No remote objects must be defined in any node class of the client or in any client thread class of the server. No remote call must be written in any method. No specific syntax (as declaring that a method throws some remote exception, or as catching such exceptions) must be added to any method. To illustrates these directives, we present in Figure 31 the involved methods of the classes concerned by the *compatibility clicked* interaction.



Normally the reference to the dispatch manager remote object should be done in this node constructor: RINode1().
In this case, it will not be done here anymore.

The call to the get_compatibility_n1() method of the remote dispatch manager is not done in this method anymore.

These methods are defined as if they were methods invoked by local objects (i.e. no remote exception throwing)

The call to the r_get_compatibility() method of the remote node 1 object is not done in this method anymore.

Figure 31 - Components involved in the Distribution aspect

Java RMI interfaces are defined as components, but they are defined with no methods. For all the remote calls, the parameters passed will be the entire objects, such as a node or a context, even if there are many informations in these objects that are not useful for the remote call. In the case of the *compatibility* interaction, the parameters passed to the remote call will be only the context object (this object contains references to any nodes where some information could be needed for this interaction).

⊕ ***Components/Aspects Interaction Representation***

		Remote calls			▨
		Distribution (1/2)			▨
		Ref. Creation	Add do-request Call	Add Answer Call	▨
		The definition of the reference to the remote dispatch manager is added	*Add the call the aspectual method for "preparing" the parameters (*`do-request`*)*	*The effective call of the remote method of the node 1 object is added*	▨
		No	No	No	▨
▨	▨	After wrapping	After wrapping	After wrapping	M
`RINode1`	`new()`	▨			
	`compatibility_clicked()`		▨		
`ThGet Compatibility1`	`answer_client()`			▨	

		Remote calls			▨
		Distribution (2/2)			▨
		Adding do-request	Remote answer method	Remote request method	▨
		Extract the necessary elements (access and selected series) from the parameters object (node1 and context) and call the remote method (`compatibility_request`*) only with these parameters.*	*The method receiving remote answers is added. The answer is just passed to another method of the node (*`r_get_compatibility`*).*	*The method receiving remote requests is added. The request parameters are just passed to another method (*`get_compatibility_n1`*).*	▨
		No	No	No	▨
▨	▨	Aspect method	Method adding	Method adding	M
`I_Dispatch Manager`	`compatibility_request()`			▨	
`I_RINode1`	`compatibility_answer`		▨		
`Distribution`	`do_request`	▨			

III.4.4) Server

⊗ **POINT I – ASPECTS IDENTIFICATION**

Many features could be expressed with aspects in a server application, such as connections control, multi-threading concurrency control. We decided to focus specially on the server policies. This area includes the different strategies that can be used in the server to “control” the client connections. We designed four aspects in this area:

The **displaying server messages** aspect groups all the modifications required to display in the client application messages coming from the server.

The **time bound** aspect includes all the modifications to add a time boundary to client request satisfaction. Basically it means that once a client has connect the server to submit a request, if the server cannot answer before a given time threshold, a message will be return to the client to tell him that his request was taking too much time on the server.

The **size bound** aspect limits the results answer addressed to clients in size. If this size is over a given threshold, a message will be return to the client to tell him that the result size is too high.

The **connection bound** aspect is used to define a maximum number of client threads running on the server. Passed this limit, new client requests are not allowed and a message will be return to the client to tell him that the server is busy.

A	Connection Control			
	Aspects used to express different connection control policies of the server			
	Displaying Server Messages	Time Bound	Size Bound	Connection Bound
	Modifications required to display on client application some messages coming from the server.	*Express all the modifications needed to include a time threshold to satisfy client requests before an error message is returned.*	*Express all the modifications needed to include a size threshold for the results answer to the client. An error message is returned if the threshold is overcome..*	*Express all the modifications needed to define a maximum number of client threads running on the server. If this number is over, an error message is returned to each new client request.*
A	Optional	Optional	Optional	Optional
C	No	Displaying Server Messages aspect required	Displaying Server Messages aspect required	Displaying Server Messages aspect required

⊗ **POINT II – COMPONENTS DESIGN**

We use the same components design for the server application than in the conventional object-oriented approach. This design was explained in Section III.3.2.2.

The client application design differs lightly from the conventional object-oriented approach. In fact a client thread of the server calls a client object method to send him the results of its request. For the aspect-oriented approach we add a parameter to all these methods that are called by the client threads of the server to send back results to clients object. This parameter is a `String` which is by default empty. This `String` parameter has to be added to all the methods used to receive answers in the client application[27], which are:

- `RIDecorator0.r_new_user_init_n1(..)`
- `RIDecorator0.r_init_n1(..)`
- `RINode1.r_get_Series(..)`
- `RINode1.r_get_compatibility(..)`
- `RINode211.r_get_elements(..)`
- `RINode212.r_get_elements(..)`
- `RINode212.r_init_n3(..)`
- `RINode222.r_get_elements(..)`
- `RINode222.r_init_n3(..)`
- `RINode3.r_init_n3d(..)`

For example the new signature of the method `r_get_compatibility` of the class `RINode1` used to receive the answer to the request to get compatibility between series is:

```
public void r_get_compatibility(String[][] compatibility, String message);
```

⊕ ***Components/Aspects Interaction Representation***

		Connections Control		
		Displaying Server Messages		
		Test for Showing Messages	Showing Messages	
		A test is done to see if the message parameter of the method is empty. If so, the normal body is executed. If not, there is a call to the `show_message` *method of this aspect, and the end of the method is not executed.*	*This method shows a string to the user in a new frame.*	
		No	No	
Component (and Aspect)		Before wrapping	Aspect Method	
`RIDecorator0`	`r_new_user_init_n1()`	▨		
	`r_init_n1()`	▨		
`RINode1`	`r_get_series()`	▨		
	`r_get_compatibility()`	▨		
`RINode211`	`r_get_elements()`	▨		
`RINode212`	`r_get_elements()`	▨		
	`r_init_n3()`	▨		
`RINode222`	`r_get_elements()`	▨		
	`r_init_n3()`	▨		
`RINode3`	`r_init_n3d()`	▨		
`Displaying Server Messages`	`show_message(String)`		▨	

[27] It also means adding this `String` parameter in the RMI interfaces definition. This must be carefully check in the Distribution aspect.

		Connections Control			▨
		Size Bound			▨
		Size Limit Constant	Checking result size	Results size check behavior	▨
		Adding the size limit constant that will be used as a parameters in client threads answers to client objects	*This method tests if the size of the* `ResultSet` *results is more than a given threshold.*	*This member call the result size test, and if the results is to large, answer the client with the special parameter* `SIZE_LIMIT`	▨
		No	No	Yes[28]	▨
▨	▨	Variable adding	Method adding	Before wrapping	▨
`ClientThread SQL`	`SIZE_LIMIT`[29]	■			
	`check_Size_ Limit()`		■		
`Panel`	`SIZE_LIMIT`	■			
`ThNewUser`	`answerClient()`			■	
`ThInit1`	`answerClient()`			■	
`ThGetSeries1`	`answerClient()`			■	
`ThGet Compatibility1`	`answerClient()`			■	
`ThGet Elements211`	`answerClient()`			■	
`ThGet Elements212`	`answerClient()`			■	
`ThGet Elements222`	`answerClient()`			■	
`ThGetDetails3`	`answerClient()`			■	

[28] The generic join point expression is: `* !abstract answerClient(..)`

[29] The entire signature of this variable is in Java: `static final String SIZE_LIMIT = “Size Limit”`. It is also valid for the size limit variable added to the `Panel` class.

		Connections Control			
		Time Bound (1/2)			
		Time Limit Constant	Answer Synchronizer	Locking Answer Synchronizer	Destroying Lasting Threads
		Adding the time limit constant that will be used as a parameters in client threads answers to client objects	*This variable is used to synchronize the access to the* `answerClient` *method by a thread or its corresponding count down thread*	*This method is synchronized. If the* `sync_answer` *variable is false, then it is changed to true. Else the calling thread is put to wait.*	*This synchronized method destroys the corresponding client thread and the count down thread.*
		No	No	No	No
		Variable adding	Variable adding	Method adding	Method adding
`ClientThread`	`TIME_LIMIT`[30]	■			
	`sync_answer`[31]		■		
	`before_Answer()`			■	
	`after_Answer()`				■
`Panel`	`TIME_LIMIT`	■			

		Connections Control		
		Time Bound (2/2)		
		Calling `before_answer`	Calling `after_answer`	Starting Count Down
		The method `after_asnwer` *is called.*	*The method* `after_asnwer` *is called.*	*Inner class* `CountDown` *definition. This inner class is instantiated and started.*
		`* !abstract answerClient(..)`	`* !abstract answerClient(..)`	No
		Before wrapping	After wrapping	Before wrapping
`ThNewUser`	`answerClient()`	■	■	
`ThInit1`	`answerClient()`	■	■	
`ThGetSeries1`	`answerClient()`	■	■	
`ThGet Compatibility1`	`answerClient()`	■	■	
`ThGet Elements211`	`answerClient()`	■	■	
`ThGet Elements212`	`answerClient()`	■	■	
`ThGet Elements222`	`answerClient()`	■	■	
`ThGetDetails3`	`answerClient()`	■	■	
`ClientThread`	`start()`			■

[30] The entire signature of this variable is in Java: `static final String TIME_LIMIT = "Time Limit"`. This is also the signature for the time limit variable added to the `Panel` class.

[31] The entire signature of this variable is in Java: `protected boolean sync_answer = false`.

- ***Remark about the inner class*** `CountDown`:

This inner class extends the Java `Thread` class. The idea is to have a thread running in parallel of each client thread which will limit this client thread maximum execution time. This can avoid server CPU overflow by limiting threads taking too much time to complete their tasks. The `CountDown` class is instantiated when the start method of a client thread begins. The resulting thread is also started at this moment. This thread will wait a given time (a constant of the class `CountDown`) and then call the `answerClient` method of its corresponding client thread, with a special parameter (`TIME_LIMIT`) which indicates that the maximum time before the server answers is over. If this method call is executed before the normal one happening when a normal answer is sent to the client, the client will know that the time limit is over, and the client thread of the count down object will be destroyed, letting free the resources for the other client threads.

		Connections Control			
		Connection Bound (1/2)			
		Clients Count	Reference Dispatch Manager	Initialization of the Dispatch Manager Reference	
		This integer represents the number of client threads running currently on the server.	*This public static variable holds the reference to the dispatch manager*	*The static variable of the class `ClientThread` is assigned to the new dispatch manager object.*	
		No	No	No	
		Variable adding	Variable adding	After wrapping	Member Type
`DispatchManager`	`clientsCount`	▨			
	`new()`			▨	
`ClientThread`	`dspchManager`		▨		

		Connections Control			
		Connection Bound (2/2)			
		Count Increment	Count Decrement	Server Busy Constant	
		In a synchronized block (on `clientsCount`), if a given threshold is over, call the `answerClient` method with the `SERVER_BUSY` parameter. If not, increment `clientsCount`.	*The method will be a synchronized block (on `clientsCount`) where this variable is decreased.*	*Adding the Server Busy constant that will be used as a parameters in client threads answers to client objects*	
		No	No	No	
		Before wrapping	Method adding	Variable adding	
`ClientThread`	`start()`	▨			
	`destroy()`		▨		
	`SERVER_BUSY`			▨	
`Panel`	`SERVER_BUSY`			▨	

III.4.5) *Database*

⊗ **POINT I – ASPECTS IDENTIFICATION**

The main feature that can be addressed in the database access part of RECINTERNET is the specificity of the databases. We can then define an aspect area called *Database Specificity* that groups the aspects that deals with concerns including issues specific to the kind of database used.

In this aspects area we have decided to address two aspects. One is the **Structural Database Connection** aspect, which deals with opening and closing the connection with the structural database. The other one is the **SINTESE Database Connection** aspect, which deals with opening and closing the connection with the SINTESE database.

A	Database Specificity *Aspects used to express concerns involving the specificity of the databases used*	
	Structural Database Connection *All the specific actions for opening and closing a connection with the structural database.*	SINTESE Database Connection *All the specific actions for opening and closing a connection with the SINTESE database.*
A	Required	Required
Plugging Constraints	No	No

⊗ **POINT II – COMPONENTS DESIGN**

We use the same idea than in the conventional object-oriented approach: having a database manager object responsible for proceeding the requests sentences (class `DBManager`). We will also have two subclasses of this class, one for each database (`RIDBManagerSQL` and `RIDBManagerSINTESE`). The concurrent access to the databases by clients threads will be managed by synchronizing the methods `proceedQuery` of the two sub-classes. In all these classes nothing related with opening or closing connections will be defined. The Figure 32 shows these classes definition.

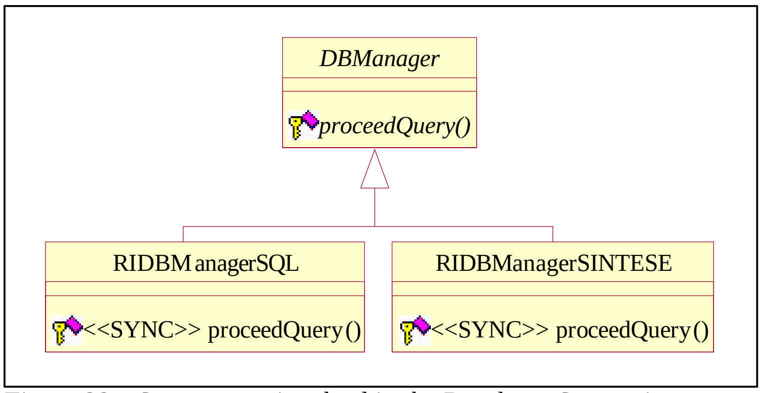


Figure 32 - Components involved in the Database Connection aspects

⊕ ***Components/Aspects Interaction Representation***

		Database Connection				
		Structural Database Connection (1/2)				
		Database URL Variable	Driver Variable	Open Connection	Close Connection	
		This variable will hold the address of the structural database.	*This variable will hold the name of the driver class to use with the structural database.*	*In this method are realized all the operations for the connection.*	*In this method are realized all the operations for closing the connection.*	
		No	No	No	No	
		Variable adding	Variable adding	Method adding	Method adding	
RIDBManagerSQL	`DatabaseURL`	▨				
	`DriverName`		▨			
	`open_connection()`			▨		
	`close_connection()`				▨	

		Database Connection				
		Structural Database Connection (2/2)				
		Calling Open Connection	Calling Close Connection	Login Variable	Password Variable	
		A call to the `open_ connection` *method is added.*	*A call to the* `close_ Connection` *method is added.*	*The login variable used for the connection is added.*	*The password variable used for the connection is added.*	
		No	No	No	No	
		After wrapping	Before wrapping	Variable adding	Variable adding	
RIDBManagerSQL	`new(..)`	▨				
	`login`			▨		
	`password`				▨	
`Server`[32]	`stop(..)`[33]		▨			

We will not represent the components/aspects interaction for the **SINTESE Database Connection** aspect since the same aspect members are used than in the **Structural Database Connection** aspect. The body of the different methods are changing, but the aspects design table is the same. It has to be noticed that the *Driver Variable* aspect member is not used any more for this aspect since it is specific to the connection to the structural database.

[32] This class is in fact the class that is responsible for starting and stopping the server.
[33] This method is in fact the method where the server will be stopped.

III.4.6) Conclusions

In this section we have explained our aspects/components decomposition in the RECINTERNET design. In this aspect-oriented approach, components design is represented through conventional object-oriented diagrams (such as UML class diagrams), and aspects design as well as aspects/components interactions are represented with aspects design tables. These different models and tables provide then a good support for the RECINTERNET implementation.

It has to be noticed that in this aspect-oriented design of RECINTERNET, most parts of the components design are similar to the conventional object-oriented design presented in Section III.3. We have just extracted from the components level the different features related to concerns that can be beneficially expressed in aspects. We can see the aspect-oriented approach as a two-dimensions decomposition, with one dimension for the components definition and one for the aspects definition. With aspects design tables we address particularly these two dimensions representation and structuration as well as the interactions between them. In fact we believe that using this two-dimension decomposition is an appropriate way to view aspects design.

An important remark is that our aspects definition is quite dependent from the current AspectJ possibilities, and consequently from Java. This provides a close support to aspects implementation, but can be a limit to this design reuse with new versions of AspectJ, or even for design reuse with other aspect languages and other programming languages. During this design, we have been also bound by a specific characteristic of AspectJ. When several aspects are woven to components code in a same weaving, it is not possible to specify the order in which the aspects are taken to apply their modifications to the code. For instance, for two aspects realizing both an after-wrapping modification on the same method, the order of these two aspects weaving can be very important. A solution is to realize two weavings, with only one aspect at each time, or to detect the possible conflict and solve it in another way. It has to be noticed that for this second solution the design aspects table helped us consequently.

An important improvement in using aspects in web-based three-tiers applications design is that it is possible to express in aspects some functionalities (as seen with the different aspects used in the RECINTERNET server application) that will be simply added to the application by plugging in the corresponding aspect to the components code. This is a specific feature of aspect-oriented programming that plays an important role in application design.

For this aspect-oriented design we had chosen only to develop few aspects in each part of the application. Anyway there were several conflicts between the different aspects and components. Using aspects design tables helped us to structure this design and to track some of the conflicts. In this section we only presented the result of our design, but of course, this had been achieved with many modifications and after many errors.

III.5) Comparison between object-oriented and aspect-oriented designs

In this section we draw a comparison of the conventional object-oriented design of RECINTERNET presented in Section III.3 and the aspect-oriented design of the same application presented in Section III.4. This comparison is organized into distinct points: first we look at the two approaches in terms of design process, that is to say how is realized the design. Then we highlight the separation and relationships between components and concerns, before comparing the way these two designs support implementation. We finish this comparison with some specific points for the two approaches. In all these sections we will highlight some important points that were revealed during the design realized for the two approaches.

III.5.1) Design process

Object-oriented design methodologies have been and still are an important field of software engineering research. This field as been largely exploited and maturated. There are many examples of object oriented real-world systems design, using different methodologies and representations to achieve efficient design models. Many tools, representation and techniques are available for the different stages of object-oriented design.

In the case of RECINTERNET, we used for the client application the OOHDM methodology [SRB96], [SR98], some design patterns [GHJV94], [LRS98], [RSG97], and some existing way of programming object-oriented systems proven as good and reusable solutions (three-tiers architectures, distributed programming with object-oriented interfaces, etc...). We could use some well-defined design language (UML[34]), as well as efficient tools supporting design and automatic code generation (Rational Rose[35]). So there was a wide range of technologies and techniques usable for a conventional object-oriented design.

AOP [Aop] is an emerging programming paradigm, and then is not yet well structured and supported as the object-oriented one. Frameworks to support aspects implementation and weaving are often developed just as experiments (such as [Kai98] or [DVDH99])and are not appropriate for real-world applications development. AspectJ is one of the most developed tool for implementing aspects, but it is still available as a beta version evolving permanently, and AspectJ users for large applications are considered as software engineering pioneers! As AOP is basically related to components implementation, designing with aspects is still a very broad research area, where methodologies, representations and tools are missing. In comparison with the numerous existing object-oriented design supports, the aspect-oriented design support possibilities were quite not existing, and we had to create our own way to design and to support our design. In this context we developed an original approach to aspects design illustrated with this concrete aspect-oriented design.

We developed our step by step decomposition and the aspects design table in order to provide the appropriate support for implementation. With this quite basic technique we provided an appropriate support for RECINTERNET implementation. Finally, an important difference between the object-oriented and the aspect-oriented design approaches used was the difference of available technologies for design support, that we tried to compensate by creating new support for aspect-oriented design.

Another important point of the design process for the comparison of these two approaches is the way to design specific concerns cross-cutting the basic functionality components structure. In the object-oriented approach there was no possibility to use any abstraction to express cross-cutting concerns such as events handling for instance. We were then obliged to design the different components involved by this concerns in respect to it, inserting methods or variable that will be used for this concern. It was particularly complex when different concerns were involving same components, because these components had to be designed to support all these different concerns in the same time.

The main advantage of AOP is to be able to express cross-cutting concerns in separated modules. By only identifying join points with components, it is possible to cleanly express complex concerns in distinct aspects separated from components and from other aspects. This has been a great advantage in the RECINTERNET design compared to the object-oriented approach. And this modifies consequently the design process. It is

[34] UML Resource Center. Home Page: http://www.rational.com/uml
[35] Rational Rose 98. Rose Enterprise Evaluation Edition. See Rose Home Page: http://www.rational.com/rose

possible to concentrate on complex concerns one by one, and almost independently from other ones. Anyway aspects composition as to be carefully foreseen at design level. It was possible with the aspects design tables, that clearly identify the aspects/components interactions and enable to efficiently track conflicts between different aspects. In this way the design process in the aspect-oriented approach was clearly more structured than in the object-oriented approach.

Finally we had less support for the aspect-oriented design than for the object-oriented one, but because of the possibility of separating concerns with AOP and to structure components/aspects decomposition with aspects design table the aspect-oriented approach provided an efficient and more structured design.

III.5.2) Components/Concerns

As we have briefly introduced it in the design processes comparison, an important point of web-based three-tiers applications design is the separation and the relationships between components (conventional object-oriented entities) and concerns (functional or non-functional ones, such as distribution, events handling, server functionalities, etc...).

In conventional object-oriented approaches, such concerns are flatten into components structure. As a consequence concerns are scattered throughout components code, and moreover we have all concerns and components expressed in the same code. In applications such as RECINTERNET it can lead to complex design and complex implementation. At design level it is then difficult given a component to recognize which parts of it are associated to which concern. In the other way it is also difficult to find out all the components associated to a given concern.

On the other hand, the aspect-oriented approach can be seen as adding a dimension in the design process: we will have the components dimension (as in the conventional object-oriented approach) and the aspects dimension. Links between the entities of these two dimensions are expressed with join points identification. The design process is then simplified because it is possible to abstract some complex concerns from the components dimension by expressing them in the aspects dimension. We then have a clear separation between concerns definition and their related components, and a localized definitions of the links between them in join points. Once again, the different aspects design tables shown in Section III.4 provide the appropriate support for representing this aspects/components separation and relationships.

In the classical aspect-oriented approach, join points are defined in order to link aspects to components. It is then possible to track the different components involved in each aspect. With aspects design tables we go one step further since we provide a way to concretely visualize aspects/components interaction in two ways: given a component, it is possible to know in which concerns it is involved, and given an aspect it is possible to know which components it involves.

To put it in a nutshell, the aspect-oriented approach support a clear separation of concerns which is not possible with conventional object-oriented approaches. This is a consequent improvement in software design process, and particularly in complex applications involving several complex concerns such as web-based three-tiers applications and particularly RECINTERNET. Moreover with aspects design tables, separation of concerns can be cleanly organized and visualized. The different aspects/components interactions can also be simply developed and visually addressed in these aspects design tables. For these reasons the aspect-oriented design can be considered as more efficient than object-oriented design for systems involving complex concerns.

III.5.3) Implementation support

Some tools exist for linking conventional object-oriented design to implementation automatically. For instance with Rational Rose it is possible to automatically generate code (for Java, CORBA, C++...) out of design models. This a great advantage in the conventional object-oriented approach since it simplifies consequently the implementation stage, and it enables to have an implementation respecting closely the design.

