Rapid Development of Electronic Product Catalogues

Nora Koch
Ludwig-Maximilians-University of Munich
Institute for Computer Science
Oettingenstr. 67
80538 München, Germany
kochn@informatik.uni-muenchen.de

Bernd Gaede
FORWISS
University of Erlangen
Am Weichselgarten 7
91058 Erlangen, Germany
gaede@forwiss.uni-erlangen.de

Josef Schneeberger
Schema GmbH
Sulzbacherstr. 81
90489 Nürnberg
js@schema.de

Abstract

The production of electronic product catalogues (EPCs) is a creative process as well as a software design problem. This paper presents the results of the EPK-*fix* project[1]. It proposes a development model for EPCs, defines a high-level specification language for the description of catalogues and implements a set of integrated tools to support the production process of EPCs. The iterative software engineering process starts with the requirements analysis and goes on with the catalogue design, both steps are supported by special editors. An automatic catalogue generation based on the design specification is the key of the rapid development of EPCs. The process is completed with a testing phase for the static and dynamic features of catalogues.

Keywords
Electronic Product Catalogues, Development Process of Multimedia Applications, Mark-up Language.

1 Introduction

With the availability of low cost computers and high quality (graphical) user interfaces, electronic product catalogues (EPCs) become an increasingly important class of software systems. There exist many different kinds of EPCs: some present very vast numbers of products with many variants, others present only few products with complicated and detailed descriptions. While multimedia technology is improving, EPCs include more and more fancy features like videos, software animation, and sound. They also increment their functionality offering features like electronic commerce and online banking to the users.

An EPC is used as an alternative to a paper catalogue, it has to be produced rapidly with a limited budget. Nevertheless, there are usually high requirements concerning the appearance of an EPC which demands iterative design cycles and layout variants. Modern EPC development is an interdisciplinary endeavour, which involves marketing personnel, screen designer, and software engineers. Since the resulting piece of software partially presents the company -- called catalogue provider in the sequel -- there are high quality standards required in general. Nevertheless, an EPC is just a software system and its development has to take into account all the problems and activities present in software development [Gloor, 1997; Lennon, 1997].

---

[1] This work was supported by the German BMBF project EPK-*fix* under grant 01 IS 250

In the EPK-*fix* project[2] we have focused an all aspects of the development process of EPCs. It is characterised by *phases of a life cycle* and by *organisational aspects*. The *life cycle* starts with the analysis of the catalogue providers requirements, it continues with the catalogue design and ends with functional tests. The approach is supported by a collection of integrated tools, which have been designed for efficient specification, production, and validation of EPCs. All tools are based on the specification language EpkMl which is a high level HTML-like language that is particularly designed to support the description of EPCs. The *organisational aspects* characterise particular features of the EPC like page layout, multimedia components, catalogue and product structure, navigation, or added values.

In the following, we focus on the *development method* and on the *role of EpkMl* in the development process. Although the integrated tool suite of the assistance system is described elsewhere [Knapp *et al.*,1997], we give a brief overview here for the sake of completeness.

In the second section we present the development process for electronic product catalogues. The third section delineates the organisational aspects of EPCs and the section four gives an overview of the EPK-*fix* architecture. We describe the specification language in section five and the generation of an example catalogue based on its specification in section six. Related work is presented in the seventh section. Finally we give some conclusions and further steps in section eight.

2   The EPC Development Process

Electronic product catalogues are special information systems with definable application fields and well identifiable characteristics like important multimedia (especially visual) product presentations and navigation facilities. Instead of using a standard development model for general software systems, we defined a methodology adjusted to EPCs. This appropriate development model allowed to design efficient tools for the rapid prototyping and production of EPCs.

Important parts in the development model for EPCs, that have been adopted from models for expert systems [Jackson,1990,Bibel *et al.,*1989], authoring tools, graphical user-interfaces, and object-oriented software systems [Rumbaugh *et al.*, 1991] are: informal preanalysis, description through checklists, human-machine interaction, multimedia integration, formal description of catalogues, generation of prototypes, creation and management of reusable libraries of EPCs components, and quality tests for the resulting catalogues.

The development process that unifies these characteristics, requires an informal preanalysis followed by the phases*: requirements analysis, design, implementation, and test*. Combining an effective requirements analysis, a well-supported software-design (specification), automation of the error-prone implementation step, and an exhaustive, partially automated testing allows a quick (small number of iterations) and inexpensive prototype completion. The

---

[2] The partners of the EPK-*fix* project are: Bavarian Research Center for Knowledge-Based Systems (FORWISS) in Erlangen, Ludwig-Maximilians-University of Munich, Technical University of Dresden, Technical University of Darmstadt and mediatec GmbH in Nuremberg. The EPK-*fix* homepage is http://www.forwiss.uni-erlangen/fg-we/epkfix/.

resulting process is a kind of spiral model [Koch and Schneeberger, 1997; Boehm, 1988] that leads to the final EPC through successive revisions and refinements (see Fig. 1).

Each phase, described below, is supported by one of the following tools: Requirements analysis ASSIstant (RASSI), Specification ASSIstant (SASSI), Generation ASSIstant (GASSI), and Testing ASSIstant (TASSI). These tools are based on the specification language, called EpkMl, designed for the formal description of the static and dynamic aspects of a catalogue and for the definition of tool interfaces [Knapp *et al.*, 1997; Koch and Turk, 1997].
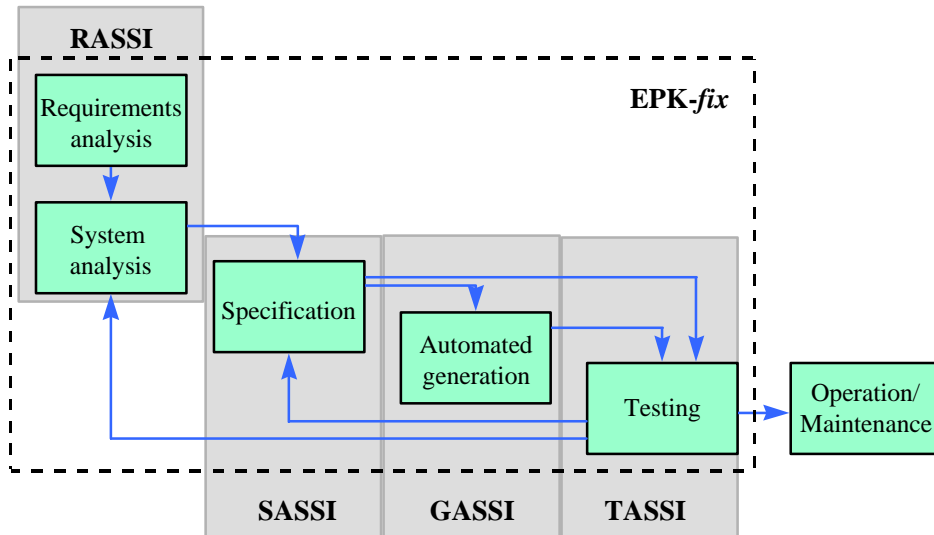

