Hypermedia Systems Development based on the Unified Process*

Nora Koch[1]

Ludwig-Maximilians-Universität München
Institute of Computer Science
Oettingenstr. 67
D-80538 München
Tel. +49 89 2178 2151
Fax +49 89 2178 2152
kochn@informatik.uni-muenchen.de

Abstract

This work presents a methodology for the development of hypermedia systems (UPHD) that is based on the Unified Process. UPHD is a hypermedia engineering approach that covers the development process, project management and quality management of hypermedia applications. It focuses on the requirement capture, analysis and design activities. A detailed description of the artifacts, workers and activities of each workflow is given as well as a description of the phases and milestones in the hypermedia applications life cycle. UML activity diagrams are used to depict the flow of control in the hypermedia development process. Maintenance of hypermedia applications is included as an additional phase of the development process.

Keywords

Hypermedia Development Method, Unified Process, Hypermedia Systems, Hypermedia Design, Requirements Capture, Web Applications.

1 Introduction

The implementation of hypermedia applications usually is performed ad hoc and it is improved in successive steps. Hypermedia systems are mostly complex software systems and require therefore, a software engineering process. Traditional development processes are seldom applied as hypermedia systems demand an appropriate development process that considers the particular aspects of the hypermedia paradigm (Lowe & Hall, 1999; Conallen, 1999; Olsina, 1998; Schwabe, Rossi & Barbosa, 1996; Isakowitz, Stohr & Balasubramanian, 1995). A comparative study of the most relevant methods for hypermedia development is presented by Koch (1999).

Particular aspects of hypermedia applications are, among others: authors have to prepare multimedia material, give hypertext format to the content, relate this material through links and perform some graphical design task for the layout. Other differences to classical software engineering can be found in the activities of the project management and quality management.

This work presents a methodology for the development of hypermedia systems that takes into account the special characteristics of hypermedia development. It is based on the Unified Process and focuses

---

* Technical Report 0003, Ludwig-Maximilians-Universität München, January 2000.

[1] also working at F.A.S.T. Applied Software Technology GmbH, Arabellastr. 17, D-81925 München, Germany, koch@fast.de

on the requirements capture, analysis and design activities. The Unified Process-based Hypermedia Systems Development (UPHD) presented in this work is characterised by:

- covering the whole life cycle of hypermedia systems,
- describing the activities, workers and artifacts of the hypermedia engineering process,
- presenting UML activity diagrams to represent the development process,
- focusing on requirements capture, analysis and design,
- defining step by step how to design a hypermedia application,
- separating conceptual, navigational and presentational aspects,
- being an object-oriented approach,
- moving through a series of iterations and increments,
- being based on the workflows of the Unified Process,
- using UML models, and
- proposing a stereotype-based extension of UML.

UPHD has some similarities with the Web Application Extension for UML (WAE) presented by Conallen, (1999). Both describe a development process for Web applications and define UML stereotypes to be used in the modeling process. WAE is based on the RUP and UPHD is more adjusted to the Unified Process. The main focus of both approaches is not identical: WAE focuses on the implementation of Web applications; UPHD focuses instead on the design. WAE present different typical Web architectures. Instead UPHD describes the phases of the iterative process and shows how activities of each workflow are related in UML activity diagrams. UPHD includes in addition to implementation tests, validation of requirements and verification of the analysis and design results.

This work is structured as follows: Section 2 describes the most important aspects of the hypermedia engineering process. In Section 3 the phases and milestones of the process are presented. Section 4 gives a detailed description of the core workflows of the development process. Finally, Sections 5 outlines some conclusions and future work. In the remainder of this work we use the masculine form for the *developers* and the feminine form for the *users* and the *customers*.

2 Hypermedia Engineering

The goal of this section is to describe the general characteristics of UPHD, a systematic hypermedia engineering approach for hypermedia systems based on the Unified Process. Web applications are a subset of hypermedia applications, therefore the UPHD is also applicable to Web applications.

The term *hypermedia development* is used to refer to the development and the maintenance of hypermedia applications in the same way as Lowe and Hall (1999) defined it. UPHD covers the whole life cycle of hypermedia applications from the creation to cessation.

Hypermedia engineering refers to the application of an engineering approach to the development process. The development process is supported by project management and quality management activities.

Hypermedia design covers only a part of the life cycle, i.e. the application of a method to generate a schema for the structure and functionality of the domain as well as the user model to be implemented. It does not cover other activities such as the iteration planning, requirements capture, implementation and testing.

Hypermedia authoring is even a more reduced set of activities in the life cycle of hypermedia. It is limited to the creation and structuring of content, usually supported by special tools.

2.1 Covering the Life Cycle

UPHD covers the whole *life cycle* of hypermedia applications, starting when the idea for a hypermedia system is conceived and ending when the product in no longer available for use. It has to be distinguished form the *software development cycle* that begins with the decision to build a hypermedia application and ends when the software product is delivered.

UPHD describes an iterative and incremental process. The iterative approach reduces the risks of the waterfall model, in which the development consists of a single sequence of phases with little feedback from each phase to the previous ones. The development based on the waterfall model produces results very late in the process making it difficult to introduce changes to the initial decisions. UPHD supports a more incremental developing process than the spiral model of Boehm (1988). In the spiral model there are four distinct cycles of development: concept, requirements, design and implementation. During each cycle the same four activities are performed. These activities are: determination of objectives and constraints, evaluation of alternatives and resolution of possible risks, development and verification, and planning for the next cycle. The spiral model does not address the maintenance activities.

UPHD is based on the Unified Software Development Process (Jacobson, Booch & Rumbaugh, 1999). This allows an incremental developing process through the inception, elaboration, construction, transition and maintenance phases. In each phase a little bit of requirements capture, a little bit of planning, a little of design, a little bit of implementation and little bit of testing is done. It uses the Unified Process terminology, whenever possible. The workflows have been adapted or extended. The methodology specifies the activities to be performed at each phase as well as the workers involved in these activities and the resulting artifacts.

UPHD establishes a priori, milestones between the phases. The goal of an iterative process with milestones is to allow a major control during the whole process mitigating the risks inherent of the development process. Steps for planning, designing, implementing, integrating and testing are performed in each iteration. In between steps, the developers can get feedback and adjust the goals of the next step.

UPHD is an object-oriented approach, that uses UML techniques for analysis and design of the hypermedia applications to be developed. The notation is extended according to the extension mechanisms supported by UML, i.e. by defining stereotypes and specifying OCL constraints.

2.2 Iterative Development

UPHD is an iterative process with a priori, a not established set of iterations. In each iteration a set of process workflows are performed; this set is called *iteration workflow*. It consists of *development process workflows* and two supporting workflows: *project management* and *quality management*.

The core workflows belonging to the development process are*: requirements capture, analysis and design*, and *implementation*. The supporting workflows are part of project or quality management. The project management consists of *risk management, iteration planning* and *iteration evaluation*. Quality management comprises *validation* of the requirements, *verification* of the artifacts resulting of the analysis and design, and *testing* of the implemented system.

The iteration workflow is shown as a UML activity diagram (see Figure 1). It depicts the flow of control (continued lines) and dependencies (dashed lines) between workflows. Swimlanes are used to indicate the different workflow groups part of project management, development process and quality

management.

Note here more differences between UPHD and the Unified Process: First, it includes development process workflows and supporting workflows in the same way the Rational Unified Process (RUP) does (Kruchten, 1998). Second, while the RUP defines the following supporting workflows: configuration and change management, project management and environment, in UPHD six supporting workflows are included. They are grouped into project management and quality management. Third, as in the RUP, analysis and design are treated in the same workflow. Finally, the more general concepts quality management is used instead of "testing".

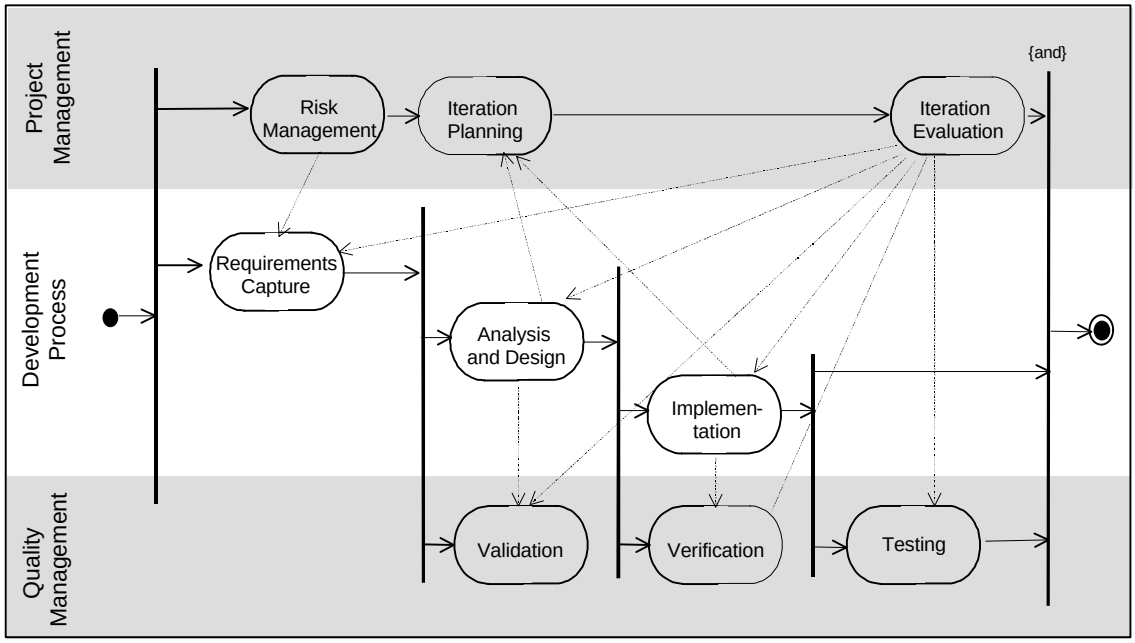


Figure 1: Iteration Workflow

The life of an application is divided by UPHD into cycles, each cycle concludes with a release The first cycle begins with the conception of the idea and the last one ends with the cessation of the use of the application. UPHD does not establish the number of iterations required in a life cycle of a hypermedia applications, but distinguishes a number of phases. Each iteration belongs then to one of following phases: inception, elaboration, construction, transition or maintenance. These phases take place over a period of time and each phase terminates in a milestone. A milestone is defined by the delivery of artifacts, such as models, code or documents and by the decisions that workers will take at this time of the process before the work of the next phase can proceed.

The number of iterations planned for each phase varies, essentially, with the complexity of the project. If the project is simple, one iteration per phase will be enough. The number of iterations of the elaboration and construction phases usually increases with the complexity of a project.

Before a detailed description of the phases of UPHD as well as of the processes and workflows are given, the terms artifact, activity, worker, stakeholder and model are defined.

An *artifact* is a tangible piece of information that is created, changed and used by workers when performing activities. It can be a model, a model element or a document.

An *activity* is a tangible unit of work performed by a worker in a workflow. The activity implies a well-defined responsibility of the worker, a well-defined result (artifacts) and a well-defined input.

A *worker* is a person with certain capabilities to perform one or more activities during a process. It is not identical to a person. One person can play the role of one or more during a project, even simultaneously.

A *stakeholder* is any person that is interested in the outcome of the project, such as a user, developer, customer, contractor or project manager.

A *model* is a simplification of reality, i.e. a semantically close abstraction of a system with the objective to better understand the system being created.

Each workflow is defined by a set of activities that are performed by a set of workers with the goal to produce some artifacts which are the measurable results of the workflow. The artifacts, workers and activities of the development workflow are briefly described in Section 4. The supporting workflows, i.e. project management and quality management are not within the scope of this work. The objective is to give only a general overview of the common aspects to the Unified Process and to detail the hypermedia specific aspects. Examples are shown in the activities subsections.

3 Phases of the Process

UPHD is based on the workflows of the Unified Software Development Process. The first four phases are the same as in the Unified Process: *inception, elaboration, construction* and *transition*. The process is extended with the *maintenance* phase as maintenance activities play an important role in the life cycle of hypermedia applications beginning very often immediately after transition has be completed. Maintenance implies changes in the content, layout and of the hypermedia structure. The last type of changes makes a relevant difference to traditional software. The following activity diagram (Figure 2) shows the iterative software engineering process including these five phases.

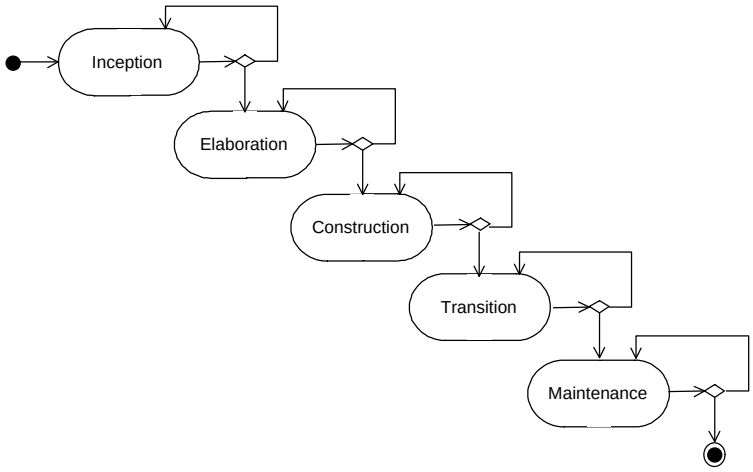


Figure 2: Software Engineering Process for one Release of an AHS

The initial state of the *inception phase* starts with just an idea or the need of a system. The final state of the inception phase is a vision of the end system and its business case. The objectives of the systems development are defined during this phase as well as a first approach of the architecture of the system, an estimation of costs and a schedule plan.

