# The epkml Language*

Alexander Knapp, Nora Koch
Ludwig–Maximilians–Universität München
{knapp,kochn}@informatik.uni-muenchen.de

Luis Mandel †
Forschungsinstitut für Angewandte Software-Technologie (FAST)
mandel@fast.de

28th November 1996

**Abstract**

Electronic Product Catalogues (EPCs) have reached the market and are gradually displacing traditional paper catalogues. EPCs usually comprise multimedia presentations, lots of special effects, video, audio, and images, besides the products being presented, their number ranging from a dozen to tens of thousands. For the specification of such catalogues, the language epkml is presented. The language is html-like, provides primitives for the construction of product families, integrates templates for the presentation of products of the same family, and allows the definition of variables as well as the definition of macros. It is window oriented, embeds SQL for database access, permits the connection with external objects, and has network capability.

**Keywords:** Electronic Product Catalogues, Multimedia, Mark-Up Language.

*This work was supported by the BMBF project EPKfix.

†The work was carried out when Luis Mandel was a member of the Institut für Informatik , Ludwig–Maximilians–Universität München.

# Contents

# Introduction

With the expansion of the World Wide Web online services and the distribution of information on CD-ROM, modern electronic support of advertising and sale of goods becomes a key factor in the marketing strategy of many companies. The technologies used to develop and deliver multimedia systems are still far from being easy and efficient and they show many weaknesses.

Due to the significant degree of difficulty in developing, producing and maintaining sophisticated multimedia software, it is necessary to get the job done by large multi-disciplinary teams of programmers, designers, media-experts, and quality control specialists. The answer to this problem lies partly in putting easy-to-use tools in the hands of a small team of software professionals and marketing experts which together would:

- help to determine the requirements,
- reduce the time to design the product,
- increment the quality testing speed,
- reduce the costs of producing and updating multimedia systems,
- simplify the maintenance of the information.

Informations systems, which focus their attention in multimedia presentation of products or services with functions that allow searching, selection, and ordering are called electronic product catalogues (EPCs). In this paper we will concentrate our attention only on this sort of information systems: EPCs on CD-ROM. All the same we are sure that most of our work will be useful for other information systems.

We think that one main issue of the problem is a connecting basic formalism, EPCs are to be developed in. `epkml` is a specification language for EPCs, that allows a declarative description of them. It is part of the EPK-fix project, that aims to develop methods and tools to support the whole life cycle of electronic product catalogues on CD-ROM. These tools will support the requirement analysis, the design of the EPCs, the realization of special services and an extensive automatic validation of the resulting catalogue. The project is carried out together with one industrial partner and three universities.

The language `epkml` was defined as an instance of the Standard Generalized Markup Language (`sgml`), based on Hypertext Markup Language (`html`) but enlarged with special features needed for the description of EPCs. `epkml` is window oriented and allows dynamic generation of layout elements. It includes templates, primitives for the control flow, variable and macro definition, among other introduced elements. The definition of the specification language was based on an object-oriented framework for EPCs, which describes the components of electronic product catalogues and the interaction among them. The determination of all the relevant EPC's components was done during the requirement analysis phase.

The first chapter presents a brief description of electronic product catalogues and states the goal of the EPK-fix project and describes the architecture of the system. The second chapter details the `epkml` language development process including the requirements analysis and the construction of the framework. Chapter three describes the characteristics and the syntax of the language `epkml`. The semantics is defined in the fourth chapter and chapter five delineates conclusions and further steps in the development and implementation. An example is given in the first appendix and the complete Document Type Definition (DTD) of the language in the second.

# 1 The EPK-fix Project

EPK-fix is a project belonging to the software engineering promotion program of the German Ministry of Education, Science, Research and Technology (BMBF). The task of this program is the research and the test of new technologies for the production of application software. The target of the EPK-fix project is to define a specification language and to develop a set of tools for the easy and low-cost production of electronic product catalogues.

## 1.1 Electronic Product Catalogues

Electronic Product Catalogues are computer controlled information systems with an important multimedia (especially visual) product presentation, navigation facilities and almost always equipped with a shopping bag administration feature. They are an inexpensive alternative to paper catalogues, but a high quality design is still related to enormous costs, because there are no appropriate tools available.
EPC developing tools would be welcome in every company or institution that wants to present its products and/or services. The catalogue design and development would be done by a teamwork of marketing experts, graphic designers, and programmers. Marketing people usually belong to the enterprise, the others may be independent catalogue developers.
In each case there are different groups or teams of people involved in the EPC business. First of all there is the person or group who makes the decision to go into the market with such a multimedia presentation for potential customers. We call them catalogue provider. They or their marketing people describe their wishes to the catalogue developer, who may design and produce it by himself or require the assistance of software experts. EPCs are designed to be used by customers or users, who are interested in the products or services that are being offered in the catalogue (we will call them users or end-users).

## 1.2 The Aim of the Project

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 project is part of a promotion program supported by the BMBF for software technology development in economy, science and engineering. Software users cooperate with software developers in each project, achieving better know-how transfer.

The EPK-fix project partners are:

- Mediatec Private Limited Company for Multimedia Solutions,
- Knowledge Acquisition Research Group of the Bavarian Center for Knowledge-Based Systems (FORWISS),
- Programming Languages and Compilers Unit of the Department of Computer Science of the Technical University Darmstadt,
- Programming and Software Technology Unit of the Institute of Computer Science of the Ludwig-Maximilians-University Munich,
- Chair for Compiler Construction of the Institute for Software Technique I, Department of Computer Science of the Dresden University of Technology.

The methodologies 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 will be a prerequisite for the use of the EPK-fix system especially in small and medium size organizations.

## 1.3 The Architecture of the EPK-fix System

The EPK-fix system components comprise a formal description language for electronic product catalogues (`epkml`) and the following four tools: the Requirements Analysis ASSIstant (RASSI), the Specification ASSIstant (SASSI), the Generation ASSIstant (GASSI) and the Testing ASSIstant (TASSI).

- `epkml` is a specification language that makes the description of the static and dynamic aspects of the electronic product catalogue possible.
- RASSI supports the informal recording of information (text, images, video) that result at the requirements analysis stage based on structured interviews. The catalogue provider expresses his desires for the catalogue to be designed during these interviews.
- SASSI is responsible for the EPC design based on the results of the RASSI tool converting them in a catalogue specification in `epkml`. Efficient and powerful editors that assist the catalogue developer are part of this component.
- GASSI makes use of the EPC specification, that was generated by SASSI, and translates the `epkml` description into a general programming language, implementing this way the electronic product catalogue.
- TASSI realizes 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.

# 2 Development of the `epkml` Language

In this section we describe the different steps we followed to develop the language. First of all an exhaustive requirements analysis was made to determine the general characteristics of catalogues and the features needed in the design of EPCs. A brief outline of our "Catalogues on CD-ROM: State of the Art" [KM96] is given. We developed a framework for the EPC components; this object-oriented view served as a careful description of the EPC elements and their attributes and led us to the next and final step: the definition of the specification language `epkml`. (See chapter 3 and chapter 4.)

## 2.1 Requirements Analysis

The analysis of the EPCs existing on the market demonstrates that they go much further than paper catalogues with cross references. They offer services like search features, demos to show end-users how to use the catalogue, enquiries through telephone communication and fax, or online ordering. Additionally, developers need features that allow an easy and low-cost production of standard catalogues but that are on the other hand, flexible enough to design more sophisticated ones. Maintenance and proof of correctness are also important subjects to be considered.