Figure 1: An overview of the EPC development model

Requirements Analysis

 EPK-*fix* provides sophisticated solutions for requirements, system, and prototype analysis during development of EPCs. The fundamental analysis activity is to carry out *structured interviews*. This interviewing process is divided into three major steps [Klausner *et al.,* 1994; Wieringa, 1996]: preparation, interview, and edition. During *preparation* a questionnaire is assembled considering all aspects of an EPC. The selection of the questions is guided by predefined generic *checklists*. The *interview* is a complete (audio-) recorded conversation between the EPC developer and the catalogue provider with any multimedia information like text, pictures, or electronic documents attached to the corresponding questions and answers of the interview. Finally, the *edition* constructs the resulting analysis documents from written (transcribed) notes, acoustic data, and multimedia documents. The software tool RASSI, supports the recording of information (text, sound, images, video) resulting from structured interviews during the requirements analysis stage.

Design

The informal catalogue description recorded in the analysis document provides the characteristics and details of the intended catalogue needed to design an EPC. The catalogue developer carries out his work with the help of *editors* for the catalogue structure and layout design, that automatically generate a formal specification of the catalogue. The design process of EPCs follows the same steps as the creation of other multimedia productions: there are

media-object generation (text, images, videos, audio, and animations), object embedding into pages, windows, or layout forms (templates), and incorporation of paths from one layout piece to another (navigation). The catalogue designer uses the specification assistant to supply the structure and the layout for the EPC.

The SASSI tool has been developed for the EPC design, which is based on the results of the Requirements Analysis Assistant and generates an EpkMl specification, that is the starting point for the next phase.

Integrated Software Generation

The result of the design phase is a specification of the EPC in EpkMl. The subsequent phase generates code for the final EPC which is either Java, HTML, or a paper catalogue in our approach. The advantage of code generation -- in contrast to hand-crafting software using some high level programming library for multimedia systems -- is obvious. EPCs can be produced more rapidly and debugging of the resulting code is omitted. Furthermore, a prototypical EPC version is generated, that includes additional interfaces to communicate with the other development tools (particularly Requirement Analysis and Testing Assistants). These interfaces provide *capture* and *replay* facilities that can, for instance, be used to present a catalogue prototype to the developer in exactly the state which is referred to by a given analysis document. The generation is based on a library of generic and reusable classes implementing all components of multimedial product presentations. When the EpkMl is changed or extended, or if implementation details have to be changed, the library has to be adapted.

Testing

Tests of complex multimedia software like EPCs require a complete testing instead of traditional sample testing. A large number of tests has to be carried out, therefore they have to be performed mostly automatically. The first step is to perform a rule-based *static test* for all elements and database objects of the catalogue specification in EpkMl. The rules are utilised to define implicit requirements and inconsistency patterns in specifications. Grouping elements, i.e. pages and windows, must be examined for group errors, for instance it could be verified, that the text colours are in adequate contrast to the background colours or that not too many different fonts are used on one page. In a second phase the actual catalogue is *dynamically tested*, therefore each branch of the EpkMl instance has to be tested and all media-objects be inspected manually. Each error found is recorded and an error protocol is generated.

The tool TASSI performs static tests on the catalogue description in EpkMl and a dynamic validation on the EPC generated by the generation tool, using test data especially prepared for that purpose.

3 Organisational Aspects of Electronic Catalogues

An EPC typically appears as a CD-ROM or as a web application and its minimal functionality is comparable to a paper catalogue enriched with multimedial objects (audio, video, and animations) and cross references [Koch and Mandel, 1997]. However, state of the art EPCs offer many more features which take advantage of the underlying computational power [Siegel, 1997]. We refer to these features as *services*. Services are, e.g.,

- search functions to find products or explanations,
- demos (animation or video) to illustrate the use of the EPC or some product,
- inquiries and orders via online connection or by fax,
- facilities to accumulate, compare, or combine products in one large order, or even
- games and animations to entertain and inform (″infotainment″) the customer.

Our analysis of EPCs focused on the organisational aspects and on the functional requirements of the software systems. The results were the basis for our definition of the specification language EpkMl (see Sec. 5) that reflects:

- *Static requirements* comprise all layout elements, i.e. windows, frames, buttons, check-boxes, pull-down menus, sliders, texts, paragraphs, headings, and listings.
- *Dynamic requirements* include every interactive situation, such as starting or stopping an animation or a video, navigating by clicking on buttons, searching, selecting help functions, ordering products, scrolling in a browser, etc.
- *Data requirements* are related to products, companies, and customers information, help text or help windows, navigation sequences, orders, and multi-lingual text for the pages.
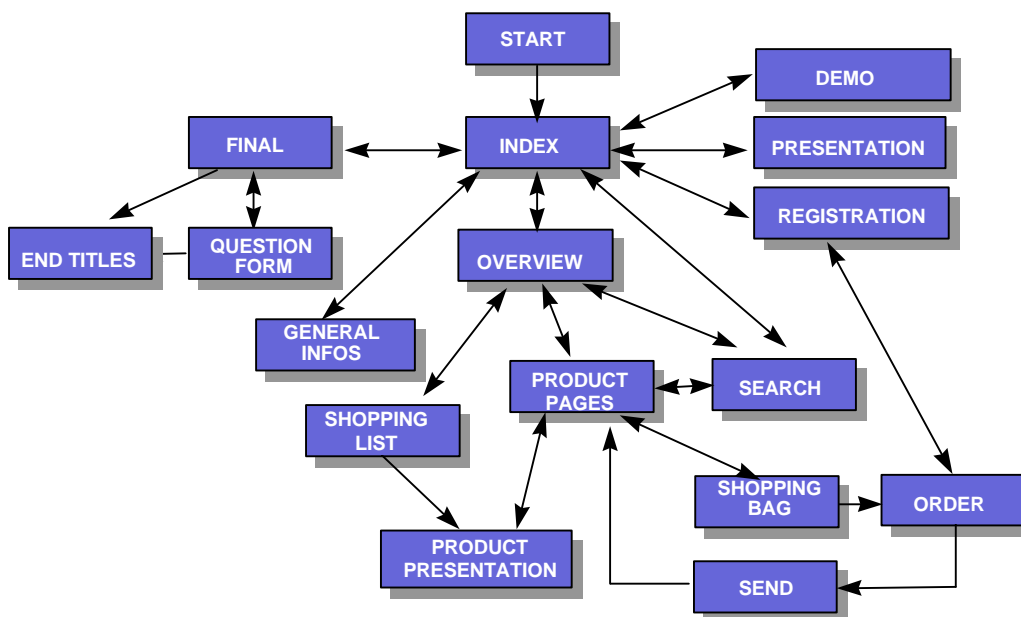
Figure 2: A typical catalogue structure