During the *elaboration phase* the architecture of the system is defined. The project manager will elaborate a plan of activities and an estimation of the resources needed to complete the project. A stable architecture as well as control over the risks are prerequisites for the next phase.

The *construction phase* focuses on the development of the system, although additional requirements elicitation and minor changes are performed during this step. The phase finalises when all the use cases are implemented.

The *transition phase* covers the period the system is tested as a complete version (normally called beta release) by a reduced group of users. Training, help-assistance and correction of defects are activities of the transition phase.

The *maintenance phase* begins when the first version is delivered and extends itself until the system is no longer used. During this period of time the system requires adjustments of different type: content updates, layout improvement, structure modifications and adaptation to new technologies or new software versions.

Five main milestones are part of the process marking the end of each of these phases. These milestones are: life cycle objectives, life cycle architecture, initial operational capability, product release and product cessation. The following sections outline the characteristics of the iteration workflow in each of these phases.

3.1 Inception

The principal objective of the inception phase is to establish the feasibility of the project, i.e. to define the business case for the system and delimit the scope of the project. This business case includes success criteria, risks assessment, budget and resource estimation as well as a phase plan with a schedule and deliver plan for the major milestones. Sometimes an executable prototype is developed during this phase.

At the beginning of the inception phase there is only an idea. The process starts with the conception. The goal is to develop and evaluate the idea for a hypermedia system, i.e. the need for hypermedia capability and benefits in developing an application based on this technology. The main aim of the feasibility study is to define the high level functional requirements for the hypermedia, to outline a first budget, produce a draft schedule plan and establish the main non-functional requirements of the system. Costs of the project must be estimated in this phase, because this is a crucial element in determining the feasibility of the project. It may be advisable to develop a prototype. The effort of building a prototype has to be minimised just in case the idea fails to live up. The results of the study performed in this phase lead to the decision whether it is worthwhile or not to develop the hypermedia application.

Techniques supporting feasibility study vary from informal *textual description* through *checklists* and *spread sheets* to implementation-oriented ones like paper or computer based *storyboarding* as proposed by Boyle (1997). P*rototyping* is a meaningful but expensive alternative.

The inception phase focuses on the requirements capture, risk management, project planning and validation of the requirements. Other activities, such as design, implementation and testing do not play an important role during inception.

The most important factors that influence the activities of the different workflows in the inception phase are:

- *current information, information sources and information structure*

  In the development of hypermedia applications, it is important to have an understanding of the types of information available and of the relationships of the information as it will be reflected in the hypermedia structure. It has to be considered how often the information changes as well as security and legal aspects related to information sources.

- *current applications*

In many cases hypermedia applications are developed to replace existing non hypermedia-based or non web-based applications. Understanding of these applications and their problems is helpful for the understanding of the scope of the new hypermedia application to develop. The navigation in the final application has to support the same or an improved behaviour and functionality of the legacy system.

- *stakeholders*

Any existing system has numerous categories of people who have a stake in the system, such as clients, users, developers and providers. Stakeholders of hypermedia systems are even a more heterogeneous group of people including a more complex and heterogeneous group of users, multimedia experts, graphic designers, hypermedia architects, public relation experts, marketing people, legal experts, e-commerce experts, etc.

- *resources*

This factor includes personnel, budget, time, information, expertise, hardware and tools.

- *technological limitations*

Technological limitations are constraints that are conditioning the possible and appropriate solutions. Limitations include processing power, bandwidths, equipment costs, equipment's reliability and security, for example. Frequently it will be necessary to analyse and test new technologies.

- *constraints*

Constraints may be inherent to the domain or be imposed by the client.

At the end of the inception phase the decision is taken whether to proceed with the development or not. The first milestone, i.e. *the life cycle objectives*, marks the end of the inception phase. The deliverables for the inception phase are:

- a first version of a domain model,
- a first version of the use case model,
- a first draft of the architecture description,
- a prototype to proof the concepts or a new technology (optional),
- a risk study,
- a plan for the whole project,
- a business case, including success criteria, risk analysis, budget estimation, and
- an architecture validation and requirements review report.

3.2 Elaboration

The principal objective of the elaboration phase is to capture the remaining requirements, to establish a sound architectural baseline, to elaborate a guide for the construction based on the models, to identify additional risks and review already known risks and to detail the project plan.

At the beginning of the elaboration phase, after the first milestone of the process the principal inputs that are available are the draft architecture, draft use case and domain model, a project plan, a list of risks and a business case.

The elaboration phase focuses on the analysis and design as well as on the iteration planning and verification of the design. There are many factors which influence the activities of the workflows in the elaboration phase. These factors can be classified into:

- *design creativity*

The creativity of the design requires very often the expertise of multimedia, graphic or marketing experts. Hypermedia design has to take into account cultural and perceptional aspects that will influence the success of a hypermedia application in addition to the functionality it offers to the users. It is a factor that most of the traditional software engineering environments do not support (Nanard & Nanard, 1995).

- *technical issues*

The limitations of the current technology impose restrictions related to the hardware, software, databases and authoring tools that can be used. For example the bandwidth puts a limit to the information that can be accessed in an appropriate time interval during online access. In the same way the utilisation of video or audio depends on the processing power of the user's computers. Technologies have to be compared to select the appropriate dynamic generation of pages, of adaptive content or adaptive links.

- *cognitive issues*

Cognitive issues play an important role in the development of the hypermedia applications. The designer should know how to avoid the problems of cognitive overload and lost in the hyperspace. He has to be aware of techniques to organise and navigate information.

- *non-technical issues*

Non-technical issues include aspects such as laws and security rules of the different countries that will access the application.

At the end of the elaboration phase the system is ready to be produced. The second milestone, called *the life cycle architecture*, marks the end of the elaboration phase. The deliverables for the elaboration phase are:

- a complete business or domain model,
- a new version of all models: use cases, analysis, design, deployment, implementation and testing,
- an architectural baseline and detailed description,
- an updated risk list,
- a project plan for the construction and transition,
- a complete business case, and
- an architecture verification and model review reports.

3.3 Construction

The principal objective of the construction phase is to produce a software product ready for initial operational release, the so called "beta release".

At the beginning of the construction phase the complete set of use cases has been described, all the models have been developed, the main risks have been analysed in detailed, the architecture is defined and almost all requirements are captured (in practice about 70% to 80%).

The construction phase focuses on the implementation as well as on the testing of the system. During the construction phase various components required for the application are produced, obtained or modified according to the design specifications. These components are then integrated to create a prototype, a draft version or the final application.

The most important factors that influence the activities of the different workflows in the construction phase are:

- *media components availability*

The production or adaptation of existing multimedia components contributes to the complexity of a project. It requires the participation of more experts than by the traditional software development, such as audio, video, animation designers.

- *dynamic page generation*

If pages of a hypermedia application are generated dynamically, a database has to be administrated, templates need to be defined and a page generator must be run. Performance may be a critical risk in this case.

- *usability*

The usability of the application for different platforms is a typical factor that affects the construction and required additional effort. The objective is to obtain a Web application that presents an equivalent quality with different browsers.

At the end of the construction phase the first version of the hypermedia system is ready to be used. The third milestone is the *initial operation capability* and marks the end of the construction phase. The deliverables for the construction phase are:

- the executable software itself – the initial operational capability release (last version build during construction),
- all the artifacts and models developed during the project,
- the architecture description updated according all the changes introduced during the construction,
- a draft version of the user manual to guide beta users, and
- a project plan for the transition and maintenance phases.

3.4 Transition

The principal objective of the transition phase is to integrate the product in the user's environment and correct the operational version until customers provide positive acceptance tests.

At the beginning of the transition phase the system has reached initial operational capability. In addition, the results of the testing workflow are available to remove the last bugs and inconsistencies.

The transition phase focuses on the establishment of the final product in the operational environment and the evaluation of the project. There are different ways to perform this transition depending on the type of product that has been developed. If it is a Web application for the internet for example, a beta version is tested by a group of acceptance testers before going online. This group has to be as heterogeneous as possible trying to simulate real Internet users. If the resulting product is an Intranet application within a large organisation, it is first tested by one department or section of the organisation.

The major factors that influence the activities of the different workflows in the transition phase are:

- *insufficient or inadequate tests*

The absence of sufficient or adequate tests in the construction phase has the effect that the bugs, misunderstandings and errors appear during this phase augmenting the rework. Web systems are even more difficult to test as these systems require a variety of users to test different user's behaviour.

- *schedule pressure*

Schedule pressure is often caused by insufficient time planned for the transition activities, such as installations, test, corrections and reworks. This happens when software products are supposed to be final versions at the end of the construction phase.

- *budget pressure*

  If no budget is left, it is often difficult to make corrections or improvements even if there is enough time to perform them.

At the end of the transition phase the system is ready to be released. The fourth milestone, i.e. *the product release*, marks the end of the transition phase. The deliverables for the transition phase are similar to the deliverables of the construction phase, but they are now corrected and complete:

- the executable software itself, including the installation software,
- legal documents, such as contracts, licenses, etc.,
- final version of all models and artifacts produced during the project,
- completed and corrected architecture description,
- final version of user manual,
- training material,
- references and addresses where to find additional information.

3.5 Maintenance

The principal objective of the maintenance phase is to adapt a hypermedia application to a changing environment, conditions or new resources. Sometimes maintenance involve corrections and improvements. Maintenance plays an essential role, even more for certain types of hypermedia systems such as Web applications. Changes in the content and layout improvements may require modifications in the structure. Therefore, maintenance of hypermedia applications can be more complicated and more time-consuming than for traditional software systems, where the maintenance process was restricted to data updating.

At the beginning of the maintenance phase a full operating hypermedia system is provided. During the whole maintenance process this state has to be preserved.

The maintenance phase focuses on the implementation of adaptations, corrections or improvements. There are many factors which influence the activities of the workflows in the maintenance phase. These factors can be classified into:

- *coupling and cohesion of the hypermedia application*

  In traditional software the rule is to minimise coupling. A typical characteristic of hypermedia applications instead is the associative coupling through linking. Highly coupled pages are difficult to maintain and easily lead to dangling or incorrect links. Cohesion means for hypermedia applications that each node, if possible, is based on a single concept. An appropriate chunking of the information may improve maintenance.

- *analysis and design maintenance*

  The maintenance of the documentation related to analysis and design is known as a critical factor in the maintainability of the software.

- *code documentation*

  This factor has the same weight for the maintenance of hypermedia applications than for the maintenance of traditional software.

The fifth milestone, i.e. *the products life end*, marks the end of the maintenance phase. At the end of the maintenance phase the system is replaced by another product or just taken out of order.

There are no deliverables at the end of the maintenance phase as the product life ends at that moment. In

each iteration within the maintenance phase the deliverables are the same as deliverables of the transition phase.

4 The Development Process

The development process consists of the workflows: requirement capture, analysis and design, and implementation. They are called the core workflows. Note the difference to the Unified Process that includes five core workflows: requirements capture, analysis, design, implementation and testing.

The UPHD does not include the design as a separate workflow. It is considered as a refinement process of the analysis, i.e. the first iteration of the design corresponds to a rough analysis, followed by successive refinements until a detailed and implementation-oriented design is reached in the last iteration. Therefore, in the same way as in the Rational Unified Process, the analysis model is defined as an abstraction of the design model, a generalisation or a less detailed design model. Testing is the other core workflow of the Unified Process not included here. It is part of the one of the supporting workflows: quality management process. This process includes validation, verification and testing workflows.

In the remaining part of this chapter the hypermedia engineering process is illustrated with the following example.

Example: The Film Assistant
The vision of the Web system called *The Film Assistant* is the following: It offers information about films, persons working in the film business, such as actors, actresses, directors, producers, cinematographers, costume designers, editors, composers, etc. as well as information about festivals and awards. Users are informed about news related to the film business.

4.1 Requirements Capture

The requirements capture is the process of determining or, some times under more difficult circumstances, the process of discovering what application is to be built. A *requirement* is a condition or capability to which an application must conform.

The requirements capture is not an easy task even for the development of traditional software. The difficulties are still increased in the case of hypermedia applications. Some of the difficulties are:

- The development of hypermedia applications may have different starting points, such as a vision, an existing application or a concrete description.
- Stakeholders have usually only partial knowledge about the process to be supported by the hypermedia application.
- There are changing use of adaptive applications, changing target groups, changing technologies, and changing resources.
- There is little experience in the systematic development of hypermedia applications.

Traditional approaches are based on the ability of a system analyst to elicit a list of requirements from the users. This approach has been improved by the systematic specification of the requirements, e.g. by use case models. The analyst uses them to ensure the completeness and correctness of the requirements. Use cases are utilised as base for discussions between customers, users and hypermedia analysts. Note that requirements must be described in the *customer language* without, whenever possible, formal and technical specifications. Use cases offer the advantage to be more unambiguous than textual description and understandable by customers and hypermedia analysts.