We observed that working with an electronic product catalogue can be divided into five different steps:

- *Installation*.

  In this initial step the access component of the EPC is installed onto the computer, configuring it to match multimedia hardware.

- *Presentation*.

  The user is presented with publicity messages about the company and its offerings and perhaps a demo of navigation facilities through the catalogue. During this phase the potential customer is a passive user, who is shown how the catalogue works other than in the next steps where he interacts with it.

- *Search*.

  Here the end-user enters the selecting criteria according to which the EPC then locates the matching entries.

- *Selection*.

  Alternating with the previous step the desired products are marked, thus creating the order list.

- *Order*. The list created is formatted and forwarded to the service provider or product vendor.

| | Q | O | M | R | B | S | Mi |
|---|---|---|---|---|---|---|---|
| windows | | √ | | √ | √ | | √ |
| frames | √ | | √ | | | √ | |
| overlapped frames | √ | | √ | | | | |
| modifiable windows | | | | √ | √ | | |
| fixable windows | | | | √ | | | |
| help windows | √ | √ | | √ | √ | | √ |
| help with hypertext | | | | √ | √ | | √ |
| active help | | √ | | | | | √ |
| product graph | √ | √ | | √ | √ | √ | √ |
| product comparison | | | | √ | √ | | √ |
| alternative search | √ | √ | √ | √ | √ | | |
| input for searching | | √ | | and, or, not | | √ | √ |
| communication | printer btx | printer btx | printer | printer btx | printer | printer | printer |
| animation | √ | √ | | √ | √ | | |
| audio | √ | √ | √ | √ | √ | | √ |
| special effects | √ | | √ | √ | √ | | √ |
| video | √ | √ | √ | √ | √ | | |
| games | √ | √ | | | √ | | |

Q: Quelle [Que95], O: Otto [OTT95], M: Mercedes Benz [Mer95], R: RS [RS ], B: Bosch [Bos95], S: Springer [Spr95], Mi: Microsoft [Mic95].

Figure 1: Catalogue comparison

Depending on the relative importance of the presentation, the searching, the selection and ordering steps, we distinguish between *presentation*, *searching*, and *order* catalogues.
Each step was analyzed in detail and a list of the observed elements as well as the system behaviour reacting to different user's stimuli was made. The content of the list was then checked against many catalogue examples to determine even slight differences in common features. These results have been tabulated. A small subset of one table is shown in figure 1.

We conclude that general requirements of the language `epkml` are:

- support of multimedia elements,
- incorporation of control constructions to permit interaction with the user and navigation through the catalogue,
- simple handling of catalogue standard operations like user registration, product searching, order forms, and question forms,

- good implementability,
- good validability,
- easiness to be learned.

As another conclusion of our analysis we grouped the EPC functional requirements to be considered in the design of the specification language, as follows:

- *Static requirements*.

  These mean the existence of layout elements in the language, such as window, frame, button, check-box, pull-down menu, slider, text, paragraph, heading, listing etc.

- *Dynamic requirements*.

  This group includes every interactive situation, such as starting or stopping an animation or a video, navigating by clicking on buttons, scrolling in a browser, selecting help functions, sending an order, etc.

- *Data requirements*.

  These are primarily products, companies, and customer information, help text or help windows, navigation sequences, orders, and multilingual text for the pages. All this multimedia information has to be stored in files or databases.

To document the requirements of the language `epkml` in detail, we have choose an object-oriented approach, that is the development of an application framework using techniques as described in [Dav90], [Pfl91], and [RBP+91].

## 2.2 The EPC framework

An EPC framework is a set of cooperating classes that make up a reusable design for an EPC specification language.
The following steps were performed in constructing our framework: identification of abstract and concrete classes, definition of attributes, identification of associations and aggregations between objects, choice of methods that reflect their behaviour, organization and simplification of the model using inheritance, iteration to refine the framework, grouping classes into components or modules. This steps are suggested by [RBP+91] for the Object Modeling Technique (OMT). A good overview on object-oriented methods can be found in [Bal95].
The resulting interacting components, observed in every EPC are: *structure*, *layout*, *direction*, *product database*, and *services*.
The selection of the word *direction* needs a few more words: the choice follows the idea of directing a film. The contemplation of a catalogue can be compared to the action of seeing a movie. The viewer of the movie has no chance to modify the sequence of the scenes while the catalogue user develops his own screenplay making use of the navigation facilities. This concept is supported by authoring tools like Director from Macromedia [Mac95].

- The *structure* is the skeleton of the catalogue; it comprises a graph or hierarchy of themes and pages and the navigation between them to guide the user through the catalogue.
- The *layout* is the static description of frames, windows and their contents.
- The *direction* describes the dynamic aspect. We can distinguish between macro-direction for the navigation through the catalogue and micro-direction for the activities within a frame or window.
- The *product database* component supports all the information about offers, in such a way that it can easily be searched, exchanged, and maintained.
- The *services* add some comfort to the EPC allowing for example the administration of orders, the user registration, the access to help functions, online communications, and the catalogue navigation.

It is beyond the scope of this paper to describe all classes of the EPC framework, in every section only a brief description of some relevant classes and their variables and methods are given. A detailed description can be found in the report of the EPK-fix project [KKMW96]. For the graphical representation we have chosen the OMT notation. Examples of part of the components are given.
We use a `java`-like syntax to describe each class of the framework, its instance variables, and the methods which define its behaviour. For more details about `java` see [GM95], [vHSS96], [Fla96], [AG96]. In the next sections names of classes, attributes and methods belonging to the framework are printed in typewriter font.

### 2.2.1 The Catalogue Structure

A catalogue consists of a set of themes, databases, a configuration, and the direction, that acts as interface between them. A theme is nothing else than a view of the database grouping products under some aspects. Different theme hierarchies conform different views of the same database which are not restricted to conform a partition of it, i.e., a given product can appear in two different hierarchies. Themes additionally build a navigation structure to allow the jump from one theme to another.
The catalogue's structure is organized in `Themes` and they again may structured in `Themes` and `Themes` may be contain one or more pages, we called them `VirtualPages`. The aggregation relation of these classes can be seen in figure 2. Both are classes that inherit from the abstract class `Structure`, which has only a grouping function and for that reason is not included in figure 2. The abstract classes `Layout`, `Direction`, and `Database` mentioned in the figure 2 together with classes that inherit from them are described in the following sections.
The class `Catalogue` has a method `start()`, which calls the initialization process of `Configuration`, starts the first process and the first theme. `Configuration` is responsible for the installation process. The method `start()` of `Theme` starts the first process and the first virtual page and the method `start()` of `VirtualPage` awakes the first process and the drawing of the layout elements of the first page.