With the aspect-oriented approach, we can use the same techniques to link the components design to their implementation. Anyway there is no equivalent technology or tools for linking aspects design to their implementation. The entire implementation of aspects must be done manually. As AOP is an emerging technique, there is not appropriate support for such automatic linking. This is an important drawback when comparing the conventional object-oriented and the aspect-oriented designs.

On the other hand, as we decided to use AspectJ, we had to design our aspects concretely in function of the possibilities of AspectJ, and to define their interaction with components through join points described in Java. For this reason our aspects design depends on the programming language used for join points, but also provides a closer support for implementation, as it can be seen in Section IV.3.

The decomposition principle of the aspect-oriented approach involves a modularization of the implementation code (distinct modules for aspects and components, and distinct modules for different aspects). As a result the number of modules used in the aspect-oriented approach is higher than in the conventional object-oriented case. In the case of the RECINTERNET design, the Table 7 shows the number of modules (classes or classes and aspects) for the two approaches. Even if the way large aspects are split into several smaller ones modifies consequently such numbers, this table illustrates the tendency of modules number increasing with aspect-oriented design.

	[illegible]	[illegible]	[illegible]
C	35 (36)	34+10 = 44 (37)	29%
C	7	7+1 = 8	14%
S	16	16+6 = 22	38%
T	[illegible]	[illegible]	[illegible]

Table 7 - Modules number in RECINTERNET design

In counterpart to this modules number increasing with the aspect-oriented approach, there is a consequent decreasing of classes size, and moreover of the total size of the application. Even if it is not possible to measure it at design level, we can explain it simply. In comparison to conventional object-oriented design, the aspect-oriented approach, features of specific concerns are “extracted” from the components code to be put in aspects. As a first result we then have a decreasing of the components size. Moreover, because it is possible to specify for one aspect member a generic join point referencing several members of components, same modifications in different components can be expressed only once. So finally the total size of the implemented application will be decreased with the aspect-oriented approach.

With these advantages of the aspect-oriented approach comes unfortunately an additional problem: composition conflicts. These conflicts happen in the aspects weaving after implementation of aspects and components. They can be very complex and difficult to detect. The aspects design table provides good ways to track and detect such conflicts because of its clear representation of components/aspects interaction, but also common components (through their join points) involved in distinct aspects. When defining the RECINTERNET design with the aspect-oriented approach, we represented aspects with the aspects design table and detected some basic composition conflicts. Anyway there can be subtle composition conflicts that will only be detected when evaluating the application. In this case the aspect-oriented approach creates with composition conflicts new kind of problems (new compared to the object-oriented approach) that cannot always be solved at design level.

Finally we have seen that the object-oriented design provides a good support for implementation. The aspect-oriented one is not as well supported and there is not appropriate design/implementation links. Anyway even if new problems arise because of aspects/components decomposition (composition conflicts) design, modularization and size decreasing are a concrete amelioration for the implementation of an aspect-oriented design of a system.

[36] Number of classes.
[37] Number of classes + number of aspects.

III.5.4) Specific Points

- ***Evolution***

Systems designed with conventional object-oriented techniques are difficult to evolve because several concerns can be involved in a single component. For instance if we have several complex concerns all involving an important class of a system. If this system has to evolve by sub-classing this important class for adding new functionality for example, the fact to have several concerns involved can make this evolution design very complex (we can think of some synchronization of methods in the important class that will have to be carefully re-done with the methods of the sub-class).

With the aspect-oriented design, as we have a clear separation between components and concerns, it is easier to make them evolve. Particularly it is simple to make evolve features of a concern (expressed in an aspect) without modifying at all any components design nor implementation. However for components evolution, involved aspects has to been tracked in order to see if they should be modified also or not. This tracking is particularly improved with aspects design table, where we can find easily all the aspects involving a given component. As an example of system evolution, we can imagine that one day the RECINTERNET system will use another kind of structural database (than *Interbase)* compliant with JDBC. This implies a modification of the different variables (driver name, database URL...) used for the database connection. With the aspect-oriented approach, this can be done simply by modifying the appropriate database connection aspect, but without modifying any components.

For the same example with the object-oriented design we would have to create a new subclass reusing some methods of the initial classes and overriding other ones. Modifications due to this evolution would then be scattered through different methods of this new class and we would have a complicated collaboration between redefined methods and methods of initial classes.

- ***Reuse***

Conventional object-oriented design is sometimes difficult for the same results: several concerns can be mixed in the same component. This is a concrete handicap for system understanding as well as system extending or modifying. With the aspect-oriented design, understandability is increased thanks to clean modularization, and simplification of the different modules. This a first advantage for reuse.

Aspects reuse in different systems is quite difficult because they are based on components implementation. However with a clear definition of the role of the different components involved in an aspect and of the objective of each aspect member as with the aspects design table, it is possible to “extract” an aspect from its “components context”. For instance the mechanisms for realizing events handling design in aspects could be applied to other applications than RECINTERNET.

It is also possible to reuse some design of a set of related aspects and components in different applications. As an example we could imagine to reuse the database manager classes and their corresponding aspects of RECINTERNET in other applications dealing with database access.

- ***Flexibility***

Flexibility is an important advantage of the aspect-oriented approach compared to the conventional object-oriented one. Designing with aspects provides a way to develop flexible systems that cannot be obtained simply with conventional object-oriented design. With AOP, and particularly with AspectJ, aspects weaver involves plugging facilities that can be used to obtain great flexibility. Plugging in or out an aspect from a system is then simply realized. With different aspects expressing different functionalities for a same system, it is possible to simply add or remove functionality of this system.

In comparison, with a conventional object-oriented approach, “plugging out” a concern requires first to identify the different components it involves, and to remove from the components the related features, after verifying the consequences of such changes. We have seen in the RECINTERNET design with aspects that different functionalities (time bound, size bound and connection bound) can be easily added or removed to the server application.

III.5.5) Comparison conclusions

To generalize to any kind of application some points of this comparison of a conventional object-oriented and an aspect-oriented one for the RECINTERNET application, we can first insist on the fact that AOP is an emerging paradigm that is not yet as mature as the object-oriented one. As a result there are is a huge difference between the technologies, techniques and tools available for design in the two approaches.

Anyway AOP provides concrete support for separation of concerns that cannot be done in object-oriented paradigm. The resulting aspects/components modularization in design (and consequently in implementation) is an important advantage for systems understandability, complexity and size decreasing, reuse, maintenance and evolution.

Moreover, important drawbacks of the aspect-oriented design process such as the lack of design representation tools or composition conflicts detection can be consequently ameliorated by designing components and aspects as well as representing them with aspects design table.

Finally the emerging aspect-oriented paradigm is highly promising with its interesting concepts for software engineering, but needs a phase of maturation so that appropriated and efficient tools, technologies and techniques could be developed for an appropriate support at any stage of software development cycle.

III.6) Conclusions

We have then presented how it is possible to apply separation of concerns into web-based three-tiers applications design. Based on the characteristics of AspectJ to support aspect-oriented programming, we have proposed guidelines to realize aspects/components decomposition in a step by step process. This design decomposition can be efficiently represented in aspects design tables.

In a second time we have presented two different approaches for web-based three-tiers applications design: a conventional object-oriented one and an aspect-oriented one. These two approaches have been developed for the concrete case of the RECINTERNET system. We have then draw a comparison between these two approaches for this concrete case. The interesting feature was that we draw this comparison for a concrete industrial system which involves different complex concerns, and illustrates then the benefits of an aspect-oriented approach.

The first conclusion we can draw from this section is about the methodology and representation we propose for aspects design. This step by step way of decomposing the aspects design enables a clear identification and organization of the different entities of an aspect-oriented design. Aspects/components interactions are clearly expressed, and in both ways: from an aspect to its components and from a component to its involving aspects. Aspects design tables contain the important information related to this decomposition, and provides an efficient way to visualize them. The clear aspects/components interaction visualization provides then a good support to detect eventual composition conflicts.

We have illustrated the interest of using these decomposition guidelines and aspects design tables for the aspect-oriented design in the concrete case of the RECINTERNET application. By comparing this aspect-oriented approach to a conventional object-oriented one in this same concrete case, we have shown the great interest of the aspect-oriented paradigm for web-based three-tiers applications, even with the important drawback that AOP needs maturation in order to be supported by efficient tools and techniques, particularly at design level.

Based on these two approaches to realize web-based three-tiers applications design, we present in the following section (IV) how to realize the RECINTERNET implementation.

IV - RECINTERNET IMPLEMENTATION

IV.1) *Introduction*

We have previously drawn a state of the art of the different technologies and techniques applicable to applications such as RECINTERNET (Section II). We have also realized the design of such an application with two different approaches (Section III): an object-oriented one and an aspect-oriented one.

In the current section we present how to implement a web-based three-tiers application such as RECINTERNET. It would have been a good point to extend to implementation stage the comparison realized for design between the conventional object-oriented approach presented in Section III.3 and the aspect-oriented one presented in Section III.4. Given the time constraints of this thesis it was not possible to implement the two approaches.

From the DATAPREV standpoint, the objectives of this thesis were to achieve efficient design models that could provide an appropriate support for the RECINTERNET implementation, and then to implement and evaluate entire parts of the system. Then the project could be completed by the other members of the RECSINWIN/RECINTERNET DATAPREV team, based on appropriate design models as well as entire implemented and tested parts of the system.

According to these objectives we decided then to begin with the implementation corresponding to the conventional object-oriented design of RECINTERNET[38]. This decision was motivated by the need of a concrete result in the given time, and mainly by the fact that main parts of this implementation could be reuse afterwards for an implementation of the aspect-oriented approach. As a result we implemented and evaluated parts of the RECINTERNET system based on our conventional object-oriented design. Then we left sufficient design models and guidelines in order that the RECINTERNET project could be efficiently finished by the other members of the team, as it is explained in Section V.1.

For the second approach (aspect-oriented) it was not possible to implement in the time of this thesis our aspect-oriented design of RECINTERNET. Anyway we developed some guidelines for explaining how to implement and deploy this approach. These guidelines and the aspects design tables presented in Section III.4 provide an appropriate support for a future implementation of an aspect-oriented version of RECINTERNET.

Finally in the current section we present how we implemented the conventional object-oriented design of RECINTERNET in Section IV.2, and how to implement the aspect-oriented one in Section IV.3. These sections provide a concrete illustration of the different points developed in this thesis: analysis of the technologies and techniques applicable in the RECINTERNET project and RECINTERNET design with two approaches: a conventional object-oriented design and an aspect-oriented one.

IV.2) *Conventional object-oriented implementation of RECINTERNET*

We have seen in Section II how the Java language provides an appropriate support (through different features such as Java applet, RMI, etc...) for the programming of the whole RECINTERNET system. Then the different parts of RECINTERNET were all implemented in Java, based on the version 1.2 of the Java Development Kit (JDK[39]). We used the JBuilder[40] Client/Server Suite Version 2.0 to develop our Java program, since this programming environment provides an efficient drag and drop support for GUI and simple JDBC support.

[38] This decision was approved by the RECSINWIN/RECINTERNET project responsible (Dr. Emmanuel Passos) and by our advisor from (Dr. Cabral Lima from UENF).

[39] Documentation and free download available online. Java Home Page. Sun Microsystems. http://www.java.sun.com

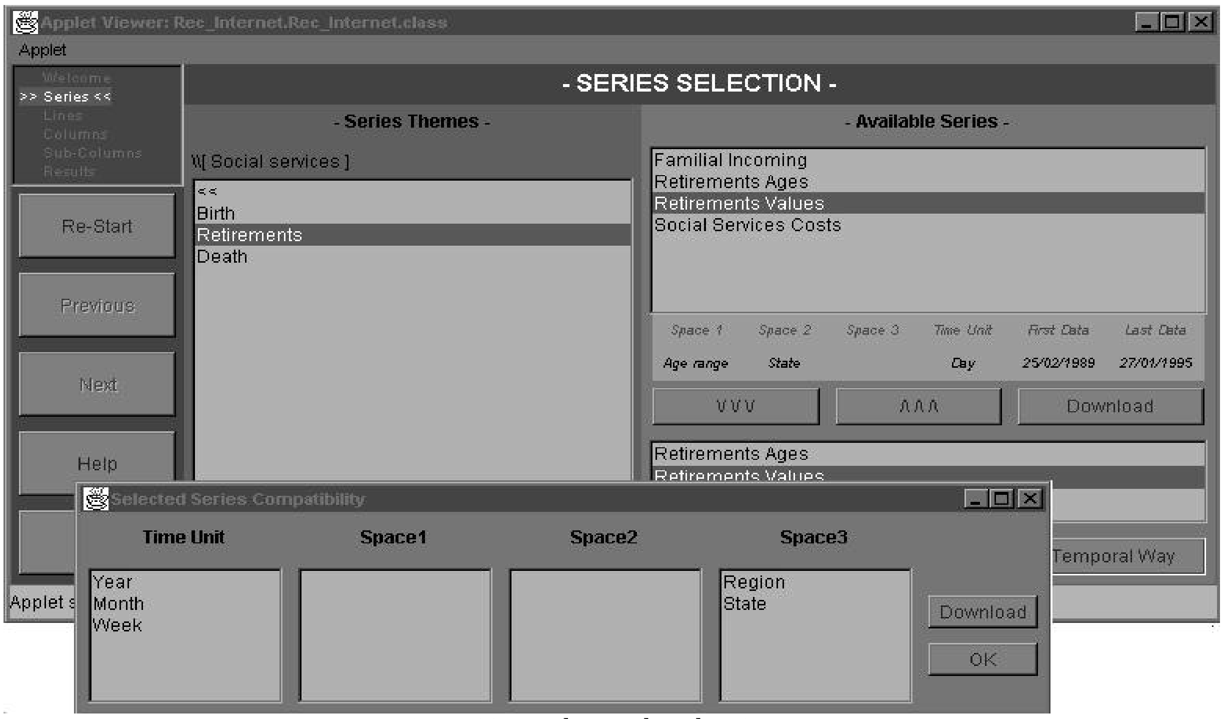
[40] Borland JBuilder^{TM} Client/Server Suite Version 2.0. Borland International. http://www.borland.com

IV.2.1) Client

We have chosen to implement the client tier of RECINTERNET as a Java applet embedded in an HTML page, as explained in Section II.5.1.5. It means that the HTML page referencing the Java applet class bytecode and the different compiled Java classes needed in this applet will be stored on the HTTP server. When a user will request this HTML page to the HTTP server through a given URL, his web browser will receive the HTML code of this page as well as the compiled classes Java bytecode. The web-browser is then able to execute the applet.

We developed then a Java applet as a support for node visualization. We implemented the different classes presented in the conventional object-oriented design to support navigation and user interaction. We also implemented the different nodes and decorators of RECINTERNET, and we used the Java Abstract Windowing Toolkit (AWT) of Java for the most part of the graphical components of each node (buttons, lists, labels, text areas...).

As RECINTERNET had to be accessible by any kind of user, using any kind of the most popular web browsers, we tested our applet with the most used web browsers (Microsoft Internet Explorer, Netscape and Hot Java). An important issue was to be able to execute the applet with different versions of these browsers. To solve this problem (without entering into details), we used a Java tool called *HTML converter*[41] that enables to modify the HTML code for the embedded applet so that *Java Plug-In*[42] is used to choose a compatible *Java Runtime Environment*[43] *(JRE)* where the applet can be normally executed. If these web-browser extensions are not available on the user machine, the user will be asked to accept their download and installation before continuing. This solution enables to execute the RECINTERNET applet in most part of used web-browsers. We evaluated our Java applet by testing successfully its functionalities with different sets of data, on different web-browsers. Finally we realized completely the RECINTERNET client tier implementation. The following figures present few screens of the RECINTERNET applet.



\\[Social services]

<<
Birth
Retirements
Death

Familial Incoming
Retirements Ages
Retirements Values
Social Services Costs

Space 1	Space 2	Space 3	Time Unit	First Data	Last Data
Age range	State		Day	25/02/1989	27/01/1995

Retirements Ages
Retirements Values

Time Unit	Space1	Space2	Space3
Year Month Week			Region State

Figure 33 - Node 1 and Node 1C screen

[41] *HTML Converter.* Available for free online: http://java.sun.com/products/plugin/1.1.2/converter.html

[42] *Java Plug-In* is a Java tool that enables to choose the Java Virtual Machine used to execute an applet from any web browser. Available for free online: http://java.sun.com/products/plugin/download/windows.html

[43] *Java Runtime Environments (JRE)* are Java Virtual Machines compatible with a given JDK version. The JRE for JDK 1.2 is available for free online: http://java.sun.com/products/jdk/1.2/jre/download-windows.html

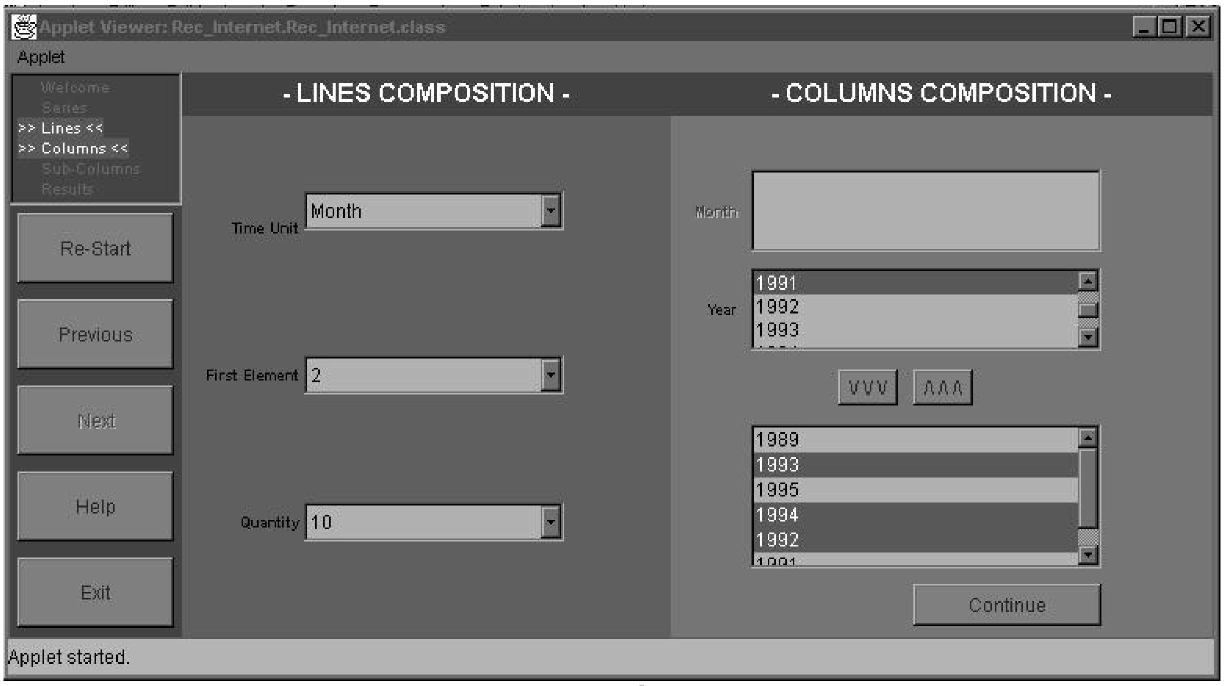
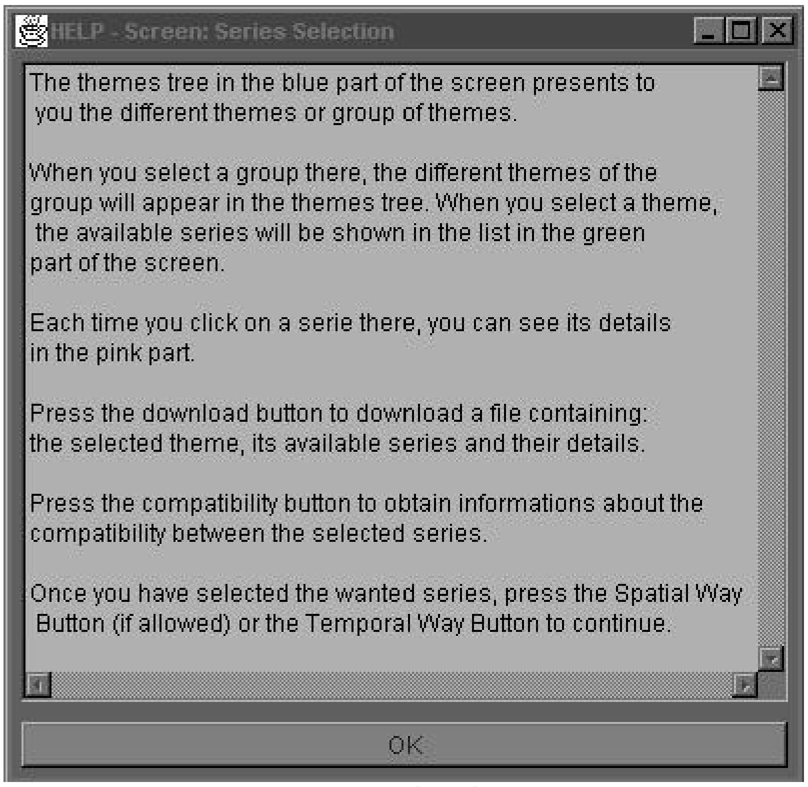


Figure 34 - Node 221 screen



The themes tree in the blue part of the screen presents to you the different themes or group of themes.

When you select a group there, the different themes of the group will appear in the themes tree. When you select a theme, the available series will be shown in the list in the green part of the screen.

Each time you click on a serie there, you can see its details in the pink part.

Press the download button to download a file containing: the selected theme, its available series and their details.

Press the compatibility button to obtain informations about the compatibility between the selected series.

Once you have selected the wanted series, press the Spatial Way Button (if allowed) or the Temporal Way Button to continue.

Figure 35 - Node Help screen

IV.2.2) Server

The server part of the RECINTERNET system was implemented as a Java application, as we explained it in Section II.5.2.4. This application will be started on the server host. This application will be executed in the Java Virtual Machine of the server host. The Java classes of the server application use the version 1.2 of JDK, and then a compatible Java Virtual Machine must be available on the server host.

It has to be noticed that this server application must be running on the same host (the same IP address) from where the client applet where downloaded. This is due to the security restrictions of Java applets: an applet cannot realize a remote communication with another host than the one from where it has been downloaded. So the server application will be run on the same machine than the HTTP server.

We implemented a main class, that can be started to initialize the different parts of the server (Java RMI registry, database manager objects). The most part of the server application implementation was the definition of the different client threads classes, as well as the dispatch manager class (which receives requests from clients and creates and starts the appropriate client thread) and the database manager classes. All this classes implementation was closely supported by the object-oriented design that we presented in Section III.3.2.

There was no particular need for graphical interface for this server application. So we just implemented an application that can be launched with a Java command line and that prints few informations about the server states on the control window.

We have tested this server application by simulating some client requests coming from client applets. We made tests with requests corresponding to each kind of client threads classes that can be executed on the server. The execution of these threads was at the end carried out on the server without problems. So finally this part of RECINTERNET was cleanly implemented and tested.

IV.2.3) Client/Server communication

As we have shown it in Section II.6.1.6, we chose to realize the client/server communication with Java RMI. This technology enables to call methods of objects running in Java Virtual Machines of different machines in a transparent way.

To realize the implementation of this communication part, we mainly defined the RMI interfaces (that describes the methods that can be called remotely), according to the design presented in Section III.3.2.1. We then have added the required modifications to the classes of the server (`DispatchManager`) and of the client (the different nodes and decorators) that had to implement these interfaces.