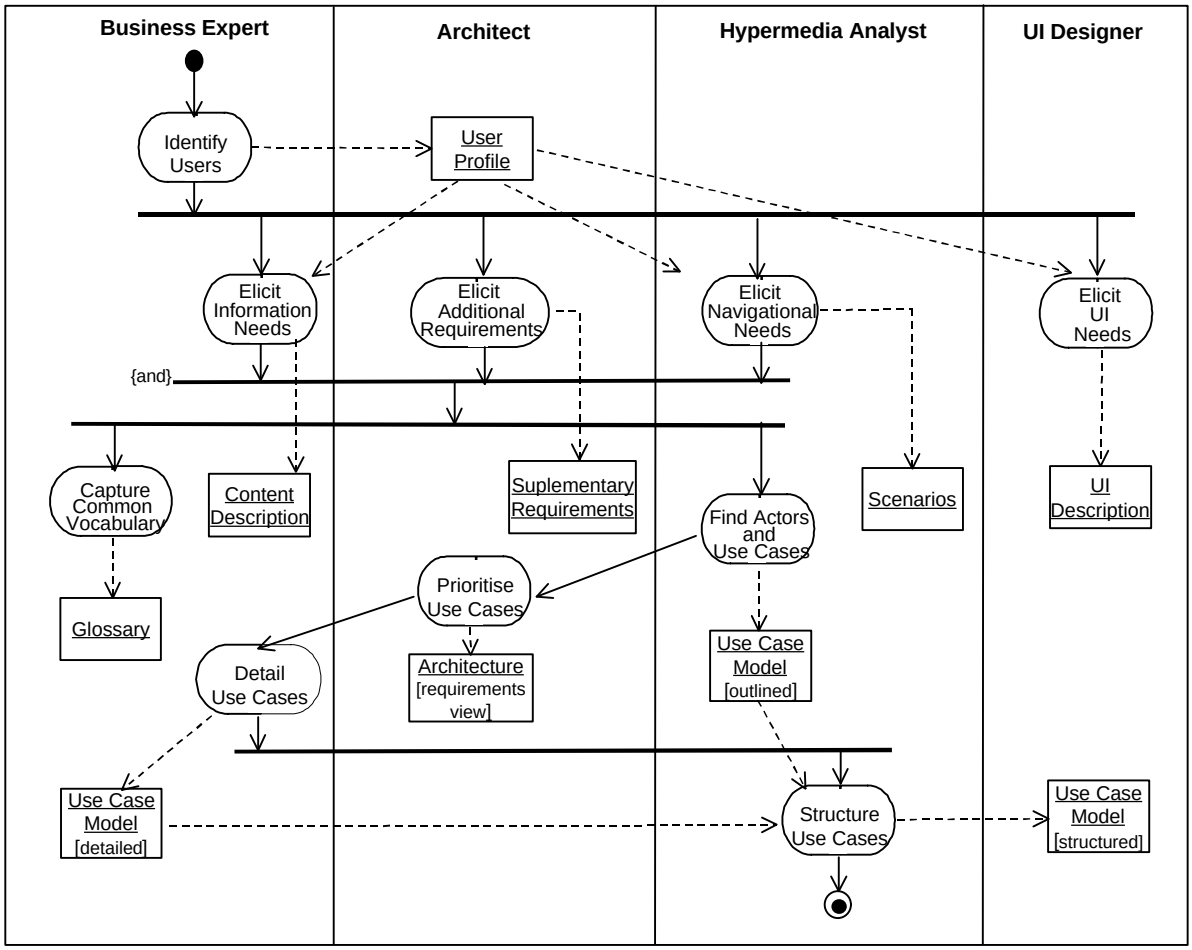


Figure 3: Requirements Capture Workflow and Object Flow

Two categories of requirements can be distinguished: functional and non-functional requirements:

- *Functional requirements* of hypermedia systems are the actions the system will be able to perform, i.e. they are used to describe the systems behaviour given certain input conditions. I distinguish the following types of functional requirements:
  - *related to the content* (e.g. images to illustrate certain pages, evaluation of exercise results)
  - *related to the structure* (e.g. navigation to the homepage is allowed from every page)
  - *related to the presentation* (e.g. layout restrictions, such as no more than 10 items in a list)
  - *related to the user profile* (user preferences or goals to be taken into account)
- *Non-functional requirements* specify systems properties, such as environment and implementation constraints, performance, reliability, extensibility, etc.

Due to the variety and complexity of requirements to be identified and analysed, the requirements capture workflow contains several activities performed by four different workers and the results are a use case model as well as documents describing the user profile, the content, scenarios, use cases and the user interface.

The requirements capture workflow is shown as a UML activity diagram in Figure 3. An object flow is added to the same diagram; it is denoted with dashed lines (Oestereich, 1999). Thus, workers, activities and artifacts of a workflow are shown all in the same diagram. A detailed description of the activities,

artifacts and workers follows. The diagram does not show artifacts that may be used as input to the requirements capture activities, such as description of a previous version of the system or a business model.

4.1.1 Artifacts

The main artifact produced in the requirements capture is the use case model composed by actors and use cases. A set of artifacts is the result of the elicitation process; there are the user profile, content description, scenarios, supplementary requirements and user interface prototype. They are used as input for the activities that produce the use case model. In addition an architecture is elaborated and a glossary has to be prepared from the beginning on.

- **User Profile**

The user profile is a description of users or user groups. This description includes user characteristics, tasks, needs, preferences related to the problem domain.

- **Content Description**

The content description consists of a detailed list of sources of the information that will be included in the application.

- **Scenarios**

Scenarios are a textual description of the typical sequences of activities performed by actors of the system. In hypermedia systems there are the typical navigation paths users follows through navigation or the steps that follow authors in the authoring process.

- **Use Case Model**

The use-case model is a model of the system's intended functions and its environment, and serves as a contract between the customer and the developers. Different versions of the use case model are build during the requirements capture workflow with increasing details. They are called the outlined, detailed and structured use case model.

- **Architecture (View of the Use Case Model)**

The architecture contains an architectural view of the use case model, i.e. focusing on the use cases that are relevant for the architecture of the system.

- **Supplementary Requirements**

The supplementary requirements are artifacts in form of documents describing non-functional requirements that can not be captured in the use-case model.

- **Glossary**

The glossary defines the terms used in the project. It is useful to reach a consensus among customers, project managers and developers regarding the definition of concepts used in the project. The objective is to reduce the risk of misunderstandings.

4.1.2 Workers

The requirements capture in the development of a hypermedia application is performed by the following four workers: business expert, architect, hypermedia analyst and user interface designer.

♦ **Business Expert**

The business expert is responsible for the business use case modeling by outlining and delimiting the organisation to be modelled. He identifies the potential users groups, their needs, i.e. he establishes what business actors and business use cases exist and how they interact. In addition, he defines the glossary, working together with the hypermedia analyst.

The business expert does not need knowledge in hypermedia engineering but experience in the use of similar hypermedia applications is helpful.

The profile of the business expert is described by the following skill: knowledge of the business domain

Example: The Film Assistant
The business expert may be a journalist specialised in film reporting and film critique.

♦ **Architect**

The architect is the main responsible in the software development process. He has to be a domain expert as well as have knowledge of software development. He is responsible for the leadership and co-ordination of all technical activities and the delivering of all technical artifacts throughout the project. He is the technical driven force of the project.

The architect does not to have project manager responsibilities. He begins his work in the inception phase and will remain active during the whole life cycle of the hypermedia application.

During the requirement capture the architect describes the architectural view of the use case model. He has to elicit the additional requirements of hypermedia applications, such as networking, distribution, browser restrictions. He has to prioritise the use cases. This prioritisation is an important input for the planning of the iterations. The architect is usually assisted in his work by other developers.

The artifacts that produces the architect during the requirements capture workflow are the *requirements view of the architecture* and the *supplementary requirements*.

The profile of the architect is given by the following skills*:*

- experience in the hypermedia technology domain and some knowledge of the business domain,
- abilities to communicate the architecture to the developers,
- leadership to co-ordinate the technical activities of the different teams and to translate decisions into activities,
- goal-oriented focusing on the resulting hypermedia application.

♦ **Hypermedia Analyst**

The hypermedia analyst is responsible for the requirements elicitation and use case modelling. He outlines the hypermedia application's functionality by finding out the access and navigation needs of the users. The hypermedia analyst delimits the systems functionality and ensures the completeness and consistency of the use case model.

The artifacts that are produced by the hypermedia analyst during the requirements workflow are the use case model and the documents describing non-functional requirements.

The profile of the hypermedia analyst is given by the following skills:

- general knowledge of the business domain and hypermedia technology domain,
- general knowledge of user modelling and adaptive systems,
- good communication skills,
- UML knowledge for the use case modelling.

♦ **User Interface Designer**

The user interface designer is responsible for the visual modeling of the user interface. Therefore, he has to capture the requirements on the user interface, including usability requirements and involving stakeholders, specially end-users in this elicitation process. He should provide a user interface description, a user interface model or a prototype, but should not implement the user interface. The focus is put on the visual shaping of the user interface. The actual implementation is done by other developers during the design and implementation workflows.

The profile of the user interface designer is given by the following skills:

- abilities to translate stakeholders ideas into hypermedia windows or web pages,
- design abilities for the creation of presentation and navigation,
- good communication skills to capture usability requirements.

4.1.3 Activities

The activities of this workflow describe the dynamics in the requirements capture. The goal is to represent these requirements as use cases (see Figure 3). The sequences of activities of the workflow is depicted with a UML activity diagram. The order these activities has not to been seen as very tight, some overlapping is also possible.

The activities needed to capture the requirements of a hypermedia application are in first place related to the domain, such as identifying users, eliciting information needs or capturing common vocabulary. Second, these activities try to find out the kind of application to be developed: elicit navigation needs, user interface needs and additional requirements as well as the prototype user interface. The last group has as objective to model the requirements captured as use cases, these activities are: find actors and use cases, detail use cases, prioritise use cases and structure use cases.

♦ **Identify Users**

Users are the principal actors of hypermedia systems. This activity consists of finding users, interviewing them (if possible), characterising and describing them. The goal is to delimit the hypermedia system from the environment and to obtain information needed to build the user model as well as define the adaptation rules. A business model can be used to start from.

The following questions may be helpful to identify users of a hypermedia system:

- Who will interact with the hypermedia application?
- What tasks do the users have?
- What functionality do they expect from the application?
- What (computer, language, cultural) background do they have?
- What background knowledge do they have?
- What experience do they have in using similar applications?
- Can different groups clearly be distinguished?
- Which attributes define these users or user groups? For example, the age range of expected users?

The activity *identify users* is performed by the business expert. The result of this activity is a user profile description.

Example: The Film Assistant
The *Film Assistant* is used by different groups of users. According to the personal attributes, such as age, sex, education the following groups can be distinguish: children, teenagers, adults, male and females as well as intellectuals. According to their preferences the users can be grouped in fans of horror films, comedies, dramas, animated cartoons, etc. If the users demand on rating qualification are taken into account, users will be classified in exigent and non-exigent. Their main objectives are to localise information related to the film business and to get recommendations about films.

♦ **Elicit Information Needs**

This activity has the goal to find out what information has to be included in the hypermedia application. It is useful to establish both, the depth and the breath of the content. If a business model is available, it can be used to obtain a first approximation to the information needed. Interviewing potential users is another technique that can be applied. In many cases the information scope is partially limited by existing content or the effort that can be invested to adapt this content to the application's requirements.

The following questions may be helpful to elicit the information users need:

- What information are the users interested in?
- What will they search for?
- What content is already available and in what form?
- What content needs to be provided?
- Who can provide the contents?
- How much time can users tolerate information that has not be updated?
- How can the risk of information overload be avoided?

The activity elicit *information needs* is performed by the business expert. The result of this activity is the content description.

Example: The Film Assistant
The users of *The Film Assistant* are interested in information about films, persons working in the film business and film events as well as news in the film business. They are also interested in suggestions about films to see. An existing film database provides the information about films, persons in the film business and film events. Films are described by a large list of attributes including title, year, language, rating, genre, country, crew, distributor, length, book, keywords, etc. Film genres are: action, thriller, comedy, documentary, etc. A film festival consists of film presentations, awards in different categories (best film, best actor, best production, best music, etc.) and a list of nominees or winners of the awards. Thus, films and persons may be honoured with awards of type nomination or winner. News are associated to films and persons; they include articles and comments. Film attributes have to be categorised and grouped with the purpose to place them in different Web pages and avoid information overloading.

♦ **Elicit Navigation Needs**

This activity has the goal to find out how the information is accessed in the hypermedia application. Interviewing potential users is a technique that can be also applied here.

The following questions may be helpful to elicit the navigation facilities users need:

- Which information do users want to see at the first glance?
- What are the typical searches they will perform?
- What are the most frequent searches they will perform?
- Which is the average length of navigation paths required? How can long navigational paths reduced?
- How can the system assist users in their navigation activities?

The activity *elicit information needs* is performed by the hypermedia analyst. The result of this activity are scenarios. These scenarios consist in the description of the typical navigational behaviour of the users.

Example: The Film Assistant
The users of *The Film Assistant* are interested primarily in information about films. Typical searches they perform are based on film names, actors, directors, film premieres, new reports about films and persons working in the film business, last film events, etc. Navigation assistance can be supported by menus and sub-menus as well as keyword search.

♦ **Elicit Additional Requirements**

The supplementary requirements are primarily non-functional requirements. They are not related to the content, navigation, or interface of the hypermedia system. These additional requirements can not be included in the use case model, therefore they are presented as a document that consists of a list of requirements.

The additional requirements can be elicited based on the following generic list. This list is not exhaustive, i.e. it can be extended.

- Budget constraints
- Time constraints
- Hardware constraints
- Software constraints
- Design constraints
- Implementation constraints
- Performance
- Security
- Availability
- Ergonomics

The activity *elicit additional requirements* is performed by the architect. The result of this activity is a list of supplementary requirements.

Example: The Film Assistant
This Web application has to be optimised for the most frequently used browsers, with the objective of a nearly identical presentation.

♦ **Elicit User Interface Needs**

This activity has the goal to find out, how the information and the navigation assistance is to be presented to the user in the hypermedia application. Interviewing the customer as well as potential users is a technique that can be applied here. Another alternative is to analyse and compare hypermedia applications, that present some similarities with the application to be developed.

The following questions may be helpful to elicit the user interface needs:

- Does the customer's organisation have a style guide for their hypermedia applications?
- What layout constraints does the customer specify?
- Has the presentation to be designed from scratch?
- How to avoid cognitive overload?

The activity *elicitation of user interface needs* is performed by the user interface designer. The result of this activity is a user interface description.

