Integrated Assistance for the Development of Electronic Product Catalogues*

Nora Koch

Ludwig–Maximilians–Universität
Institut für Informatik
Oettingenstr. 67
80538 München, Germany

kochn@informatik.uni-muenchen.de
Tel: + 49 89 2178 2177
Fax: + 49 89 2178 2175

Josef Schneeberger

FORWISS Erlangen
Am Weichselgarten 7
91058 Erlangen, Germany

jws@forwiss.uni-erlangen.de
Tel: + 49 9131 69 11 93
Fax: + 49 9131 69 11 85

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, which propose a development model for EPCs and define a high-level specification language for the description of catalogues. Based on the model and the language a set of integrated tools are implemented to support the production process of EPCs, starting with the requirements analysis, going on with the catalogue design and generation, up to the functional testing.

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

*This work was supported by the BMBF project EPK-fix under grant 01 IS 520.

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 exists many different kinds of EPCs: some present very large amounts of products with many variants, others only few products with complicated and detailed descriptions, or they may include fancy features like videos, software animation, and sound.
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. Moreover, an EPC is a software system and its development has to take into account all the problems and activities present in software development.
In this paper we present a systematic approach for developing EPCs. It is characterized by *phases of a life cycle* and by *organizational 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 (RASSI, SASSI, GASSI, TASSI), 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 *organizational aspects* characterize particular features of the EPC like page layout, multimedia components, catalogue and product structure, navigation, or added values. The EPC process model, the specification language and the tools have been developed within the scope of the EPK-fix project [1].
In the second section we present the development process. The third section delineates the organizational aspects of EPCs while the specification language is described in the fourth section. Finally we give some conclusions and further steps in section five.

2 The EPC Development Process

Electronic product catalogues are special information systems with definable applications fields and good identifiable characteristics like important multimedia (especially visual) product presentations and navigation facilities. Instead of using a standard development model for general

[1] 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

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 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 resulting process is a kind of spiral model [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 four 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].

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; Turk and Stoyan, 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 for the design of an EPC. The *cat-*

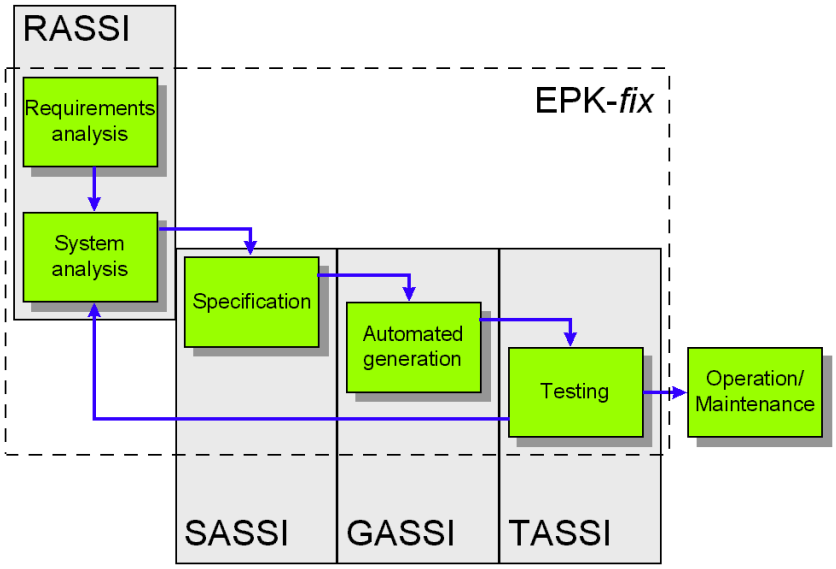

Figure 1: An overview of the EPC development model

alogue developer carries out his work with the help of *editors* which generate automatically a formal specification of the catalogue.
The design process of EPC s 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 design assistant to supply the structure and the layout for the EPC. The SASSI tool was developed for the EPC design, which is based on the results of the RASSI tool and generates an EPKML specification, that is the start 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 handcrafting software using some high level programming library for multimedia systems – is obvious. EPCs can be produced more rapidly and the debugging of the resulting code is omitted. Furthermore, a prototypical EPC version is generated, which includes additional interfaces to communicate with the other development tools (particularly RASSI and TASSI). These interfaces provide *capture* and *replay* facilities that can, for instance, be used to present a catalogue prototype to the developer in exactly that 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. The test process requires 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 utilized 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 a 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 (GASSI), using test data especially prepared for that purpose.

3 Organizational 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, 1996]. However, state of the art EPCs offer many more features which take advantage of the underlying computational power. 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 organizational 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. 4):

- *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.