These classes implemented the RMI interfaces had to be compiled with the Java command `rmic`, in order to create the appropriate RMI stubs and skeletons that will be used to call the remote objects. When starting the server application, the RMI registry must also be started so that the dispatch manager object can be registered in the Java RMI registry and called by client applets. It has to be noticed that the IP address of the host on which is running the server application must be known from the client applets. These applets can then create the appropriate references for the remote server dispatch manager and call some of its methods correctly.

We only had the possibility to test the RMI deployment of RECINTERNET on the DATAPREV intranet. It was however not possible to test the system on any HTTP server and then the RMI features with the Internet. A HTTP server will only be available when the whole RECINTERNET system will have been implemented and will be ready to be opened to the public. Anyway we have validated locally the main functionalities of the RMI communication between the client Java applets and the server application.

IV.2.4) Database access

We have described in Section II.6.2.1 and II.6.2.2 the way the RECINTERNET databases are accessed. The access to the server structural database is done using the JDBC standard. It is then possible to insert database exploitation with SQL queries inside Java code.

The server structural database must be installed on the server host. It is an *Interbase* database, that is managed by *Interbase server*, which must be started on the server machine. Specific JDBC drivers for *Interbase*[44] must be installed on the server machine in order to be able to realize the database connection.

The SINTESE database is hosted on a remote machine connected to the server host by the DATAPREV intranet. The database server on this remote machine is normally accessed through terminals connection. With Java it is possible to simulate such a terminal and to pass to it command lines (in SINTESE syntax) through TCP/IP protocol. Then it is possible to submit from the server application some requests to the SINTESE database.

We implemented the `DatabaseManager` classes described in Section III.3.2.3 to provide the appropriate structure for databases access, but for time reason we did not implemented the detailed finalization of databases connection nor results exploitation. Anyway we clearly defined the `RIDatabaseManagerSQL` and the `RIDatabaseManagerSINTESE` classes structure and the way to achieve their implementation.

Finally, based on our state of the art of possible technologies and on our conventional object-oriented design, we have implemented and tested most of the parts of the RECINTERNET system. Client applet and server application were totally implemented and were successfully tested. Client/server communication with Java RMI was entirely implemented but only tested locally, without a complete Internet deployment. Database access was partially implemented, and will be completed by the other members of the RECINTERNET team, according to the DATAPREV the definition of our thesis work objectives in the given time.

IV.3) Guidelines for the implementation of the aspect-oriented design

We did not have time to implement the aspect-oriented design of RECINTERNET we presented in Section III.4. However we present here briefly how to realize in Java this aspect-oriented implementation with AspectJ.

IV.3.1) Aspect-oriented implementation with AspectJ

The implementation mechanism with AspectJ can be decomposed as following: *components implementation, aspects implementation* and *components/aspects composition.*

- *Components implementation* is realized by implementing Java components classes in the same way than in a conventional object-oriented implementation. Java classes of the different components are implemented according to the different designs models presented in Section III.4 (most parts of these components implementation are the same than in the conventional object-oriented implementation). The result is a set of classes expressed in Java code (*.java* files).

- *Aspects implementation* is done in the same way than components implementation. Based on the aspects design tables presented in Section III.4, aspects are expressed in the AspectJ language presented in Section III.1.2.3. Aspects are defined as Java classes in modules, with different aspect members using the Java syntax. The result is a set of aspects expressed in AspectJ language (concretely these aspects are stored also in *.java* file, even if they are not expressed in conventional Java code).

[44] The all-Java JDBC driver for Interbase can be downloaded for free from the Interbase Home Page: http://www.interbase.com

• *Components/aspects composition* is automatically done with the AspectJ *weaver*. Once AspectJ is installed, after having set the different source and class paths (as explained in the AspectJ installation documentation), it is possible to use the `ajc` command to start the aspects/components weaving. This command take as parameters all the java files *(.java)* of the aspects and components to be woven. Given the options chosen, it is either possible to obtain the woven Java code either to obtain directly the corresponding compiled Java bytecode[45]. It has to be noticed that this obtain Java code (or bytecode) is created in new files (*.java* for Java code or *.class* for Java bytecode), and then the initial aspects and components code is not modified.

It is possible to plug-in or plug-out some aspects on the obtained woven Java code in a simple way. For instance, to plug-in a new aspect on a woven Java code, it is just needed to weave this aspect with the woven Java code. To plug-out one aspect, it is just needed to take back the initial components code that were left unmodified by the previous weaving and weave them again if needed. The Figure 36 presents one example of these plug-in and plug-out mechanisms.

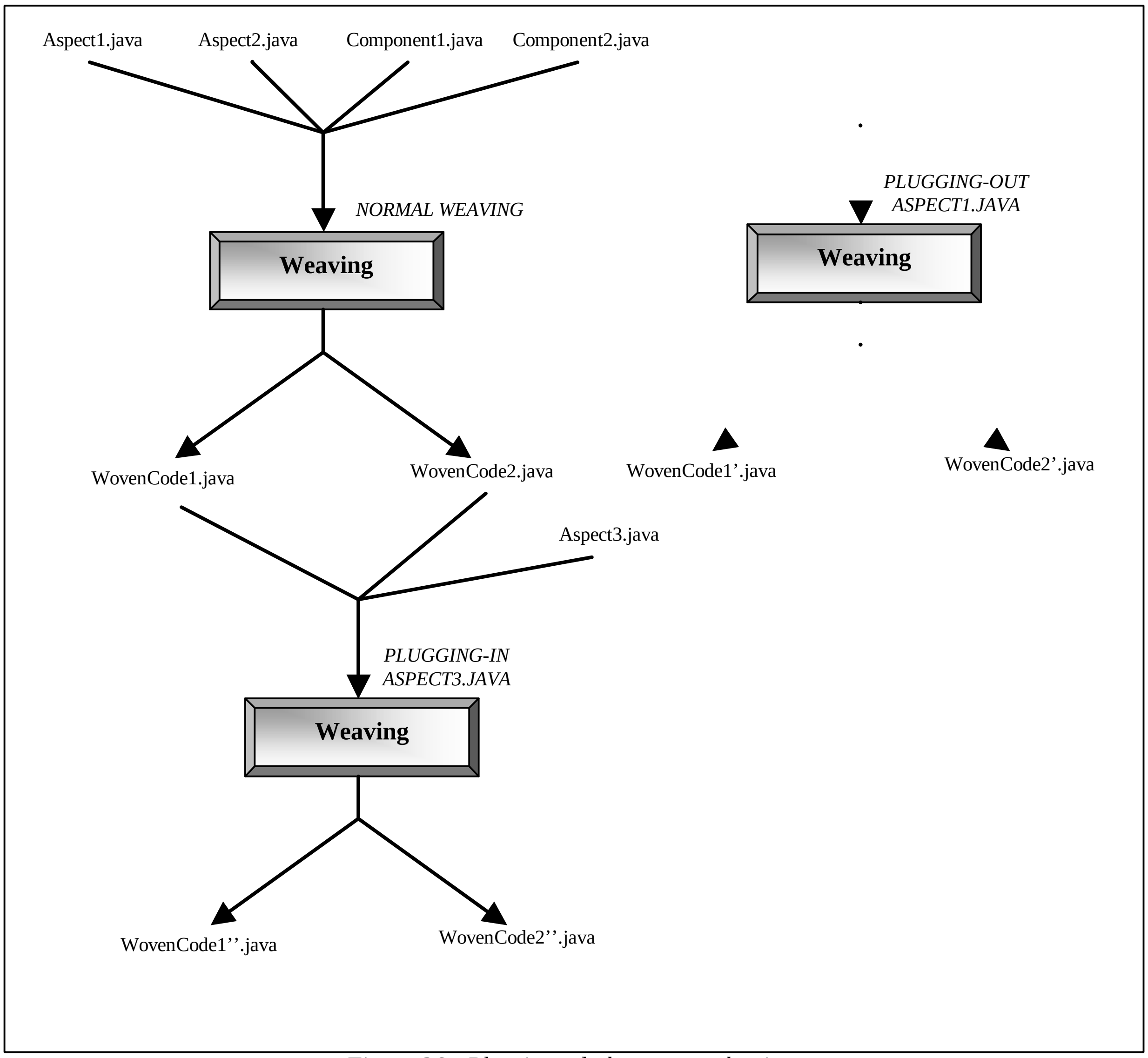


Figure 36 - Plug-in and plug-out mechanisms

[45] If this option is chosen, the weaver creates the woven Java code and calls the normal Java compiler (`javac`) to compile it and create the corresponding Java bytecode.

IV.3.2) RECINTERNET implementation

The aspects design tables presented in Section III.4 and the components design models of Sections III.4 and III.3 provide a good support for the implementation of the aspects and components of this approach, as we explain it in the current section.

With an aspect-oriented decomposition of RECINTERNET into components and aspects, the components implementation process is the same that in a conventional object-oriented approach. However, specifications about the components involved in different aspects must be carefully respected. For the aspects implementation, it is possible to closely follow the information of the aspects design table, that identify properly the different members, join points and intentions of an aspect. As an example, we give the implementation of the *Dynamic Node Load* aspect, presented in details in Section III.4.2. The summary of the aspects design table for this aspect is given in Table 8.

		Navigation				
		Dynamic Node Load				
		TargetNode access	Cache searching	Node creation	Searching node	
		Yes[46]	No	No	No	(
		Before wrapping	Method adding	Aspect method	Aspect method	
`Context`	`GetNode()`		▨			
`Transformation`	`Action()`	▨				
`Navig Transformation`	`SpecificAction()`	▨				
`Intra Transformation`	`SpecificAction()`	▨				
`Extra Transformation`	`SpecificAction()`	▨				
`RIPrevNext Transformation`	`SpecificAction()`	▨				
Dynamic Node Load	`GetNode()`				▨	
	`CreateNode()`			▨		

Table 8 - Dynamic Node Load aspects design table

The Figure 37 presents our Java implementation of the different classes involved in this aspect. Their implementation has been done according to the specifications given in Section III.4.2 about the *Dynamic Node Load* aspect, and also about the conventional object-oriented models presented in Section III.3.1.

[46] The generic join point expression is: `protected !abstract void * specificAction(..), public void Transformation.action(..).`

RIContext.java
```
package Rec_Internet;
/** This class is used to represent the navigation informations */
public class RIContext
     extends CacheContext {
// We don’t present the code of this class since it is not used for this aspect example
}
```

Transformation.java
```
package Rec_Internet;
/** This class is used to contains the necessary methods and variables to describe navigation actions */
public abstract class Transformation {

     /** Class variable: the context */
     protected static RIContext context = null;

     /** Instance variable: reference of the target node of          a navigation */
     protected Node targetNode = null;

      /** Class method context mutator */
     public static final void setContext(RIContext context_) { context = context_; }

     /** context accessor */
     public RIContext getContext() { return context; }

     /** Template method to define the structure of a navigation action */
     public void action(String target, Object args[])
          throws RIException {
          specificAction(target, args);
          targetNode.show(); }

     /** Abstract method for specific actions depending on the type of navigation */
     protected abstract void specificAction(String target, Object args[])
         throws RIException; }
```

NavigTransformation.java
```
package Rec_Internet;
/** This class is used in the case of a normal navigation between two navigational nodes */
public class NavigTransformation
  extends Transformation {

   /** In the case of a normal navigation between two navigational nodes, we need to hide the origin node, to
reset the target node, and then set in the context the current node index and the next index */
  protected void specificAction(String target, Object args[])
    throws RIException {
    context.getCurrentNode().hide();
    context.setCurrentNode(targetNode);
    context.setNextIndex(target);
    targetNode.reset(args); } }
```

IntraTransformation.java
```
package Rec_Internet;
/**This class is used in the case of a navigation from a navigational node to an intra node */
public class IntraTransformation
  extends Transformation {
  /** For a navigation to an intra node, we desactivate the navigational no de, and reset the target node */
  protected void specificAction(String target, Object args[])
    throws RIException {
    context.getCurrentNode().desactivate();
    targetNode.reset(args); } }
```

ExtraTransformation.java
```
package Rec_Internet;
/** Used for a navigation from a navigational node to an extra node */
public class ExtraTransformation
  extends Transformation {
  /** In the case of a navigation to an extra node, we just need to reset the target node */
  protected void specificAction(String targe t, Object args[])
    throws RIException {
    targetNode.reset(args); } }
```

RIPrevNextTransformation.java
```
package Rec_Internet;
/** To describe previous and Next navigation */
public class RIPrevNextTransformation
  extends Transformation {

  /** In the case of a previous or next navigation, we hide the origin node and set in the context the
target node as current node. We don’t reset the target node */
  protected void specificAction(String target, Object args[])
    throws RIException  {
    context.getCurrentNode().hide();
    context.setCurrentNode(targetNode); } }
```

Figure 37 - Components code for the Dynamic Node Load aspect

The Figure 38 presents the code of the Dynamic Node Load aspect. This aspect has been implemented according to the corresponding aspects design table.

AdynamicNodeLoad.java

```
package Rec_Internet;

/** This aspect defines all the necessary modifications related to dynamic node search a nd
creation when navigating */
aspect ADynamicNodeLoad {

    /** Node Creation member:
    This method dispatch the creation request to the appropriate node constructor */
    public static Node createNode(String name)
        throws RIException {

        if (name.equals("Node0")) return new RINode0();
        if (name.equals("Node1")) return new RINode1();
        if (name.equals("Node211")) return new RINode211();
        if (name.equals("Node212")) return new RINode212();
        if (name.equals("Node221")) return new RINode221();
        if (name.equals("Node222")) return new RINode222();
        if (name.equals("Node3")) return new RINode3();
        if (name.equals("Node1C")) return new RINode1C();
        if (name.equals("Node3D")) return new RINode3D();
        if (name.equals("Help")) return new RIHelp();

        // Throw an exception for other names
        throw new RIException(name, UNKNOWN_NODE);
    }

    /** Searching Node member:
    This method organize the node search/creation. First it calls the method to search the
node in the context, and if it is not found, call the method t o create it */
    public static Node getNode(String name, RIContext context)
        throws RIException {
        Node result = context.getNode(name);
        if (name == null) {
            return ADynamicNodeLoad.createNode(name); }
        else return result;
    }

    /** TargetNode Access member:
    We add the targetNode initialization before all the methods where this variable is used */
    advise protected !abstract void *.specificAction(..),
        public void Transformation.action(..) {
        before { targetNode = ADynamicNodeLoad.getNode(target, co ntext); }
    }

    /** Cache Searching member:
    We introduce in the RIContext class the method to search a node */
    introduce public Node RIContext.getNode(String target)  {
        for (int i=0; i<navigNode.size(); i++) {
            Node node = (Node) navigNode.elementAt(i);
            if (node.getName().equals(target))
                return node; }
        for (int i=0; i<intraNode.size(); i++) {
            Node node = (Node) intraNode.elementAt(i);
            if (node.getName().equals(target))
                return node; }
        for (int i=0; i<extra.Node.size(); i++) {
            Node node = (Node) extraNode.elementAt(i);
            if (node.getName().equals(target))
                return node; }
        return null;
    }
}
```

Figure 38 - Dynamic Node Load aspect

The weaving of these components with the aspect can then be done as explained in the previous part. We can plug-in or plug-out as wanted the different aspects of the aspect-oriented RECINTERNET system. We then obtain automatically different woven Java files (.java) that can be normally compiled to produce an executable Java bytecode (.class). The Appendix E presents the resulting code of the weaving of this aspect and its related components.

It has to be noticed that once the weaving mechanism has been achieved, when compiling the obtained code, the compilation errors refers to the woven code, and not to the initial components or aspects code. Then with a compilation error and the line where it happened, the debugger needs to track from which component or aspect the code is coming. This is facilitate by the comments generated by the aspect weaver, but however it make debugging process more difficult. In the same way run-time exceptions are shown with a line number, which refers to the woven code and not the initial one.

Another problem encountered with AspectJ is that it is not possible to specify in which order the given aspects will be woven to the components code. When weaving a set of components and aspects files, AspectJ takes the aspects one by one in a random order and weave them to the components code. There is no way to specify an aspect weaving order. The only solution when two aspects must be woven in a given order is to realize a first weaving with only the first aspect, and then a second weaving with the second aspect on the obtained woven code. Anyway it add on level of "weaving indirection" for debugging the obtained code, which can be problematic.

Anyway AspectJ provides an efficient way to implement aspect-oriented programs. RECINTERNET can then be implementing by following closely the aspects design tables of Section III.4 and the components description of Section III.4 and III.3.

IV.4) Conclusions

Given the time constraints it was not possible to extend to a complete implementation the two approaches (object-oriented and aspect-oriented) we used for RECINTERNET design. We started then with the implementation of our conventional object-oriented design and achieved most parts of this implementation. For the aspect-oriented approach we only give guidelines to explain how it could be done.

We have then presented in this section the way we realized our object-oriented implementation of the RECINTERNET system. To briefly give a synthesis of the achievement we reached in this implementation, it can be said that most parts of the RECINTERNET system have been wholly implemented: the client applet, the server application as well as the client/server communication features are entirely coded. For the database access part we have only implemented it partially. It must also be noticed that for each of these parts implemented we realized corresponding evaluation by testing their execution with sample sets for the different functionality they had to provide. Finally we have developed most parts of the RECINTERNET system according to the design we had conceived and represented in design models. We explain in Section V.1 how our implementation will be used by the RECSINWIN/RECINTERNET team to finalize the development of the RECINTERNET project.

We have also presented in the current section some guidelines for realizing the implementation of the aspect-oriented approach. For this purpose we described precisely the weaving mechanism of AspectJ to compose automatically aspects with components code. We explained also the way to develop aspects code with the support of our aspects design tables. We finally illustrated these guidelines with an implementation example of one aspect we described in our aspect-oriented design: the *Dynamic Node Load* aspect.

To conclude this section, we can say that by developing the implementation (or explaining how to realize it) of the RECINTERNET whole system with the object-oriented and aspect-oriented approaches, we gave a concrete illustration of the different research aspects developed in this thesis.

V - CONCLUSIONS AND PERSPECTIVES

In this section we draw conclusions about the work realized and we analyze the perspectives it involves. In Section V.1 we analyze the state of the RECINTERNET project at the end of this thesis and explain how it will evolve. We give in Section V.2 conclusions about the two approaches we used for developing the RECINTERNET project: the object-oriented and the aspect-oriented approaches. We finish in Section V.3 with conclusions and perspectives about our original approach for aspect-oriented design organization and representation.

V.1) The RECINTERNET project

From the industrial standpoint of DATAPREV, the work realized during this thesis had to reach different objectives. Based on the RECSINWIN project we had to conceive the RECINTERNET system, that is to say a web-interface that enables users to compose dynamically a request to the SINTESE database of DATAPREV and then visualize its results. The given objectives were to browse the existing technologies and techniques in order to create appropriate design models and to implement and evaluate most parts of the RECINTERNET system.

In this thesis we realized an analysis of the techniques, technologies and tools suitable for the RECINTERNET development. For design stage we analyzed particularly the architectural possibilities, the way to design dynamic web-interfaces and the techniques to realize separation of concerns. For implementation stage we focused on the technologies available for implementing each of the three tiers of RECINTERNET: client, server and database. We also studied some possibilities to realize the communication between these three tiers. Finally we also analyzed the existing technologies to support implementation with separation of concerns.

As a result of this analysis we proposed the following choices for the RECINTERNET development. RECINTERNET can be programmed according to the principles of separation of concerns, and particularly with Aspect Oriented Programming (AOP). In a more conventional object-oriented way, RECINTERNET can be developed according to the Object Oriented Hypermedia Design Methodology (OOHDM) and its associated design patterns. This two possibilities have been the starting point of the development of RECINTERNET with two approaches and their comparison.

We have also chosen for RECINTERNET a three-tier architecture with a specific characteristic: using a server local structural database mirroring the structure of the main SINTESE database of DATAPREV in order to make the requests composition faster and to only connect to the SINTESE database for final requests.

Our analysis led us to choose to program the client part of RECINTERNET as a Java applet embedded in a HyperText Markup Language (HTML) page, so that it can be executed within any web-browser. The server part is a Java application running on a server host and communicating with the client applets with Java Remote Method Invocation (RMI). The access to the server local structural database is achieved through the Java DataBase Connectivity (JDBC) standard, and the access to the remote SINTESE database is done through a simulation of a terminal connection to the SINTESE DataBase Management System (DBMS) with a Transmission Control Protocol/Internet Protocol (TCP/IP).

Based on these choices we realized the RECINTERNET development with two approaches: a conventional object-oriented one and an aspect-oriented one. We applied the OOHDM methodology to design the client application of RECINTERNET, as well as some of its design patterns. We also designed carefully the other parts of the system: server, client/server communication and database access. As a result we provided a set of object-oriented design models that cover the different parts of the RECINTERNET system and provide an appropriate support for its implementation.

For the design of the aspect-oriented approach we decomposed the different parts (client, server, client/server and database access) of the RECINTERNET system into aspects and components. The components design reused parts of the conventional object-oriented design models and add some modifications in order to support aspects. Aspects design was realized with a step by step decomposition methodology we developed. Finally we represented this aspects/components design with some aspects design tables that organize the decomposition and enable a clear visualization of aspects/components interactions.

We then realized the implementation of the conventional object-oriented implementation design models. We implemented and evaluated completely the client part of RECINTERNET, as well as the server application. For the client/server communication part, we implemented it totally but only tested it locally[47]. We provided a partially completed implementation of the database access part. Given the time constraints, for the implementation of the aspect-oriented approach we only proposed some guidelines on the way to do it, as well as example of aspects implementation.

Finally we have realized partially two approaches for the RECINTERNET development. For both approaches we have developed a set of design models covering the different parts of the system and providing appropriate implementation support. In the object-oriented approach we left to the project team most of the parts of the RECINTERNET system totally implemented and evaluated and guidelines to finalize the lasting parts implementation and deployment. In the aspect-oriented approach we left to the RECSINWIN/RECINTERNET project team guidelines to achieve the RECINTERNET implementation from our design models as well as illustrative examples.

- ***Evolution of the RECINTERNET project***

Based on the work realized during our thesis, and according to what had been foreseen about this work with DATAPREV, the members of the RECSINWIN/RECINTERNET project team will finalize the RECINTERNET development.

In a first step they will finalize the conventional RECINTERNET implementation. Based on the parts already implemented and on our design models, they will finish the implementation of the database access part. They will then continue with some tests on this implementation before carrying out the whole deployment of the entire RECINTERNET system: installing the required applications, database and files on the HTTP server and testing the internet access and the whole system execution.

The RECSINWIN/RECINTERNET team agreed to use our aspect-oriented approach to become more familiar with the aspect-oriented way of programming that they had never used previously. They were particularly interested in the possibility to program some server functionalities (client thread execution time control, results size control and number of client threads running on the server control) with easily pluggable aspects as we proposed in our aspect-oriented approach.

At this point of the project we also have some concrete propositions for the enhancement of the RECINTERNET system:

A first improvement in the soon evolution of RECINTERNET should be to realize its Brazilian version (in Portuguese language). It means that any information appearing in the client applet should be written in Portuguese. Actually the client applet implementation we realized is only in English and then cannot be deployed to a large Brazilian public. Even if the data provided by the databases are all in Portuguese language (such as the name of the states of a serie, the months where a serie is defined and so on), the graphical and navigational informations of the client applet are in English (such as *continue, exit, help,* etc...). We can imagine to modify these words hard-coded in the client applet implementation. Another idea should be to develop some "language" aspects (English, Portuguese aspects, or even more) that can be plugged to the client applet code to define language variables (and their corresponding values which will be the words in the given language) that will be read each time a word must appear to the user in the applet. By plugging-in one of these aspects it could be then simple to obtain a different language version of the client applet.