Similarly to the international standards ODA (Office Document Architecture) [ISO8613, 1988] and SGML [Goldfarb,1994], various aspects of EPCs can be distinguished and handled separately. Office documents have a (logical) *structure* and a presentation *layout*. In addition, an EPC includes *multimedia* elements, it makes use of an underlying *database*, and it offers *services* and *navigation* facilities. All aspects are simultaneously present in any state of the EPC at runtime. For example, a page of the EPC that presents product information being retrieved from the database. Layout elements like frames, buttons, and menus appear on the screen complemented by multimedia elements like sound or animation. Using buttons and

hyperlinks, it is possible to navigate to a help window or to an arbitrary (e. g. the next or previous) page in the catalogue [Schneiderman, 1998].

The aspects we considered in our design are: the *structure*, the *layout*, the *direction*, the *database*, and the *services*.

- The *structure* is the skeleton of the catalogue; it comprises a hierarchy or graph of themes (product families) and pages.
- The *layout* is the static description of pages, frames, windows, and their contents. It describes (sub-)sets of catalogue pages by abstraction from particular contents (see *templates* in Sec. 5).
- The *direction* describes the dynamic facilities for pages and themes that permit user interaction and navigation through the catalogue. It comprises the micro-direction for each page and the macro-direction for the connection between pages.
- The *database* component provides all the information about products and offers, such that it can easily be searched and maintained.
- The *services* add functionality in order to work efficiently with the EPC. For example, services include administration of orders (shopping bag feature), calculation of financial or configuration parameters, user registration, access to the help system, and online communications (e.g. ordering).

Working with EPCs can be divided into the phases installation, presentation, search, selection, and order. Depending on their relative importance [Knapp *et al.*, 1996], we distinguish between the catalogue types: presentation, search, and order catalogues. Fig. 2 illustrates a typical catalogue structure and the navigation through its units.

4   The Architecture of the EPK-*fix* System

The goal of the EPK-*fix* project, as already said, is the development of methods and a collection of integrated tools for efficient specification, production, and validation of EPCs. The EPK-*fix* system comprise the following five tools: the Requirements Analysis ASSIstant, the Specification ASSIstant, the Generation ASSIstant, the Testing ASSIstant and the Organisation ASSIstant (see [Koch and Turk, 1997]. The formal description language EpkMl is the base of almost all EPK-*fix* interfaces.

The EPK-*fix* architecture integrates the tools described below, their user interfaces and a repository for catalogue descriptions [Shaw-Garlan,1996]. These descriptions are HTML-documents for the requirement analysis and test protocols, EpkMl catalogue specifications, and automatically generated Java-code for the EPCs.

The methodology and specific tools support the complete life cycle of EPCs starting with the catalogue providers requirements, continuing with the catalogue design up to the functional tests. These tools must be easy to use, reduce the amount of EPC development time and permit a low-cost production of catalogues. These conditions are a prerequisite for the acceptance of the EPK-fix system especially in small and mid size enterprises (SMEs).

RASSI (Requirements Analysis Assistant) supports the informal recording of information (text, images, video) that result from the requirements analysis stage based on structured

interviews. This assistant contains the following submodules to manipulate and convert five types of basic objects, i.e. checklists, questionnaires, interviews, protocols, and documents:

- The *questionnaire and checklist editor* is used to create new checklists either from scratch or based on existing ones by reordering or deleting themes, adding subthemes, complementing with annotations, etc. Additionally, it supports assembling a questionnaire from a checklist by formulating questions and adding documents.
- The *interview assistant* performs a complete audio recording of the interview. Linking to multimedia material and the inclusion of typed notes during the interview is also possible. Segments of the recorded dialogue are associated to the themes of the questionnaire through mouse clicks.
- The *protocol editor* allows for reproduction of the whole audio sequence, adjusting reassignment of the audio links and an automatic reorganisation of the theme structure.
- The *presentation assistant* integrates multiple interviews generating an analysis document in HTML or RTF format.

SASSI (Specification Assistant) is responsible for the EPC design based on the results of the RASSI tool and the conversion into a catalogue specification in EpkMl. The editors that assist the catalogue developer in this step are:

- The *structure editor* is used by the catalogue designers to define the theme structure and navigation paths through the catalogue.
- The *layout editor* assists the user in the detailed visual specification of the presentation and the integration of special services.
- The *database editor* permits the access to a database. The queries are formulated using SQL-statements.

A three-window approach supports the developers in the design process. They produce the catalogue design in the central window working with one of the editors (structure, layout or database) based on the RASSI-documentation that is visible in the left window. Objects from the left and central window are automatically linked through object identifier. In the right window the developer observes the EpkMl description of the generated catalogue.

GASSI (Generation Assistant) makes use of the EPC specification generated by SASSI and translates the EpkMl description into Java implementing this way the electronic product catalogue. The generation of this formally specified multimedia system is done by analogy with compiler phases in the following steps:

- The *parsing* step comprises lexical and syntactical analysis of a given EpkMl specification. An EpkMl conform document defines a tree structure of EpkMl elements that can be parsed by a standard SGML-parser yielding text output that is more easy to process. GASSI calls the parser *nsgmls* and uses its output for further processing.
- An *Intermediate Representation* of syntactically correct specifications is used for semantic analysis and optimisation. The internal representation of a specification consists of specification objects arranged in a tree structure, yielding an extended

document object model. Each node of this tree structure contains an SGML element and allows the access to the attributes that had been set in the specification. The intermediate representation is also responsible for the replication of globally defined specification elements (e.g. *stylesheets*, that might define the size of other elements) to make them locally available where needed.

- The basis of the *Sourcecode Generation* is a class library realising common features of EPCs which are used for a specific implementation either by parameterised direct instantiation or by subclassing and instantiation of the new subclass. For these classes miscellaneous Java-syntax-conform expressions can be retrieved or composed, e.g. unique Java identifiers; constructor calls; variable declarations or templates for subclasses.

The generation of EPCs relies on a library of extensible generic classes (that are reusable, and reliable due to their automated testing by TASSI) providing EPC-specific components ranging from simple layout elements up to modules that perform *services* (e.g. online-help, ordering-facilities).