Similarly to the international standards ODA (Office Document Architecture) [8613, 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 particular page of the EPC presents product information which was retrieved from the database. Layout elements as frames, buttons, and menus appear on the screen complemented by multimedia elements as sound or animation. Using buttons and hyperlinks, it is possible to navigate to a help window or to the next (or previous) page in the catalogue (structure).
The aspects we considered in our design are: the *structure*, the *layout*, the *direction*, the *database*, and the *services*. Fig. 2 illustrates how these components are assembled to the static and dynamic aspects of a catalogue: presentation and navigation.

- The *structure* is the skeleton of the catalogue; it comprises a graph or hierarchy of themes (product families) and pages.

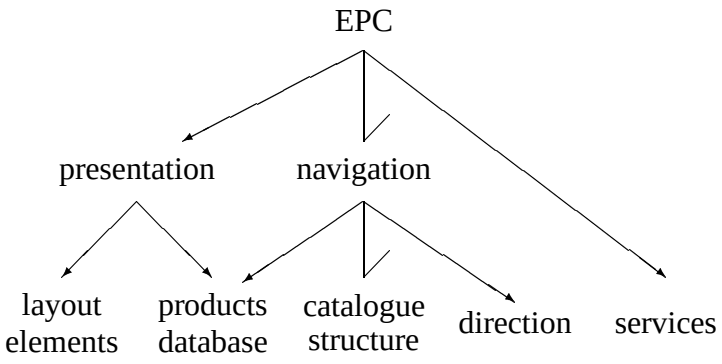

Figure 2: EPC aspects

- The *layout* is the static description of pages and their contents. It describes (sub-)sets of catalogue pages using abstracts from a particular contents (see the templates in Sec. 4.
- 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 are the administration of orders (shopping bag feature), the 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 [Koch and Mandel, 1996], we distinguish between the following catalogue types: presentation, search, and order catalogues.

4 The Language EpkMl

The specification language EPKML was defined for allowing a 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.
EPCs on CD-ROM and online are an alternative to traditional paper catalogues for product sale and service offer. Since these catalogues may coexist, we developed an HTML-like *declarative language* that permits an easy specification of all kinds of catalogues. 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 organizational 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.
The most important characteristics of the language are:

- *hierarchical organization of themes*, that means themes may include sub-themes with
- *automatic navigation* through the theme structure;

- *windows-oriented layout* expanding the graphical and functional possibilities in comparison with frame-oriented layout;
- *styles* for a simply way to define layout templates;
- *multimedia features* like video, audio, and slide-show;
- *embedded 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, including in a shopping bag, or ordering products.

We restrict ourselves in this paper to only a few aspects of the syntax and pragmatics of each component of the language EPKML (referring to the example below). A detailed description of the language [Knapp *et al.*, 1996] and a formal structural operational semantics is presented in [**?**].

```
01 <theme name=products>
02   <extension result=on-sale>
03     <sql>
04       select name,id,price,image,audio
05       from database
06       where database.price<100
07     </sql>
08     <template name=group-template>
09       <window name=group-window style=on-sale-products>
10         <button name=shopping disabled>
11           <on-click>
12             <open name="shopping-bag">
13           </on-click>
14         </button>
15         <frame name=product-description>
16           <img src='$which-picture$'>
17           <audio src='$which-audio$'>
18         </frame>
19         <browser>
20           <make-options from=on-sale>
21             <p> '$on-sale.name$'
22             <on-selected>
23               <set name=shopping.disabled value=''>
24               <redraw name=shopping>
25               <set name=which-picture value='$on-sale.image$'>
26               <set name=which-audio value='$on-sale.audio$'>
27               <open name=product-description>
28             </on-selected>
29             <on-deselected>
30               <set name=shopping.disabled value='disabled'>
31               <redraw name=shopping>
32               <close name=product-description>
```

```
33              </on-deselected>
34            </make-options>
35          </browser>
36          <back-button>
37        </window>
38      </template>
39    </extension>
40 </theme>
```