[47] As we explained in Section IV.2.3, it was not possible to realize complete tests because we did not have the possibility to use an HTTP server.

As an ulterior evolution we propose to extend the RECINTERNET system in order to provide web-users the possibility to define their requests in a different way. As it is possible in the RECSINWIN system, it could be possible to express requests to the SINTESE database in natural language. The web-user could then write a request with his own words or choose between a set of pre-defined sentences. The same approach than the one used in RECSINWIN could be used to filter and correct such sentences and translate them in to SINTESE queries. It has to be noticed however that this extension should modify consequently the RECINTERNET system since it requires some language dictionaries where are referenced the different words that can be used. A good solution could be to store these dictionaries on the server host and to access them with Java RMI from the client applet.

We propose also an improvement in the database access part of the RECINTERNET system. The idea should be to light up the connections to the SINTESE database by using a kind of cache mechanism. We thought about storing in the server local structural database not only the structure of the SINTESE database but also the series entries that are requested by users very frequently. In fact there are series of the SINTESE database that are requested more frequently than others, and so it could ameliorate consequently the requests answering mechanisms to only consult the server local structural database for very common requests. Moreover as the SINTESE database is only updated in given period, all at a time, updating the server local structural database for these cached series entries would not require more specific attention than in the actual updating mechanism.

A last enhancement that we propose as an ulterior modification for RECINTERNET aims to address the problem of server overflow. If RECINTERNET becomes used by a high number of clients, the unique CPU of the server where client threads are running can be rapidly insufficient. As a solution we can propose to replace the middle tier (the server) of RECINTERNET by several identical servers, running on different hosts. The architecture will then evolve to a multi-tiers architecture. All the client request would still be addressed to a unique server with Java RMI, but a load-balancing functionality will be defined to choose to execute the corresponding client thread on the "less busy" of the different servers.

To conclude this part we can say that we have carried out the most part of the development of a first version of RECINTERNET, which will be finalized by the RECSINWIN/RECINTERNET team and be ameliorated and modified in future versions.

V.2) Object-oriented and Aspect-oriented approaches of RECINTERNET development

We have also explained in this thesis how we realized the RECINTERNET project with two approaches: a conventional object-oriented one and an aspect-oriented one.

In the conventional object-oriented approach we have split the RECINTERNET system into four parts: client, server, client/server communication and database access. Using parts of the OOHDM methodology we have decomposed the RECINTERNET client application into functional units. These units are encapsulated in objects, which are described by their classes. We have described these classes using conventional design models for object-oriented programming: class diagrams and state diagrams in UML, classes descriptions. These models are language independent. They describe different parts of the client application, such as the structure of the entities of the databases that will be used *(conceptual model),* classes framework for navigation, extension of this framework for RECINTERNET *(navigational model)* and graphical representation *(interface model).*

We also described the other parts of the RECINTERNET system (server, client/server communication and database access) with these conventional object-oriented design models. The main particularities of this design are the use of Java RMI interface for describing remote calls between client and server, the use of client threads classes representing each kind of client requests, the fact that these client threads directly communicate with the client object to send back the results of a request and the fact that the two databases are only accessed through a database manager object. All these points are concretely described in the different language independent UML diagrams and explanations that we provided for the object-oriented approach.

Based on these design models, the object-oriented implementation of RECINTERNET can be characterized by the fact that we could automatically generate an important part of the code from the design models with the Rational Rose tool. All our implementation was done in Java, using the programming facilities of JBuilder such as graphical interface development facilities and debugging.

For the aspect-oriented approach we realized a decomposition of the RECINTERNET system into different aspects and components, based on the possibilities of the AspectJ tool, which works with components expressed in Java. In order to realize this decomposition at design level, we followed a methodology we developed to organize aspects conception. We reused several parts of the conventional object-oriented design models to define the different components of the aspect/components decomposition. We realized some modifications in these design models in order to support the different aspects of this decomposition. So the components design of this approach were described with conventional object-oriented language independent design models, with few comments specific to Java in order to specify some points for aspects support.

Aspects design was achieved through our step by step decomposition. We identified some aspects in the different parts of the RECINTERNET system (client, server, client/server communication and database access). We designed aspects for dynamic node loading, event handling, distribution, server functionalities (client threads execution time limit, results size limit and number of client threads running on the server limit) and database connection. To represent them we introduced our aspects design tables, which enable to clearly structure aspects and components and to visualize aspects/components interaction. Finally we modeled the aspect-oriented design with conventional object-oriented diagrams for components and with aspects design tables for aspects and aspects/components interaction.

We provided elements to guide the implementation of the aspect-oriented approach. Components implementation can be done in the same way than in the conventional object-oriented approach, using for example automatic code generation from design models. Aspects can be implemented with AspectJ by following closely the aspects design tables. However there is no automatic aspects implementation support nor facilities such as debugging as in object-oriented implementation.

We compared these two approaches in the case of RECINTERNET development, mainly for the design stage. We highlighted that AOP is an emerging paradigm that need time to maturate in order to support more efficiently design and implementation processes with appropriate tools and techniques. For this point object-oriented design and implementation is clearly supported in a better way with appropriate tools and design techniques. In this context we illustrated the benefits of our aspects design tables for aspects implementation support.

On the other hand we insisted on the advantages of the aspect-oriented approach, and particularly for RECINTERNET and web-based three-tiers applications. By supporting separation of concerns programming, AOP is clearly advantageous compared to object-oriented programming. To sum up the advantages we highlighted for aspect-oriented design and implementation of RECINTERNET, we can say that AOP introduces a notion of high modularity (distinguish components and aspects as well as aspects between themselves) that is cruelly missing in object-oriented programming. This modularity improves consequently programs complexity and size decreasing, understandability, flexibility and reuse.

Finally we have shown that AOP presents important advantages compared to object-oriented programming. It is then an interesting and promising emerging paradigm, which needs time in order to reach maturation and to be supported by efficient tools and techniques for each stages of software development.

V.3) Aspects design tables and step by step aspects design

As aspects are mainly based on components implementation it is difficult to clearly express them at design level. Moreover as AOP is an emerging programming technique, when developing the RECINTERNET project with the aspect-oriented approach, we were confronted to a lack of appropriate techniques and representation for aspects design.

In this context we developed an original approach for organizing and representing aspects design. The main idea is to decompose a system design in two dimensions: a components dimension and an aspects one. We proposed a step by step methodology to structure this decomposition, divided in four points: aspects identification, components design, aspects characterization and aspect members definition.

The different steps of this methodology gives an appropriate support to characterize and organize these two dimensions of a design, as well as clearly define the interactions between these two dimensions. Through a clear identification and description of join points between aspects and components, it is then possible to define precisely the way these two dimensions collaborate. Aspects intention is carefully described through explanations of their aspect members objectives and the way they involve components.

Aspects design tables sum up the main information of aspects/components decomposition with this methodology. Aspects are represented in table columns and components in table lines. In a further level aspect members are represented in sub-columns and components join points in sub-lines. Aspects/components interactions are represented in the crossing of these sub-columns and sub-lines. Aspects design tables also contains additional information such as aspect type and plugging constraints, member type and generic join points expression and several descriptions of the different entities of the two dimensions.

We have illustrated that this simple way of representing aspects design presents several advantages for design realization as well as implementation support.

A first advantage comes from the visualization possibilities of aspects design tables. The decomposition in two dimensions and the way these two dimensions are linked can simply be visualized with aspects design tables. Interactions between aspects and components can be easily identified in lines and columns crossing. These interactions are defined at components members[48] and aspects members[49] level. An important characteristic is that aspects design tables provide interaction visualization in both directions: from a given component it is possible to simply find the different aspects where it is involved, and from a given aspect it is possible to find the components it involves. Moreover this simple interactions visualization goes one step further since it is also applicable to the level of components members and aspects members.

Another important advantage of aspects design tables is for detecting aspects composition conflicts. Aspects conflicts can appear when two or more aspects are woven to the same component code but describes some incompatible code modifications. Aspects interferences can be very subtle, but aspects design tables enable to track and detect simply some of them. Shared join points between aspects can be a source of composition conflicts since they represent different modifications at the same place of components code. Components involvement in aspects can be efficiently visualized with aspects design tables, and then it is easy to track shared join points and to localize potential conflicts between aspects members.

Last, but not least, aspects design tables provide an efficient way of representing aspects design and then an appropriate support for aspects implementation. All the important informations for implementing aspects are represented in aspects design tables and can be closely followed during aspects implementation. Columns and sub-columns describe closely the different aspects modules as well as the aspects structure (the different members of an aspect) and the components code modifications that will be defined in aspects members.

We illustrated the applicability and the benefits of our methodology and representation in the concrete case of RECINTERNET design. All the aspects of the system were conceived with our methodology and represented in aspects design tables. Finally we illustrated the way aspects design tables provide a good support for implementation by implementing one of the aspects of the RECINTERNET design (the *Dynamic Node Load* aspect) out of its aspect design table.

- ***Possible improvements of our methodology and its aspects design tables***

The step by step aspects/components decomposition process and its aspects design tables we proposed are just a simple way to organize and visualize aspects design. However it presents consequent advantages and we

[48] Component members are the variables, constructors or methods of a class.
[49] Aspects members are the variables, constructor, methods, introduce members or advise members of an aspect.

thought about few improvements that could make it more benefic in aspect-oriented software development. We also stay open to any comments and suggestions for improving our methodology and representation.

A simple first enhancement could be to simply color differently the cell corresponding to the crossing of a component join point with an aspect member which creates a new component member (one of the aspect member types we defined as *variable adding*, *constructor adding* and *method adding).* This would provide more direct information of components structure modifications (i.e. members adding), but would mainly highlight composition conflicts happening if aspect weaving is not realized in the appropriate order. For instance if an aspect *A1* add a new method *M* in a component *C* and another aspect *A2* realizes modifications on this method *M* (such as an *after wrapping* modification for example), the aspect *A1* must be woven to the component *C* code before the aspect *A2*. If not there will be a composition conflict since the aspect *A2* describes modifications on an non-existing method. Coloring specially the cell corresponding to the crossing of method *M* line and the corresponding aspect member of aspect *A1* column would provide an efficient way to detect this kind of conflicts with aspects design tables.

Another improvement we propose is to express inside the crossing cells the different methods or variable calls that will be done in the corresponding aspect member. For instance we can imagine that we have an aspect *A1*. This aspect has an aspect member *A1* that add a method *M1* to a component *C1*. In the description of this method *M1* is explained that there will be a call to a variable *V1* of a component *C2* and a call to a method *M2* of the aspect *A1*. Then we can write in the corresponding cell (C1.M1:A1.AM1) the name of the involved calls that we represented for this example by C2.V1 and A1.M2. This provides one more level in information of what will be coded in aspect members, and also one more level for composition conflicts detecting. This proposition will in fact refine the possibility to track the different modifications involved by an aspect and to potentially detect more subtle composition conflicts.

For the RECINTERNET design case, we have developed this methodology and representation specifically for the Java language and the AspectJ tool and its associated aspect language. An interesting extension could be done to be independent of components languages but also (as much as possible) independent from aspect languages. This last point could be done by extending the range of aspect members type to the new kind of modifications involved by other aspects language than AspectJ, as well as new possibilities for join points expression.

As ulterior improvements, we also thought of automatic potential composition conflicts detection. With computerized aspects design tables it seems possible to develop a small application that browses the different join points of a design and the type of the aspect member in which they are involved. Taking one by one the components members represented in an aspects design table, this tool could search for the different aspects members involving this join point and eventually check the type compatibility of the found aspects members. The developer could then be automatically warned for each join point involved in several aspect members, with particular warning messages about the possible incompatibility problems due to the involved aspects members type (as we described it in the table[50] representing the possible conflicts related to aspect member types).

The last enhancement we propose is more ambitious. The idea is to automatically generate aspect code out of the information contained in aspects design tables. In the case of AspectJ, we can imagine a tool that browses the different aspects of the table and creates the code corresponding to the aspect declaration, and also the structure of its aspect members (declaring in this aspect introduce or advise members, or aspects methods, variables and constructors) with the expression of their join points. This would provide an efficient way to simplify the transposition of aspects design to aspects implementation.

In the short period of this thesis it was unfortunately not possible to realize these different improvements for our aspect-oriented development support. However by keeping in touch with the members of the RECSINWIN/RECINTERNET project team, and by following the future evolutions of the RECINTERNET project we will continue on working and ameliorating the original support we provided for aspect-oriented development.

[50] Table 6 in Section III.2.3.3

To put it in a nutshell we have developed a methodology and its associated representation to simply organize the two dimensions (aspects and components) of an aspect-oriented design. We illustrated its benefits for design and implementation in the concrete case of an industrial application development. This simple proposition is bound to evolve in order to provide an efficient solution for designing and implementing systems in the emerging aspect-oriented paradigm, as this one is concretely based on components implementation and does not provide yet efficient techniques and tools for design.

To conclude, even with the particular constraints of realizing this thesis in the same time as a Master of Science thesis and as an industrial engineering project, we have achieved an intensive and important work during these 6 months of thesis. We believe that this thesis was realized adopting a rigorous scientific process so that we could conciliate in a same “real-world” project research aspects resulting in original propositions and concrete industrial development for a project which will be continued and also useful for Brazilian social welfare.

FIGURES INDEX

TABLES INDEX

REFERENCES

Separation of Concerns and Aspect Oriented Programming

[AJ] AspectJ^{TM} Home Page. Xerox Parc Corporation
http://www.parc.xerox.com/spl/projects/aop/aspectj

[AJPrimer] The AspectJ^{TM} Primer. A Practical Guide for Programmers. Xerox Parc Corporation
http://www.parc.xerox.com/spl/projects/aop/aspectj/primer

[Aop] Aspect Oriented Programming Home Page. Xerox Parc Corporation
http://www.parc.xerox.com/spl/projects/aop

[AT98] M. Aksit and B. Tekinerdogan, *Solving the Modeling Problems of Object-Oriented Languages by Composing Multiple Aspects Using Composition Filters,* AOP'98 workshop position paper, 1998.

[AW99] Veronica Argañaraz and Thomas Wallet. *Aspect Oriented Programming vs. Subject Oriented Programming,* in Separation of concerns workshop, organized by Carine Lucas, for the EMOOSE Master. Nantes, February 1999.

[Ber94] L. Bergmans, *The Composition Filters Object Model,* Dept. of Computer Science, University of Twente, 1994.

[Beu99] Antoine Beugnard. *How to make aspects re-usable, a proposition.* Published in proceedings of ECOOP'99. Lisbon, June 1999

[Cza98] K. Czanercki. *Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Components Models (Chapter 7).* Ph.D. Thesis, Technische Universität Ilmenau. Germany, 1998

[DVDH99] Kris De Volder and Theo D'Hondt. *Aspect-Oriented Logic Meta Programming.* Published in proceedings of Reflection'99, St Malo, July 1999.

[FS99] Andrés Farías and Mario Südholt. *Definition of the security aspect in Java.* Thesis report of the EMOOSE Master. Ecole des Mines de Nantes, August 1999.

[HL95] Walter L. Hürsch and Cristina Videira Lopes. *Separation of Concerns.* Northeastern University technical report NU-CCS-95-03, Boston, February 1995

[Kai 98] Kai Böllert. *Aspect-Oriented Programming. Case Study: System Management Application.* Diplom-Wirtschaftsinformatiker (FH) Thesis. Flensburg University. November 1998.

[Kic98] Gregor Kiczales. *Aspect-oriented Programming: Going Beyond Objects for Better Separation of Concerns in Design and Implementation.* Aspect-Oriented Programming talk, 1998
(http://www.parc.xerox.com/spl/projects/aop/invited-talk)

[KLM+97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Christina Videira Lopes, Jean-Marc Loingtier, John Irwin. *Aspect Oriented Programming.* Published in proceedings of ECOOP'97, Finland, June 1997.

[Lam99] John Lamping. *The role of the base in aspect oriented programming.* Published in proceedings of ECOOP'99, Lisbon, June 1999

[Lie92] Karl J. Lieberherr. *Component Enhancement: An Adaptive Reusability Mechanism for Groups of Collaborating Classes,* in Component Enhancement: Information Processing '92, 12th World Computer Congress, pages 179-185, J. Van Leeuween. Madrid, 1992.

[LK97] Christina Videira Lopes and Gregor Kiczales. *D: A Language Framework for Distributed Programming.* Xerox Palo Alto Research Center. Technical report SLP97-009, P9710044. February 1997.

[LL96] Christina Videira Lopes and Karl J. Lieberherr. *AP/S++: Case-study of a MOP for Purposes of Software Evolution.* Published in proceedings of Reflection'96.

[LO97] Karl J. Lieberherr and Doug Orleans, Preventive Program Maintenance in Demeter/Java (Research Demonstration), in International Conference on Software Engineering, pages 604-605, ACM Press. Boston 1997.

[MLTK97] K. Mens, C. Lopes, B. Tekinerdogan and G. Kiczales. *Aspect Oriented Programming.* In Jan Bosch and Stuart Mitchell, editors. ECOOP 97 Workshop Reader, Lecture Notes in Computer Science, pp 483-496. Springer Verlag 1997.

[OHBS94] Harold Ossher, William Harrison, Frank Budinsky, and Ian Simmonds, *Subject-Oriented Programming: Supporting Decentralized Development of Objects,* Proceedings of the 7th IBM Conference on Object-Oriented Technology, July, 1994

[OKK+96] H. Ossher, M. Kaplan, A. Katz, W. Harrison, V. Kruskal, *Specifying Subject-Oriented Composition,* Theory and Practice of Object Systems, volume 2, number 3, 1996, Wiley & Sons

[SW96] R.J. Stroud and Z. Wu. *Using Metaobject Protocols to Satisfy Non-Functional Requirements,* Technical Report 533, Department of Computing Science, University of Newcastle upon Tyne, 1995.

Three-tiers architectures

[C/S] Client/Server Software Architectures – An Overview. *Software Engineering Institute. 1999*
http://www.sei.cmu.edu/str/descriptions/clientserver_body.html

[GR96] J. Gallaugher and S. Ramanathan. *Choosing a Client/Server Architecture. A Comparison of Two-Tier and Three-Tier Systems.* Information Systems Management Magazine. 1996

[Hun98] Ching-Ho Hung. *The Implementation of Web-Databases by the Approach of Java Database Connectivity: JDBC.* (*Chapter 2: Components and Architecture of Web Database).* Master Thesis. Knowledge Systems Institute. Illinois, 1998

Hypermedia Navigation

[GHJV94] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison Wesley. 1994

[GSP93] F. Garzotto, D. Schwabe, P. Paolini. *HDM – A Model Based Approach to Hypermedia Application Design.* ACM Transaction on Information Systems. Vol. 11, pp 1-26. 1993

[ISB95] T. Isakowitz, E. Stohr and P. Balasubramaniam. *RMM, A methodology for structured hypermedia design.* Comm ACM., pp 34-48. August 1995

[LRS98] F. Lyardet, G. Rossi, D. Schwabe. *Using Design Patterns in Educational Multimedia Applications.* Proceedings of EDMedia'98. Freiburg, Germany, 1998

[RSG97] G. Rossi, D. Schwabe and A. Garrido. *Pattern Systems for Hypermedia.* Proceedings of PLOP'97 - University of Illinois, USA, 1997

[SRB96] Daniel Schwabe, Gustavo Rossi and Simone D. J. Barbosa. *Systematic Hypermedia Application Design with OOHDM.* Proceedings of Hypertext'96. Washington DC, USA, 1996

[SR98] Daniel Schwabe and Gustavo Rossi. *Developing Hypermedia Applications using OOHDM.* Workshop on Hypermedia Development Processes, Methods and Models. Hypertext'98. Pittsburgh, USA, 1998

Technical solutions for Three-tiers Architectures

[ActX1] ActiveX Controls. Microsoft COM Technologies Home Page. Microsoft Corporation.
http://www.microsoft.com/com/tech/activex.asp

[ActX2] ActiveX. Active Group Home Page.
http://www.activex.org

[ACW98] David Aries, Laurent Cailleteau and Thomas Wallet. *CORBA et la Migration d'Objets (CORBA and Object Migration).* Computer Sciences Formation Transversal Project. Ecole des Mines de Nantes. Nantes, 1998

[Ada] Ada and the Web and Java. Home page. Kempe Software Capital Enterprises (KSCE).
http://www.adahome.com/Resources/Ada_Java.html

[Applet] Applets. Java^{TM} Technology Home Page. Sun Microsystems, Inc.
http://www.javasoft.com/applets/index.html

[Can98] Marco Cantu. *Dominando o Delphi 4.0 – A Bíblia (Mastering Delphi 4.0 – The Bible), Chapters 23 (Programação para Internet) e 24 (Programando WEB no lado do servidor).* Makron Books; São Paulo, 1998.

[Cgi] A guide to HTML and CGI scripts. Mike Smith. University of Brighton. July 1995.
http://snowwhite.it.brighton.ac.uk/~mas/mas/courses/html/html.html

[Dcom] DCOM. Microsoft COM Technologies Home Page. Microsoft Corporation. 1999 http://microsoft.com/com/tech/DCOM.asp

[DD97a] H. M. Deitel and P. J. Deitel. Java^{TM} How to Program. *Chapter 1: Introduction to Computers and Java Applets*, pp 2-59. Published by Prentice-Hall International, Inc. ISBN 0-13-286163-1. New Jersey, 1997.

[DD97b] H. M. Deitel and P. J. Deitel. Java^{TM} How to Program. *Chapter 16: Networking*, pp 816-859. Published by Prentice-Hall International, Inc. ISBN 0-13-286163-1. New Jersey, 1997.

[FC97] Jim Flynn and Bill Clarke. Visual J++^{TM} – Programando em Java^{TM}. *Parte III: Construindo Programas em Java*, pp 141-167. Published by MAKRON Books do Brasil Editoria Ltda. ISBN 85-346-0805-9. São Paolo, 1997

[GGM97] Jean-Marc Geib, Christophe Gransart and Philippe Merle. *CORBA : Des concepts à la pratique (CORBA: From concepts to practical use).* Published by Editions Masson. ISBN 2225830460. Paris, France, October 1997

[GmG95] James Gosling & Henry McGilton. *The Java(tm) Language Environment: A White Paper*. (*Multithreading chapter*). Sun Microsystems, Inc. 1997 http://www.pas.com.au/java/doc/javawhitepaper_1.html

[Html1] W3C Home Page for HyperText Markup Language. http://www.w3.org/MarkUp

[Html2] HTML Tutorial. Tim Clinton and CultureNet, March 1995 http://www.ffa.ucalgary.ca/cnet/html-course/contents.html

[Jdk] JDK^{TM} 1.2 Documentation on Javasoft Home Page. Sun Microsystems, Inc. 1999 http://www.javasoft.com

[MAB+98] Martin Murhammer, Orcun Atakan, Stefan Bretz, Larry Pugh, Kazunari Suzuki, David Wood. *TCP/IP Tutorial and Technical Overview.* IBM Redbook GG24-3376-05. Part I: Architecture and Core Protocols, pp 1-93. ISBN number 0738412007. 1998

[ND98] Robert Niles and Jeffry Dwight. *CGI em Exemplos*. Translated from *CGI By Examples* by Eduardo Nunes. Makron Books. São Paulo, 1998.