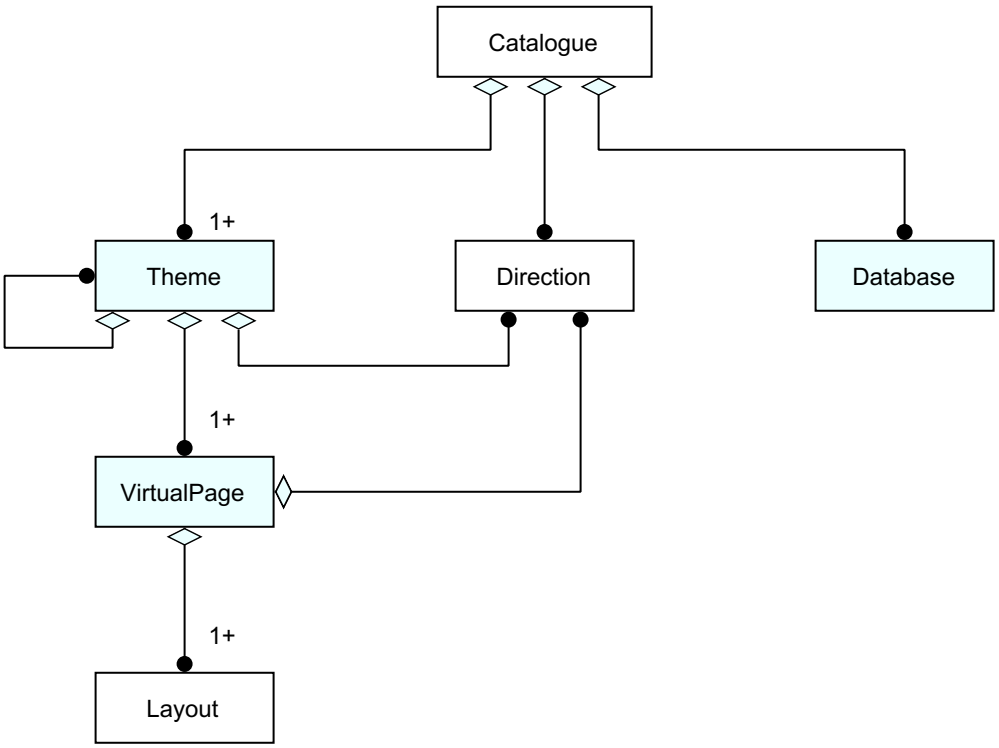


Figure 2: The catalogue structure

```
class Catalogue extends Object
{
  private String name;
  private String version;
  private String author;
  private String copyright;
  private List<Theme> contents;
  private List<Database> data;
  private List<Thread> direction;
  private Configuration configuration;

  public Catalogue(String name, String version,
                   String author, String copyright,
                   List<Theme> contents, List<Database> data,
                   List<Direction> direction);

  public start();
}
```

```
class Theme extends Structure
{
  private String name;
  private List<Theme> subthemes;
  private List<VirtualPage> pages;
  private List<Thread> direction;

  public Theme(String name, List<Theme> subthemes,
               List<VirtualPage> pages, List<Thread> direction);

  public start();
}


class VirtualPage extends Structure
{
  private String name;
  private List<Layout> layout;
  private List<Thread> direction;

  public VirtualPage(String name, List<Layout> layout,
                     List<Thread> direction);

  public start();
}
```

### 2.2.2 Layout

Layout elements are grouped into other layout elements and these can again be grouped into frames and so on. The inheritance diagram of most of the layout classes is shown in figure 3. Each frame has a border (`padding`) defined as the distance from the content to the frame and which is considered part of the object. The elements are placed in relative form, or if coordinates are given, the `position` will be absolute. Other important attributes that inherit every layout object from the `Layout` class are `layer`, `size`, `margin`, `visible`, and `display`. Margin is the distance to the surrounding objects, which differs from padding because it does not belong to the object. If `visible` is false the object will not be seen, but it has already an assigned position. The method `draw` is responsible for the representation of the object on the screen.

```
abstract class Layout extends Object
{
  private Boolean visible;
  private Integer layer;
  private Position position;
  private Extension size;
  private Padding padding;
  private Padding margin;
  private UserMode usermode;
  private Display display;

  ...

  public draw();
}
```

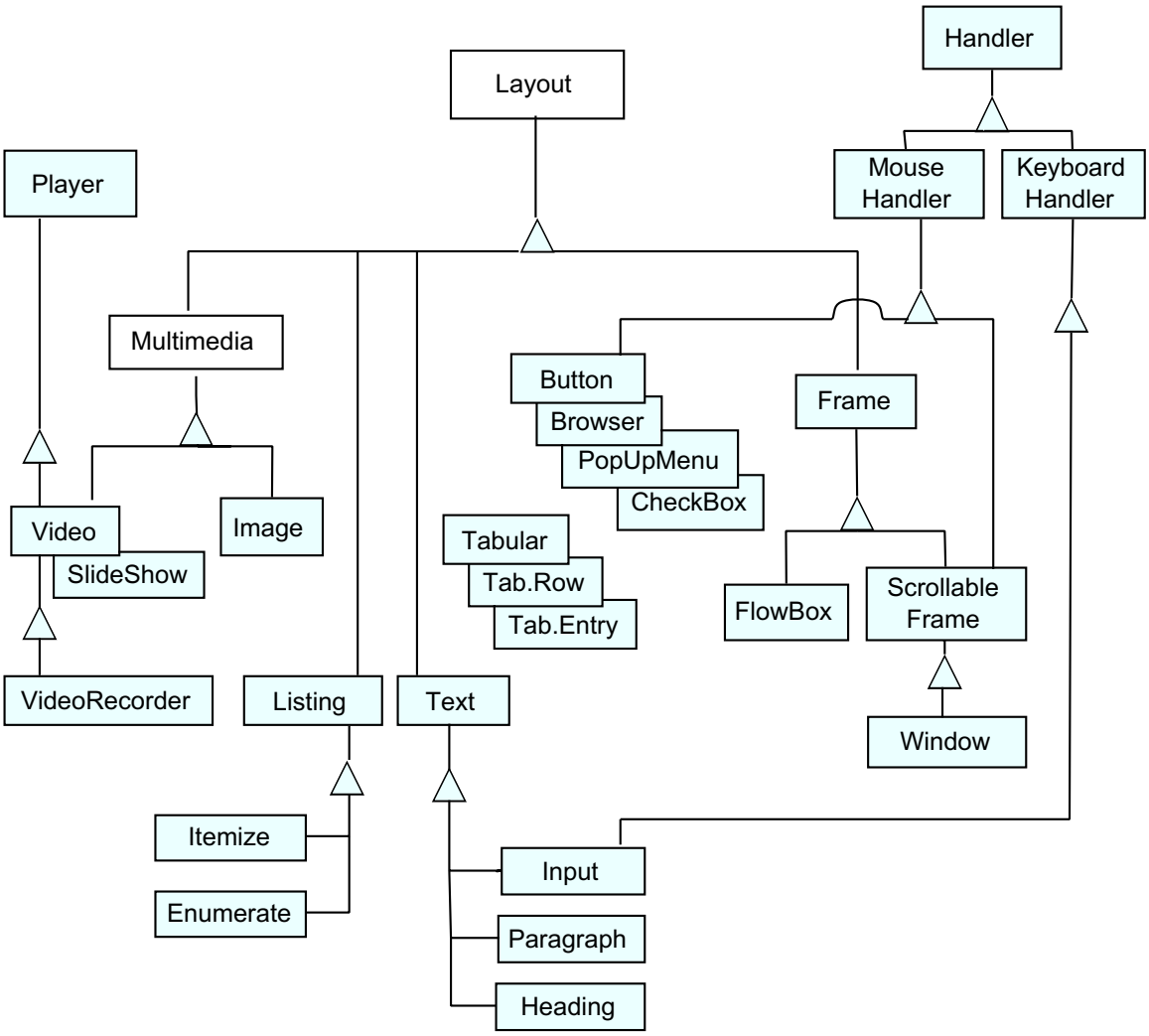


Figure 3: Diagram of some layout components

Many classes inherit from class `Layout`, redefining the method `draw` if necessary and adding specific variables. We will mention here only those we think are the most significant.
Class `Frame` serves to group layout elements allowing the definition of a background. Variables `alignment` and `distribute` are set to arrange elements in a different manner in the frame. The class `ScrollableFrame` adds an horizontal and a vertical slider to the frame and methods to handle them. `Windows` are frames with the capability to be moved within the screen, to change their size, to be reduced to an icon, and to be closed with the variables `moveable`, `sizeable`,

iconizable, and closeable respectively.

```
class Window extends ScrollableFrame
{
  private Layout title;
  private Image icon;
  private Button iconizable;
  private Button closeable;
  private Button sizeable;
  private Boolean moveable;
  private Extension minsize;
  private Extension maxsize;

  public Window(List<Layout> contents, Layout title, Image icon,
                Button iconizable, Button closeable,
                Button sizeable, Boolean moveable,
                Extension minsize, Extension maxsize,
                Slider horizontalslider, Slider verticalslider);

  ...

  public moveWindow();
  public resizeWindow();
  public fullSizeWindow();
  public iconizeWindow();
  public hideWindow();
}
```