TASSI (Testing Assistant) realises static tests on the catalogue description in EpkMl and a dynamic validation on the EPC generated by GASSI, using for that purpose especially prepared test data. TASSI supports automatic and manual tests with a strong support for manual tests. However, structure-based testing of Java applets or databases that are treated as black-boxes is out of the systems scope.

The characteristics of TASSI are:

- fully graphical user interface,
- declarative specification of general and catalogue-specific automatic tests via rules,
- automatic execution of static tests of all media-objects,
- support of dynamic tests by a test agent (automatic navigation of the EPC to untested objects),
- automatic execution of static tests of dynamic objects,
- error classification via browser,
- context-sensitive specification and requirements presentation,
- capture and replay of manual test input,
- complete test state administration, and
- automatic test document generation.

OASSI (Organisation Assistant) is responsible for the version management and the communication of co-operating experts working in a typical distributed environment. RASSI supports the interviewing process with the customer while SASSI, GASSI and TASSI are used in the development area. Therefore, a central repository and a communication tool has been implemented that controls the assistants registration and disconnection. It is an Internet-based tool that provides information about the current state of the ongoing catalogue projects.

5 The Language EpkMl

The specification language EpkMl has been defined to enable declarative description of EPCs and as basis for the communication between the tools in the catalogue production process. Its design was guided by some basic considerations like easiness of learning and extensibility as well as more EPC-specific considerations like the integration of database features or navigational support. A central aspect of the specification language is the description of a catalogue structure that permits the definition of navigation contexts and an automatic generation of navigation path through these contexts.

Design Decisions

EPCs on CD-ROM and online are an alternative to traditional paper catalogues for product sale and service offer. The goal of the project was to allow specification of different types of catalogues in parallel. An HTML-like *declarative language* achieves this goal. Thus, the language EpkMl is defined as an instance of SGML using mark-up tags [Goldfarb,1994] and supports all components that were observed during the analysis of the organisational aspects of an EPC (see Sec. 3): *structure*, *layout*, *direction* (control constructions), *database*, and *services*. An important requirement is the simple handling of catalogue standard operations (services), like user registration, product search, order forms, shopping bag, table of contents, and question forms.
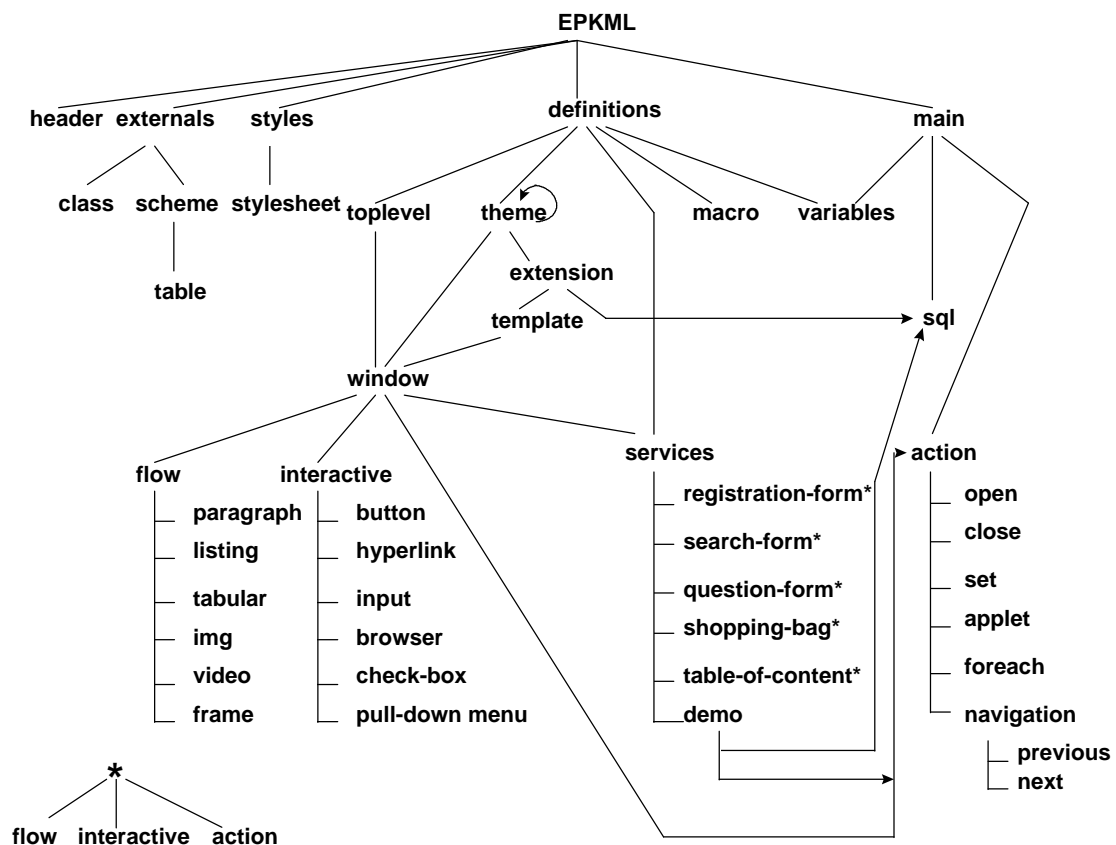

Figure 3: Structure of the EpkMl language

The most important characteristics of the language can be summarised as follows:

- *Hierarchical organisation of themes*. It means that products are organised in hierarchies, so-called themes. For each theme the developer can define the products belonging to the theme (sub-themes) and the presentations of these products.
- *Automatic navigation* through the theme structure. This structure is used to generate navigation facilities. Additionally, the user may be guided through the catalogue in a controlled fashion.
- *Windows-oriented layout* expanding the graphical and functional possibilities in comparison with frame-oriented layout.
- *Styles* for a simple way to define layout templates.
- *Multimedia features* like video, audio, and slide-show. Multimedia has been integrated in a transparent way such that the designer may concentrate on its declarative aspects.
- *SQL-statements*, that are integrated for access to relational databases.
- *Primitives for control flow*, that allows the user to navigate through the catalogue structure.
- *Connection to external languages* as in HTML via applets; and
- *Special services* present in most EPCs like searching, adding products to a shopping bag, or ordering products are included as built-in functions.

We schematise the structure of the EpkMl language with the graph shown in Fig. 3. For clarity only a part of the language elements are represented graphically.