Example: The Film Assistant
The presentation has to be designed from scratch. No style guide is available. A film page includes only one film poster. The persons pages may include a picture. ...

The remaining activities of the requirements capture workflow are only briefly explained. They have the same objectives and are performed in the same way as the activities with the same name of the Unified Software Development Process.

♦ **Find Actors and Use Cases**

During this activity all the requirements elicited during the above described activities is used to define the actors of the system and find the use cases that describe the functionality of the system.

The users are the main actors of the hypermedia application. The following questions help to find candidates for other actors:

- Who will perform the authoring work?
- Who will support and maintain the hypermedia system?
- What are the system's external resources?
- What other systems will need to interact with this one?

A brief description of each actor should include information about what or who the actor represents, why the actor is needed and what interests the actor has in the system.

The best way to find use cases is to consider what each actor requires of the system. The set of functional requirements, i.e. informational, and navigational, gives answers to this question. Workshops or interviews can also be used to understand what use cases are needed. Each use case is briefly described.

The activity *find actors and use cases* is performed by the hypermedia analyst. The result of this activity is an outlined use case model.

Example: The Film Assistant
The actors identified are the *User* and the *Film Assistant Administrator*. *User* A User represents a person who is browsing in the *Film Assistant*, navigating the hyperspace defined by this application. *Film Assistant Administrator* A *Film Assistant Administrator* represents a person who is in charge of the update and maintenance of the system. Some of the identified uses cases are: Search Title, Search Film People, Search Festival, Look at News,

Example: The Film Assistant
Ask for Film Recommendations, Update Film, Update Film People, Update Festival, Change News, Enter Keywords. The description of one use case is included here to exemplify this step. *Look at News* This use case is used by both actors, the User and the Registered User. They can specify the period of time for the news. If they do not enter any period, in the case of a User the default period of a week is used.

- **Detail Use Cases**

The objective of detailing each use case is to describe its flow of events in detail including how the use case starts and ends as well as how actors interact with these use cases. The business expert performs this activity adding his domain knowledge to the use case model specification. The result is a detailed description of a use case in text and diagrams (Schneider & Winters, 1998).

This detailed description of the use case includes a use case name, the list of actors communicating with the use case, the priority, the status of the development of this use case, pre- and post-conditions that must be true at start and after end of the use case, a list of use cases that "extend" this use case, a list of use cases that are "included", a flow of events describing primary scenarios. Optionally, secondary scenarios (alternatives and exceptions not shown in the primary flow of events), activity diagrams, user interface, sequence diagrams, views of participating classes and other artifacts as well as other requirements and open questions can be added to description of a use case.

Example: The Film Assistant

Use case name: "Search Festival"

Actors: User, Film Assistant Administrator

Priority: 1

Status: Requirements capture

Pre-condition: Search frame must be visible

Post-condition: Page of the festival with the most important awards is shown

Flow of events:

1. The use case starts when the user selects the "festivals" option.
2. The user enters the name or a keyword of the festival.
3. The user starts the search mechanism
4. If the result is more that one festival then
   a) the system displays a list of festivals matching the user's input
   b) the user selects one

   end if
5. The system displays the information about the festival
6. The system searches awards for the categories: best film and best direction.
7. The system displays the winners of these awards.
8. The use case ends

Secondary scenario:

1. If the system does not find a matching festival, the user is asked to re-enter the keyword or festival's name.

♦ **Prioritise Use Cases**

The purpose of this activity is to determine with which priority each use case has to be developed, i.e. designed, validated and implemented. This activity is performed by the architect and the result is visible in the architectural view of the use case model that only includes the critical actors and use cases.

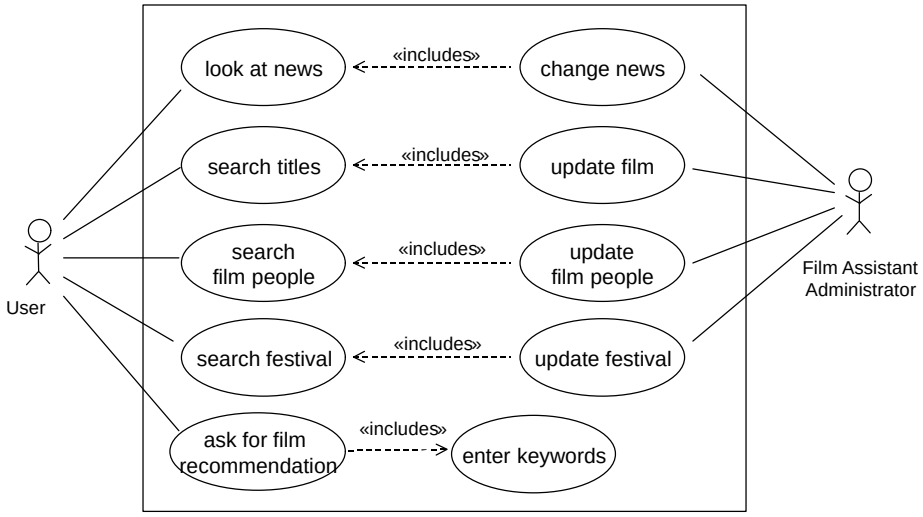
Example: The Film Assistant
The "search" use cases and the use case "ask for film recommendations" have the highest priority. The implementation of these use cases will give to the customer the "*look and feel*" of the system. Priority two is assigned to the use cases defining the update of the database through special forms. In a third step the "news" features will be designed and implemented, i.e. the functionality to add comments and documents related to films and film people as well as the functionality to perform a full text search in these documents.

♦ **Structure Use Cases**

During this activity a complete use case model is build. Therefore, the relationships of actors and use cases are analysed in detailed. Relationships of type "include" and "extends" are established between use cases as well as generalisations between actors and between use cases. A detailed flow of events with a pre-condition and a post-condition can be defined in textual form as well as represented with a UML statechart diagram. The result is a detailed use case model.

Example: The Film Assistant

The following use case model shows part of the use cases, actors and relationships identified in *The Film Assistant.*



♦ **Capture a Common Vocabulary**

The objective of this activity is to define a common vocabulary that can be used in all textual descriptions of the system, especially in use case descriptions. This common vocabulary is the base for communication with all stakeholders. The result is a glossary produced by the business expert.

Example: The Film Assistant
.... **active help:** short description for each navigation button, may be suppressed by the user. **film people:** includes actors, actresses, directors, producers, cinematographers, costume designers, editors, composers, etc. **news:** are data, documents and articles related to films and film people as well as festivals recently included in the database. .....

4.2 Analysis and Design

The purpose of the analysis and design workflow is to translate the requirements description (obtained in the previous workflow) into a specification that describes how to implement the hypermedia application. Analysis focuses on the application's functional requirements ignoring non-functional requirements and implementation constraints. Design adapts the analysis results to the conditions imposed by the non-functional requirements. The design is a refinement process of the analysis. In this work, both, analysis and design are described together. When the term "design" is used, it means analysis as well.

The design workflow of UPHD is a model-based and user-centred approach for building hypermedia applications. It is model-based for almost all activities a UML-model is build. It is user-centred because it takes into account user properties for the construction of these models. The design comprises the following activities:

- architectural design,
- conceptual design,
- navigational design,
- presentational design,
- detail design classes,
- define subsystems and interfaces.

The artifacts produced during these activities are the implementation view of the architecture, the conceptual model, the navigational model, the presentational model, design classes, subsystems and interfaces. For the construction of the models the UML extension presented by Koch and Mandel (1999) is used. It is based on UML class diagrams and UML statecharts with special stereotypes for the navigational and presentational design.

The purpose of the *architectural design* is to outline the design and deployment models as well as their architecture. It requires the identification of architecturally significant design classes, subsystems and their interfaces, nodes and their network configurations as well as special requirements on persistency, distribution and performance.

The main objective of the *conceptual design* is to capture the domain semantics and to present it as an object-oriented class model. Based on this model the next two activities take into account the special characteristics of the hypermedia paradigm: the navigational functionality and the multimedia user interface.

The *navigational design* defines the structure of the hypermedia application describing how navigation can take place. The basis of the navigational design is the conceptual model and the outcome is a navigational model, which can be seen as a view over the conceptual model. The navigational model is defined in a two step process. In the first step it is specified *which* objects can potentially be reached through navigation and in the second one *how* these objects are reached. Additional objects are utilised

to access navigational objects and a grouping concept is required to achieve an optimal navigation (Baumeister, Koch & Mandel,1999).

The static and dynamic aspects of the user interface are modelled during *the presentational design* depicting the layout in a schematic way. The outcome of the presentational design is the static and dynamic presentational model. While static presentational model associates to each navigational object at least one presentational object, the dynamic presentational model describes the behaviour of these presentational objects.

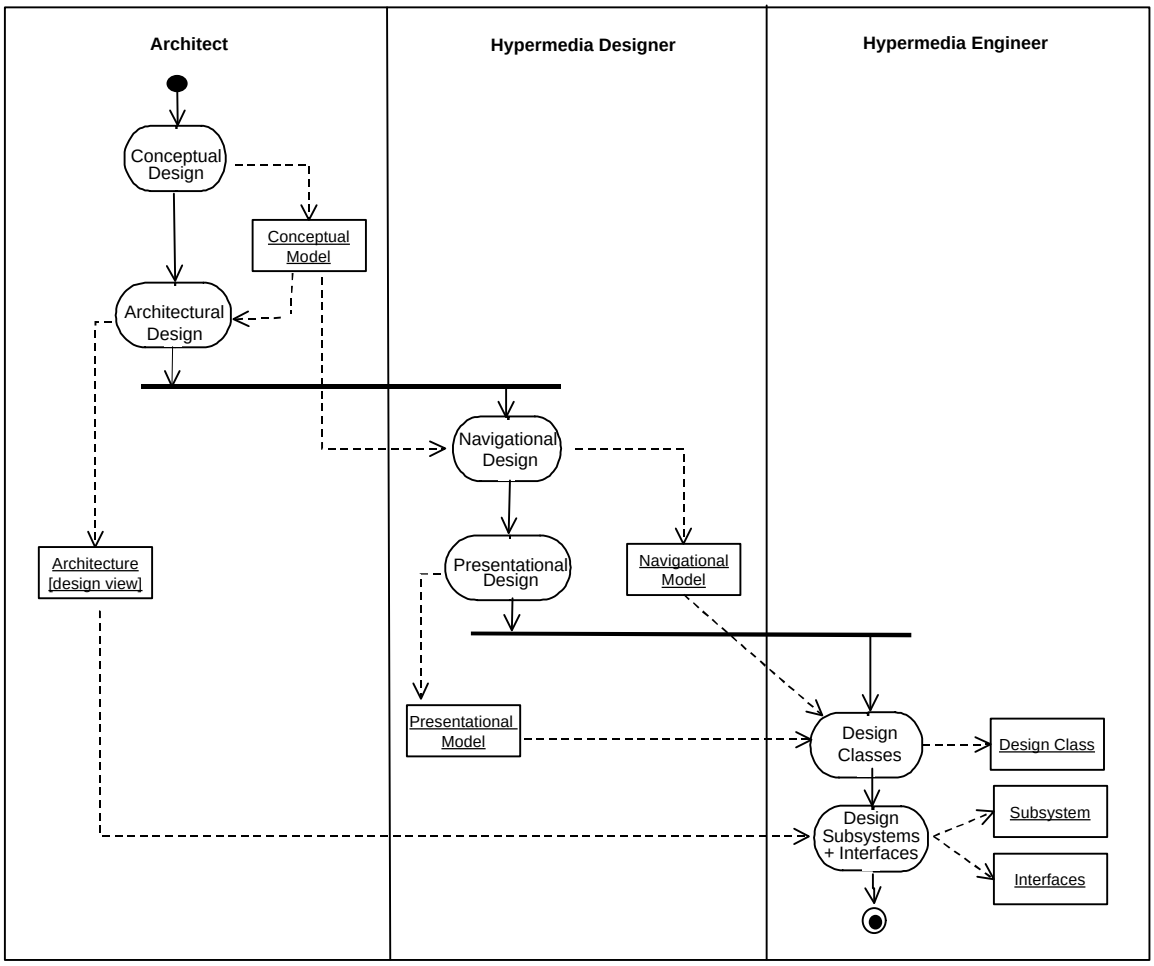


Figure 4: Analysis and Design Workflow

The *conceptual, navigational* and *presentational design* activities provide a clear separation of the information the user can access, how this information is structured and how it is presented to the user. This separation allows for a more modular and reusable design.

The last two activities of the list, i.e. *detail design classes* and *define subsystems and interfaces* are performed by the hypermedia engineer in the late iterations of the design. The goal is to detail the design class and group them into subsystems preparing them for the next workflow where the focus is put on implementation.

4.2.1 Artifacts

The artifacts produced in the analysis and design workflow are a design view of the architecture, a set

of models, design classes, subsystems and interfaces. These models are the user model, conceptual model, navigational model and presentational model.

- **Conceptual Model**

The conceptual model is a model of the problem domain trying to leave out navigational and presentational aspects that are a particular characteristic of hypermedia applications. Therefore, the conceptual model is the same as the business or domain model for traditional software development. A conceptual model is represented as a class model.

- **Navigational Model**

The navigational model is build in two steps. In the first step a *navigational class model* is constructed based on the conceptual model. The result of the second step is a *navigational structure model* that is build on the navigational class model.

Navigational Class Model

The *navigational class model* defines a view on the conceptual model showing *which* classes of the conceptual model can be visited through navigation in the application. The navigational class model is a UML class diagram that is built with a set of navigational classes and associations between these navigational classes. Classes and relationships are obtained from the conceptual model, i.e. each navigational class and each association of the navigational class model is mapped to a class, respectively to an association in the conceptual model.