The class FlowBox allows text flow horizontally around layout elements.
For objects of the class Text the usual features are offered: selection of a font, fontsize, style and colour for them. The text itself is stored in source, which together with the method changeLanguage makes it possible to support multilingual catalogues. Classes Paragraph and Headings inherit from class Text. Paragraph includes attribute indent for indentation and baselineSkip for the separation between lines. Heading has a leftMargin and a number to be added to the text and a style for the same.
Classes for the definition of lists and tables are also part of the layout component of the framework. Classes Listing, Enumerate, and Itemize are lists, which differ only in the symbols that precede the text. The class Tabular together with the classes TabularRow and TabularEntry permit the creation of tables.
Multimedia elements are described in the classes Video, Image, and SlideShow, which are subclasses from the abstract class Multimedia.
Very important layout elements are buttons, because they are indispensable by laying out the navigation through the EPC. A Button can be not clickable. Methods enable and disable allow the status to be changed.

```
class Button extends GroupLayout, MouseHandler
{
  private List<Layout> contentsdisabled;
  private List<Layout> contentsnotclicked;
  private List<Layout> contentsclicked;
  private Integer default;

  public Button(List<Layout> contentsdisabled,
```

```
                List<Layout> contentsnotclicked,
                List<Layout> contentsclicked,
                Integer default);

  ...
  public enable();
  public disable();
}
```

Three different contents for the `Button` can be defined with `contentsclicked`, `contentsnotclicked`, and `contentsdisabled`.
The class `GroupLayout` is not represented in figure 3 for simplicity reasons, grouping is the only purpose of this abstract class. Other interactive classes that inherit from class `Layout` are the well-known: `Browser`, `PopUpMenu`, `CheckBox`, `RadioButton`, `Slider`, and `PullDownMenu` (the last three are not included in figure 3, they can be drawn below checkbox). They inherit as well from the class `MouseHandler`.

### 2.2.3 Direction

The dynamic direction is defined separately from the static layout. This separation is got over by the use of multiple inheritance.
Independent and parallel threads communicate through events and are synchronized with clocks are the basic direction objects. There are two conditions for the implementation of this direction model: the existence of a background process, the event manager, and a system clock with the necessary precision.
From the abstract class `Direction` the classes `Clock`, `Event`, and `Thread` inherit the variables `usermodus`, and `limit` (see figure 4). The first specifies the users permissions and the second provides boundaries for the number of active processes, events, and clocks.

```
abstract class Direction extends Object
{
  private UserMode usermode;
  private Integer limit;

  public setUserMode(UserMode usermode);
  public UserMode getUserMode();
  public setLimit(Integer limit);
  public Integer getLimit();
}
```

With the help of `Clock` events can be synchronized over the time. `Timer` and `PeriodicalTimer` have additionally alarm and periodical alarm functionality respectively. Processes are informed through events of any change in the actual state. An abstract class `Event` resumes the characteristics of the different type of events. These are `FocusEvent`, `ClockEvent`,`MouseEvent`, `ActionEvent`, `KeyboardEvent`, and `InteractionEvent`. Some of these classes have subclasses that describe accurately what happens when the mouse or keyboard key is up or down or the effect of selecting an object with the mouse.

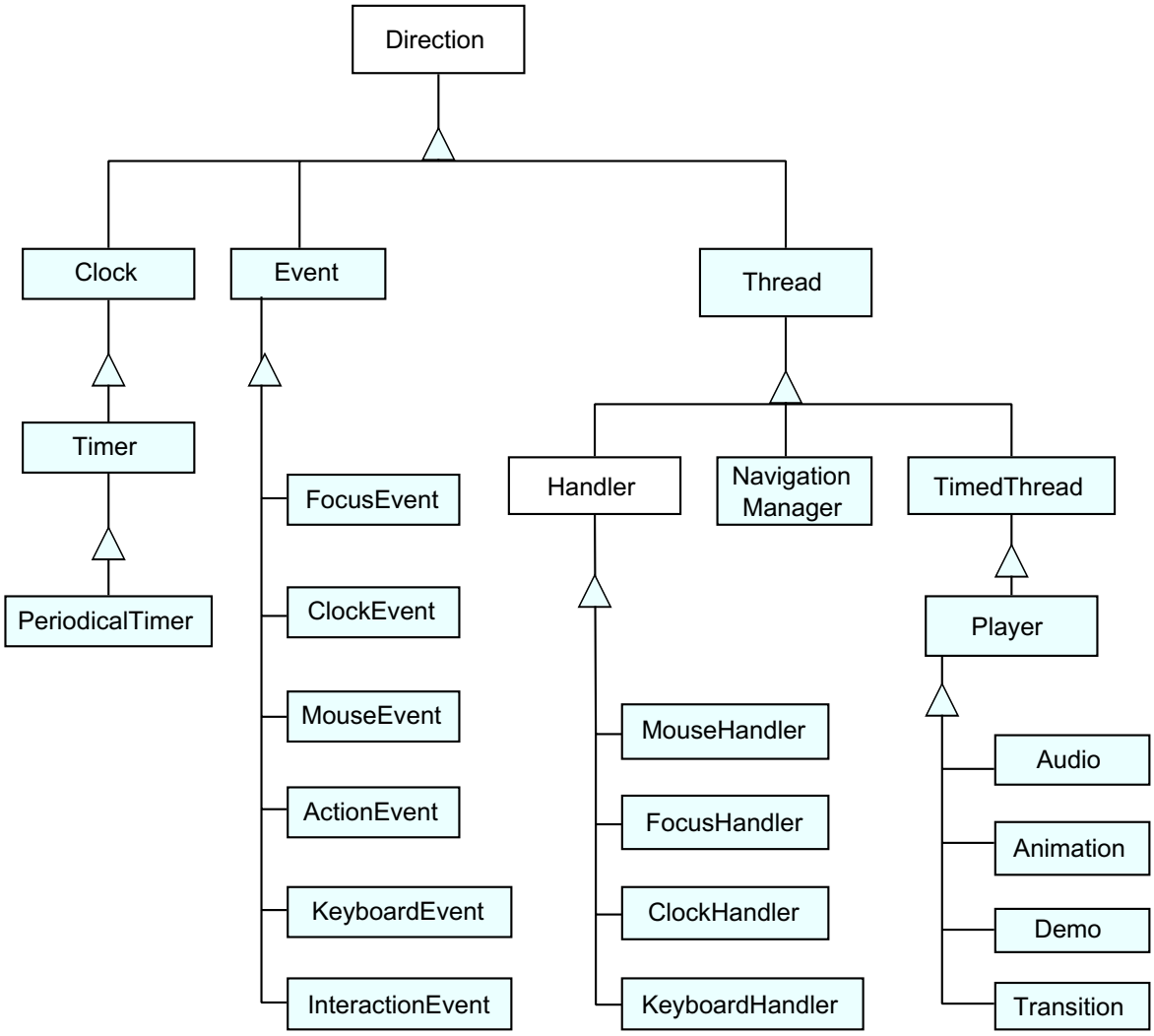


Figure 4: Some classes from the component direction

```
abstract class Event extends Direction
{
  private Time stamp;
  public setStamp(Time stamp);
  public Time getStamp();
```

```
  public Boolean signal();
  public Boolean asignal();
}
```