Document Type Definition

EpkMl as an instance of SGML is defined in a *Document Type Definition* (DTD) and uses mark-up tags. Each block begins with `<name-of-the-tag>` and ends with `</name-of-the-tag>`. The closing tag is defined as optional for some tags; it is indicated with '- 0' in the DTD specification. As in the HTML Document Type definition for each element the contents is defined ('+' means at least once, '?' optional, '*' zero or more times). For more clarity, we concentrate ourselves in this paper to the keywords, i.e. the elements of EpkMl, only some attributes are mentioned. A small part of the DTD definitions of the EpkMl elements is shown below. It can be observed how the structure given in Fig. 3 is mapped into the specification language.

```
<!ELEMENT epkml - - (header, externals, styles, definitions, main) +(expand | variant)>

<!ENTITY  % oid "oid CDATA #REQUIRED">

<!ENTITY % attributes "name     %IDENT;
          style        %IDENTS;
          invisible (invisible) #IMPLIED
          layer        CDATA '#0'
          xpos         %DIMEN;
          .......
          botmrg      %DIMEN;          ">

<!ENTITY % properties "properties CDATA "
          status (opened | closed | suspended) opened
          attribs %IDENTS;
          elems %IDENTS; ">
```

```
<!ENTITY % flow "p | heading |
            listing | itemize | enumerate |
            tabular | img | video | slide-show |
            flowbox | frame">
<!ENTITY % interactive "button | next-button | previous-button |
                back-button | hyperlink | input | scribble |
                pop-up | browser | multiple-browser |
                radio-button | checkbox | pull-down |
                vertical-slider | horizontal-slider">
<!ENTITY % toplevel "window | page">
<!ENTITY % services "demo | registration-form | question-form |
            search-form | shopping-bag | shopping-list |
            table-of-contents">

<!ENTITY % open "open | redraw">
<!ENTITY % close "suspend | close ">
<!ENTITY % database "sql">

<!ELEMENT (%open;) - O (attribute | element)*>
<!ATTLIST (%open;)
 %oid;
 name            CDATA #REQUIRED>

<!ELEMENT (%database;) - O (#PCDATA)>
<!ATTLIST (%database;)
 %oid;
 result   %IDENT;           >

<!ENTITY % cntrlbtns "play-button | stop-button | pause-button |
                forward-button | rewind-button">

<!ELEMENT audio - O (%cntrlbtns;)*>
<!ATTLIST audio
 %oid;
 name            %IDENT;
 %properties;
 %audio-format;
 duration        %TIME;
 src             CDATA           #REQUIRED>

<!ELEMENT (%cntrlbtns;) - O (disabled?, clicked?, (%flow;)*,
                   on-click?)>
<!ATTLIST (%cntrlbtns;)
 %oid;
 %attributes;
 %properties;
 disabled        (disabled)      #IMPLIED>

<!ELEMENT click - O EMPTY>
<!ATTLIST click
 name            CDATA           #REQUIRED>

<!ELEMENT window - - (%flow; | %interactive; | dialog-window |
            %services; | %action;)*>
<!ATTLIST window
 %oid;
 %attributes;
```

```
    %properties;
    title          CDATA          %void;
    iconized       (iconized)     #IMPLIED
    background     CDATA          %def-background;>

<!ELEMENT theme - - ((page|window)*,(extension,exceptions*)*,theme*)>
<!ATTLIST theme
  %oid;
  name           CDATA          #REQUIRED>

<!ELEMENT extension - - (sql,template,empty?)>
<!ATTLIST extension
  %oid;
  result   %IDENT;>

<!ELEMENT template - - (page | window)>
<!ATTLIST template
  %oid;
  name           CDATA          #REQUIRED>

<!ELEMENT styles - O (stylesheet)*>

<!ELEMENT main - O (var | %action;)+>
<!ATTLIST main
  %oid;>
```

Language description

The five catalogue aspects considered in the design (structure, layout, direction, database, and services) are specified with a group of elements and their attributes in the EpkMl language. It follows a short description of some interesting elements.

- *Structure*
  Each `<theme>` is specified through an `<extension>` that includes an SQL-statement declaring the products that will be retrieved from the database and through a `<template>` describing the layout aspects of these products.
  The theme description may contain sub-themes. `<exceptions>` may be specified for products of the extension to be presented with their own `<template>`. Templates are predefined forms for structured data presentation. Their gaps can be filled "on the fly" with values obtained via SQL-statements. The results of a database query are held in variables, which names have to be surrounded by $...$ and which values may be assigned by the `<set>` construct.
  Theme hierarchies build-up tree structures, in which navigation takes place by special commands `<next>`, `<previous>`, `<up>`, `<down>`, and `<back>`. These instructions branch to the next or previous theme in a given hierarchy, to the one below or above, or back in the history of visited themes, respectively.

- *Layout*
  The visual (layout) features of EpkMl are a superset of those of HTML. Different text fonts and styles are provided as `<p>` (paragraphs), `<image>`, `<frame>` and other HTML-like elements. Interactive elements such as `<browser>`, `<checkbox>`, `<pulldown-`

menu>, <button>, <input> among others have also been included. We add <window> (thus making EpkMl window-oriented instead of screen-oriented) and <flowbox> for images inside texts. Multimedia is integrated by adding the time-dependent elements <video>, <slide-show>, and <audio>.

All these elements may be customised in advance defining <stylesheet>s, which set values for certain attributes. These values can be overloaded by individual settings in the element, that includes the defined <style>.

The HTML set of interactive elements is extended and these elements are provided with specifiable methods, e.g. <on-click> for <button> that are invoked if an interaction takes place. For the most common interaction facilities, such as navigation through the catalogue structure, there are precustomised elements with standard behaviour like <previous-button>, <next-button>, and <back-button>. This behaviour can be changed or extended in the specification.

- *Direction*
  Navigation through the catalogue is specifiable with a set of user controlled tags. Conditional branching may be achieved with the <empty> and <non-empty> tags on the basis of the result of a database query.
  For unconditional branching there are several possibilities: navigation through the theme structure, opening and closing of elements, and use of interactive elements. In the first case, there may be a change between themes by the use of <next>, <previous>, etc., already mentioned. Second, a layout element or a theme may be called directly via an <open> statement (provided with a name) with the effect of element visualisation and execution of its statements. Conversely, elements may be closed with <close> with no effect on the control flow.
  Last but not least, the control flow will be changed if interaction with the catalogue takes place, e.g. by clicking a button (<on-click> tag in a <button> or selecting an option <on-selected> in a <browser>).

- *Database*
  Access to databases is specified with the <sql> tag. Statements under the scope of this tag must be written in standard SQL (see [Melton and Simon,1993]. The result of an SQL-statement can be casted to options of a browser by using the <make-options> tag or to items of a list by using the <make-items> tag.