[Odbc] ODBC. Microsoft ODBC Technologies Home Page. Microsoft Corporation. 1999 http://www.microsoft.com/data/odbc

[Omg] Object Management Group Home Page. Object Management Group, Inc. 1998 http://www.omg.org

[Rmi] RMI. The Java^{TM} tutorial. Sun Microsystems, Inc. 1999 http://java.sun.com/docs/books/tutorial/rmi/index.html

[Servl1] Servlets. The Java^{TM} tutorial. Sun Microsystems, Inc. 1999 http://java.sun.com/docs/books/tutorial/servlets/overview/index.html

[Servl2] Java Web Server. JavaServer Group Home Page. Sun Microsystems, Inc. 1999 http://jserv.javasoft.com/index.html

[Stk] HP Distributed Smalltalk. Hewlett-Packard Company Home Page. http://www.hp.com

[Tan89] Andrew S. Tanenbaum. *Computer Networks (Chapter 10).* Published by Prentice Hall. ISBN 0-13-162959-X. New Jersey, 1989

[Wil95] Arthur Wilson. *An introduction to CGI Scripts and HTML Forms.* Workbook. University of Edinburgh. November 1995

[Zei99] Stefan Zeiger. *Servlets Essentials.* April 1999 http://www.novocode.com/doc/servlet-essentials

Miscellaneous

[Lim95] J. C. M Lima. *Knowledge representation in software packages aimed to know about their users.* In "Advances in Database and Expert Systems". IIAS Editions, ISBN 0921836228, Windsor, Canada, pp 110-114, 1995.

[Lim97] J. C. M. Lima. *Teaching intelligently by computers: a formal model based on an object notation*. Universidade Estadual do Norte Fluminenese, Centro de Ciência e Tecnologia, Research report 12. 97, Campos, Brasil, 1997.

[RLKS98] A. Romanczuck-Requile, J. C. M. Lima, C. Kaestner, E. Scalabrin. *A Contextual Help System Based on Intelligent Diagnosis Processing Aiming to Design and Maintain Object-Oriented Packages.* In "Lecture Notes in Computer Science", ISBN 3-540-65460-7 1543, Springer-Verlag, pp. 64-65, 1998.

[RSW99] *Projeto Lógico/Físico de Modernização do Sintese para Windows (Logical/Physical Project of the Sintese Modernization for Windows).* Internal Technical Report. DATAPREV. Brazil, January 1999.

APPENDIX A – RECSINWIN GRAPHICAL RELATIONAL MODEL

This appendix presents the graphical relational model used to describes the tables structure used for the local structural database of the RECSINWIN application (adapted from [RSW99]).

Fig. 2.1.1

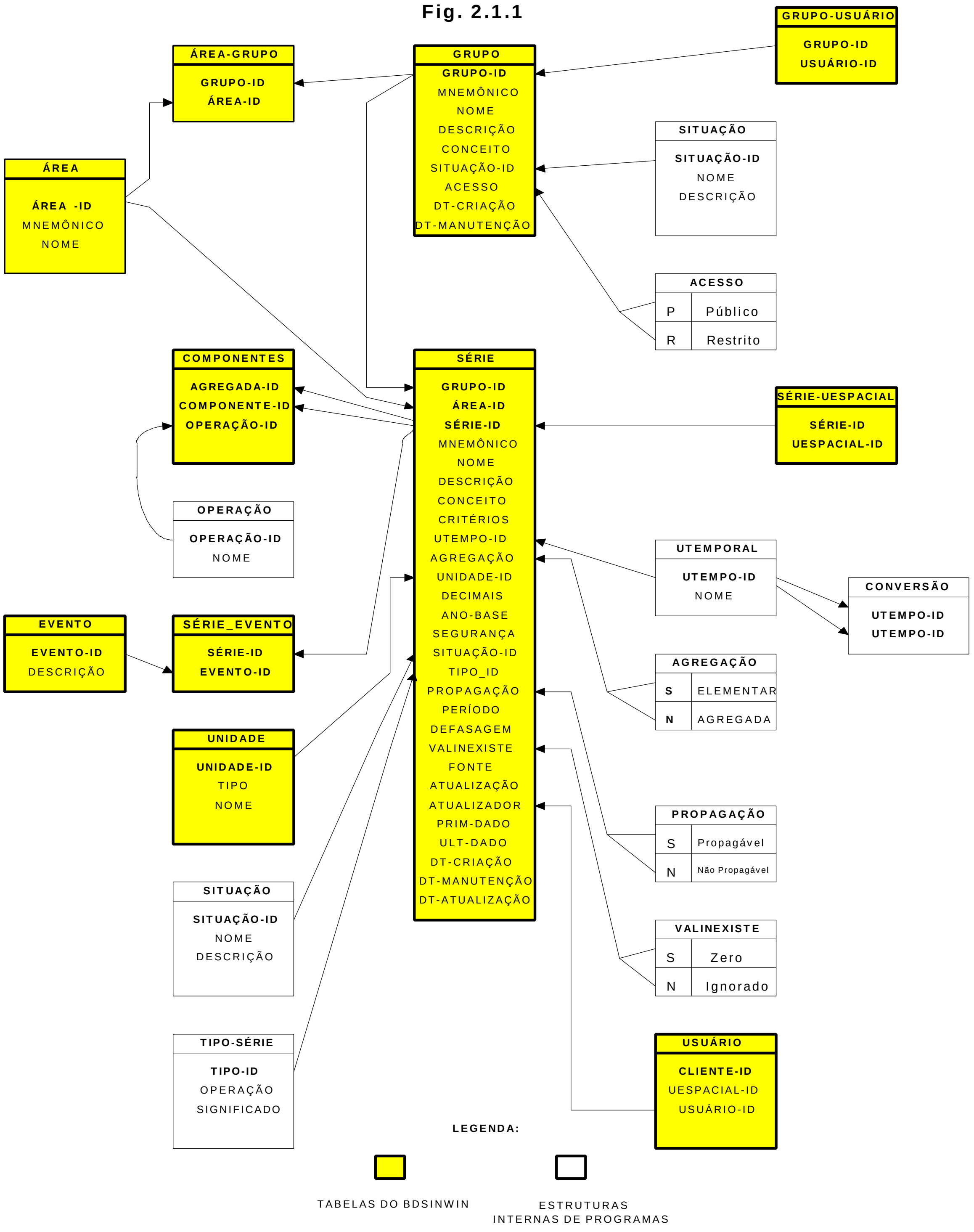


Fig. 2.1.2

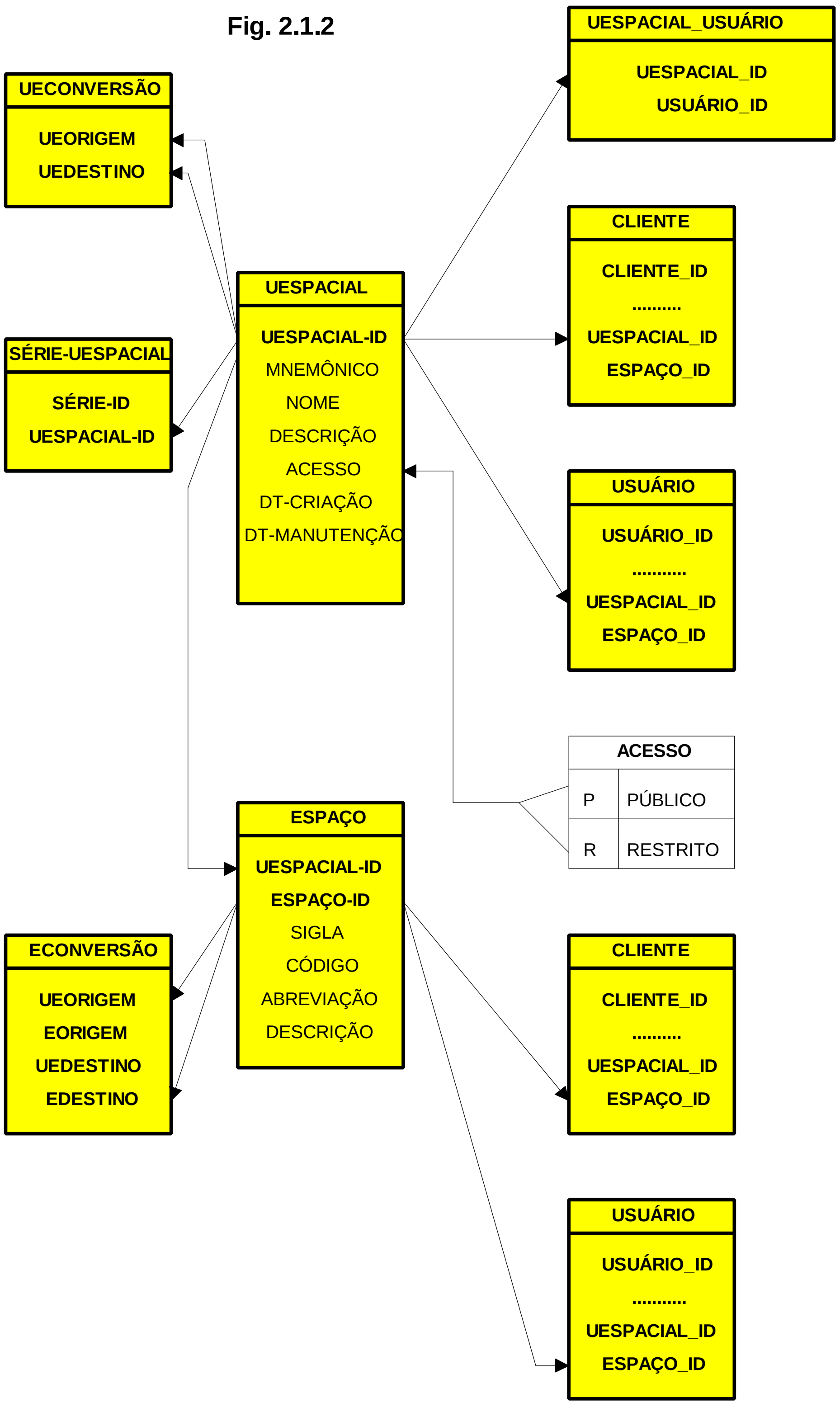
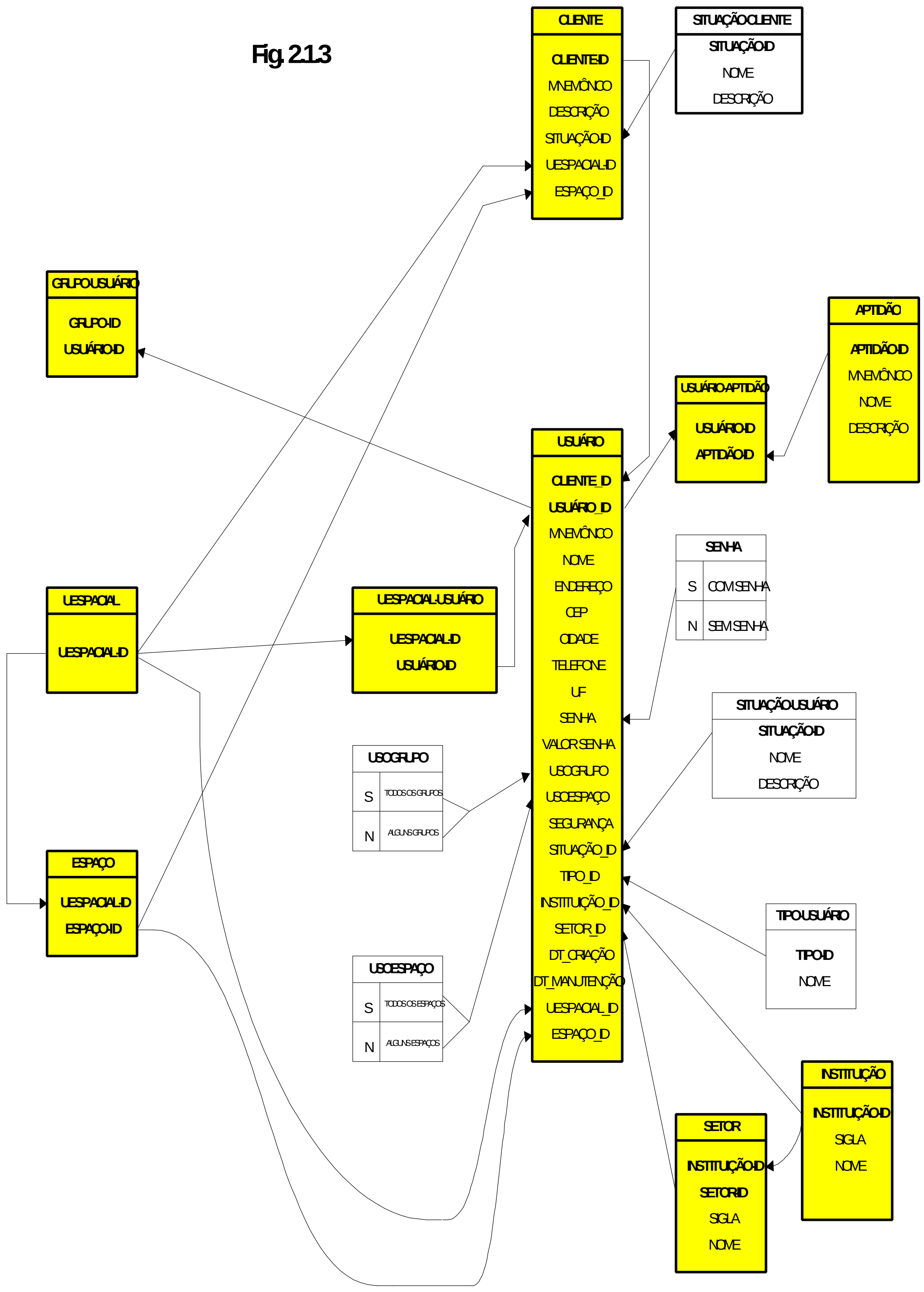

UESPACIAL_USUÁRIO
UESPACIAL_ID
USUÁRIO_ID
UECONVERSÃO
UEORIGEM
UEDESTINO
CLIENTE
CLIENTE_ID
....
UESPACIAL_ID
ESPAÇO_ID
UESPACIAL
UESPACIAL-ID
MNEMÔNICO
NOME
DESCRIÇÃO
ACESSO
DT-CRIAÇÃO
DT-MANUTENÇÃO
SÉRIE-UESPACIAL
SÉRIE-ID
UESPACIAL-ID
USUÁRIO
USUÁRIO_ID
....
UESPACIAL_ID
ESPAÇO_ID
ACESSO
P PÚBLICO
R RESTRITO
ESPAÇO
UESPACIAL-ID
ESPAÇO-ID
SIGLA
CÓDIGO
ABREVIAÇÃO
DESCRIÇÃO
ECONVERSÃO
UEORIGEM
EORIGEM
UEDESTINO
EDESTINO
CLIENTE
CLIENTE_ID
....
UESPACIAL_ID
ESPAÇO_ID
USUÁRIO
USUÁRIO_ID
....
UESPACIAL_ID
ESPAÇO_ID

Fig. 21.3


CLIENTE
CLIENTE-ID
MNEMÔNICO
DESCRIÇÃO
SITUAÇÃO-ID
UESPACIAL-ID
ESPAÇO_ID
SITUAÇÃO CLIENTE
SITUAÇÃO-ID
NOME
DESCRIÇÃO
GRUPO USUÁRIO
GRUPO-ID
USUÁRIO-ID
APTIDÃO
APTIDÃO-ID
MNEMÔNICO
NOME
DESCRIÇÃO
USUÁRIO-APTIDÃO
USUÁRIO-ID
APTIDÃO-ID
USUÁRIO
CLIENTE_ID
USUÁRIO_ID
MNEMÔNICO
NOME
ENDEREÇO
CEP
CIDADE
TELEFONE
UF
SENHA
VALOR SENHA
USO GRUPO
USO ESPAÇO
SEGURANÇA
SITUAÇÃO_ID
TIPO_ID
INSTITUIÇÃO_ID
SETOR_ID
DT_CRIAÇÃO
DT_MANUTENÇÃO
UESPACIAL_ID
ESPAÇO_ID
SENHA
S COM SENHA
N SEM SENHA
UESPACIAL
UESPACIAL-ID
UESPACIAL-USUÁRIO
UESPACIAL-ID
USUÁRIO-ID
USO GRUPO
S TODOS OS GRUPOS
N ALGUNS GRUPOS
SITUAÇÃO USUÁRIO
SITUAÇÃO-ID
NOME
DESCRIÇÃO
ESPAÇO
UESPACIAL-ID
ESPAÇO-ID
USO ESPAÇO
S TODOS OS ESPAÇOS
N ALGUNS ESPAÇOS
TIPO USUÁRIO
TIPO-ID
NOME
INSTITUIÇÃO
INSTITUIÇÃO-ID
SIGLA
NOME
SETOR
INSTITUIÇÃO-ID
SETOR-ID
SIGLA
NOME

APPENDIX B – THE *NODE AS A NAVIGATIONAL VIEW* HYPERMEDIA SYSTEM PATTERN

This description is extracted from [RSG97].

Node as a Navigational View

- ***Problem:*** How to add navigation capabilities to the components of an object-oriented (OO) application, therefore adding hypermedia functionality to the application ?

- ***Forces:***

  - We want to add hypermedia functionality to existing applications
  - Original interface behavior must be preserved, and hypermedia behavior must be added
  - Modifications of objects in the original application is undesirable
  - Redefining the original GUI to include hypermedia capabilities (bookmarks, backtracking, history maintenance, to name a few) is undesirable and many times unfeasible
  - It is difficult to include dynamic links to the existing interface

- ***Solution:*** Define a navigational layer between the application to be enhanced and its graphical interface, build up of objects' observers are called *nodes.* Implement the navigational behavior in nodes. Then define each node's GUI adding means of activating node's behavior.

This solution implies defining a hypermedia node as dependent of an object or group of objects, thus separating hypertext functionality from the behavior of the application and its interface.

- ***Known uses:*** The OOHDM methodology defines the concepts of Node as a navigational view over a conceptual model [Schwabe96]. In [Bieber95] the authors present an architecture for adding hypertext functionality, where nodes are defined as representations of the objects of interest to the application. A similar approach has been used in the Devise Hypermedia Model [Gronbaek94b].

- ***Related patterns:*** Navigation is performed by links *(Link as a Relationship View* pattern) and activated by anchors *(Anchor* pattern). Nodes are Observers [Gamma94].

- ***References:***

[Bieber95] M. Bieber and C. Kacmar. "Designing Hypertext Support for Computational Applications". *Communications of the ACM 38* (8), August, 1995

[Gamma94] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison Wesley. 1994

[Gronbaek94b] K. Gronbaek. "Composites in a Dexter-Based Hypermedia Framework". *Proceedings of the European Conference on Hypermedia Technology,* ECHT'94, Edinburgh, Scotland, September 1994, pp. 59-69.

[Schwabe96] Daniel Schwabe, Gustavo Rossi and Simone D. J. Barbosa. *Systematic Hypermedia Application Design with OOHDM.* Proceedings of Hypertext'96. Washington DC, USA, 1996

APPENDIX C – NAVIGATIONAL FRAMEWORK CLASSES DESCRIPTION

This appendix describes the classes of the navigational framework presented in Part III.3.1.2. Some methods are described in Java to explain how they should be implemented (even if it could be in another programming language).

Panel class

This abstract class is used to model a frame that contains some components.

Class Variables

- context: a public `Context`. Will be used by the instances of the subclasses (`Node` and `Decorator`) for their initializing.

Instance Variables

- components[ ]: a private `Vector` of `Component`. Used to reference the different components of the Panel.
- name: a public `String`

Methods

- reset( ): a public abstract method that will be overridden in he subclasses to provide a way to initialize (or re-initialize) the elements of the container.
- show( ): a public abstract method which will be overridden to make visible the panel
- hide( ): a public abstract method which will be overridden to make invisible the panel
- activate( ): a public method used to make accessible the panel. The body of this method depends of the programming language and of the interface kind.
- desactivate( ): a public method used to make inaccessible the panel. The body of this method depends of the programming language and of the interface kind.
- AddComponent(Component c): a public method used to add a component to the panel.
-

Node class

This abstract subclass of `Panel` is used to model perceivable navigational units. A node will contain different components or decorators.

Instance Variables

- decorators[ ]: a private `Vector` of `Decorator`. Here are stored the different decorators applied to the node.

Methods

- reset( ): this public template method is used to initialize or re-initialize the elements composing the node (decorators and components), thanks to the context.
-

```
public void reset() {
     for (int i=0, i<decorators.size(), i++)
          decorators[i].reset();       //
Decorators initialisation
     }
     componentsReset();    // Components
initialisation
}
```

- resetComponents( ): this abstract public method will be overridden in the subclasses to describe the way the components of the node are initialized.

- addDecorator(Decorator d): this public method is used to add a decorator to the node.

Decorator class

This abstract subclass of `Panel` is used to model node's decorators. This class will be sub-classed for each new kind of decorator.

NavigNode class

This abstract subclass of Node is used to model the navigational nodes. This class will be sub-classed in the application for each new navigational node.

Methods

- show( ): a public method which make visible the navigational node. The body of this method depends of the programming language and of the interface kind.
- hide( ): a public method which make invisible the navigational node. The body of this method depends of the programming language and of the interface kind.

FrameNode class

This abstract subclass of `Node` is used to model both intra and extra nodes. Concretely these nodes will appear as a new frame on top of the current navigational node.

Instance variable

- frame: a private `Frame` used to show the node

Methods

- showFrame(): a protected method used to show the frame. Will be called by the subclasses `show` method. s
- closeFrame(): a protected method used to close the frame. Will be called by the subclasses `hide` method.
- addComponent(Component c): a public method used to add a component to the frame

IntraNode class

This abstract subclass of `FrameNode` is used to model the intra nodes. This class will be sub-classed in the application for each new intra node.

Methods

- show( ): a public method which makes visible the intra node. The body of this method depends of the programming language and of the interface kind.
- hide( ): a public method which make invisible the intra node. The body of this method depends of the programming language and of the interface kind.

ExtraNode class

This abstract subclass of Node is used to model the extra nodes. This class will be sub-classed in the application for each new extra node.

Methods
- show( ): a public method which makes visible the intra node. The body of this method depends of the programming language and of the interface kind.
- hide( ): a public method which make invisible the intra node. The body of this method depends of the programming language and of the interface kind.

Component class

This abstract class is used as a super class of all the different graphical components that will be used.

Methods
- show(): a public method to make visible the component.
- hide(): a public method to make invisible the component.
- activate() : a public method to make available the component.
- desactivate(): a public method to make unavailable the component.

Anchor class

This subclass of Component is used to encapsulate a navigational link. It will be instantiated for each new anchor of each node.

Instance Variables
- link: a private `Link`

Methods
- proceedLink(): a public method that calls the `navigate` method of the link.

Link class

This class is used to model navigational links. Each link of an anchor will be a new instance of this class, with its own target and one of the possible transformations.

Instance Variables
- target: a private `Target`
- transformation: a private `Transformation`

Methods
- navigate( ): a public method used to define the navigation.

```
void navigate(){
     String targetName = target.getTarget();
     transformation.action(targetName);
}
```

Target class

This class is used to define the target of a navigation.