For the class `Thread` there is defined a variable `priority` and a list of instances of subclasses of `Event` which is interested in that kind of processes. If an event matches the interest of a thread, the method `handler` of the process is called and an event of lower priority may be interrupted.

```
class Thread extends Direction
{
  private Integer priority;
  private PriorityList<Event> interests;

  public Thread();
  public Thread(Integer priority, PriorityList<Event> interests);

  public Boolean start();
  public Boolean stop();
  public run();
  public Boolean suspend();
  public Boolean resume();
  public Boolean kill();
  public Boolean yield();
  public Boolean join(Time wait);
  public Boolean handler(Event event);
  public setPriority(Integer Priority);
  public Integer getPriority();
  public setInterests(PriorityList<Event> interests);
  public PriorityList<Event> getInterests();
}
```

We defined three subclasses for `Thread`, there are `Handler`, `NavigationManager`, and `TimedThread`, a brief description is given below. `MouseHandler`, `FocusHandler`, `ClockHandler`, and `KeyboardHandler` are subclasses of the class `Handler` and they allow through multiple inheritance the easy connection of the layout with the direction. For example, the specification of the class `KeyboardHandler` is as follows:

```
class KeyboardHandler extends Handler
{
  public onKeyDown(KeyboardEvent event);
  public onKeyUp(KeyboardEvent event);
}
```

The class `NavigationManager` is defined to control the catalogue navigation. Therefore it includes a `stack` of virtual pages and a variable `actual` to register the visited and the actual page. Pages can be incorporated in a `bookmarks` list. The methods `back`, `next`, `bookmark`, and `unbookmark` have been defined to administrate these lists of pages.

```
class NavigationManager extends Thread
{
  private List<VirtualPage> stack;
  private VirtualPage actual;
```

```
  private List<VirtualPage> bookmarks;

  public NavigationManager();

  public VirtualPage back();
  public VirtualPage next();
  public register(VirtualPage obj);
  public skip();
  public bookmark(VirtualPage obj);
  public unbookmark(VirtualPage obj);
  public List<VirtualPage> getHistory();
  public List<VirtualPage> getBookmarks();
}
```

To model uniform synchorized events the class `TimeThread` is provided with a clock. To control different interactive multimedia elements the following classes are available: `Audio`, `Animation`, `Demo`, and `Transition`. Their general behaviour is summarized in the class `Player`, that inherits from `TimedThread`.

```
abstract class Player extends TimedThread
{
  private List<Marker> markers;

  public play();
  public halt();
  public reverse();
  public forward(Time time);
  public rewind(Time time);
  public goto(Marker marker);
  public setMarkers(List<Marker> markers);
  public List<Marker> getMarkers();
}
```

### 2.2.4 Database

Information about the products, the orders, the company, and about the EPC structure must be stored to allow the catalogue construction, visualization, and navigation. (No graphic representation is included here.)
The abstract class `Database` describes their general properties and the class `DbObject` the properties of the database entries. At the moment we restricted us to relational databases. The catalogue provider will normally own a product database, which in most of the cases will be reorganized by the developer to obtain an adequate product hierarchy, independent from the layout and the navigation.
Products can be grouped in product groups and these again in groups. To describe these hierarchies the classes `Product` and `ProductGroup` are provided. Each product is characterized only by a `number`, a `name`, a list of `properties`, a `description`, and a `price`. Every property that depends on the nature of the product can be defined separately and has a `name` and a `contents`.

Class `Order` is defined to allow the storage of products that the end-user marks or selects to be part of the shopping bag and then decided to order. Products that had already been ordered in previous catalogue sessions by the same user can also be located.

```
class Order extends DbObject
{
  private String number;
  private User user;
  private Date date;
  private Product product;
  private Integer quantity;
  private Boolean marked;

  public Order(Date date, User user, Product product,
               Integer quantity, Boolean marked);
  public Order(String number, User user, Product product,
               Integer quantity, Boolean marked);

  ...
}
```

Class `Help` offers `pages` and `keywords` to assist the end-user. The methods `show` and `hide` permit the visualization or the hiding of help pages related to a keyword.
Methods of the class `Company` will manage the information for the general presentation of the company and maybe the presentation of the catalogue itself, showing facilities and advantages.
To make navigation possible information about every virtual page that has been visited will be stored with the assistance of a class `Dialogue`. Catalogue personalization is achieved by class `User`, which has variables and methods to register and retrieve all data that identify the end-user.

### 2.2.5 Services

This group of classes adds some comfort and is a great help for the catalogue developer, who can incorporate these special catalogue features without additional effort. We designed standard forms for situations that require user input. Their description can be found in the following classes: `RegistrationForm` which is a template for the input fields, with it the end-user can achieve the catalogue personalization, `SearchForm` for special queries to the database, `HelpForm` for assistance by the catalogue use, and `QuestionForm` if the provider is interested in the users feedback. (See figure 5.)
The shopping bag functionality is supported by the class `ShoppingBag`, which allows to add products to the list, delete products, change the quantity to order, and compute and store the total amount of the selected products.

```
class SearchForm extends Window, KeyboardHandler
{
  private PopUpMenu index;
  private Browser show;
  private OKButton ok;
  private Backbutton back;
  private Helpbutton help;

  public SearchForm(List<Layout> contents, PopUpMenu index,
```

```
                    Browser show, OKButton ok,
                    BackButton back, HelpButton help);
}


class ShoppingBag extends Window
{
  private MultipleBrowser orders;
  private Text total;
  private OKButton ok;
  private CancelButton cancel;
```