- *Services*
  Services provide standard functionality for a catalogue. To serve to that purpose EpkMl includes the following tags:

  - <table-of-contents>: allows the definition of an introduction window or page as an index of the different alternatives of the EPC (company's presentation, demo, tutorial, different views of the product database, ordering).
  - <registration-form>: permits the personalisation of the catalogue. Data entered in this form will appear in the order form.
  - <search-form>: with this tag it is possible to define which kind of search will be done onto the database every time the end-user fills in the corresponding form with adequate keywords.

- `<shopping-bag>`: is a template to maintain a list of products to buy. Update functions are supported to modify this list.
- `<shopping-list>`: serves to administrate the list of products selected all together at the beginning and to be visited during navigation.
- `<order>`: defined to send a buy order to the provider. This function has different semantics depending on the hardware configuration. An order can be sent by Internet, by E-mail, by dialing a telephone number by modem, by fax, or can be printed. When the catalogue is installed the semantics of this tag is settled.
- `<question-form>`: to be filled in by the end-user to return feedback to the catalogue provider about the success of the catalogue or to criticise it.

A more detailed description of the language is given in [Knapp *et al.*,1996] and a formal structural operational semantics is presented in [Knapp and Kosiuczenko,1997].

6   Generating a Catalogue from an EpkMl Specification

We exemplify the catalogue specification and the generation process with an electronic product catalogue developed for *Sanacorp Pharmahandel AG*, a pharmaceutical company. This company provided us with the product database. We produced a catalogue with EPK-*fix* delivered on a CD-ROM as a prototype used to design future EPCs. The EpkMl specification of the required catalogue was defined based on the interviews realised during the requirements analysis phase. This design step was performed using the structure and layout editors of the Specification Assistant. In the following the EPKML catalogue specification of one page (Fig. 4) is described and the automatically generated Java-code is presented. The numbers in brackets reference lines of code.

The catalogue specification

The specification begins with the template declaration for the layout: The element *stylesheet* (1) assigns a name of this pattern for the catalogue pages. This template defines a white background colour (2) for pages within which three elements can be seen: an image (3), a navigation tree (4-6) representing the catalogue structure, and a button (7-15) for finishing the navigation process. The button layout is described through an image (8-9), his functionality through the included element *on-click* (13-16), i.e. closing the active (12) page, opening another one (13).

```
1  <stylesheet oid="stylesheet21" name="Sanacorp1">
2    <page oid="Page7" width=800 height=600 background="white">
3    <img oid="Image1763" xpos=16 ypos=16 src="epk/img/p-vert.gif">
4    <applet oid="applet1764" name="EpkTree" xpos=640 ypos=16
5     width=144 height=448 function="Navigation">
6    </applet>
7    <button oid="Button19" xpos=256 ypos=528 width=128 height=48>
8      <img oid="Image20" xpos=640 ypos=528 width=128 height=48
9      src="epk/img/beenden.gif">                              // finish
10     <on-click>
12       <close oid="close99011" name="Page7">
13       <open oid="open99012" name="Theme27">
14     </on-click>
15     </button>
16   </page>
17  </sylesheet>
18  [...]
```

The specification continues with a *theme* (19) description. A theme can define just one catalogue page or a set of pages through the reference of a *template* in the *extension* sub-element. The example page references the above defined stylesheet (21) for the basic layout. Additionally, it is enriched with *frame* (24), a table (25) and a text (30-33) as well as other buttons. The table shows how the result of an *sql*-query is used from the EpkMl-element *make-table* (26) and *table-column* (27-28). The specification of the buttons beginning in (35) is done in analogy to the button in the *stylesheet* element.

```
19  <theme oid="Theme10" name="Allg">
20    [...]
21    <page oid="Page29" style="Sanacorp1">
22      <sql oid="SQL2469" result="ALLG">
23       SELECT * FROM PCLAVBAY WHERE SLAV = '1' </sql>
24      <frame oid="P799" xpos=144 ypos=16 width=480 height=504>
25       <table oid="Table8 xpos=144 ypos=16 width=480 height=480>
26         <make-table from=ALLG>
27          <table-column name="Einheit"> $ALLG.EIN$              // unit
28          [...]
29       </table>
30       <p oid="P179" xpos=144 ypos=500 width=480 height=20
31        font="helvetica" fontsize=14 fontcolor="black">
32         Allgemeiner Praxis- und Sprechstundenbedarf
33       </p>
```

```
34     </frame>
35     <button oid="Button459" xpos=128 ypos=528 width=128 height=48>
36     <img oid="Image2076" xpos=128 ypos=528 width=128 height=48
37      src="epk/img/weiter.gif">                                    // continue
38       <on-click>
39         <next oid="next99012">
40       </on-click>
41     </button>
42     [...]
43   </page>
44   [...]
45 </theme>
```

The generated source code

The code generation process generates Java program code (a new class "EpkPageSanacorp1") from the specification. The new class implements the desired layout and behaviour based on the stylesheet using the standard components EpkPage, EpkButton and EpkImage. This class is then used to build subclasses for the catalogue pages that are derived from the class.