Instance Variables
- target: a private String which is the name of the target.

Methods
- getTarget( ): this public method return the name of the target.

DynamicTarget class

This abstract class is used to model a `Target` whose target name will be define dynamically in the navigation time. This dynamicity generally requires to use the context. This class will be sub-classed for each new dynamic target.

Class Variables
- context: a private static `Context`. It will be used to define dynamically the target name.

Methods
- ***getTarget( ): public template method***

```
public String getTarget() {
     defineTarget();
     return target;
}
```

- defineTarget( ): public abstract method that will be overridden for each specific dynamic target. Here should be defined the way a target is dynamically created.

Transformation class

This abstract class is used to model the actions to do for a navigation between two nodes.

Class Variables
- context: a private static `Context`. It will be used in the different actions.

Methods
- action(String target): a template public method.
-

```
public void action(String target) {
     Node targetNode =
context.getNode(target);
     specificAction(target);
     targetNode.show();
}
```

- specificAction(String target): an abstract protected method that will be overridden to define actions specific to some kinds of transformations.

NavigTransformation class

This class represents transformations happening when navigating from a navigational node to another one.

Methods
- specificAction(String target): this method describes the specific actions realized for this kind of navigation:

```
protected void specificAction(String target)
{
       context.getCurrentNode().hide();
       context.setCurrentNode(context.getNode
(target));
       context.getCurrentNode().reset();
}
```

IntraTransformation class

This class represents transformations happening when navigating from a navigational node to an intra node.

Methods

- specificAction(String target): this method describes the specific actions realized for this kind of navigation:

```
protected void specificAction(String target)
{
      context.getCurrentNode().desactivate();
      context.getNode(target).reset()
}
```

ExtraTransformation class

This class represents transformations happening when navigating from a navigational node to an extra node.

Methods

- specificAction(String target): this method describes the specific actions realized for this kind of navigation:

```
protected void specificAction(String target)
{
        context.getNode(target).reset();
}
```

Context class

This class is used to model the navigational context. This class will be instantiated once for each new user. This object will encapsulate all the needed information about the user's navigation.

Instance Variables

- currentNode: a private `Node` that refers the current node of the navigation.
- nodesManager: a private `NodesManager` that will be used to create new nodes.

Methods

- getCurrentNode(): a public method that return the current node.
- init(): a public method used to initialize the context.
- setCurrentNode(node n): to set the current node.
- getNode(String target): this method will call the `getNode` method of the nodes manager.

CacheContext class

This class is used as a navigational context but also as a cache for nodes. Instead of always creating new nodes, they are cached in this context
and simply re-initialized when needed.

Instances Variables

- navigNodes[ ]: a public vector of `Node`. Here are referenced the navigational nodes that have been created.
- intraNodes[ ]: a public vector of Node. Here are referenced the intra-nodes that have been created.
- extraNodes[ ]: a public vector of Node that references the extra nodes.
- currentExtraNode: a private `ExtraNode` that reference the last extra node reached.

Methods

- addNode(Node n): a public method used to add a node in the cache context. This method uses the class of the node to add it to the appropriate vector (`navigNodes`, `intraNodes` or `extraNodes`).

NodesManager class

This abstract class is used to manage nodes demands. It is used to find an existing node or to create a new node. This class will be overridden to define the nodes creation for a specific application.

Instance Variables

- context: a private `Context`.

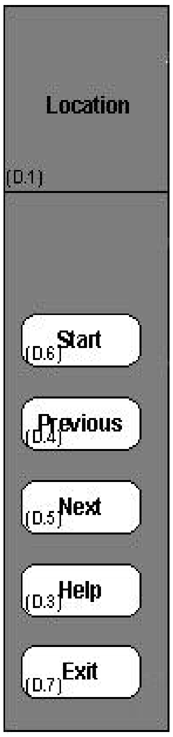
Methods

- getNode(String node): a public method to get a node out of its name. If the context is not a `CacheContext`, this method just calls the `createNode` method. If it is, it will look for the appropriate node in the three nodes vectors of the cache context (`navigNodes`, `intraNodes` or `extraNodes`) to find the wanted node. If this node does not exist, the `createNode` method will be called.
- CreateNode(String node): a protected abstract method that will be overridden to define the creation of nodes out of their name (which is specific to each application).

APPENDIX D – RECINTERNET INTERFACE DESCRIPTION

In this appendix we present the different nodes and decorators used in RECINTERNET. The drawings shows the different elements as well as the way they are organized. The final appearance of the implemented interface can be different from these drawings since we use them just to identify the different components of each node and organize them. The description of each node gives the characteristics of all its components (type, behavior, etc...).

Decorator 0



This decorator will be added to all the navigational nodes. It factor out some more functionalities and information.

- ***(D.1) Location element:*** this element shows the user where he is in his navigation. For that it shows the name of the navigational nodes, and differentiates the current node from the others.
- Type**:** text area
- Modification: impossible
- Visible: always

- ***(D.3) Help anchor:***
- Type**:** anchor
- Visible: always
- Available: always
- Modification: click
- Target: static: *Help extra-node.*
- What for**:** provide appropriate help to the user
- On click**:** navigation to the *Help* extra-node with the information corresponding to the current context. Search by topics and navigation are possible into this help system node.

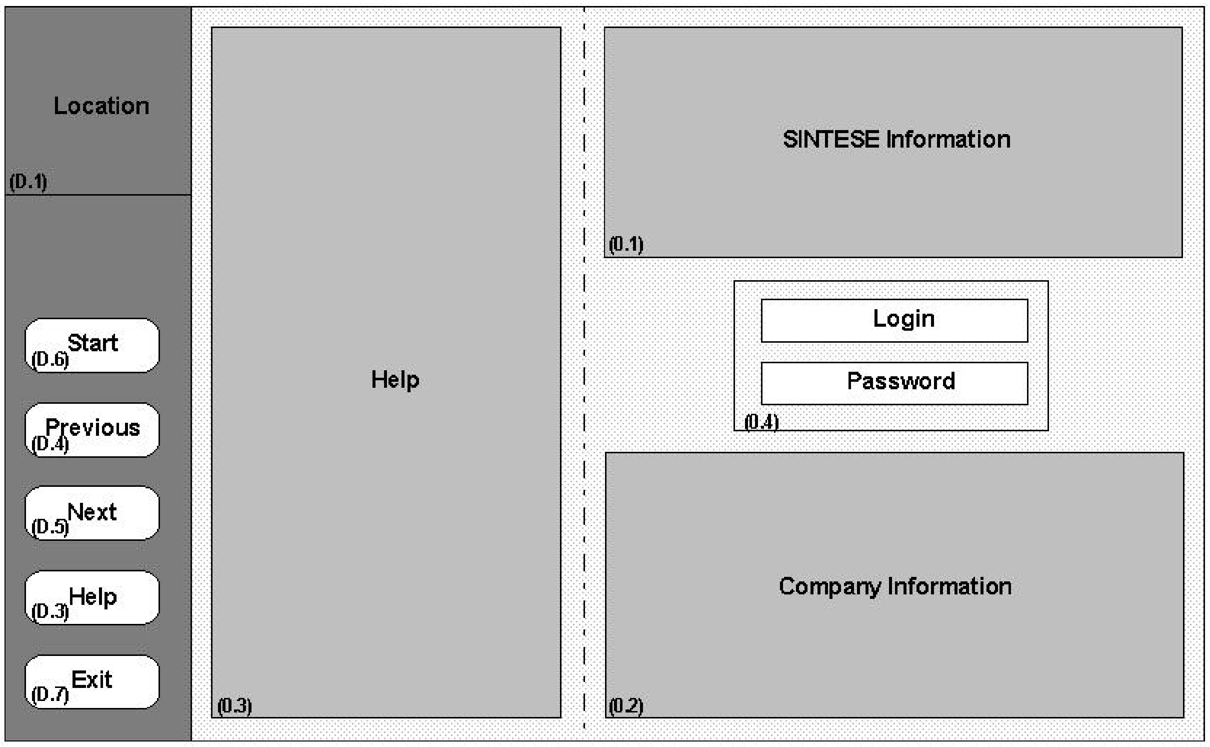
- *(D.4) Previous anchor:*
- Type: anchor
- Visible: always
- Available: in Node 211, 212, 221, 222, 3
- Modification: click
- Target: dynamic: from *Node 211* and *Node 221,* the target is *Node 1.* From *Node 212,* the target is *Node 211.* From *Node 222,* the target is *Node 221. From Node 3,* the target is either *Node 212* or *Node 222,* depending of the node where the user was previously.
- On click: navigation to the previous node (as specified in *Target*). When leaving the origin node, the states of all the elements of the node are stored in a context object. When reaching the target node, the user finds the elements of this node exactly how they were when he quitted this node.

- *(D.5) Next anchor:*
- Type: button
- Visible: always
- Available: only from a node that has been reached with the *Previous* navigation.
- Modification: click
- Target: dynamic: from *Node 1,* the target is either *Node 211* or *Node 221,* depending of the node where the user was previsously. From *Node 211,* the target is *Node 212.* From *Node 221,* the target is *Node 222.* From *Node 212* or *Node 222,* the target is *Node 3.*
- On click: navigation to the next node (as specified in *Target*). When reaching this target node, the user finds the elements of this node exactly how they were when he quitted this node (with the previous anchor).

- *(D.6) Start anchor:*
- Type: button
- Caption: "Start" in Node 0, "Re-start" in Node 1, Node 211, Node 212, Node 221, Node 222 and Node 3.
- Visible: always
- Available: always
- Modification: click
- Target: static: *Node 1*
- On click: navigation

- *(D.7) Exit anchor:*
- Type: button
- Visible: always
- Available: always
- Modification: click
- Target: static: *Node 0*
- On click: navigation

Node 0 – Welcome

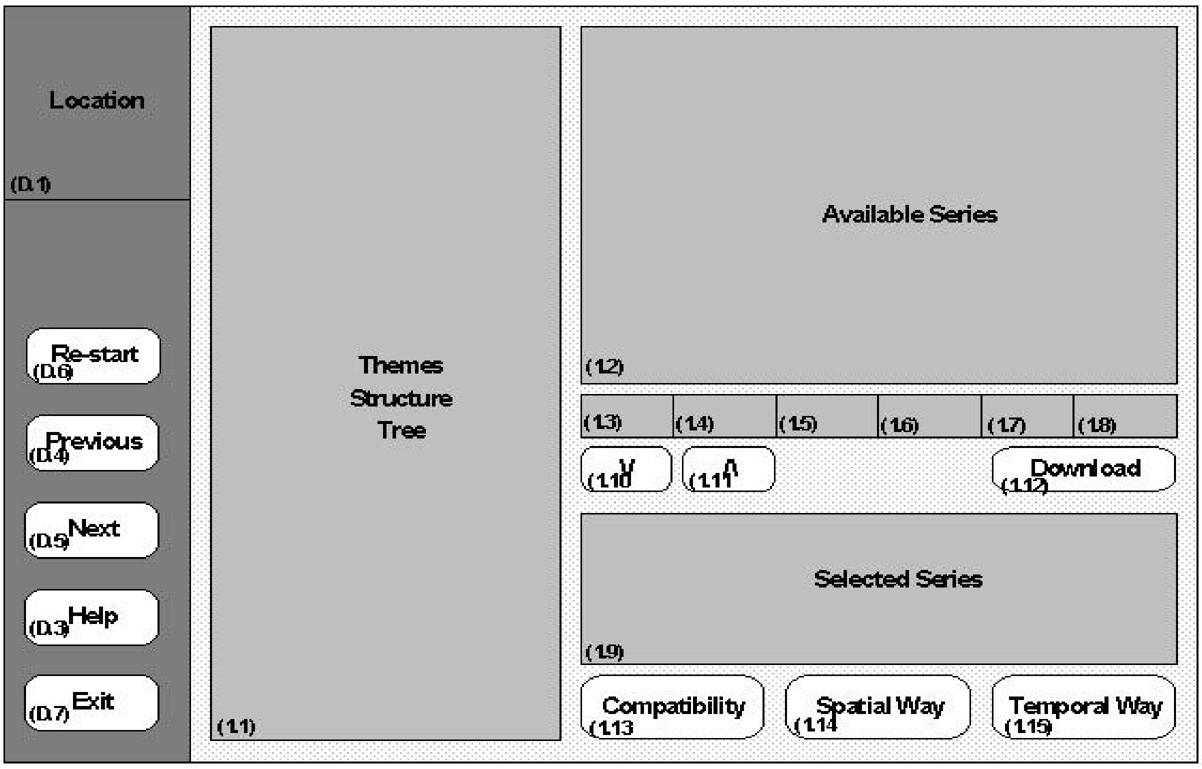


This is the welcoming node.

- *(0.1) SINTESE Information element:* this element provides information about SINTESE
- Type: text area
- Node 0 init: visible
- Modification: impossible

- *(0.2) Company Information element:* this element provides information about the company (logo...)
- Type: text area
- Node 0 init: visible
- Modification: impossible
-
- *(0.3) Help element:* Description of how to use the different commands of the decorator
- Type: text area
- Node 0 init: visible
- Modification: impossible
-
- *(0.4) Login/Password element:* this element enables the user to write his login and his password
- Type: text field + password field
- Node 0 init: visible
- Node 0 default value: empty and empty
- Modification: write
- Available: always

Node 1 – Series Selection



In this node, the user will be able to choose a subset of series.

Data elements

- *(1.1) Themes Structure Tree element:* this element is an arborescent list of the different themes of series. They are organized in directories grouping themes of series. This structure is extracted from the database of the web server.
- Type: arborescent list
- Node 1 init: visible
- Node 1 init values: the different directories grouping themes of series
- Node 1 init expansion: collapsed
- Modification: extend or collapse directory, selection
- Available: always
- Selection: one item or one directory
- On selection update:
- If a directory is selected, the *Available Series element (1.2)* is set to empty.
- If an item is selected, the *Available Series element (1.2)* is filled in with the available series corresponding to the selected theme.

- *(1.2) Available Series element:* in this element appear the different series available for the theme selected in the *Themes Structure Tree element (1.1)*.
- Type: list
- Node 1 init: visible
- Node 1 init values: empty
- Modification: selection
- Available: when not empty
- Selection: one item
- On selection update: the First, Second, Third Dimension Space Unit elements (1.3 to 1.5), the Time Unit element (1.6), the First Data element (1.7) and the Last Data element (1.8) are filled in with the information corresponding to the selected serie.

- (1.3 to 1.5) First, Second and Third Dimension Space Unit elements: these element provides the space units of the selected serie
- Type: 3 text fields
- Node 1 init: visible
- Modification: impossible

- *(1.6) Time Unit element:* this element provides the time unit of the selected serie
- Type: text field
- Node 1 init: visible
- Modification: impossible

- *(1.7) First Data element:* this element provides the date of the first data of the selected serie
- Type: text field
- Node 1 init: visible
- Modification: impossible

- *(1.8) Last Data element:* this element provides the date of the last data of the selected serie
- Type: text field
- Node 1 init: visible
- Modification: impossible

- *(1.9) Selected Series element:* in this element appear the different series that the user have chosen.
- Type: list

- Node 1 init: visible - empty
- Modification: selection
- Available: when not empty
- Selection: one item
- On selection update: nothing

Functional elements

- *(1.10) Select Serie element:*
- Type: button
- Node 1 init: visible
- Modification: click
- Available: when a serie is selected in the *Available Series Element (1.2)*
- On click: the selected serie of the *Available Series element (1.2)* is added to the *Selected Series element (1.9)* if it was not already in.

- *(1.11) Unselect Serie element:*
- Type: button
- Node 1 init: visible
- Modification: click
- Available: when a serie is selected in the *Selected Series Element (1.9)*
- On click: the selected serie of the *Selected Series Element (1.9)* is removed.

- *(1.12) Download element:*
- Type: button
- Node 1 init: visible
- Modification: click
- Available: when the *Available Series Element (1.2)* is not empty.
- Function: provide the user a text file containing the list of the available series for the selected theme.
- On click: starts a web browser classical downloading process.

Intra-navigational elements

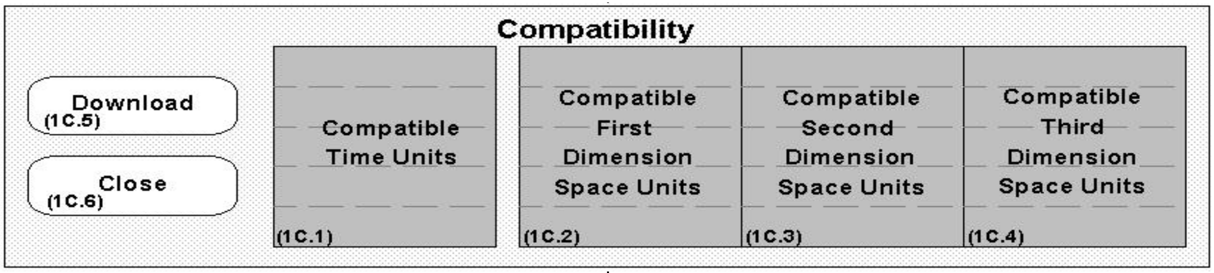
- *(1.13) Detail Compatibility element:*
- Type: anchor
- Node 1 init: visible
- Modification: click
- Available: when the *Selected Series Element (1.9)* is not empty.
- Target: static: Node 1 + Series Compatibility
- On click: navigation

Navigational elements

- *(1.14) Spatial Way element:*
- Type: anchor
- Node 1 init: visible
- Modification: click
- Available: when the *Selected Series Element (1.9)* is not empty, and when we have space compatibility between the selected series
- Target: static: *Node 211*
- On click: navigation

- *(1.15) Temporal Way element:*
- Type: anchor
- Node 1 init: visible
- Modification: click
- Available: when the *Selected Series Element (1.9)* is not empty
- Target: static: *Node 221*
- On click: navigation

Node 1C – Series Compatibility Details


Compatibility
Download (1C.5)
Close (1C.6)
Compatible Time Units (1C.1)
Compatible First Dimension Space Units (1C.2)
Compatible Second Dimension Space Units (1C.3)
Compatible Third Dimension Space Units (1C.4)

This intra node shows some extra information about the compatibility of the selected series.

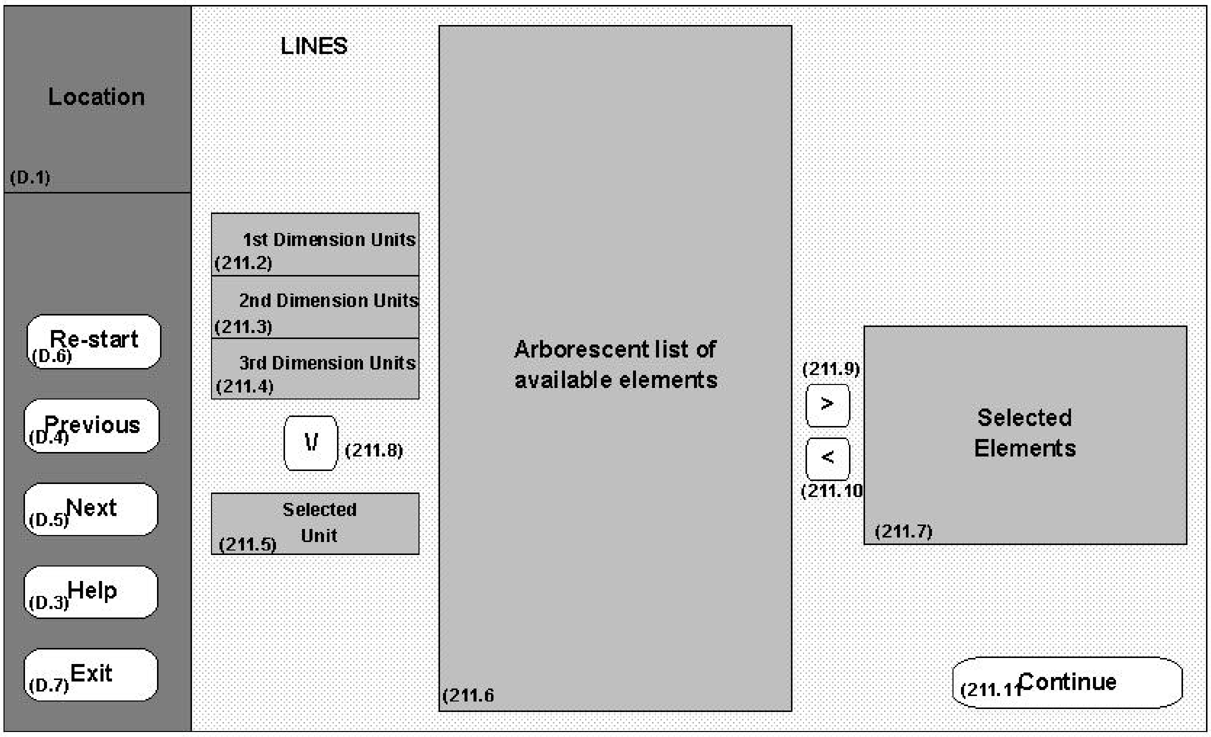
(1C.1) Compatible Time Units element: this element shows which are the compatible time units for the selected series

- Type: set of text fields
- Node 1C init: visible
- Modification: impossible

- (1C.2) to (1C.4) Compatible First, Second and Third Dimension Space Units elements: these elements show which are the compatible space units for the selected series
- Type: 3 sets of text fields
- Node 1C init: visible
- Modification: impossible
-

- *(1C.5) Download element:*
- Type: button
- Node 1C init: visible
- Modification: click
- Available: always
- Function: provide the user a text file containing the compatibility information of the selected series
- On click: starts a web browser classical downloading process.
-
- *(1C.6) OK element:*
- Type: button
- Node 1C init: visible
- Modification: click
- Available: always
- On click: exit from this node to go back to the *Node 1*

Node 211 – Lines Composition – Spatial Way



In this node the user defines what spatial elements will appear in the lines of the result sheet.

Data elements

- *(211.2) to (211.4) First, Second and Third Dimension Units elements:*
- Type: linked lists
- Node 211: visible
- Node 211 init values: for the available element(s), the units (in the corresponding dimension) for which the set of selected series are defined.
- Node 211 init selection: none
- Available: for the element(s) corresponding to the dimensions for which the set of selected series are defined, always. For the other(s), never.
- Modification: selection
- On selection update: when the user selects something in one of the three lists, the two others are set to no selection.

(211.5) Selected Unit element: this element shows the unit chosen by the user.

- Type: text field
- Node 211 init: visible
- Node 211 init default value: nothing
- Modification: impossible

- *(211.6) Arborescent List of Available Elements element:* this element provides the user different possibilities of selections related to the selected unit of the *Selected Unit element(211.5)*
- Type: arborescent list
- Node 211 init: visible
- Node 211 init values: nothing
- Modification: extend or collapse directories, selection
- Available: when not empty
- Selection: multiple selection of items of a same directory. Multiple selection of items from different directories is not possible. Selection of directories is not possible.
- On directory collapse: nothing
- On selection update: nothing
- On directory extension: the directory is open and the items corresponding to the directory (from the web server database) appear

- *(211.7) Selected Elements element:* in this element appear the different elements chosen by the user
- Type: list
- Node 211 init: visible
- Node 211 init values: empty
- Modification: selection
- Available: when not empty
- Selection: one item
- On selection update: nothing

Functional elements

- *(211.8) Select Unit element:*
- Type: button
- Node 211 init: visible
- Modification: click
- Available: when a unit is selected in one of the First, Second and Third Dimension Units elements (211.2 to 211.4)
- On click:
- If the unit of the Selected Unit Element (211.5) is the same that the unit selected in the First, Second and Third Dimension Units elements (211.2 to 211.4), nothing happens.