In the navigational class model navigability is specified for associations, i.e. direction of the navigation along the association is shown through the arrow attached to the end of the association's line. We distinguish for each link a navigational source object and a navigational target object; the latter one may be determined dynamically.

A navigational class is defined as a stereotyped class (see Figure 5) with the same name as the corresponding class of the conceptual model. *Navigational objects* are instances of these navigational classes connected by *links* (in UML terms) that are instances of the associations of the navigational model.

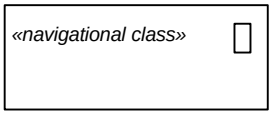


Figure 5: Stereotype for navigational class

Some conceptual classes may be reduced to attributes of navigational classes. The values of these attributes can be computed from some conceptual objects. The formula to compute the derived attribute is given by an OCL expression. A derived attribute is denoted in UML by a slash(/).

Navigational Structure Model

The *navigational structure model* defines the navigation of the application, i.e. how navigational objects are visited. It is based on the navigational class model. Additional model elements are required to perform the navigation between navigational objects: *menus, indexes, external nodes* and *navigational contexts*. The concept of *navigational* node is introduced to refer to navigational objects as well as the above mentioned model elements.

Navigational Contexts

A *navigational context* (context for short) consists of a sequence of navigational nodes. This concept was introduced by OOHDM (Schwabe & Rossi, 1996) to permit different groupings of the navigational objects. This way a navigational object can be navigated in different contexts.

Example: The Film Assistant
The information about the film “Casablanca” is shown as one of the “films of the year 1942” and one of the “films with Humphrey Bogart”.

The set of navigational nodes belonging to a navigational context are usually connected through an aspect, such as being objects of a given class or being related to another class through an association. Navigational contexts allow the organisation of the navigational space in sequences that can be traversed following a predefined order. They include the definition of links that connect each navigational node belonging to a navigational context to the previous and to the next navigational node. In addition, links to the first object and to the last object as well as circular navigation may be defined. Navigational contexts may be nested. Navigation is usually performed within a navigational context, but navigational context changes are possible.

Example: The Film Assistant
For example if “Casablanca” is visited in the context of “films with Humphrey Bogart”, it is possible to continue with other “films of the same genre” or “films of the same year”.

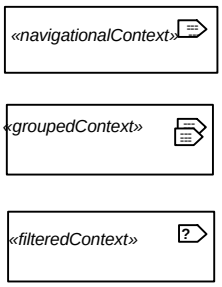


Figure 6: Navigational Context Types

Three different types of context are distinguished: *navigational context* (simple context), *grouped context* and *filtered context.*

A *filtered context* allows a dynamic selection of a collection of elements from a navigational context satisfying a property. This property is supplied usually by the user in a query.

Stereotyped classes *«navigationalContext»*, *«groupedContext»* and *«filteredContext» are defined* (Figure 6), which have associated an OCL-expression defining the sequence of navigational nodes.

Navigational contexts for the same navigational class are grouped in a UML package. Navigational contexts are related through special associations which allow for context changes. This is possible since the same object is part of different sequences (navigational contexts).

A stereotyped association *«change»* is defined to permit navigation in another context and to return to the starting point before the navigational context is changed. The possibility to change from one context to another within a package is the default semantics for a package of contexts. In such a case context changes need not explicitly be drawn. When only some changes are allowed, a diagram of the package must show the permitted context changes.

Example: The Film Assistant
An example of a simple navigational context is „all festivals“ (festivals by event). A *grouped context* is a sequence of sequences of navigational nodes, such as „films by actor“ (denotes a sequence of actors, each of them has played roles in a sequence of films). The result of the query: “all films nominated for the category best original music in 1998” is an example of a filtered context. This can be specified with the following OCL constraint: Film :: filter (cat:String,y:Integer) : Set(Film) **post:** self**.** AllInstances → select (f:Film | f**.**awards**.**category = cat and f**.**awards**.**year = y and f**.**awards**.**nomination) For example in *The Film Assistant* it is possible to change from any context of class Film to the context News by film, which will show all news for the given film.

Access Structures and External Nodes

In a navigational model it is necessary to specify how navigational contexts are accessed. The model elements defined for this access (called access primitives) are: *indexes, guided tours, queries* and *menus*. Stereotyped classes *«index»*, *«guidedTour»*, *«query»*, *«menu»* and *«external node»* are defined for them and for external nodes. A graphical notation for these stereotypes is shown in Figure 7.

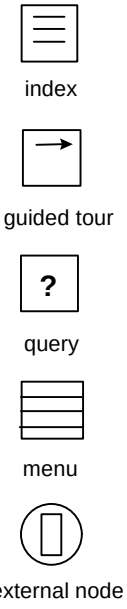


Figure 7: Access primitives and external node

- An *index* allows direct access to each element within a navigational context. It comprises a list of descriptions of the nodes of the navigational context from which the user selects one. These descriptions are the starting points of the navigation. Sometimes navigation to the neighbours (next, previous) is suppressed.
- A *guided tour* gives access to the first object of a navigational context. Objects are navigated then sequentially. Guided tours may be controlled by the user or by the system.
- A *query* is an input form; when evaluated it produces a filtered context.
- A *menu* is an index on a navigational context of a set of navigational nodes. Hypermedia applications have at least one entry point or initial node, so called main menu (start page).
- An *external node* is a navigational node belonging to another hypermedia application, i.e. this node is not part of the application that is being modelled.

♦ **Presentational Model**

The presentational model is the representation of an *abstract user interface*, showing how the navigational structure is presented to the user. The same navigational structure may yield different presentations depending on the restrictions of the target platform and the technology used.

Most of the methods for hypermedia design suggest the development of prototypical pages for this activity. In this work instead, we propose first to define a presentational model as a composition of user interface objects. UML model elements and UML diagrams are chosen as technique in the same way as for the conceptual and navigational model. The presentational model is a rough design of the user interface; decisions about details such as size, colour or font of user interface elements are taken when a prototype is developed or will be taken in the implementation phase.

For each navigational object at least one presentational object has to be defined. If the presentation of an object depends on the navigational context within the navigational object is visited, one presentational object for each context has to be specified.

The presentational model consists of the static presentational model and the dynamic presentational model. The first one is represented by UML composition diagrams that describe how the user interfaces are built. The second one is represented by UML state diagrams describing the behaviour of the components, i.e. how navigation is activated and which user interface transformations take place.

Static Presentational Model

The *static presentational model* provides hints how the navigational nodes of the navigational model are presented to the user. For example, it gives a first approach of position and size of the user interface elements relative to each other. It consists of a collection of user interface objects that are represented by UML composite objects showing the composition of user interface objects by other user interface objects and relationships between these user interface objects (Koch, 1998; Koch & Mandel, 1999).

A user interface object can be either a primitive user interface object like text, image and button, or a composition of user interface objects. A presentational object is a special kind of composite user interface object based on a navigational object.

For the most frequently used user interface objects we define stereotypes according to the extension mechanism provided by UML. These user interface objects are*: anchor, text, image, audio, video, form, button, collection and anchored collection* (see Figure 8). The meaning of these user interface objects is the following:

- An *anchor* is a clickable area that is the starting point of a navigation establishing this way the relationship to other nodes. An anchor consists of a presentation together with a link. The presentation may be either a text (even a single character), an image, a video, a group of mixed media, any interactive object or a whole document. There are seldom presented in the literature as independent objects (De Bra & Calvi, 1998 and Halaz & Schwarz, 1994), mostly as part of hyperlinks. Since anchors are one of the most frequent used objects in the hyperspace model, a special notation is defined for them: an underlined description in analogy to the notation used usually by browsers.
- A *text* is a sequence of characters together with formatting information.
- A *button* is a clickable area, which has an action associated to it. Example of actions are playVideo, displayImage and stopAudio. Note that anchors may be implemented as buttons that means that buttons can also trigger navigation.
- A *form* is used to request information from the user. They supply information in one or more input fields or select options from a browser or checkbox. The semantics of this model element includes the display of the content, the waiting for the user activity, the evaluation of the input and the trigger of the defined event.
- *Image, audio* and *video* are multimedia objects. An image can be displayed; audio and video can be started, stopped, rewinded and forwarded. To provide these functionality active user interface objects may be associated to these multimedia objects, such as buttons or anchors.
- *Collections* and *anchored collections* are model elements introduced to provide a convenient representation of composites that are frequently used. It avoids the textual description by comprehension or by extension. A collection consists of a set of user interface objects. Anchored collections are a set of anchors. It is not specified whether the collection will laid out horizontally or vertically; objects still may be arranged as a table.
- *External applications* are model elements of other applications not related to the navigational objects of this application. They can be implemented for example by applets, Active-X components or embedded objects.

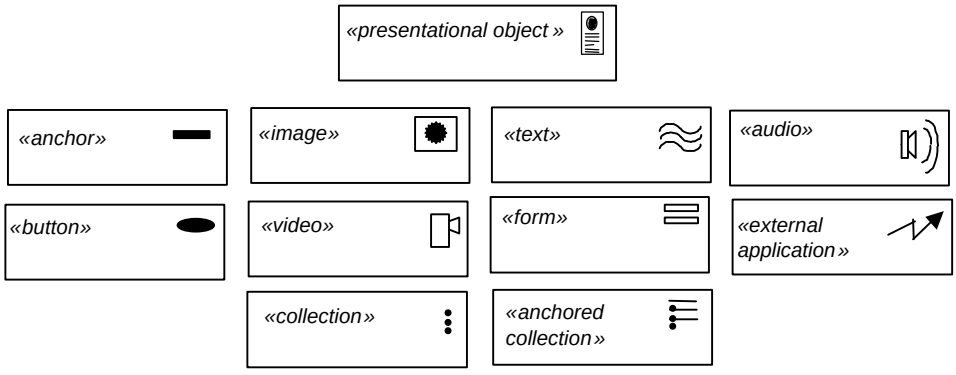


Figure 8: Stereotypes for User Interface Objects

Dynamic Presentational Model

The *dynamic presentational model* describes the changes on the user interface when the user interacts with it or when the system reacts to internal events such as timeouts.

UML state diagrams have been chosen to depict the dynamic presentation model. The behaviour of a presentational object is defined through states and through transitions between states. A state is specified by a name, entry and exit actions, internal transitions, and/or sub-states. A transition has an action associated to it, i.e. it is triggered by an event. In the case of user interfaces most of the events are generated by the user, such as mouse focus, mouse clicks, or keyboard inputs. Complex behaviours can be modelled easily in UML with sub-states. Two different types of sub-states are possible: sequential and concurrent [2,4].

The design of these UML state diagrams is expensive and usually so many details are not necessary for user interface objects with well known behaviour. Therefore, we use them only for complex composite user interface objects.

♦ **Architecture (design view)**

The architecture description from the design view depicts the architecturally important artifacts. They are:

- the subsystems, which the system is divided into,
- the interfaces of these subsystems,
- key design classes that represent some generic design mechanisms and have many relationships to other classes that are not necessarily detailed in the architectural view.
- design classes that are related to the realisation of important use cases.

♦ **Design Class**

During the first iterations in the analysis and design workflow classes are outlined, i.e. more relevant attributes and operations are described. In successive iterations details are added to finally produce a design class. A hypermedia system requires the definition and the detailed description of design classes for the conceptual, navigational and presentational models. For each of these design class:

- all attributes and operations are defined,
- visibility of attributes and operations is often specified,
- relationships in which the design class is involved sometime; this implies the addition of attributes to the design class,
- the methods, i.e. the realisation of the operations, is detailed in natural language or in pseudo-code,
- active classes, i.e. classes, whose objects maintain their own threads, are identified.

♦ **Subsystem**

Subsystems are defined to group artifacts of the design models in more manageable pieces or for a separation of design concerns. A subsystem consists of design classes, interfaces, use cases or other subsystems. A subsystem can be used to represent legacy systems or part of them or to represent reusable software components. Another purpose of subsystems is to encapsulate the classes contained, showing behaviour only through the interfaces of the subsystem.

♦ **Interface**

An interface is used to specify the operations provided by design classes and subsystems. A design class that provides an interface has also to provide methods that realise the operations of the interface. A subsystem that provides an interface has to contain a design class that provides this interface.

4.2.2 Workers

The analysis and design in the development of hypermedia application is performed by at least the following workers: architect, hypermedia designer and hypermedia engineer. The role of the hypermedia designer is very often performed by more than one hypermedia expert, such as navigation designer, multimedia expert and graphic designer.

♦ **Architect**

The architect is responsible in the design for the integrity of the *user model*, the *conceptual model* and the *design view of the architectural model*.

The profile of the architect has been defined in the requirements capture workflow.

♦ **Hypermedia Designer**

The hypermedia designer is responsible for the use case realisation. The functionality described in the use cases has to be reflected in the navigational structure, dynamic page generation and the adaptive user interface of the hypermedia application. The hypermedia designer develops the navigational model and presentational model.

The profile of the hypermedia designer is given by the following skills:

- knowledge about the functional and non-functional requirements of the system,
- general knowledge of the business domain and hypermedia technology domain,
- experience in the design of the structure of a hypermedia system, and
- UML knowledge for the modeling activities.

♦ **Hypermedia Engineer**

The hypermedia engineer details and maintains the attributes, operations, methods, relationships and implementation requirements of one or more *design classes* as well as the integrity of one or more *subsystems* and *interfaces*. It is often appropriate to let the same hypermedia engineer be responsible for a subsystem and the model elements contained in the subsystem. The implementation is done then by the same hypermedia engineer profiting from his knowledge of these model elements.

The profile of the hypermedia engineer is given by the following skills:

- UML knowledge for the modeling activities,
- experience in the implementation language that has be chosen,
- the technologies with which the system will be implemented, and
- experience in HTML, JavaScript, database definition and generation, multimedia design and/or integration, etc.