In our example the specified page is generated as a subclass of the previously generated standard page class (15). Now the specific contents is considered in addition to the already defined elements. Therefore, some variables are declared (17-25). The body of the generated method *init* (36-81) includes the initialisation of these elements based on the call of the parametrised constructors (41, 47, 58, 65). Other statements assign the unique identifiers needed for extended development support and determine the position of the elements. At runtime the concrete catalogue page is generated as an instance of this class incorporating the concrete data of the product database. The code that is responsible for this instantiation is located in the main class of the application.

```
1 // Classfile for EpkPageExt39
2 // The following code was generated by GASSI Version 1.0
3 // Tue Oct 14 11:27:58 GMT+00:00 1997
4 //--------------------------------------------------
5
6 /*** ## Package declaration ## ***/
9  package sanaEpk;
8
9  /*** ## The imports ## ***/
10 import java.awt.*;
11 import java.util.*;
12 import epk.*;
13
14 /*** ## Class declaration ## ***/
15 public class EpkPageExt39 extends EpkPageSanacorp1 {
16
17 /*** ## Variable declarations ## ***/
18 protected Epkcontainer frameFrame799;
19 protected EpkTable table8;
20 protected EpkTTA ttaP799;
21 protected EpkButton buttonButton459;22  protected EpkButton buttonButton460;
23 protected EpkButton buttonButton461;
24 protected EpkButton buttonButton462;
25 protected EpkButton next_buttonButton2072;
26
```

```
27  /*** ## The Constructor ## ***/
28  /** GASSI generated method. **/
29  public EpkPageExt39(Frame dummy) {
30
31    super(dummy);
32
33  }
34
35  /*** ## The methods ## ***/
36  /** GASSI generated method. **/
37  public void init() {
38
39    super.init();
40
41    frameFrame799 = new Epkcontainer();
42    frameFrame799.setOId("Frame799");
43    frameFrame799.setBounds(144, 16, 480, 504);
44    Epk.objTable().put("Frame799", frameFrame799);
45    this.add((ComponentIF)frameFrame799);
46
47    table8 = new EpkTable("SELECT * FROM PCLAVBAY WHERE SLAV='1'");
48    table8.setOId("Table8");
49    table8.setBounds(144, 16, 480, 480);
50    table8.setColumn("EIN", "Einheit");
51    table8.setColumn("NAM", "---- Artikelbezeichnung ----");
52    table8.setColumn("INFO", "Info?");
53    table8.setColumn("LAVK", "AVK*");
54    table8.setColumn("KZZUZAH", "Erstattung");
55    Epk.objTable().put("Table8", table8);
56    frameFrame799.add((ComponentIF)table8);
57
58    ttaP799 = new EpkTTA("Allgemeiner Praxis- und Sprechstundenbedarf");
59    ttaP799.setOId("P799");
60    ttaP799.setBounds(144, 500, 480, 20);
61    Epk.objTable().put("P799", ttaP799);
62    ttaP799.setFont(new Font("Helvetica", Font.PLAIN, 14));
63    frameFrame799.add((ComponentIF)ttaP799);
64
65    buttonButton459 = new EpkButton("epk/img/weiter.gif", "epk/img/weiter.gif",
66                      "epk/img/weiter.gif", Navigation.NEXT);
67    buttonButton459.setOId("Button459");
68    buttonButton459.setBounds(128, 528, 128, 48);
69    Epk.objTable().put("Button459", buttonButton459);
70    this.add((ComponentIF)buttonButton459);
71
72    buttonButton460 = new EpkButton("epk/img/hilfe.gif", "epk/img/hilfe.gif",
73                      "epk/img/hilfe.gif", Navigation.COMP);
74    buttonButton460.setOId("Button460");
75    buttonButton460.setBounds(256, 528, 128, 48);
76    Epk.objTable().put("Button460", buttonButton460);
77    buttonButton460.setOIDsForActions(" Page29 Theme27 ");
78    buttonButton460.setActionIDs(" 2009 2008 ");
79    this.add((ComponentIF)buttonButton460);
80  [...]
81  }
82 }
```

The source code of all the classes generated for an application is compiled and transformed into an executable program. The user of the GASSI generation tool can decide, if he wants to include the new classes in the catalogue class library. Therefore unique names to identify them have to be chosen. Reuse of these classes is then possible.

7   Related Work

Approaches related to our work come from four areas: First of all, numerous approaches focus on development of multimedia documents in general. There exist process models as well as development tools which support the production, publication, and management of hypermedia documents. The generality of these approaches is their drawback to our scenario. They do not provide any help for the particular task of electronic product catalogue development. Second, there are tools and approaches which focus on the particular aspects of commerce using modern "electronic" media. These approaches often lack the ability to create a vast range of variations of catalogues, which is desirable since customers usually like to give their catalogues a characteristic appearance. Third, framework-based software engineering is also apt to improve development of EPCs. The last related field of activity is the use of XML (eXtensible Markup Language) with respect to extended WWW presentation possibilities.

Process models and development tools for electronic catalogues are aspects of particular interest, both are likewise important. We need a wide flexibility to model catalogues in all variants of hypermedia, as well as appropriate support for frequent commercial features.

Well-known commercial authoring tools like Macromedia Director™ and Toolbook™ support the design and implementation of catalogues, but do not cover the whole life cycle. There are a lot of other different approaches regarding authoring environments. None of them covers requirements analysis assistant or extensive testing integrated with rapid prototyping through automatic code generation. We now briefly describe two interesting recent approaches with different focus: SchemaText™ System and OOHDM-Web.

The SchemaText System is an object-oriented WWW authoring environment to construct and maintain large and complex electronic documents [Schema, 1998]. It supports the authoring process through initial prototyping, authoring-in-the-large (large-scale design), authoring-in-the-small (produce text, embed graphics) as well as multi-platform production (WWW, MS-Windows Help, SGML). It uses Scheme (IEEE Standard 1178-1990) as its scripting language, a dialect of Lisp. The schema design can be done either in a top-down or in a bottom-up fashion. In spite of lots of elaborate features, the system doesn't generate programs and functionality beyond navigation has to be implemented by hand.

EPK-Editor [Rosewitz and Timm, 1998] is an example for a specific e-commerce authoring tool. It makes the production of EPCs with standard services like search and ordering facilities easier, faster and cheaper. Disadvantages are the missing portability of resulting EPCs (MS Windows™ only) and minor design features due to layout restrictions.

OOHDM-Web is an environment that allows prototyping of web applications designed with OOHDM [Schwabe and Pontes, 1998]. Therefore it maps the navigation and interface constructs onto a library of functions in the CGI-Lua environment. OOHDM-Web supports the dynamic generation of pages based on CGI scripts, predefined-templates, and content from a

database (DB). The designer then builds page templates that mix HTML tags with special commands that are interpreted by CGI-Lua scripting environment. These special commands build a bridge to the database. Unfortunately, OOHDM schemata first have to be transformed into tables of a relational database and the dependence of the entire approach on CGI and a database system restrict conceivable extensions.

Framework-based software development has nowadays become an important strategic issue, since software vendors "have indicated that as much as eighty percent of their development cost is spent writing and supporting the basic, non-competitive functions that are essentially the same for any application solution offered in a specific domain. (...) IBM's San Francisco project addresses these problems by providing application developers with a base set of object oriented infrastructure and application logic which can be expanded and enhanced by each developer in the areas where they chose to provide competitive differentiation" [IBM, 1998]. EPCs with tight integration into enterprise IT applications could be built onto this kind of applications by means of these extensions.

The generation of code for arbitrary applications is usually based on frameworks, too. The most general approach known to the authors is developed in the Jakarta project and described in [Batory *et al.*, 1998]. It deals with the creation of largely domain-independent programming infrastructure (e.g., languages for component specification, languages for component composition, etc.). The Jakarta Tool Suite (JTS) is designed for constructing software component technologies and their generators. In contrast to general application development, visual design of graphical user interfaces is supported by ubiquitous IDEs (Integrated Development Environments) but these offer restricted multimedia and no electronic commerce facilities at all.