- Else:

- The Selected Unit element (211.5) field is set to the name of the selected unit in the First, Second and Third Dimension Units elements (211.2 to 211.4)

- The *Arborescent List of Available Elements element (211.6)* is initialised with the items corresponding to the selected time unit and organised in parts (let's suppose for these explanations that the selected time unit is *Posto),* and by default all the directories are collapsed:
- *Part 1:* this part contains only the item "All the *Postos* of Brasil".
- *Part 2:* this part contains a directory called "*Postos* of Brasil". The items of this directory are all the *Postos* of Brasil, ordered alphabetically.
- *Part 3, 4, 5...:* the unit *Posto* can be converted into bigger units (like *Estado, Regiao...).* There will be one part for each of this converted units. Each of these parts will contain one directory. Its name will be for example "All the *Postos* of an *Estado*". The items of this directory will be the different *Estados*. If the user selects for example the state "*Rio de Janeiro"* in this directory, it means the lines will be all the *Postos* of the state *Rio de Janeiro,* one per line.
- When a directory contains more items than a defined limit, this directory will appear, but it will not be possible to open this directory to select some items.

- The Selected Elements element (211.7) is set to empty

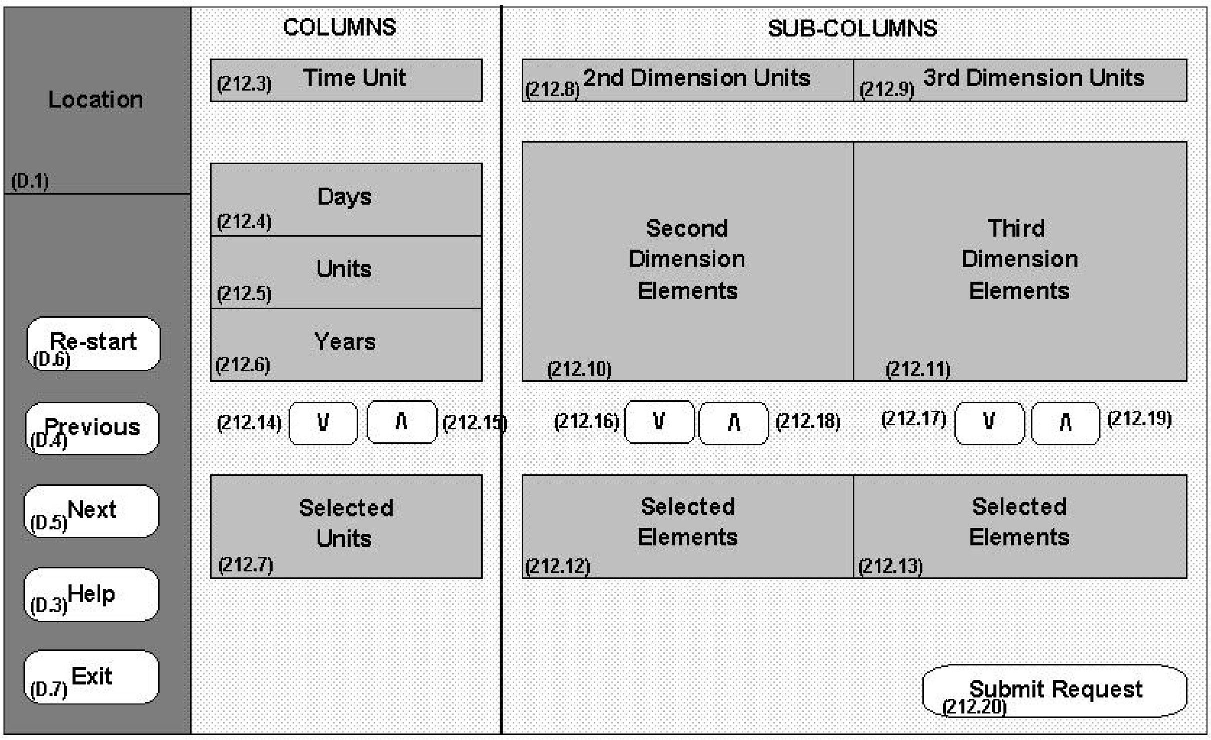
- *(211.9) Select Element element:*
- Type: button
- Node 211 init: visible
- Modification: click
- Available: when at least one element is selected in the *Arborescent List of Available Elements element (211.6)*

- On click:
- If the Selected Elements element (211.7) was empty:
- The selected elements of the Arborescent List of Available Elements element (211.6) are added to the Selected Elements element (211.7)
- Else:
- If the elements of the Selected Elements element (211.7) are from the same directory than the selected elements of the Arborescent List of Available Elements element (211.6):
- The selected elements of the Arborescent List of Available Elements element (211.6) that are different from the elements of the Selected Elements element (211.7) are added to the Selected Elements element (211.7)
- Else:
- The Selected Elements element (211.7) is set to empty, and then the selected elements of the Arborescent List of Available Elements element (211.6) are added to it.
-
- *(211.10) Unselect Element element:*
- Type: button
- Node 211 init: visible
- Modification: click
- Available: when an element of the *Selected Elements element (211.7)* is selected
- On click: the selected serie of the *Selected Elements element (211.7)* is removed.

<u>Navigational elements</u>

- *(211.11) Continue element:*
- Type: anchor
- Node 211 init: visible
- Modification: click
- Available: when the Selected Elements element (211.7) is not empty
- Target: static: *Node 212*
- On click: navigation

Node 212 – Columns & Sub-Columns Composition – Spatial Way



In this node the user defines how will be organized the columns and sub-columns of the result sheet.

Data elements

- *(212.3) Time Unit element:* this element provides the user the possibility to choose between the time units available for the selected series (that are included in: day, week, decade, fortnight, month, bimestre, trimestre, quadrimestre, semestre, year).
- Type: List
- Node 212 init: visible
- Node 212 init default selection: none
- Node 212 init values: the time units for which the set of selected series is defined
- Available: always
- Modification: select one element
- On selection update:
- If the selection is “year”:
- The Days element (212.4) and the Units element (212.5) are set to invisible.
- The Selected Units element (212.7) is initialised to empty.
- Setting of the Years element (212.6):
- Values: years of the union of the periods for which the selected series are defined.
- Selection: none

- Else If the selection is “day”:
- Setting of the Days element (212.4):
- Visible: yes
- Values: integers from 1 to 31
- Selection: none
- Setting of the Units element (212.5):
- Visible: yes
- Caption: months
- Values: the names of the 12 months of the year
- Selection: none
- Setting of the Years element (212.6):
- Values: the years of the union of the periods for which the selected series are defined.
- Selection: none
- The Selected Units element (212.7) is initialised to empty.
-
- Else:
- The *Days element (212.4)* is set to invisible.
- The Selected Units element (212.7) is initialised to empty.
- Setting of the Units element (212.5):
- Visible: yes
- Caption: name of the time unit selected
- Values: integers from 1 to *MAX*, where *MAX* is the number of the time unit selected in a year.
- Selection: none
- Setting of the Years element (212.6):
- Values: the years of the union of the periods for which the selected series are defined.
- Selection: none
-
- *(212.4) Days element:*
- Type: list
- Node 212 init: invisible
- Available: always
- Modification: selection (one item)
- On selection update: nothing
-
- *(212.5) Units element:*
- Type: list
- Node 212 init: invisible
- Available: always
- Modification: selection (one item)
- On selection update: nothing

-
-
- *(212.6) Years element:*
- Type: list
- Node 212 init: visible
- Available: always
- Modification: selection (one item)
- On selection update: nothing
-
- *(212.7) Selected Units element:* in this element appear the different units instances chosen by the user.
- Type: list
- Node 212 init: visible
- Modification: selection
- Available: when not empty
- Selection: one item
- On selection update: nothing

- *(212.8) to (212.9) Second and Third Dimension Units elements:*
- Type: lists
- Node 212 init: visible
- Node 212 init values: for the available element(s), the units (in the corresponding dimension) for which the set of selected series are defined + the item "no selection"
- Node 212 init selection: item "no selection"
- Available: only for the element(s) corresponding to the dimensions for which the set of selected series are defined.
- Modification: selection
- On selection update: the corresponding *Xth Dimension Elements element (212.10 or 212.11)* is filled in with the elements of the web server database corresponding to the selected unit + an item "all elements".
-
- *(212.10) to (212.11) Second and Third Dimension Elements elements:* lists of the available elements corresponding to the selected unit
- Type: lists
- Node 212 init: visible
- Node 212 init values: nothing
- Available: when not empty, and only for the element(s) corresponding to the dimensions for which the set of selected series are defined.
- Modification: selection
- On selection update: nothing
-
- *(212.12) to (212.13) Second and Third Dimension Selected Elements elements:* lists of the elements chosen by the user for each dimension
- Type: list
- Node 212 init: visible
- Modification: selection
- Available: when not empty, only for the element(s) corresponding to the dimensions for which the set of selected series are defined.
- Selection: one item
- On selection update: nothing

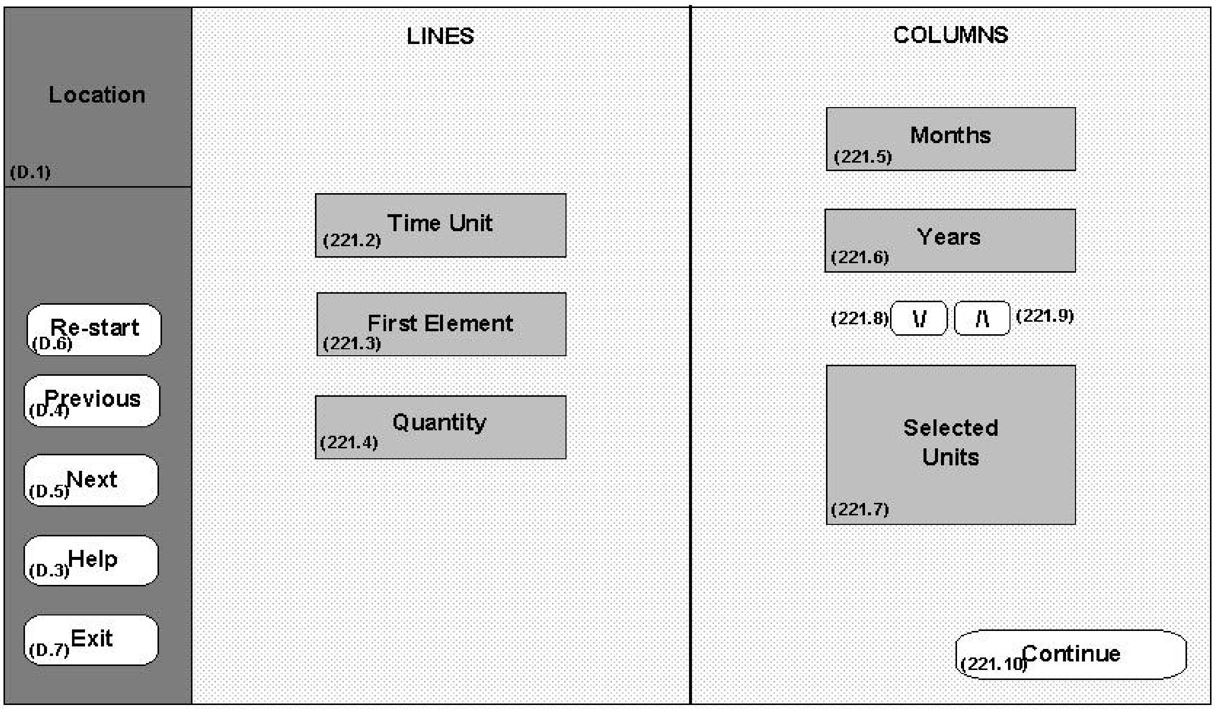
<u>Functionnal elements</u>

- *(212.14) Select Unit element:*
- Type: button
- Node 212 init: invisible
- Modification: click
- Available: if the elements *Days element (212.4), Units element (212.5)* and *Years element (212.6)* that are visible have each one item selected.
- On click: a composition of the selection of the *Days element (212.4)* (if it is visible), the selection of the *Units element (212.5)* (if it is visible) and the selection of the *Years element (212.6)* is constructed and added to the *Selected Units element (212.7)* if it is a valid composition. A composition is valid when the selected day effectively exists in the given month (Months with 30 or 31 days, February with 28 or 29 days).
-
- *(212.15) Unselect Unit element:*
- Type: button
- Node 212 init: invisible
- Modification: click
- Available: when an element of the *Selected Units element (212.7)* is selected
- On click: the selected serie of the *Selected Units element (212.7)* is removed.
-
- *(212.16 to 212.17) Second and Third Dimension Select Element elements:*
- Type: buttons
- Node 212 init: visible
- Modification: click
- Available: if one item is selected in the corresponding *Xth Dimension Elements element (212.10 or 212.11),* and only for the element(s) corresponding to the dimensions for which the set of selected series are defined.
- On click: the selected element of the corresponding Xth Dimension Elements element (212.10 or 212.11) is added to the corresponding Xth Dimension Selected Elements element (212.12 or 212.13) if it was not already in.
-
- *(212.18 to 212.19) Second and Third Dimension Unselect Element elements:*
- Type: buttons
- Node 212 init: visible
- Modification: click
- Available: when an element of the corresponding *Xth Dimension Selected Elements (212.12 or 212.13)* is selected, and only for the element(s) corresponding to the dimensions for which the set of selected series are defined.
- On click: the selected element of the corresponding *Xth Dimension Selected Elements 212.12 or 212.13)* is removed.

<u>Navigational Elements</u>

-
- *(212.20) Submit Request element:*
- Type: anchor
- Node 212 init: visible
- Modification: click
- Available: if:
- The Selected Units element (212.7) contains at least one item
- For each available dimension in the *Sub-Columns Part:*
- The corresponding *Xth Dimension Selected Elements element (212.12 or 212.13)* contains at least one element, except if the selection of the corresponding *Xth Dimension Units element (212.10 or 212.11)* is "no selection".
- Target: static: *Node 3*
- On click: navigation

Node 221 – Lines & Columns Composition – Temporal Way



In this node the user defines what temporal elements will appear in the lines and the columns of the result sheet.

Data elements

- *(221.2) Time Unit element:* this element provides the user the possibility to choose between the time units available for the selected series (that are included in: day, week, decade, fortnight, month, bimestre, trimestre, quadrimestre, semestre, year).
- Type: list
- Node 221 init: visible
- Node 221 init default values: the time units for which all the selected series are defined
- Node 221 init default selection: none
- Available: always
- Modification: select one element
- On selection update:
- If the selection is "year":
- Setting of the First Element element (221.3):
- Visible: yes
- Values: the years of the unions of the periods for which the selected series are defined.
- Selection: the first of these years.
- Setting of the Quantity element (221.4):
- Visible: yes
- Values: integers from 1 to 999
- Selection: 1
- All the elements of the *Columns Part* are set to invisible.
-
- If the selection is not "year":
- If the selection is "day":
- The First Element element (221.3) and the Quantity element (221.4) are set to invisible.
- The Months element (221.5), Selected Units element (221.7), Select element (221.8) and Unselect element (221.9) are set to visible.
- Setting of the Years element (221.6):
- Visible: yes
- Values: the years of the unions of the periods for which the selected series are defined.
- Selection: the first of these years.
-
- If the selection is not "day":
- Setting of the First Element element (221.3):
- Visible: yes
- Values: integers between 1 and *MAX*, where *MAX* is the number of the time unit selected in a year.
- Selection: 1
- Setting of the Quantity element (221.4):
- Visible: yes
- Values: integers from 1 to 999
- Selection: *MAX*
- The *Months element (221.5)* is set to invisible.
- The Selected Units element (221.7), Select element (221.8) and Unselect element (221.9) are set to visible.
- Setting of the Years element (221.6):
- Visible: yes
- Values: the years of the unions of the periods for which the selected series are defined.
- Selection: the first of these years.

- *(221.3) First Element element:*
- Type: list
- Node 221 init: visible
- Node 221 init default value: none
- Available: when something different from "day" is selected in the *Time Unit element (221.2)*
- Modification: selection
- On selection update: nothing

- *(221.4) Quantity element:*
- Type: list

- Node 221 init: visible
- Node 221 init default value: none
- Available: when something different from "day" is selected in the *Time Unit element (221.2)*
- Modification: selection
- On selection update: nothing

- *(221.5) Months element:*
- Type: list
- Node 221 init: invisible
- Node 221 init default values: the 12 months of the year
- Node 221 init default selection: none
- Available: always
- Modification: selection (one item)
- On selection update: nothing

- *(221.6) Years element:*
- Type: list
- Node 221 init: visible
- Node 221 init default values: none
- Node 221 init default selection: none
- Available: when something different from "year" is selected in the *Time Unit element (221.2)*
- Modification: selection (one item)
- On selection update: nothing

- *(221.7) Selected Units element:* in this element appear the different units instances chosen by the user.
- Type: list
- Node 221 init: visible
- Modification: selection
- Available: when not empty
- Selection: one item
- On selection update: nothing

<u>Functional elements</u>

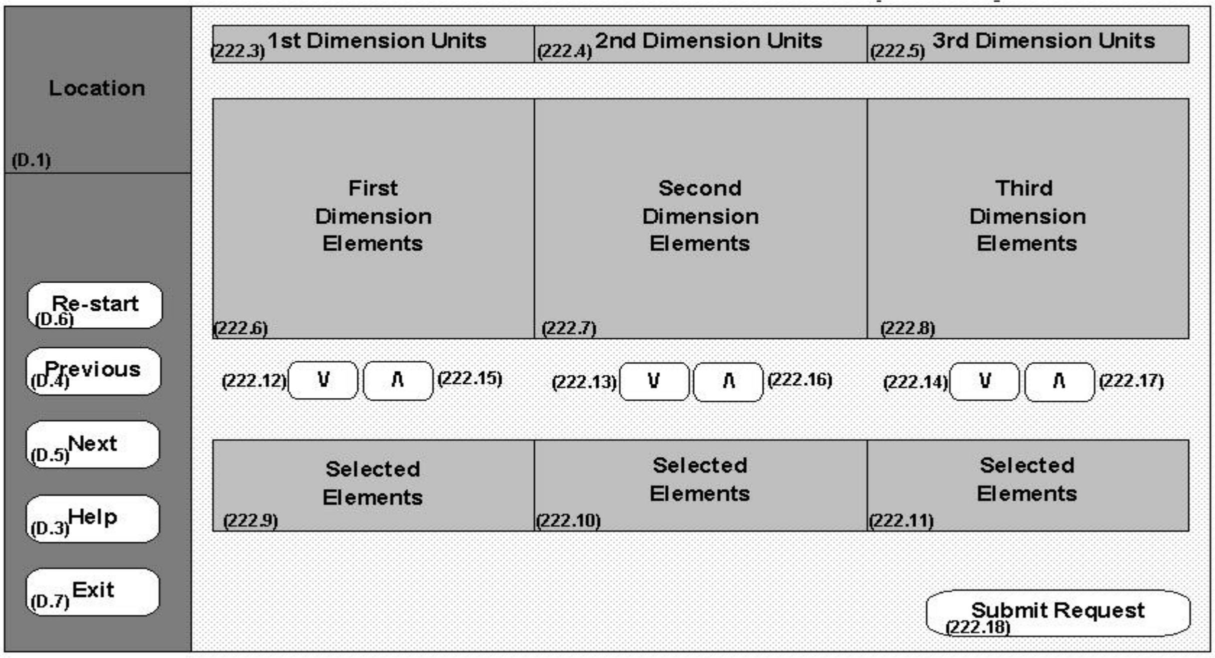
- *(221.8) Select Unit element:*
- Type: button
- Node 221 init: visible
- Modification: click
- Available: if the elements *Months element (221.5)* and *Years element (221.6)* that are visible have each one item selected
- On click: a composition of the selection of the selection of the *Months element (221.5)* (if it is visible) and the selection of the *Years element (221.6)* is constructed and added to the *Selected Units element (221.7)* (if not already in).

- *(221.9) Unselect Unit element:*
- Type: button
- Node 221 init: visible
- Modification: click
- Available: when an element of the *Selected Units element (221.7)* is selected
- On click: the selected serie of the *Selected Units element (221.7)* is removed.

<u>Navigational elements</u>

- *(221.10) Continue element:*
- Type: anchor
- Node 221 init: visible
- Modification: click
- Available: when an item is selected in the *Time Unit element (221.2)* and:
- If the Years element (221.6) is available, the Selected Units element (221.7) must not be empty
- Target: static: *Node 222*
- On click: navigation
-

Node 222 – Sub-Columns Composition – Temporal Way



In this node the user defines how will be organized the sub-columns of the result sheet.

Data element

- *(222.3) to (222.5) 1st, 2nd and 3rd Dimension Units elements:*
- Type: lists
- Node 222 init: visible
- Node 222 init values: for the available element(s), the units (in the corresponding dimension) for which the set of selected series are defined + the item "no selection"
- Node 222 init selection: item "no selection"
- Available: only for the element(s) corresponding to the dimensions for which the set of selected series are defined
- Modification: selection
- On selection update: the corresponding *Xth Dimension Elements element (222.6, 222.7 or 222.8)* is filled in with the elements of the web server database corresponding to the selected unit + an item "all elements".

- *(222.6) to (222.8) 1st, 2nd and 3rd Dimension Elements elements:* lists of the available elements corresponding to the selected unit
- Type: lists
- Node 222 init: visible – values: nothing
- Available: when not empty
- Modification: selection
- On selection update: nothing

- *(222.9) to (222.11) 1st, 2nd and 3rd Dimension Selected Elements elements:* dimension elements lists chosen by the user
- Type: list
- Node 222 init: visible
- Modification: selection
- Available: when not empty
- Selection: one item
- On selection update: nothing

Functional elements

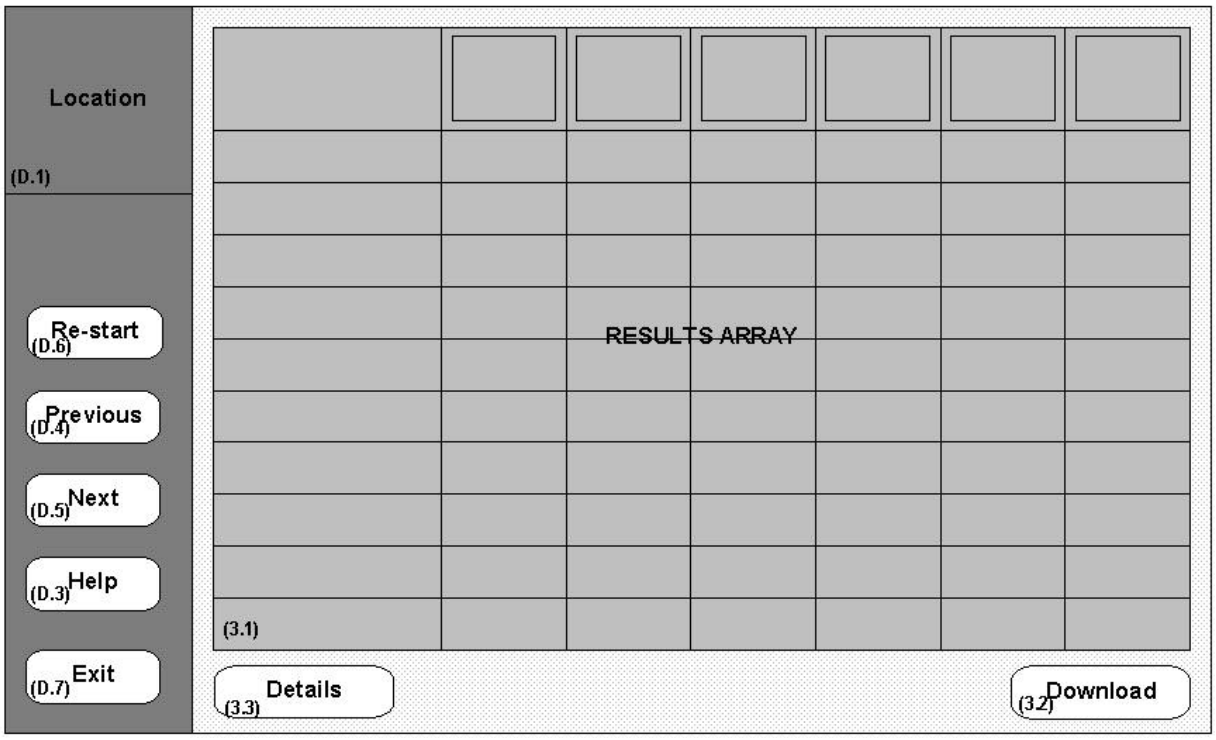
- *(222.12 to 222.14) 1st, 2nd and 3rd Dimension Select Element:*
- Type: buttons
- Node 222 init: visible
- Modification: click
- Available: if one item is selected in the corresponding *Xth Dimension Elements element (222.6, 222.7 or 222.8)*
- On click: the selected element of the corresponding Xth Dimension Elements element (222.6, 222.7 or 222.8) is added to the corresponding Xth Dimension Selected Elements element (222.9, 222.10 or 222.11) if it was not already in.