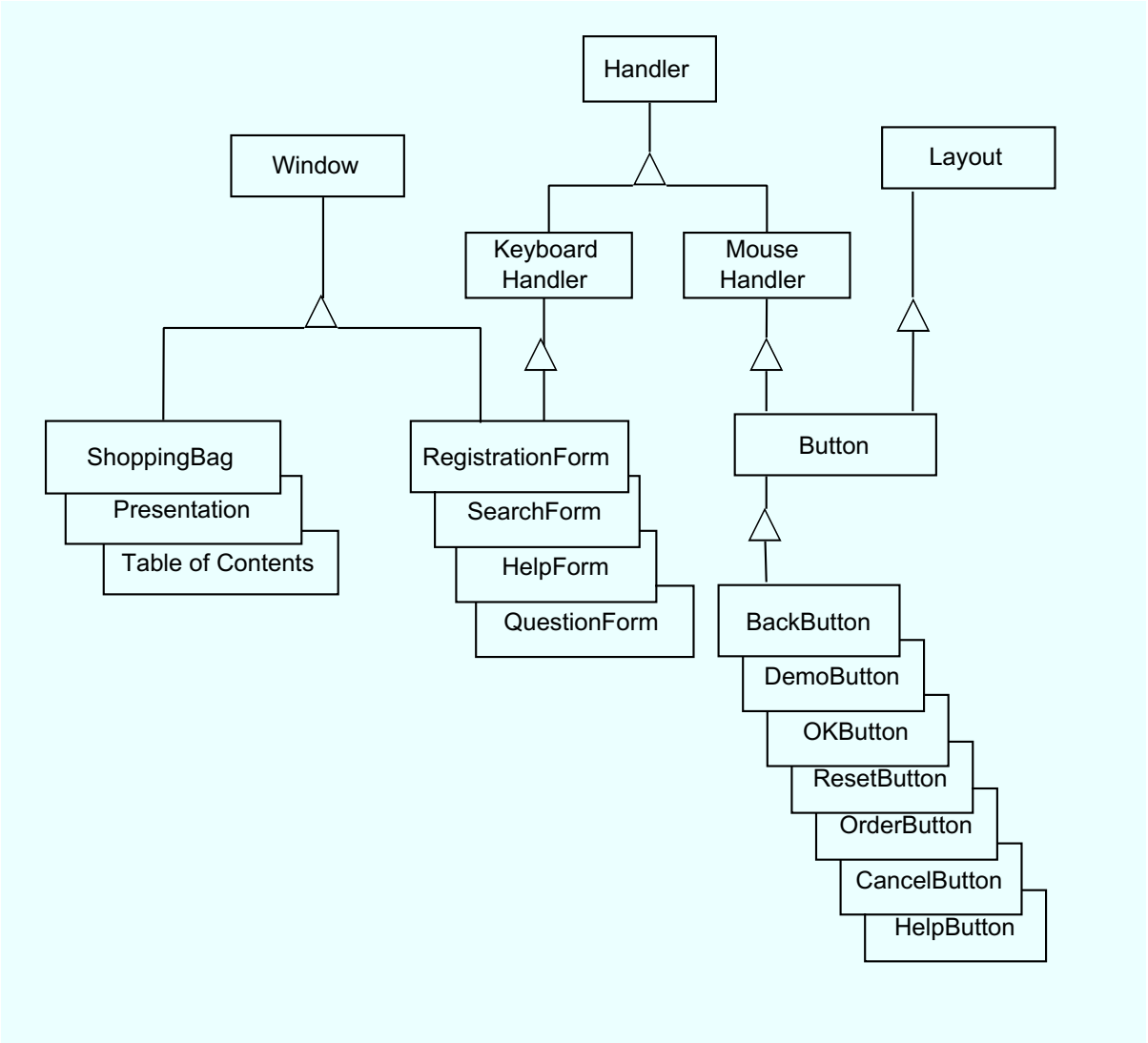


Figure 5: Services

```
  private HelpButton help;
  private OrderButton order;
  public OrdersBag(Window window, MultipleBrowser orders,
                   Text total, OKButton ok, CancelButton cancel,
                   OrderButton order, HelpButton help);
  public sum();
}
```

Usually an EPC begins with a company presentation and follows with an index with references to different sections. For the company overview we provide a special class `Presentation`, which include the possibility of a demo. For the index we designed a `TableOfContents`. It is a class that comprises a list of `themebuttons`, a list of `themes`, and three "special" buttons: `demo`, `help`, and `exit`. With the last the user can leave the catalogue.
Services are also supported by special buttons like `BackButton`, `DemoButton`, `OKButton`, `ResetButton`, `OrderButton`, `CancelButton`, and `HelpButton`.

# 3 The Syntax of `epkml`

The specification language `epkml` is result of the exhaustive requirements analysis made and is based on the framework described in the previous chapter.
EPCs on CD-ROM have similarities with hypertext documents distributed on the World Wide Web (WWW), in what concern the layout and the navigation. Web pages are described with HyperText Markup Language (`html`), an instance of the Standard Generalized Markup Language (`sgml`). For more details on `sgml` and `html` see [Gol94] [vH94]) [Gra95].
On account of the advantages of a standard, `epkml` is defined as an instance of `sgml`. Even though in `epkml` most of the `html` layout elements can be found, it has been enlarged with important additional capabilities incorporating ideas from `java`, `framemaker`, TEX, and authoring tools.
In the first section of this chapter we enumerate the features that characterize the language which we present in this report. The second section outlines the most important elements of the language with subsections for layout, structure, control, database, and services. In the last section we give an overview of the elements defined for the catalogue structure description, there are header, externals, styles, definitions, and main.

## 3.1 Characteristics of the `epkml` Language

The basic features of this language are the following:

- `epkml` *is* `html` *like.*

  `epkml` is an instance of `sgml`. This decision gives us two advantages: it is a standard (ISO 8879). A public domain software, i.e. `sgmls` can be used to parse every Document Type Definition (DTD). Everybody who knows rudiments of `html` will not find it difficult to learn.

- `epkml` *admits the hierarchical organization of catalogues themes.*

  An EPC can be seen as a front-end of a database. The products are organized in hierarchies. For the construction of this "class hierarchy" we have built–in facilities in the language. Using these facilities tree hierarchies —called themes— can be defined and for each theme the developer has the possibility to define the products belonging to the theme and the presentation of these products. This is possible using the templates mentioned below.

- `epkml` *integrates templates.*

  Usually it is desirable to have the same presentation for all the products belonging to the same family. For example, for an EPC for fashion, the catalogue developer may want to give a blue logo to all the products of winter fashion. Such a presentation is defined using a template which may include generic variables for the products to be displayed.

- `epkml` *offers special services.*

  EPCs have some commonalities. Usually an EPC provides the possibility to order a product, it has a help function, a demo, a quit button, etc. Such features are included in `epkml` as built-in functions.

- `epkml` *allows variable definition.*

  The language offers the possibility to define variables. That is, in an EPC one can define new variables which later can be used either as entities for reuse of code or as an attribute of a tag.

- `epkml` *includes macro definition and expansion.*

  The variable mechanism is not enough for name abstraction, so that the facility to define macros has been added to the language. Macros are stated to allow parameters of two different types: attribute and elements.

- `epkml` *has primitives for control flow.*

  Some control flow primitives are present in the language. In principle any layout element (i.e. window, frame, paragraph, text, etc.) can be opened or closed by the use of tags. Some active elements like radio-buttons, browsers, buttons, sliders, etc. react to external inputs such as mouse clicking. For these objects we have defined special tags. Control elements that are specified under the scope of such a tag will be executed only when the respective layout element is clicked, focused, etc.

- `epkml` *is window oriented.*

  In contrast to others "screen oriented" languages we have developed a window oriented language with the full functionality of windows, i.e. overlapping, iconization , etc.

- `epkml` *embeds SQL-statements.*

We assume that the products of the EPCs are organized in a relational database. For such a database we use SQL as standard query language. A tag is provided for database queries and general access.

- `epkml` *allows connection to external languages.*

  A connection to the external world is provided via applets as it is in `html`. Applets allow the specification of parameters. This is especially suitable for connection with external objects and the operating system.

- `epkml` *has dynamic generation of layout elements.*

  The result of a query to the database can be cast to be the contents of a browser or the contents of an itemized list. It is possible to do this "on the fly" in `epkml`.

- `epkml` *provides network capability.*

  The SQL-statements are in fact requests sent to a database server. This database could be local, in the simplest case or refers to an external server. The locality of a request is transparent to the user.