XML is used to specify application-specific structured description languages that can be used to support production and management of hypermedia documents. Its basic renunciation of considering layout aspects is bypassed via XSL, the eXtensible Style Language. An introduction to concepts of XML and more detailed information about it can be found in [Megginson, 1998]. However, XML is no tool with application functionality but only provides standards. Compliance with these makes it possible to use existing or emerging tools to produce and view documents.

8   Conclusions and Further Steps

A study of the current state-of-art of electronic product catalogues [Knapp *et al.*,1997] on the market showed the need for a comfortable specification language for EPCs and easy-to-use tools. Through the features and services observed and tested, we identified the components and characteristics of the language EpkMl and the features of each assistant. The catalogue developer is assisted by special editors and a presentation assistant from the beginning of the first *interview*. During the catalogue *design* and *generation* phase the assistance is realised with other editors and class libraries. *Tests* accompany the entire development process including the final EPC.

Based on the methodology we developed, four assisting tools (RASSI, SASSI, GASSI, and TASSI) were implemented for the EPC production. The first experiences with small example applications proved the practical advantage of our approach and systems. We are currently

starting to produce EPCs for more complex domains. Experiences with these applications will lead to refinements and improvement of our specification language and the tools.

In our approach, an EPC is the result of cooperating experts working at various places on different aspects of the catalogue. In order to keep track of the correct versions of the documents and programs produced so far, *version management* is required. The WWW based project server OASSI has been implemented, that allows to define and manage users and projects with appropriate access capabilities. Furthermore, the server manages revisions of documents and identifies official releases of software modules.

Further *user modeling* aspects will be incorporated into the development process of EPCs, thereby producing adaptive catalogues. For a first adaptive prototype the users will supply their preferences, goals, interests, and tasks filling in the registration form. In a second step this information will be obtained from the knowledge acquisition component, which inferences from the user's behaviour. The user model will be instantiated from stereotypes stored in the knowledge base.

References

[Bibel *et al.*, 1989]   W. Bibel, J. Schneeberger and E. Elver. *The Representation of Knowledge.* In H. Adelij, editor, Knowledge Engineering. Mc. Graw Hill, New York, chapter I, 1989.

[Batory *et al.*, 1998]   D. Batory, B.e Lofaso, and Y. Smaragdakis. *JTS: A Tool Suite for Building GenVoca Generators.* Accepted for publication in the 5th International Conference on Software Reuse, Victoria, Canada, 1998.

[Boehm, 1988]   B.W. Boehm. *A Spiral Model of Software Development and Enhancement.* IEEE Computer 21(5):61-72, 1988.

[Gloor, 1997]   P. Gloor. *Elements of Hypermedia Design.* Birkhäuser Verlag, 1997.

[Goldfarb, 1994]   C. Goldfarb. *The SGML Handbook.* Clarendon Press, Oxford, 1994.

[IBM98]   IBM: International Business Machines Corporation: *San Francisco Project Technical Summary*. http://www.ibm.com/Java/Sanfrancisco/prd_summary.html (investigated October '98), 1997.

[ISO8613, 1988]   8613, ISO 1988. Information Processing - Text and Office Systems - Office Document Architecture (ODA) and Interchange Format. Vol i-III, parts 1-8.

[Jackson, 1990]   P. Jackson, *Introduction to Expert Systems.* Addison Wesley, Reading, 1990.

[Klausner *et al.,* 1994]   J. Klausner, G. Kraetzschmar, J. Schneeberger and H. Stoyan. *The Knowledge Mining Center.* In L. Steels, G. Schreiber and W. van der Velde, editors. Position paper of the 8th European Knowledge Aquisition Workshop EKAW´94, Technical Report, Hoegaarden, Vrije Universiteit Brussel, 1994.

[Knapp and Kosiuczenko,1997]   A. Knapp and P. Kosiuczenko. *Developing Formal semantics of EpkMl.* Technical Report 9704, Ludwig-Maximilians-Universität München, 1997.

[Knapp *et al.*,1996]   A. Knapp, N. Koch and L. Mandel. *The Language EpkMl.* Technical Report 9605, Ludwig-Maximilians-Universität München, 1996.

[Knapp *et al.*,1997]	A. Knapp, N. Koch, M. Wirsing, J. Duckeck, R. Lutze, H. Fritzsche, D. Timm, P. Closhen, M. Frisch, H.-J. Hoffmann, B. Gaede, J. scneeberger, H. Stoyan and A. Turk. *EPK-fix: Methods and Tools for Engineering Electronic Product Catalogues.* In R. Steinmetz and L. Wolf, editors, Interactive Distributed Multimedia Systems and Telecommunication services, LNCS 1309, Springer Verlag, 1997.
[Koch and Mandel, 1997]	N. Koch and L. Mandel. *State of the Art and Classification of Electronic Product Catalogues.* International Journal of Electronic Markets, University of St. Gallen, Vol. 7(3), 28-31, 1997.
[Koch and Turk, 1997]	N. Koch, and A. Turk, *Towards a Methodical Development of Electronic Catalogues,* International Journal of Electronic Markets, University of St. Gallen, Vol. 7(3), 16-21, 1997.
[Koch and Schneeberger, 1997]	N. Koch, and J. Schneeberger, *Integrated Assistance for the Development of Electronic Product Catalogues,* Proceedings of Symposium of Software Technology (SoST´97), 101-112, 1997.
[Lennon,1997]	J. Lennon, *Hypermedia Systems and Applications*. Springer Verlag, 1997.
[Megginson, 1998]	D. Megginson. *Structuring XML Documents*. In Charles F. Goldfarb Series on Open Information Management, Upper Saddle River, NJ, Prentice Hall, 1998.
[Meltin and Simon, 1993]	J. Melton and A. Simon. *Understanding the new SQL*. Morgan Kaufmann, 1993.
[Rosewitz and Timm, 1998]	M. Rosewitz and U. Timm. *Editor für Produktberatung*. WIRTSCHAFTSINFORMATIK 40, 1/1998.
[Raumbaugh *et al,* 1991]	J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen. *Object-oriented Modelling and Design*. Prentice Hall, 1991.
[Shaw and Garlan, 1996]	M. Shaw and D. Garlan. *Software Architecture*. Prentice Hall, 1996.
[Shema, 1998]	http://www.schema.de
[Schneiderman, 1998]	B. Schneiderman. *Designing the User Interface*. Addison Wesley, 1998.
[Shwabe and Pontes, 1998]	D. Schwabe and R. Almeida Pontes. OOHDM-Web: Rapid Prototyping of Hypermedia Applications in the WWW, Technical Report PUC-Rio Inf MCC 08/98, 1998.
[Wieringa,1996]	R. Wieringa. R*equirements Engineering*. John Wiley & Sons, 1996.