- *(222.15 to 222.17) 1st, 2nd and 3rd Dimension Unselect Elements:*
- Type: buttons
- Node 222 init: visible
- Modification: click
- Available: when an element of the corresponding Xth Dimension Selected Elements element (222.9, 222.10 or 222.11) is selected
- On click: the selected element of the corresponding Xth Dimension Selected Elements element (222.9, 222.10 or 222.11) is removed.

Navigational Elements

- *(222.18) Submit Request element:*
- Type: anchor
- Node 222 init: visible
- Modification: click
- Available: If for each available dimension the corresponding *Xth Dimension Selected Elements element (222.9, 222.10 or 222.11)* contains at least one element, except if the selection of the corresponding *Xth Dimension Units element (222.6, 222.7 or 222.8)* is "no selection".
- Target: static: *Node 3*
- On click: navigation

Node 3 – Results Visualization



The aim of this node is to present to the user the result of the request he submitted.

Data elements

- *(3.1) Results Array element:* this element is an array that formats the answer given to the request submitted by the user in the *Node 212* or *222.*
- Type: array
- Node 3 init: visible
- Modification: column selection (one)
- Available: always
- On selection update: nothing
-

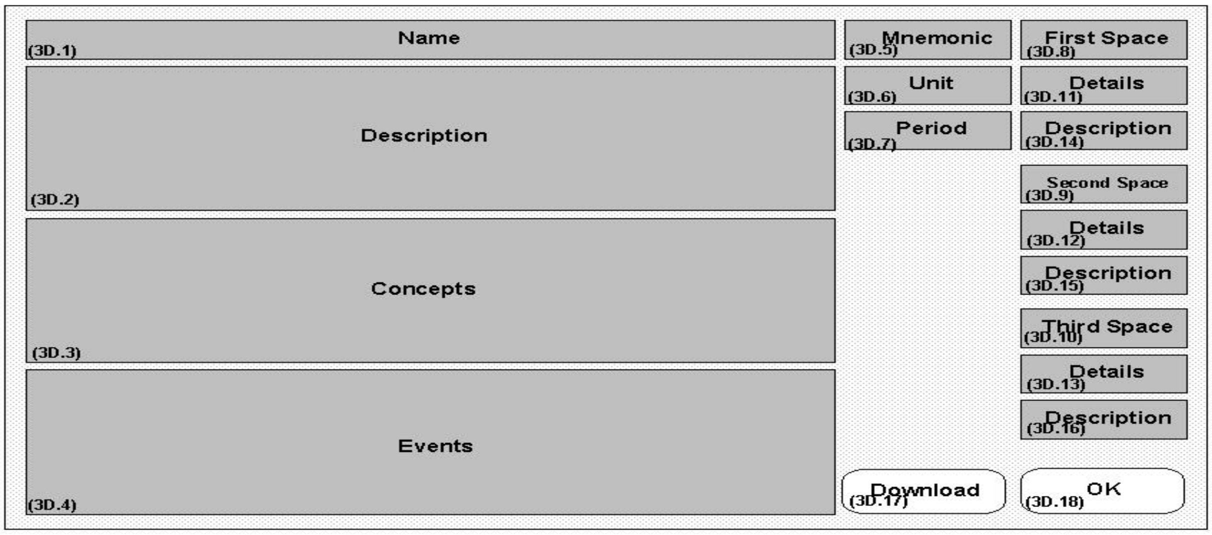
Functional elements

-
- *(3.2) Download element:*
- Type: button
- Node 3 init: visible
- Modification: click
- Available: always
- Function: provide the user a text file containing a text translation of all the information of the results array.
- On click: starts a web browser classical downloading process.
-

Intra-navigational elements

- *(3.3) Details element:*
- Type: anchor
- Node 3 init: visible
- Modification: click
- Available: when a column is selected in the *Results Array element (3.1)*
- Target: static: *Node 3 + Details*
- On click: navigation
-

Node 3D – Results Details



- This node presents some extra details about the column selected in the results table.

- *(3D.1) Name element:* this element provides the name of the serie of the selected column
- Type: text area
- Node 3D init: visible
- Modification: impossible

- *(3D.2) Description element:* this element provides the description of the serie of the selected column
- Type: text field
- Node 3D init: visible
- Modification: impossible

- *(3D.3) Concepts element:* this element provides the concepts informations of the serie of the selected column
- Type: text field
- Node 3D init: visible
- Modification: impossible

- *(3D.4) Events element:* this element provides the events informations of the serie of the selected column
- Type: text field
- Node 3D init: visible
- Modification: impossible

- *(3D.5) Mnemonic element:* this element provides the mnemonic of the serie of the selected column
- Type: text field
- Node 3D init: visible
- Modification: impossible

- *(3D.6) Unit element:* this element provides the unit of the serie of the selected column
- Type: text field
- Node 3D init: visible
- Modification: impossible

- *(3D.7) Period element:* this element provides the period of the selected column
- Type: text field
- Node 3D init: visible
- Modification: impossible

- *(3D.8 to 3D.10) First, Second and Third Space elements:* these element provide the three spaces of the serie of the selected column
- Type: 3 text fields
- Node 3D init: visible if the serie is defined for the corresponding dimension
- Modification: impossible

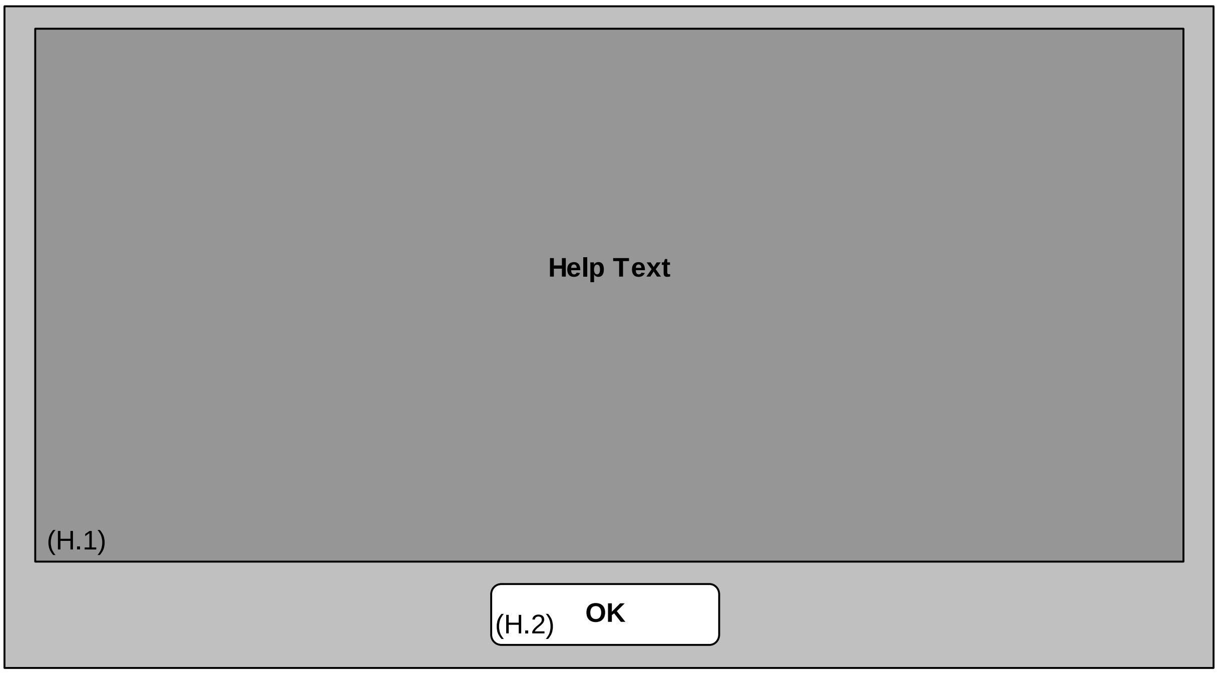
- *(3D.11 to 3D.13) First, Second and Third Dimension Details elements:* these element provide details about the three spaces of the serie of the selected column
- Type: 3 text fields
- Node 3D init: visible if the serie is defined for the corresponding dimension
- Modification: impossible

- *(3D.14 to 3D.16) First, Second and Third Dimension Description elements:* these element provide descriptions of the three spaces of the serie of the selected column
- Type: 3 text fields
- Node 3D init: visible if the serie is defined for the corresponding dimension
- Modification: impossible

- *(3D.17) Download element:*
- Type: button
- Node 3D init: visible
- Modification: click
- Available: always
- Function: provide the user a text file containing the details about the selected column
- On click: starts a web browser classical downloading process.

- *(3D.18) OK element:*
- Type: button
- Node 3D init: visible
- Modification: click
- Available: always
- On click: exit from this node to go back to the *Node 3*

Help Node



This extra node provides some contextual help about the current navigational node.

- *(H.1) Help Text element:* this element contains the help description
- Type: text field
- Node H init: visible
- Modification: impossible

- *(H.2) OK element:*
- Type: button
- Node H init: visible
- Modification: click
- Available: always
- On click: close this node

APPENDIX E – WOVEN CODE OF THE DYNAMIC NODE LOAD ASPECT

ExtraTransformation.java

```
package Rec_Internet;
/** Used for a navigation from a navigational node to an extra node */
public class ExtraTransformation extends Rec_Internet.Transformat ion {
  /*
   * This is the orginal method body for
   *  protected  void  specificAction(java.lang.String  target,  java.lang.Object[]  args)  throws
RIException
   * it is now called from there inside of any advise weaves.
   */
  protected    final    void    Rec_Inter net_ExtraTransformation$specificAction(java.lang.String
target, java.lang.Object[] args) throws RIException {
    targetNode.reset(args);
  }

  /** In the case of a navigation to an extra node, we just need to reset the target node */
  /*
   * The body of this member was replaced by aspectj
   * At the core of this code is a call to
   *  protected  final  void  Rec_Internet_ExtraTransformation$specificAction(java.lang.String
target, java.lang.Object[] args) throws RIException
   * which holds the original method body.
   * Around this call is the code for all advise
   * weaves that apply to this member.
   */
  protected  void  specificAction(java.lang.String  target,  java.lang.Object[]  args)  throws
RIException {
    {
      /*
       * Generated by aspectj
       * This implements the before advice protected void specificAction()
       * weave from the aspect ADynamicNodeLoad. (ADynamicNodeLoad.java:35)
       */
      {
        java.util.Enumeration _enumeration = _aspects.elements();
        while (_enumeration.hasMoreElements())  {
          java.lang.Object _thisAspect = _enumeration.nextElement();
          if (_thisAspect instanceof Rec_Internet.ADynamicNodeLoad)  {
            final            Rec_Internet.ADynamicNodeLoad            thisAspect            =
(Rec_Internet.ADynamicNodeL oad)_thisAspect;
            {
              targetNode = Rec_Internet.ADynamicNodeLoad.getNode(target);
            }
          }
        }
      }
      Rec_Internet_ExtraTransformation$specificAction(target, args);
    }
  }
}
```

IntraTransformation.java

```
package Rec_Internet;
/**This class is used in the case of a navigation from a navigational node to an intra node */
public class IntraTransformation extends Rec_Internet.Transformation {
  /*
   * This is the orginal method body for
   *  protected  void  specificAction(java.lang.String  target,  java.lang.Object[]  args)  throws
RIException
   * it is now called from there inside of any advise weaves.
   */
  protected    final    void    Rec_Internet_IntraTransformation$specificAction(java.lang.String
target, java.lang.Object[] args) throws RIException {
    context.getCurrentNode().desactivate();
    targetNode.reset(args);
  }
  /** In the case of a navigation to an intra node, we need to desactivate the navigational
node, and to reset the target node */
  /*
   * The body of this member was replaced by aspectj
   * At the core of this code is a call to
   *  protected  final  void  Rec_Internet_IntraTransformation$specificAction(java.lang.String
target, java.lang.Object[] args) throws RIException
   * which holds the original method body.
   * Around this call is the code for all advise
   * weaves that apply to this member.
   */
  protected  void  specificAction(java.lang.String  target,  java.lang.Object[]  args)  throws
RIException {
    {
      /*
       * Generated by aspectj
       * This implements the before advice protected void specificAction()
       * weave from the aspect ADynamicNodeLoad. (ADynamicNodeLoad.java:35)
       */
      {
        java.util.Enumeration _enumeration = _aspects.elements();
        while (_enumeration.hasMoreElements())  {
          java.lang.Object _thisAspect = _enumeration.nextElement();
          if (_thisAspect instanceof Rec_Internet.ADynamicNodeLoad)  {
            final            Rec_Internet.ADynamicNodeLoad            thisAspect            =
(Rec_Internet.ADynamicNodeLoad)_thisAspect;
            {
              targetNode = Rec_Internet.ADynamicNodeLoad.getNode(target);
            }
          }
        }
      }
      Rec_Internet_IntraTransformation$specificAction(target, args);
    }
  }
}
```

NavigTransformation.java

```
package Rec_Internet;
/** This class is used in the case of a normal navigation between two navigational nodes */
public class NavigTransformation extends Rec_Internet.Transformation {
  /*
   * This is the orginal method body for
   *  protected  void  specificAction(java.lang.String  target,  java.lang.Object[]  args)  throws
RIException
   * it is now called from there inside of any advise weaves.
   */
  protected   final   void   Rec_Internet_NavigTransformation$specificAction(java.lang.String
target, java.lang.Object[] args) throws RIException {
    context.getCurrentNode().hide();
    context.setCurrentNode(targetNode);
    context.setNextIndex(target);
    targetNode.reset(args);
  }

  /** In the case of a normal navigation between two navigational nodes, we need to reset the
target node, and then set in the context the current node index and the next index */
  /*
   * The body of this member was replaced by aspectj
   * At the core of this code is a call to
   *  protected  final  void  Rec_Internet_NavigTransformation$specificAction(java.lang.String
target, java.lang.Object[] args) throws RIException
   * which holds the original method body.
   * Around this call is the code for all advise
   * weaves that apply to this member.
   */
  protected  void  specificAction(java.lang.String  target,  java.lang.Object[]  args)  throws
RIException {
    {
      /*
       * Generated by aspectj
       * This implements the before advice protected void specificAction()
       * weave from the aspect ADynamicNodeLoad. (ADy namicNodeLoad.java:35)
       */
      {
        java.util.Enumeration _enumeration = _aspects.elements();
        while (_enumeration.hasMoreElements())  {
          java.lang.Object _thisAspect = _enumeration.nextElement();
          if (_thisAspect inst anceof Rec_Internet.ADynamicNodeLoad)  {
            final             Rec_Internet.ADynamicNodeLoad             thisAspect             =
(Rec_Internet.ADynamicNodeLoad)_thisAspect;
            {
              targetNode = Rec_Internet.ADynamicNodeLoad.getNode(target);
            }
          }
        }
      }
      Rec_Internet_NavigTransformation$specificAction(target, args);
    }
  }
}
```

RIContext.java

```
package Rec_Internet;
/** This class is used to represent the navigation informations */
public class RIContext extends Rec_Internet.Ca cheContext {
  /*
   * Generated by aspectj
   * This implements the introduce public Node getNode(java.lang.String target)
   * weave from the aspect ADynamicNodeLoad. (ADynamicNodeLoad.java:38)
   */
  public Node getNode(java.lang.String target) {
    org.aspectj.runtime.JoinPoint thisJoinPoint = null;
    for (int i = 0; i < navigNode.size(); i++)  {
      Node node = (Node)navigNode.elementAt(i);
      if (node.getName().equals(target)) return node;
    }
    for (int i = 0; i < intraNode.size(); i++)   {
      Node node = (Node)intraNode.elementAt(i);
      if (node.getName().equals(target)) return node;
    }
    for (int i = 0; i < extra.Node.size(); i++)  {
      Node node = (Node)extraNode.elementAt(i);
      if (node.getName().equals(target)) retur n node;
    }
    return null;
  }
}
```

RIPrevNextTransformation

```
package Rec_Internet;
/** To describe previous and Next navigation */
public class RIPrevNextTransformation extends Rec_Internet.Transformation {
  /*
   * This is the orginal method body for
   *  protected  void  specificAction(java.lang.String  target,  java.lang.Object[]  args)  throws
RIException
   * it is now called from there inside of any advise weaves.
   */
  protected  final  void  Rec_Internet_RIPrevNextTransformation$specificAction(java.la ng.String
target, java.lang.Object[] args) throws RIException {
    context.getCurrentNode().hide();
    context.setCurrentNode(targetNode);
  }

  /*
   * The body of this member was replaced by aspectj
   * At the core of this code is a call to
   * protected final void
Rec_Internet_RIPrevNextTransformation$specificAction(java.lang.String target,
java.lang.Object[] args) throws RIException
   * which holds the original method body.
   * Around this call is the code for all advise
   * weaves that apply to this member.
   */
  protected  void  specificAction(java.lang.String  target,  java.lang.Object[]  args)  throws
RIException {
    {
      /*
       * Generated by aspectj
       * This implements the before advice protected void specificAction()
       * weave from the aspect ADynamicNodeLoad. (ADynamicNodeLoad.java:35)
       */
      {
        java.util.Enumeration _enumeration = _aspects.elements();
        while (_enumeration.hasMoreElements())  {
          java.lang.Object _thisAspect = _enumeration.nex tElement();
          if (_thisAspect instanceof Rec_Internet.ADynamicNodeLoad)  {
            final            Rec_Internet.ADynamicNodeLoad            thisAspect            =
(Rec_Internet.ADynamicNodeLoad)_thisAspect;
            {
              targetNode = Rec_Internet.ADynamicNodeLo ad.getNode(target);
            }
          }
        }
      }
      Rec_Internet_RIPrevNextTransformation$specificAction(target, args);
    }
  }
}
```

Transformation.java

```
package Rec_Internet;
/** This class is used to contains the necessary methods and  variables to describe navigation
actions */
public abstract class Transformation extends java.lang.Object {
  /** Class variable: the context */
  protected static Rec_Internet.RIContext context = null;
  /** Instance variable: reference of the target node  of
       a navigation */
  protected Node targetNode = null;
  /** Class method context mutator */
  public static final void setContext(Rec_Internet.RIContext context_) {
    context = context_;
  }

  /** context accessor */
  public Rec_Internet.RIContext getContext() {
    return context;
  }

  /*
   * This is the orginal method body for
   * public void action(java.lang.String target, java.lang.Object[] args) throws RIException
   * it is now called from there inside of any advise weaves.
   */
  protected    final    void    Rec_Internet_Transformation$action(java.lang.String    target,
java.lang.Object[] args) throws RIException {
    specificAction(target, args);
    targetNode.show();
  }

  /** Template method to define the structure of a navigation action  */
  /*
   * The body of this member was replaced by aspectj
   * At the core of this code is a call to
   *   protected   final   void   Rec_Internet_Transformation$action(java.lang.String   target,
java.lang.Object[] args) throws RIException
   * which holds the original method body.
   * Around this call is the code for all advise
   * weaves that apply to this member.
   */
  public void action(java.lang.String target, java.lang.Object[] args) throws RIException {
    {
      /*
```

```
       * Generated by aspectj
       * This implements the before advice public void action()
       * weave from the aspect ADynamicNodeLoad. (ADynamicNodeLoad.java:35)
       */
      {
        java.util.Enumeration _enumeration = _aspects.elements();
        while (_enumeration.hasMoreElements())  {
          java.lang.Object _thisAspect = _enumeration.nextElement();
          if (_thisAspect instanceof Rec_Internet.ADynamicNodeLoad)  {
            final            Rec_Internet.ADynamicNodeLoad            thisAspect             =
(Rec_Internet.ADynamicNodeLoad)_thisAspect;
            {
              targetNode = Rec_Internet.ADynamicNodeLoad.getNode(target);
            }
          }
        }
      }
      Rec_Internet_Transformation$action(target, args);
    }
  }

  /** Abstract method that should describe some specific actions depending on the type of
navigation */
  protected abstract void specificAction(java.lang.String target, java.lang.Object[] args)
throws RIException;

  protected java.util.Vector _aspects = new java.util.Vector();
  public java.util.Vector getAspects() {
    return _aspects;
  }
}
```

AdynamicNodeLoad.java

```
package Rec_Internet;
class ADynamicNodeLoad extends java.lang.Object {
  public static Node createNode(java.lang.String name) throws RIException {
    if (name.equals("Node0")) return new RINode0();
    if (name.equals("Node1")) return new RINode1();
    if (name.equals("Node211")) return new RINode211();
    if (name.equals("Node212")) return new RINode212();
    if (name.equals("Node221")) return new RINode221();
    if (name.equals("Node222")) return new RINode222();
    if (name.equals("Node3")) return new RINode3();
    if (name.equals("Node1C")) return new RINode1C();
    if (name.equals("Node3D")) return new RINode3D();
    if (name.equals("Help")) return new RIHelp();
    throw new RIException(name, UNKNOWN_NODE);
  }

  public static Node getNode(java.lang.String name, Rec_Internet.RIContext context) throws
RIException {
    Node result = context.getNode(name);
    if (name == null)  { return Rec_Internet.ADynamicNodeLoad.createNode(name); }
    else return result;
  }

  private java.util.Vector _objects = new java.util.Vector();
  public java.util.Vector getObjects() { return _objects; }

  public void addObject(Rec_Internet.RIPrevNextTransformation object) {
    if (!_objects.contains(object))  {
      object.getAspects().addElement(this);
      _objects.addElement(object);
    }
  }

  public void removeObject(Rec_Internet.Transformation object) {
    object.getAspects().removeElement(this);
    _objects.removeElement(object);
  }
}
```