4.2.3 Activities

Throughout the analysis and design workflow, the developers will perform a set of activities to model the content, structure and interface of the hypermedia. These models consists of classes and

relationships that can be grouped into subsystems.

The following activities are included in these workflow: conceptual design, navigational design, presentational design, architectural design, detail design classes, define subsystems and interfaces.

♦ **Conceptual Design**

The activity conceptual design aims to build a domain model including all the concepts that are relevant to the application and the different users or user groups identified in the requirements capture workflow. The main objective is to capture the domain semantics with as little concern as possible of the navigation paths, presentation and interaction aspects. Decisions as whether each concept corresponds to one hypermedia page, a hypermedia document or if the page is generated on-the-fly based on the frame-based internal representation of domain concepts, are postponed to the implementation phase.

Activities of the conceptual design are typical object-oriented modeling activities, such as:

- to find classes,
- to specify the most relevant attributes and operations,
- to determine relationships between classes,
- to define inheritance hierarchies,
- to find dependencies,
- to identify interfaces.

Well-known object-oriented modeling techniques are used for the modeling, such as composition, generalisation and specialisation. UML packages can be used to group classes and associations. OCL constraints can be included in the diagram or specified separately (see example below).

The results of this activity is a UML class model of the problem domain. Classes and associations defined in this step are used during navigational design to derive nodes of the hypermedia structure. Relationships will be used to derive links.

Example: The Film Assistant

For the model of the *Personal Film Assistant* the following classes have been identified: Film, Festival, Person, Award, News, Genre, Country and Keyword. Persons are part of Films and Awards are part of Festivals. A Film is specified by its attributes and the classes Genre, Keyword and Country. A Film can be related to other Films. News are related to Films and to Persons taking part in a film production.

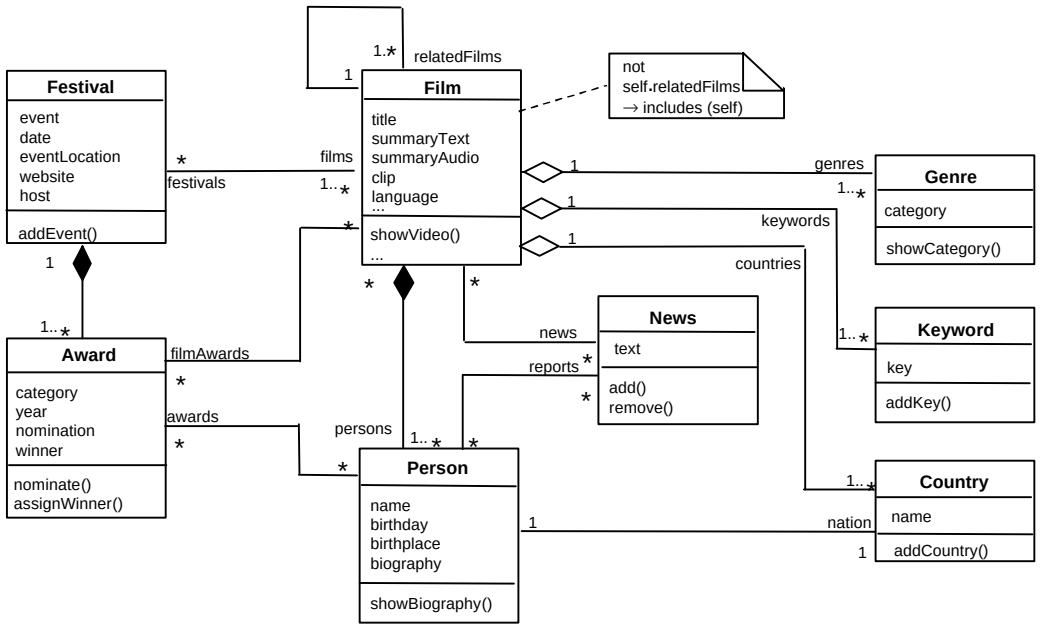
The Conceptual Model of the *Film Assistant* is depicted below.

The graphical representation of the conceptual model includes an invariant to establish that films are not related to themselves. Another invariant is separately specified. It specifies that if an person is honoured with an award, this occurs in relation to a film, i.e. the film is honoured with the same prize. This can be expressed by the following OCL expression:

Film

```
inv:  self.persons.awards → forAll ( a:Award |  self.filmAwards →
        exists ( aw:Award | a = aw ) )
```

Example: The Film Assistant



♦ **Navigational Design**

Navigational design is a critical step in the design of every hypermedia application. Even simple applications with a non-deep hierarchical structure will become complex very soon by the addition of new links. Additional links improve navigability on the one hand but imply on the other hand higher risk to lose orientation. Building a navigational model not only is helpful for the documentation of the application structure, it also allows for a more structured increase of navigability.

The navigational design defines the structure of the hypermedia application. A navigational model is build in two steps. The first one describes the rough navigation structure based on navigational classes and links. It is called the *navigational class model*. The second one refines this model describing how objects can be grouped to optimise navigation and collaborate to fulfil the navigation. Navigational contexts are defined and access structures are added in this step. The result is a *navigational structure model*.

Step 1: Constructing the Navigational Class Model

The *navigational class model* can be seen as a sub-graph of the conceptual model where some classes which are not relevant for the navigation are eliminated and/or reduced to attributes of other classes. The following "rules" can be applied to construct the navigational class model on basis of the conceptual model:

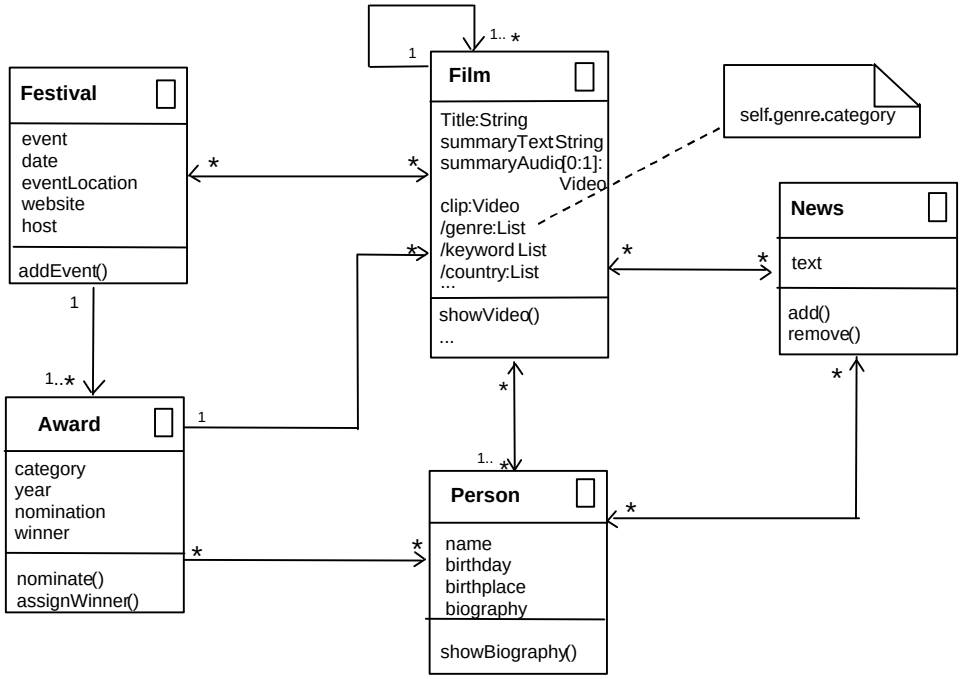
- Navigational classes map to conceptual classes.
- An attribute of a navigational class may map on a conceptual class which is irrelevant for the navigation process but whose content is part of the presentation.
- Navigational relationships (links) map directly on conceptual relationships (associations).
- Some conceptual relationships are excluded if a class has been suppressed or if the relationship is irrelevant for navigation.

- Attributes of navigational classes map directly attributes of the corresponding conceptual class.
- Attributes are excluded if they are neither needed for navigation nor for presentation.
- Relationships are added to improve navigation if necessary.
- Navigability is added to the association ends indicating the source navigational class and the target navigational class.

For navigational classes a stereotype *«navigational class»* is defined (as shown in Figure 5). Navigational classes differ from corresponding conceptual classes in attributes and methods that are added or eliminated. Navigational classes and associations with navigability are graphically represented in a UML class diagram.

Example: The Film Assistant

The resulting *navigational class model* for *The Film Assistant* example is shown in Figure below. The first activity is to identify the classes of the conceptual model that are relevant as nodes for the hypermedia application. In this example, all classes of the conceptual model with the exception of Genre, Keyword and Country are included in the navigational model. These three classes are not relevant for the navigation as they will not have pages associated to them. Therefore, Genre, Keyword and Country are incorporated as derived attributes in the navigational class Film. These derived attributes are denoted in UML by a slash(/) before its name as shown in the navigational class model of *The Film Assistant*.



Step 2: Constructing the Navigational Structure Model

In this step the *navigational structure model* is build. It defines the navigation of the application, i.e. how navigational objects are visited. The start point for this constructing activity is the navigational class model, but also additional model elements has to be defined for navigation between navigational objects: *navigational context and access primitives, such as menus, indexes, external nodes*. A UML object diagram is build to show how these navigational contexts, access primitives and external nodes collaborate.

The activities that are performed to pass from the navigational class model to the navigational structure model are the following:

- *defining navigational contexts*. For each navigational class at least one navigational context has to be defined. A simple navigational context is required to access the context from a menu through an index. A filtered context is appropriate when a query is used to reach the context. For each association with navigability and multiplicity “zero or more” or “ one or more” a context “by the source class” has to be included. These navigational contexts are grouped into a UML package for graphical representation’s purposes.
- *determining context changes*. Context changes are added to the model according to the typical navigation activities of the user that were determined with a use case or a scenario based analysis.
- *adding access primitives*. One type of access to navigational contexts has to be chosen in each case. Context changes usually require the definition of additional indexes.

Example: The Film Assistant

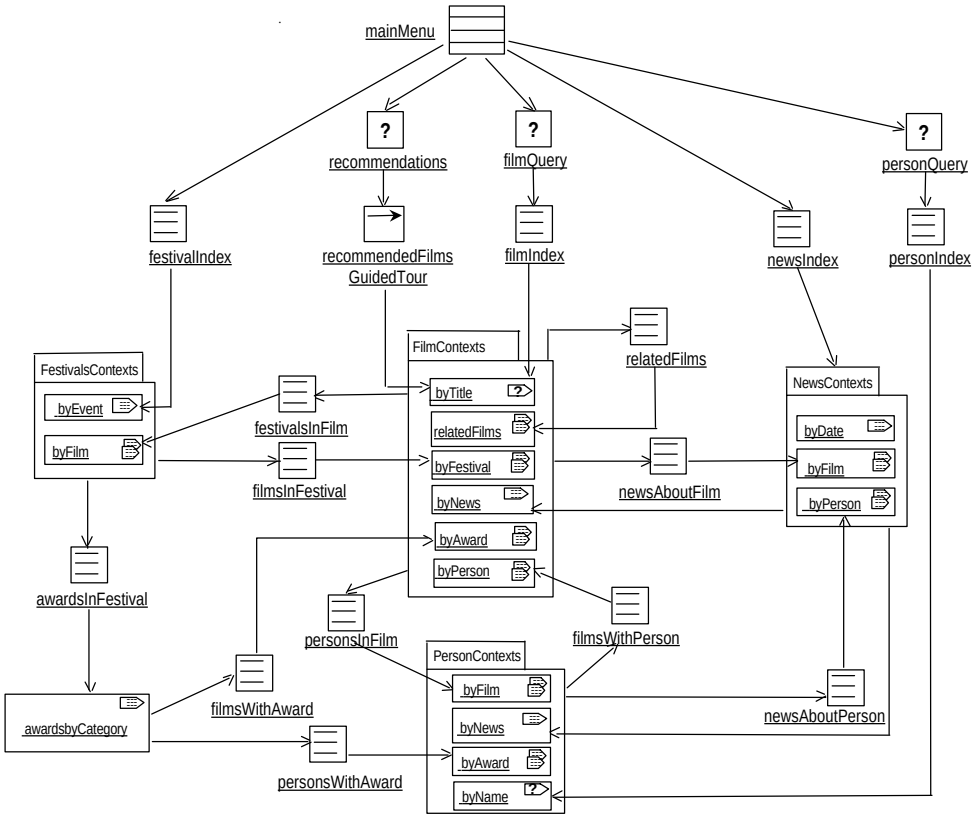
The Film Assistant is used as an example to illustrate the navigational structure model. Given the navigational classes and associations defined in the navigational class model and according to the first activity mentioned above, the following are some of the navigational contexts needed: film by title, related films, film by person, person by film, awards by category, news by date, news by film, etc.

Each navigational context consisting of more than one navigational node requires at least one access primitive. A main menu is the starting point to access different navigational contexts related to the defined navigational classes.

Indexes are added to permit direct access to the objects of a navigational context. In this case there are filmIndex, personIndex, festivalIndex and newsIndex. A guided tour through recommended films is incorporated to assist users trying to select a film they have not seen yet based on some user preferences. Queries to find films and persons according to users input or choices in a pull down menu are also possible to be modelled.

The figure below shows the UML object diagram for the navigational structure model of *The Film Assistant.*

Example: The Film Assistant



♦ **Presentational Design**

Presentational design means defining the way in which navigational objects will appear, which interface objects will activate navigation and which interface transformations will take place. The developer has to decide in this design step which type of adaptation is chosen for the hypermedia application.

In this work it is proposed to model the user interface with the help of a static and dynamic object-oriented model. In addition, a prototype can be built. The models only include a rough design of the user interface elements, details such as position, size or colour are decided when the prototype is constructed or are postponed to the implementation phase.