## 3.2 Elements of the Language

The language `epkml`, as already said, is defined as an instance of `sgml`. That means that the language uses mark-up tags. They are written `<name-of-the-tag>` for the opening tag and `</name-of-the-tag>` for the closing tag, thus separating a block. For some tags the closing tag is defined as optional.
For the full grammar in form of a DTD (document type definition) and some information on `sgml` see Appendix B. In the sequel, some familiarity with `html` is assumed [Gra95].

### 3.2.1 Layout Elements

The layout features of `epkml` are a superset of those of `html`. Thus different text fonts and styles are provided, paragraphs (`<p>`), `<image>`, `<frame>`, etc. may be used. Also the known interactive elements such as `<browser>`, `<checkbox>`, `<input>`, and so on, are available.
According to the framework, we added `<window>` (thus making `epkml` window oriented instead of screen oriented), `<flowbox>` for images inside texts, and some other miscellanea such as `<pulldown-menu>` or `<button>`. Additionally, the time-dependent elements `<video>`, `<slide-show>`, and `<audio>` were introduced.
Any layout element may now be positioned absolutely by means of attributes `xpos` and `ypos`. They may also be provided with some margin and padding, giving extra space surrounding the element. If no positioning information is given, for these elements the layout takes place as in `html`.

```
<frame name=hello
       xpos=10pt
       ypos=20pt>
  <p>Hello world!
  <button name=quit>
    <p>Quit
    <on-click>
      <exit>
  </button>
</frame>
```

The interactive layout elements were enriched by specifyable methods, e.g. <on-click> for <button> that are invoked if an interaction takes place. The details are described in section 3.2.3.

### 3.2.2 Structure

The theme hierarchy is the heart of the language. <theme> implements theme and virtualPage of the framework.
Each <theme> includes its <extension>: an SQL statement declaring the products it covers, <page>: a form to be filled with actual product contents, <exceptions>: products of the extension to be treated specially with their own <page>, and perhaps some sub-themes.
Pages are some predefined templates for structured data presentation. Its gaps can be filled "on the fly" with values obtained via SQL-statements.
The exceptions may be used on the one hand for showing products that do not fit the norm syntactically or, on the other hand, for rendering products that shall attract more attention than the other ones, e.g. those that are on sale.

```
<theme name = general>
  <extension result = general-result>
    <sql>
      ...
    </sql>
  </extension>
  <page name = general-template>
    ...
  <exceptions>
    <sql>
      ...
    </sql>
    <page name = exception-template>
      ...
  <theme name = sub-general>
    ...
</theme>
```

Through the theme hierarchies a forest structure is build up in which navigation takes place by the special commands <next>, <previous>, <up>, <down>, and <back> (implementing

the NavigationManager). 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.

### 3.2.3 Control

Besides the new layout elements, the database access feature, and the theme hierarchies, we added control elements to the language epkml.

**Variables.** First of all, variables were introduced. They are written as $name$. Their values may be modified by the <set> tag. Variables can contain both element and attribute contents, but they must be used consistently.

```
<var name = months value = 12>
<var name = pic>
   <img src = pic.gif>
</var>
<set name = pic>
  <img src = pic2.gif>
</set>
```

There are some predefined variables for special uses. These variables are $title$, $author$, $date$, $last-modified$, $curdate$, $curtime$, $dimension-unit$, and $time-unit$.

**Macros.** To achieve more convenient programming, there is a <macro> tag to abbreviate some often used constructs.

```
<macro name = dummy
    attribs = "a"
      elems = "e">
  <img src = $dummy.a$>
  <p>$e$</p>
</macro>
```

The attribute attribs is used to specify the list of parameters which will be passed. They are used as sgml attributes whereas the attribute elems has the same target but for sgml elements. It is not allowed to use a parameter which has been defined to be an attribute as an element and vice versa.
Macros are expanded using the <expand> tag.

```
<expand name=dummy>
  <attribute name=a value=12>
  <element name=e>
    Hello world!
  </element>
</expand>
```

**Control Flow.** The control of the flow of a catalogue is sequential from top to bottom, but it may be modified by the use of special 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: First, structurally, there may be a change between themes by the use of <next>, <previous>, etc., already mentioned. Second, any layout element and any theme may be called directly via an <open> statement provided with a name: the element called is shown and its statements are executed. Conversely, elements may be closed with <close>, but this has no effect on the control flow.
For a finite number of repetitions there has been defined <foreach>, which cause the sequence of tags between the opening-tag and the matching end to be executed for each database entry value indicated by <in>.
Last but not least, the user may affect the control flow. Whenever he interacts with the catalogue by e.g. clicking a button, the statements that are declared inside the action method of the interactive element (i.e. <on-click> in case of <button> and similarily for browsers, sliders, etc.) are executed. Those interactions are registered when the catalogue is in a waiting state by executing the <wait> statement. It can wait indefinitely or until a certain condition (an alarm, the end of a video, etc.) is fulfilled.

```
<button style=sport-style>
  <img src=stop.gif>
  <on-click>
    <close name=sport-presentation>
  </on-click>
</button>
```

Besides the standard mode of control flow there is a special demo mode during which the principal features and facilities of the catalogue may be shown automatically. Actions under the scope of the tag <demo> are executed in sequence, but without the possibility of user interactions. For the simulation of these interactions the <click> tag is supplied.

```
<demo name=general-demo>
  <set name=time-unit value=sec>
  <open name=winter-theme>
  <click name=help>
  <wait end-of=5>
  <close name=help-window>
  <open name=pres-video format=mov>
  <wait end-of=pres-video>
  <next theme>
  <wait end-of=5>
  <exit demo>
</demo>
```

**External Functionality.** The tag <applet> permits to call external functions in the same way as in the newest versions of html. The specification of parameters in applets allows among other possibilities the connection with external objects and the operating system.

### 3.2.4 Database

Database access is reached via the <sql> tag. Statements under the scope of this tag must be written in standard SQL, see [MS93]. The result of a SELECT statement of SQL is bound to the name specified via the attribute result.

```
<sql result=trousers>
  SELECT *
  FROM fashion
  WHERE kind=trousers
</sql>
```

When an SQL-statement is executed the result can be cast to be options of a browser by the use of the <make-options> tag or to be items of an itemized list by the use of the <make-items> tag. An example of the use of the tag sql is given in Appendix A.

### 3.2.5 Services

Services are the standard functionalities provided 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). It corresponds to the class with the same name in the framework presented in section 2.2.
- <registration-form>: permits the personalization 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 done onto the database every time the end-user fills in this form with the adequate keywords.
- <shopping-bag>: is a template for the products list that has been selected as "product to buy", supposed to allow update functions as modification of the quantity to order or to eliminate any product of the list.
- <shopping-list>: serves to administrate the list of products selected all together at the begining 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 decided.
- <question-form>: to be filled in by the end-user to provide some feedback to the catalogue provider about the success of the catalogue or to criticize it.

## 3.3 The Structure of an epkml Catalogue

As is shown in the example listed below, an epkml catalogue is divided into five sections. We give a brief explanation of each of them.

```
<epkml>
  <header
    title = "The Shortest EPKML Catalogue"
    author = "Me"
    date = "03/10/96"
    last-modified = "03/10/96">
  <externals>
  <styles>
  <definitions>
  <main>
    <exit>
</epkml>
```

### 3.3.1 Header

The first section is the <header>, which only comprises documentation about the catalogue. Under this tag the attributes title, author, date and last-modified are mandatory and they generate values for the variables $title$, $author$, $date$, and $last-modified$.