Structure. Each <theme> (01) is specified through an <extension> (02) that includes an SQL -statement (04-06) declaring the products it covers and through a <template> (08) describing the layout aspects. The theme description may content sub-themes and <exceptions> to specify products of the extension to be treated specially 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 $...$ (16,17,21) and which values may be assigned by the <set> (25,26,30) construct.
Through the theme hierarchies tree structures are build-up, 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, (<p>) (paragraphs) (21), <image> (16), <frame> (15), etc. as well as interactive elements such as <browser> (19), <checkbox>, <pulldown-menu>, <button> (10), <input> among others. We add <window> (09) (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> (17).
All these elements may be customized in advance defining <stylesheet>s, which associate values for certain attributes. This values can be overloaded by individual settings in the element, which includes the defined <style> (09). HTML set of interactive elements is extended and these elements are provided with specifiable methods, e.g. <on-click> (11) for <button> (10) that are invoked if an interaction takes place. For the most common interaction facilities, such as navigation through the catalogue structure, there are precustomized elements with standard behaviour like <previous-button>, <next-button>, and <back-button> (36). This behaviour may 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, open and close 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>` (12,27) statement (provided with a name) with the effect of element visualization and execution of its statements. Conversely, elements may be closed with `<close>` (32), but that has no effect on the control flow. Last but not least, the control flow will be changed if interaction with the catalogue takes place, by e.g. clicking a button (`<on-click>` (11) tag in a `<button>` (10)) or selecting an option (`<on-selected>` (22) in a `browser>` (19)).

Database. Access to databases is specified with the `<sql>` (04) 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 cast to be options of a browser by the use of the `<make-options>` (20) tag or to be items of an itemized list by the use of the `<make-items>` tag.

Services. EPKML includes the following tags to provide standard functionalities in a catalogue: `<table-of-contents>` to build an index page, `<registration-form>` for the personalization of the catalogue, `<search-form>` for starting a database exploration, `<shopping-bag>` (12) to maintain a list of products to buy , `<shopping-list>` to select products to been viewed, `<question-form>` to obtain feedback of the users, `<order>` for ordering products. An order can be sent by Internet, by e-mail, by dialing a telephone number by modem, by fax, or can be printed. The semantics of this order function will be established at installation time.

5 Conclusions and Further Steps

A study of the current state-of-art of the electronic product catalogues [Koch and Mandel, 1996] 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 from the beginning of the first *interview* by special editors and a presentation assistant. During the catalogue *design* and *generation* phase the assistance is realized with other

editors and class libraries. *Tests* accompany the entire development process including the final EPC. Based on our methodology we developed, four assistant tools (RASSI, SASSI, GASSI, and TASSI) were implemented for the EPC production.

The first experience 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. A WWW based project server is currently being implemented, which 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 to the development process of EPCs, thus 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 one stereotype stored in the knowledge base.

References

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

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

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

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

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

[Klausner *et al.*, 1994] Klausner, J.; Kraetzschmar, G.; Schneeberger, J.; and Stoyan, H. 1994. The Knowledge Mining Center. In Steels, L.; Schreiber, G.; and Van de Velde, W., editors 1994, *Position papers of the "8th European Knowledge Acquitision Workshop EKAW'94"*, number 94-2 in Technical Report, Hoegaarden. Vrije Universiteit Brussel, Artificial Intelligence Laboratory.

[Knapp *et al.*, 1996] Knapp, Alexander; Koch, Nora; and Mandel, Luis 1996. The Language EPKML. Technical report 9605, LMU München.

[Knapp *et al.*, 1997] Knapp, Alexander; Koch, Nora; Wirsing, Martin; Duckeck, Jochen; Lutze, Rainer; Fritzsche, Hartmut; Timm, Dietrich; Closhen, Patrick; Frisch, Martin; Hoffmann, Hans-Jürgen; Gaede, Bernd; Schneeberger, Josef; Stoyan, Herbert; and Turk, Aandreas 1997. EPK-fix: Methods and Tools for Engineering Electronic Product Catalogues. In Steinmetz, R. and Wolf, L.C., editors 1997, *Interactive Distributed Multimedia Systems and Telecommunication Services*, LNCS 1309. Springer-Verlag Berlin-Heidelberg. 199–209.

[Koch and Mandel, 1996] Koch, Nora and Mandel, Luis 1996. Catalogues on CD–ROM: The State of the Art. Technical report 9610, Ludwig–Maximilians–Universität München.

[Melton and Simon, 1993] Melton, Jim and Simon, Alan R. 1993. *Understanding the new SQL*. Morgan Kaufmann, San Mateo, California.

[Rumbaugh *et al.*, 1991] Rumbaugh, James; Blaha, Michael; Premerlani, William; Eddy, Frederick; and Lorensen, William 1991. *Object–Oriented Modelling and Design*. Prentice Hall, Englewood Cliffs, N. J.

[Turk and Stoyan, 1996] Turk, A. and Stoyan, H. 1996. Erfassung, Verarbeitung und Dokumentation natürlichsprachlicher Äußerungen in der Anforderungsanalyse. In Ortner, E.; Schienmann, B.; and Thoma, H., editors 1996, *Natürlichsprachlicher Entwurf von Informationssystemen*. Universitätsverlag Konstanz. 32–46.