The presentational design supports the modeling of an *abstract user interface*, showing how the navigational structure is presented to the user. To achieve this purpose, a static and a dynamic presentational model is built. Presentational design means defining the way how navigational nodes will appear, selecting user interface objects to activate navigation, and determining which interface transformations will take place. The same navigational structure may yield different presentations depending on the restrictions of the target platform and the technology used.

Most of the methods for hypermedia design suggest the development of prototypical pages for this step. Instead, we propose first to define a paper presentational model as a composition of user interface objects.

For each navigational node at least one presentational object has to be defined. If the presentation of an object depends on the navigational context within the navigational object is visited, one presentational object for each context has to be specified.

Step 1: Building the Static Presentational Model

The *static presentational model* provides hints how the navigational nodes of the navigational model are presented to the user. For example, it gives a first approach of position and size of the user interface elements relative to each other. It consists of a collection of user interface objects, which are represented by UML composite objects showing the composition of user interface objects by other user interface objects and relationships between these user interface objects. A user interface object can be either a primitive user interface object like text, image and button, or a composition of user interface objects. A presentational object is a special kind of composite user interface object based on a navigational object.

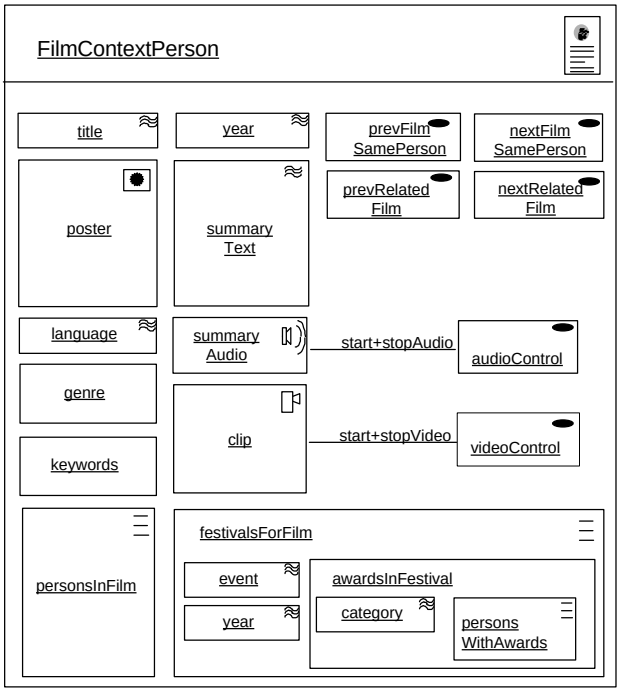
The following activities are performed to built the static presentational model based on the navigational structure model:

- At least one presentational object is defined for each navigational node.
- A presentational object is shown as the composition of:
  - objects obtained from the attributes of the corresponding navigational node,
  - interactive objects to control audio and video,
  - anchors or buttons to allow for navigation, and
  - indexes or guided tours to access to contexts defined for other navigational classes.

When the presentational objects have to fulfil special requisites, they can be specified with OCL constraints.

Example: The Personal Film Assistant

For each presentational object of the example a UML diagram for a composition object is drawn. This diagrams show how attributes of a object are represented with appropriate user interface objects, such as images, text, buttons, anchors, etc. The example below shows the film in the context person. Therefore, it includes buttons to link to the previous and next film of the same person (actor, director, etc). These are the previous and next objects in the context. In addition, this film is linked to related films thorough the buttons “prev related film” and “next related film”.



Step2: Describing the Dynamics of the Presentational Model

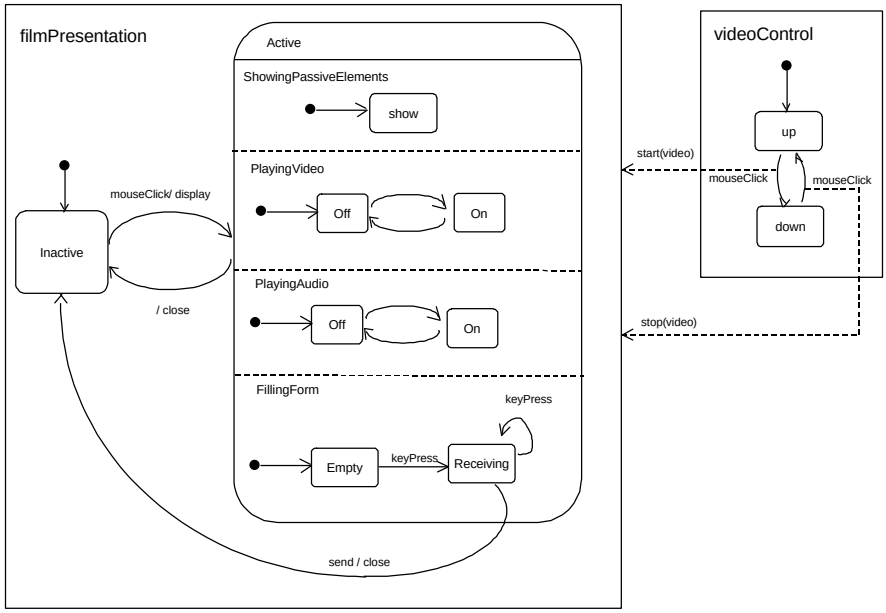
The goal of the *dynamic presentational design* is to describe the behaviour of the presentational objects, i.e. the changes on the user interface when the user interacts with it or when the system reacts to internal events such as timeouts.

Each presentational object may be active or inactive; and perceptible for the user or not perceptible. Perceptible means audible in the case of audio and visible in case of all other user interface objects. Only one user interface object can be active at the same time. We use two variables (*activation* and *perception*) which contain the active user interface object and the list of the currently perceptible objects, respectively. Whenever an element is added to the *perception* variable the internal event *show* is generated and whenever it is removed the event *hide* is generated.

The contents of these variables are changed when an action is triggered by a transition in the dynamic presentational model. We can distinguish between macro navigation and micro navigation according to the total or partial replacement of the user interface object. In the context of Web design, for example, macro navigation can be implemented by fetching a new HTML page from the server, while micro navigation can be implemented using frames, which allow to replace only part of the display.

Example: The Film Assistant

In the example concurrent substates are used to denote that the state Active state of a filmPresentational object has the following substates: showing PassiveElements, playingVideo, fillingForm and playing Audio. The object is in any sequential state from each one of the concurrent substates. The first of these substates has only the state show. It can be excluded from the state diagram as it will permanently remain in this state. The state diagram for the object AudioControl is not included as it is similar to VideoControl state diagram.



♦ **Architectural Design**

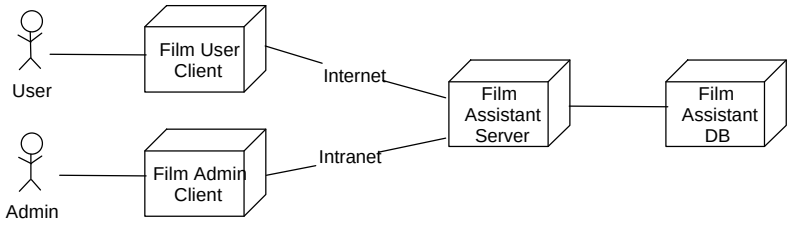
The purpose of the architectural design is to outline the architecture (design view) by identifying the following:

- subsystems and their interfaces,
- design classes, that are relevant for the architecture,

- generic design mechanisms to handle functional and non-functional requirements
- reuse possibilities, such as reusing parts of similar systems or general software products.

Example: The Film Assistant

The following is a simple architecture for the this application.



♦ **Detail Design Classes**

During the previous modeling activities of the analysis and design workflow: conceptual design, navigational design, presentational design and architectural design a set of classes have been outlined. During the first iterations usually the classes are named, some attributes are defined, sometimes some operations are identified.

In successive iterations the classes have to be detailed. The activity *detail a class* has to be performed by the hypermedia engineer because he knows the implementation requirements for classes. This activity includes the following sub-activities: define the class operations, define class attributes, identify the relationships it participates in (aggregation, association, inheritance, dependency), describe its methods, determine its states and establish the requirements relevant to its implementation.

Example: The Film Assistant

The description of the design class for the "film presentation in the context person" is included as example.

Name: FilmContextPerson

Description: contains the information needed to present a film that in the navigation context of film related to a person (actor, director, etc.).

Inherits from class: Film

Attributes: linkPreviousFilm and linkNextFilm (for the same person)
audioControl and videoControl
.....

Operations: getTitle(), getPoster(), ...

Relationships: Film, Person, PersonContextFilm, Festival, Awards.

Diagrams: Navigational Structure Diagram

Special requirements: None

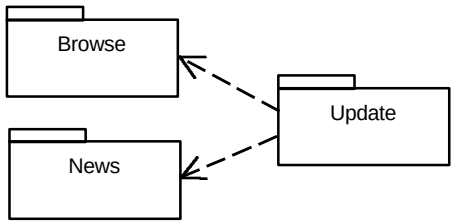
Trace: navigational classes Film and Person.

♦ **Define Subsystems and Interfaces**

The objective of dividing the system in subsystems is to obtain a set of subsystems that are as independent as possible one from each other. This independence permits that each subsystem may be implemented by another hypermedia engineer. Subsystems have to provide interfaces to fulfil their purposes. The number of dependencies from one subsystem to the others has to be minimised as much as possible.

Example: The Film Assistant

The following subsystems have been identified in the application: browse, update and news presentation.



4.3 Implementation

Implementation consists of transformation of the results of the design phase, i.e. design classes, subsystems and interfaces into an implemented system in terms of components, i.e. source code, scripts, executables, etc. Implementation is accomplished in successive activities starting with generation of components, assembling subsystems and interfaces and finishing with software integration. The typical components that need to be produced during implementation of hypermedia systems are:

- media components, such as text, images, video, audio and animation (they constitute the content of the hypermedia application);
- databases to store the content, i.e. a relational database or a mark-up file system organised into a hierarchical or graph structure;
-
- structure components required by the hypermedia paradigm, such as menus, indices, guided tours and links;
- user interface components, such as windows, frames, buttons, logos, forms and banners;
- components for the dynamic page generation, such as CGI scripts or Java servelets;
- search engines;

Implementation is the focus during construction, but implementation may also be carried out during inception when a prototype is created, during elaboration to create the baseline architecture, during transition to handle late defects and during maintenance to update the current version.

The implementation activities are performed by five different types of workers: the architect, content provider, multimedia designer, hypermedia engineer and system integrator. They produce the following artifacts: the implementation view of the architecture, the implementation model, the deployment model, the content, the hyperspace structure, the user interface, the integration plan and the hypermedia system.

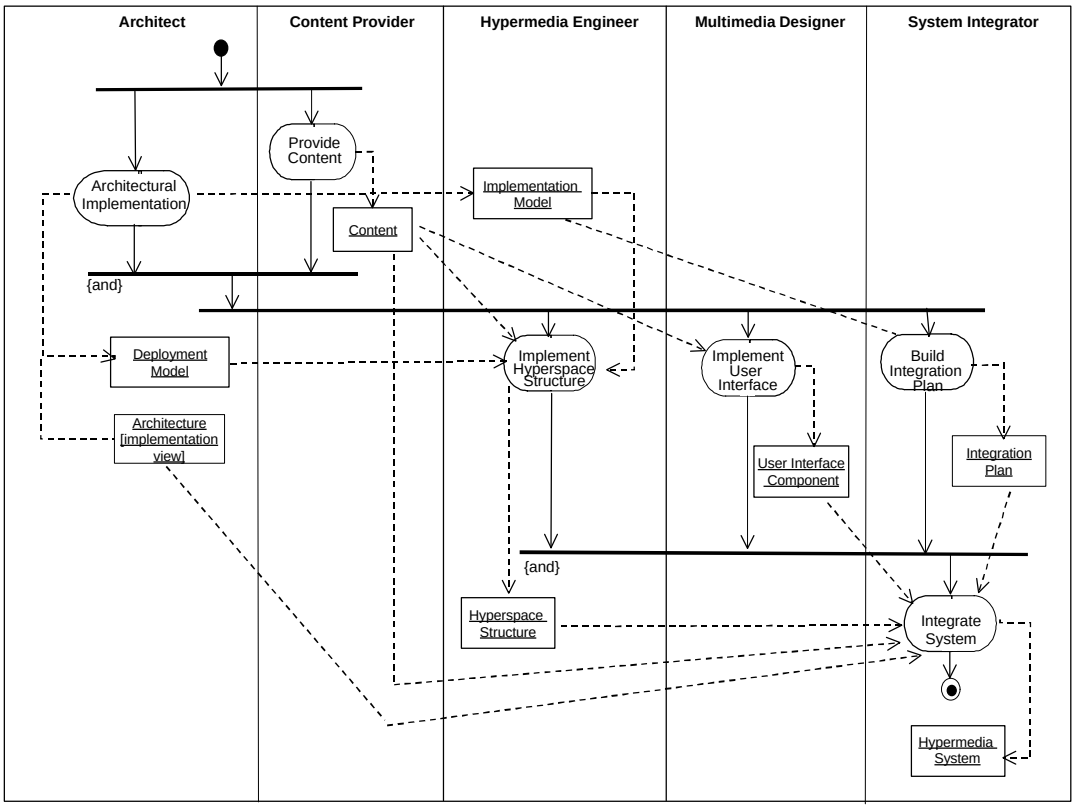


Figure 9: Implementation Workflow and Object Flow

♦ **. Artifacts**

During the implementation workflow the artifacts produced are the components of the system. They are related to the content, user interface and hyperspace structure. Supporting the development of these artifacts some models are build or refined. They are the implementation model, deployment model and the implementation view of the architecture. The integration plan is also an artifact of this workflow as well as the hypermedia system that is the result of the integration activity.

♦ **Implementation Model**

The implementation model describes how the elements of the design model are implemented in terms of components of source code, executables, existing components. It also describes how the components are organised and grouped, and how they depend on each other. The artifacts in the implementation model are components, subsystems and interfaces. According to the functionality in the hypermedia system of these model elements, we classify them into content model elements, structure model elements and presentational model elements.

♦ **Deployment Model**

The deployment model is an object model that describes the physical distribution of the system among computational nodes. It shows the mapping between the software architecture and the system architecture, i.e. the hardware architecture.

♦ **Architecture (implementation view)**

The architecture description contains an architectural view of the implementation model showing the

artifacts that are relevant for the architecture. These artifacts are: the subsystems, interfaces and their relationships as well as key components of the application.

- **Integration Plan**

The integration plan describes the sequence of incremental iterations required to construct the hypermedia system by successive integration of subsystems and components. It details the parts that are to be added in each iteration and the functionality that is expected to be implemented after each integration. The partial system construct in each iteration is called by the Unified Process *system build.*

- **Content**

The content includes different types of media, such as text, images, audio, video and animation. Each media type requires a special treatment and appropriate tools to handle them during creation or adaptation.

Text is usually the predominant media type in hypermedia applications. If a text is not specially created for a hypermedia application, usually it requires some adaptation work. Texts included in hypermedia applications must be, whenever it is possible, specific, concrete and precise. Two types of images are used as complement for the text: bitmap graphics and vector graphics. The major sources of bitmap work are photographs, scanned images, screen dumps and pictures created with special paint programs.

Audio, video and animation are dynamic time based media. The effective use of sound seldom can substitute written information, but has the advantage to attract the user's attention. Video provides a rich source of documentation. The quality of the video material plays an important role for its usefulness in a hypermedia application. Animation adds impact to a presentation and contributes enormously e.g. to a learning process.

The storage of this multimedia content requires some kind of organisation, such as multimedia databases or a set of files with a hierarchical directory organisation.

- **Hyperspace Structure**

The hypermedia structure is given by a hierarchical HTML file organisation or by a set of templates which are dynamically filled with data by CGI-scripts at run-time when these pages are requested. The dynamic generation of the pages assures the separation of the content from the structure and presentation facilitating maintenance. In addition, menus and indices have to be created to include additional navigational support.

- **User Interface**

The user interface is the component of the application with which the user has the most direct contact to. It provides to the user the look and feel of the application. Multimedia designer have to find the right balance of size, colours and position of user interface objects avoiding cognitive overloading..

- **Hypermedia System**

The hypermedia system is the sum of the content, hyperspace structure and user interface components that are integrated to provide the full functionality of the application.

4.3.1 Workers

The implementation of hypermedia applications require a more heterogeneous group of workers than the

development of traditional software. Content providers and multimedia designers are required in addition to the architect, the component engineer and the system integrator. The component engineer is called hypermedia engineer since his profile contains specific skills required for hypermedia development.

♦ **Architect**

During the implementation phase, the architect is responsible for the integrity of the implementation model and ensures that the components of this implementation model are implemented and integrated. He is also responsible of the mapping of executable components into physical nodes.

The profile of an architect is given in Section 4.1 when the responsibilities of architect are defined in the scope of the requirements capture workflow.

♦ **Content Provider**

The content provider is responsible to provide all the raw material that will be included in the hypermedia application. This material consists of text, images, video, audio and/or animations. He has to digitise them and chunk them into appropriate pieces of information.

The profile of the content provider is given by the following skills:

- experience with the legacy applications which are the source of data,
- text composer experience, and
- knowledge about tools to manipulate images, video, audio and animations.

♦ **Multimedia Designer**

The multimedia designer is responsible for the production of all multimedia elements that the hypermedia application requires, such as windows, buttons, logos, images, etc.

The profile of the multimedia designer is given by the following skills:

- knowledge about tools to create and manipulate images, video, audio and animations,
- experience in the design of multimedia elements, and
- creativity skills.

♦ **Hypermedia Engineer**

The hypermedia engineer defines and maintains the source code of one or several file components that implement the structure of the hypermedia application. He assures that these components implement the correct functionality, i.e. the functionality specified by the design classes. Usually the hypermedia engineer of a complex application is responsible for all the components of one subsystem.

The profile of the hypermedia engineer is outlined in the Section 4.2 together with his activities in the analysis and design workflow.

♦ **System Integrator**

The responsibility of the system integration can not lie by any of the hypermedia engineers. Instead, a system integrator is assigned to plan the sequence of system's builds in each iteration and the successive integration of the subsystems. The result of his planning activities is the integration plan.

The profile of the system integrator is given by the following skills:

- good communication skills,
- general domain knowledge,
- experience in system integration, and
- experience in the implementation language that has been chosen.

4.3.2 Activities

The main goal of the implementation workflow is to obtain an implemented system. To achieve this goal the system architect outlines the key components of the implementation model. Based on the implementation model and the content provided (by the content provider) the hypermedia engineer implements the hyperspace structure and the system integrator builds the integration plan. The multimedia designer implements the user interface using the implementation model and the content therefore. The objective of the system integration activity is to combine the content, hyperspace structure and user interface and to test the functionality of each integrated system build.

♦ **Architectural Implementation**

The objective of the architectural implementation is to outline the implementation model, its architecture and to identify the components that are relevant for the architecture, such as executable components and to map components to nodes of the network configuration.

♦ **Provide Content**

An important activity during the implementation process is the capture or generate the underlying data for the content and convert it into an appropriate form. It has not to be underestimated. The conversion process is required for example when the data to be used in the hypermedia application comes from legacy applications, such as paper-based documentation, manuals or image, audio and video archives.

This process involves several activities such as:

- obtaining the raw data from legacy records or by new recording of information,
- scanning text and images,
- digitising images and video,
- capturing audio,
- applying character recognition to scanned text,
- adjusting quality, size, colours of images,
- chunking the data into appropriate pieces of information, etc.

The main problem is the difficulty to automate some of these activities, such as the adjustment of quality of images and the correction of scanned text.

♦ **Implement Hyperspace Structure**

Once the suitable data was obtained, the data needs to be organised according to the structure defined in the navigational and presentational design activities. Thus, the implementation of the hyperspace structure requires the following activities:

- implementation of templates,
- introduction of linking mechanisms,
- storage and organisation of the data,

- creation of mechanisms of automatic page generation.

♦ Implement User Interface

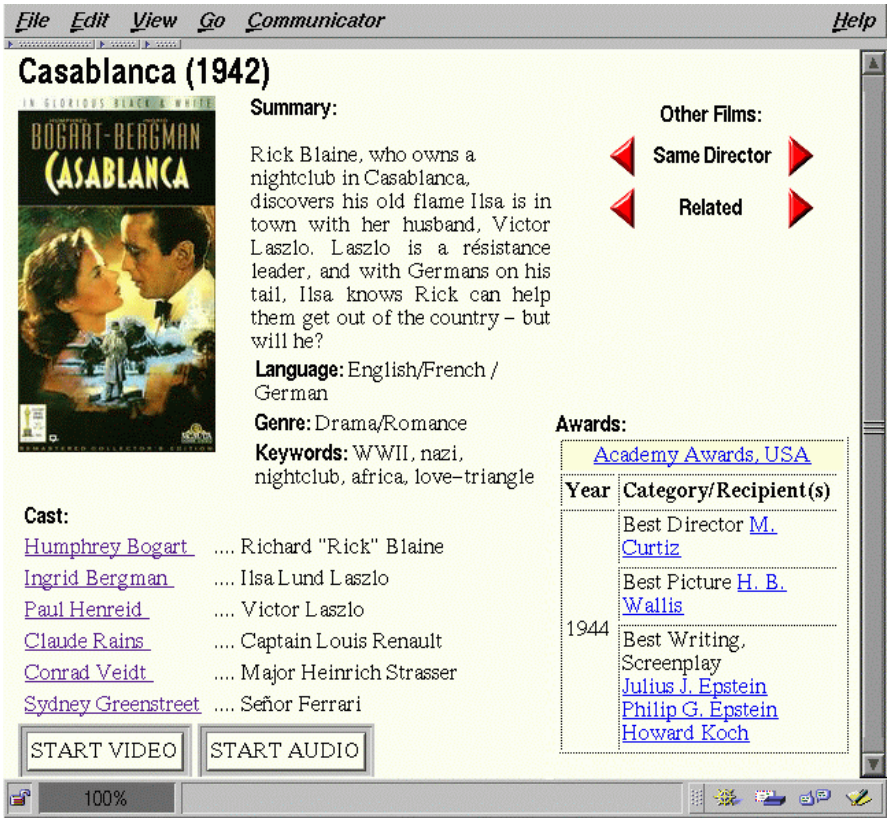
The implementation of the user interface is one of the greatest consumers of implementation time in the development process of hypermedia applications.

During this activity a user interface is implemented based on the presentational design. The logical design determines which user interface elements are needed while the physical design gives a first approach of the visual aspects and distribution of these user interface elements.

The activity *implement user interface* is performed by the multimedia designer, who produces a user interface prototype during the first iterations and a final version in the last one.

Example: The Film Assistant

The following screen shot shows a Web page of this application.



♦ Build Integration Plan

The objective of this activity is to plan the new components and/or functionality to be included in the previous system build. The build of the current iteration should not include too many new components or improvements. It should be easy to test and should allow for easy documentation of the changes. The following steps can be used as guide to construct an integration plan:

- decide which use case to include,
- identify the subsystem or the design classes that participate in the use case realisation,
- identify the implementation subsystem or components in the implementation model tracing to

the subsystem or design classes of the second step,

- plan to include the implementation of the subsystem or the components in the subsequent build.

♦ **Integrate System**

Following the plan elaborated during the activity "build integration plan" the components are included in the current system, compiled and linked. The new build is then ready for the testing process.

5 Conclusions and Future Work

The methodology for the development of hypermedia applications (UPHD) presented in this work is based on the Unified Process and uses UML modeling techniques. It covers the whole life-cycle of hypermedia applications including supporting workflows for project and quality management.

UPHD focuses on the requirements capture, analysis and design of hypermedia applications Maintenance is treated as an additional phase of the development process.. It proposes a UML extension for the design of the navigational and presentational models of hypermedia applications. This extension is defined according to the UML extensions mechanism. For the analysis and design of hypermedia applications a set of steps are given that support the construction of such applications based on the separation of content, structure and presentation. Activities, workers and artifacts are described separately following the schema of the Unified Process and the Rational Unified Process documentation.

Future work involves in first place a description of the supporting workflows, i.e. project management and quality management. This description will focus on the special project management aspects that are required for the development of hypermedia applications as well as on the quality control in the hypermedia engineering process. Quality management in UPHD emphasises the validation of the requirements captured and the systematic verification of the design results trying to introduce corrections in early stages. In addition, a more detailed description will be given for architectural and implementation aspects.

References

Balasubramanian V., Bieber M. and Isakowitz T. (1996). Systematic hypermedia design. *Technical report, CRIS Working Paper Series. Stern School of Business, New York University,* 1996.

Baumeister, H., Koch, N. and Mandel L. (1999). Towards a UML Extension for Hypermedia Design. *In Proceedings UML'99 – The Unified Modeling Language: Beyond the standard Conference*. Eds. France R. and Rumpe B. LNCS 1723. Springer.

Boehm B. (1988). A spiral model for software development and enhancement. *Computer*, May, 61-72.

Booch G., Rumbaugh J. and Jacobson I. (1999). *The Unified Modeling Language: A User Guide.* Addison Wesley.

Boyle T. (1997). *Design for Multimedia Learning.* Prentice Hall.

Conallen J. (1999). *Building Web Applications with UML.* Addison Wesley.

De Bra P. and Calvi L. (1998). AHA: A generic adaptive hypermedia system. *Proceeding of the 2nd Workshop on Adaptive Hypertext and Hypermedia, HYPERTEXT´98.*

Halasz F. and Schwartz M. (1994). The Dexter Hypertext Reference Model. *Communications of the ACM 37(2), Grønbæk K. and Trigg R. (Eds.),* 30-39.

Isakowitz T., Stohr E. and Balasubramanian P. (1995). A methodology for the design of structured hypermedia applications. *Communications of the ACM*, 8(38), 34-44.

Jacobson I., Booch G., and Rumbaugh J. (1999). *The Unified Software Development Process.* Addison Wesley.

Koch N. (1998). Towards a methodology for adaptive hypermedia systems development. *Proceedings of the 6th Workshop ABIS-98: Adaptivity and User Modelling in Interactive Software Systems,* Timm U. and Rössel M. (Eds.), FORWISS, 41-52.

Koch N. (1999). A Comparative Study of Methods for Hypermedia Development. Technical Report 9905, Institute of Computer Science, Ludwig-Maximilians-University Munich.

Koch N. and Mandel L. (1999) Using UML to design hypermedia applications. Technical Report 9901, Institute of Computer Science, Ludwig-Maximilians-University Munich.

Kruchten P. (1998). *The Rational Unified Process: An Introduction.* Addison Wesley.

Lowe D. and Hall W. (1999). *Hypermedia and the Web: An engineering approach.* John Wiley & Sons.

Nanard J. and Nanard M. (1995). Hypertext design environments and the hypertext design process. Communication of the ACM, August 1995, Vol 38 (8), 49-56.

Oestereich B. (1999). *Developing Software with UML: Object-Oriented Analysis and Design in Practice.* Addison-Wesley.

Olsina L. (1998). Building a Web-based information system applying the Hypermedia Flexible Process Modeling Strategy. *1st International Workshop on Hypermedia Development, Hypertext´98.*

Schneider G. and Winters J. (1998). *Applying Use Cases: A practical guide.* Addison Wesley.

Schwabe D., Rossi G. and Barbosa S. (1996). Systematic hypermedia design with OOHDM. In *Proceedings of the ACM International Conference on Hypertext, Hypertext´96.*