### 3.3.2 Externals

The <externals> section is planned primarily to declare objects, that are external to the language. Such is the case of the relational database scheme, declared for consistency checks purpose. Then every SQL-statement can be checked against the scheme of the database in order to detect unknown tables or columns.
The SQL-statements can also be seen as requests to a database server. These requests may be local or remote to the database. Using the attribute <path> of the <scheme> tag, the developer can specify an URL which is the path to the (local/remote) database. (The implementation of this feature has not yet been done. Such requests/answers can be implemented using a new MIME-type or using a database proxy server. I.e. an answer to such request will have a Content-Type:application/sql to be interpreted by the epkml-browser.)
<class> is another allowed tag in <externals>. It is used to declarate new tags allowed in the epkml-language. In the example given below a new tag called <my-window-with-logo> whose only attribute is logo is declared. Thus new objects of the class <my-window-with-logo> are created. It is also possible to indicate via the attribute methods of the <my-window-with-logo> tag to which methods they must respond.

```
<externals>
  <scheme name = my-database>
   <table name = "prod-desc"
       columns = "code price desc">
```

```
  </scheme>
  <class name = my-window-with-logo
        slots = "logo"
      methods = "redraw iconize">
</externals>
```

### 3.3.3 Styles

A style is a collection of stylesheets and a stylesheet is a set of defaults for differents tags collected under a name. `epkml` allows the declaration of styles for layout elements via the `<style>` and `<stylesheet>` tags. Every attribute of a layout element can be set to a given stylesheet.
A stylesheet can extend or inherit from other stylesheets and they can be reused by the specification of the attribute `extend` of the `<stylesheet>` tag. Multiple inheritance of stylesheets is allowed and name clashes are solved from left to right in the `extend` list or by using the `<default>` tag.

```
<styles>
  <stylesheet name = catalogue-style>
    <default>
      <p lftmrg = 1cm>
    </default>
  </stylesheet>
  <stylesheet    name = my-catalogue-style
              extends = catalogue-style>
    <default  extends = other-catalogue-style>
      <p baselineskip = 12pt>
    </default>
  </stylesheet>
</styles>
```

### 3.3.4 Definitions

Global specifications are done with the `<definitions>` tag. It was originally thought for global definitions of variables, macros and frames, but later it was extended to allow global definitions of any layout element. Definitions made under these tag can be used everywhere and may be be redefined under the scope of another tag. Then the rules of lexical scope will be applied.
An example of the employment of the `<definitions>` tag is shown in Appendix A.

### 3.3.5 Main

The `<main>` part of an EPC is composed by the sequence of instructions to be executed when the EPC is loaded. Basically under `<main>` the developer specifies the presentation of the whole catalogue, help facilities, navigation solutions through the structure and an exit option.

# 4 The Semantics of epkml

In what follows we give a short overview of a formal structural operational semantics (cf. [NN92]) of the language epkml. We describe a model of states that reflects the necessary information to control the execution flow of an epkml-program, but which abstracts from the exact layout and database internals. The possible transitions between these states are given by rules that show how an epkml-statement affects the state in which it is executed. Some of these rules may be annotated by external events thus extending conventional structural operational semantics.
The semantics provides a static binding for the variables and the procedures (i.e. the layout elements), a block structure for the scopes of variables, and "call-by-value" parameter handling. It takes care of possible interactions by the user and of the time aspects the language offers.

## 4.1 The Model

The set of all possible epkml-statements (the programs) is called $S$. Any statement that does not contain free variables is a value of our semantics.
Some statements contain actions (e.g. <on-click> in <button>); we define a function

act : $S \hookrightarrow S$

that returns this action for a given statement, whenever this is possible. In general, we define functions with names as used in the syntax of epkml to yield that part of a statement that their syntactical counterpart surrounds, say

template : $S \to S$

returns for a statement $s$ the text contained in the first occurring pairs of <template> and </template> in $s$, according to the block-structure in the lexicographic ordering.
The set of all epkml-variable names is called $N$.
The semantics works on states, elements of a set $\Sigma$ that are quintuples of a theme structure graph (from a set $G$), an environment (from $E$), a database instance (from $D$), the laid out elements (from $L$), and a memory store (from $M$):

$$\Sigma = G \times E \times D \times L \times M.$$

The memory store $M$ is a set of locations $C$, where a pair of an epkml-statement and an environment can be saved.

$$M = C \hookrightarrow S \times E$$

This is due to the static binding. An operation

new : $M \to C$

returns a new location for a given memory state, not in the domain of this memory state. The operations

stm : $C \to S$ and env : $C \to E$

return for a location the statement and the environment contained, respectively.
An environment is a list (a stack) of partial mappings from the variable names $N$ to locations,

$$E = (N \hookrightarrow C)^*$$

written as $[e_1, \ldots, e_n|\varepsilon]$ (where $e_1, \ldots, e_n$ are partial mappings and $\varepsilon$ another environment).. This is due to the block structure and the static binding as well. The application of an environment (seen as a partial function) to a variable name yields the value of the first (the uppermost) applicable mapping in the list for this variable. An environment may be extended by

$$\cup : E \times N \times C \to E$$

which manipulates the first mapping of the list or a value of an already defined variable may be changed by a recursive search in the stack:

$$\ddot{\cup} : E \times N \times C \to E.$$

The set of theme structure graphs $G$ comprises all lists of trees that are describable in `epkml`, the vertices of which are marked with a name, an `epkml`-statement (the contents of `<theme>` without the sub-themes), an environment, and a boolean value to mark a theme as the current or not.

$$G = (N \times (S \times E) \times \mathbf{B} + (N \times (S \times E) \times \mathbf{B}) \times G)^*$$

On these graphs,

$$\text{pos} : G \times N \to G$$

returns for a given graph and a name a new graph, with the boolean mark set only for the theme with the given name,

$$\text{nxt} : G \to N$$

returns the name of the vertex with the next theme.The function

$$\text{bld} : G \times N \times (S \times E) \times G^* \to G$$

constructs for a given graph, a mark, and a list of graphs, a new one that extends the first graph by a tree out of the mark and the list of graphs. As for variable names,

$$\text{env} : G \to E$$

returns for a theme structure graph the environment of the vertex that is marked as current.
The set of database instances $D$ contains lists of tuples according to the relevant database schemes. We do not model these schemes. We assume the required SQL-statements as being given.
Finally, the elements of the layout $L$ are sets of locations the contents of which is supposed to be visible on the screen.

$$L = \wp(C)$$

Besides these states, we model an external surrounding that supplies events (i.e. user interactions and timings) to trigger state transitions. These events are denoted as conditions. For a user interaction with an element $l$ we write $\iota : l$, for the expiration of a timer $\tau : t$ set to duration $t$, and for the end of a process $p$ (such as a video) $\eta : p$.

## 4.2 The Rules

We only state rules for so-called normalized `epkml`-programs. These are syntactically correct statements in which every attribute assignment `status=open` is substituted by `status=closed` and an `<open>`-statement, after the owning statement. Obviously, this is no restriction of the possible `epkml`-programs.
To take into account side effects of the interactive elements, we think of their variable changes as being permanent (the memory of their location is changed, too); we only partially state the necessary memory manipulations.
Every rule transforms a pair of an `epkml`-statement and a state either merely to a state, saying that the statement has been fully executed, or again to a pair of a statement and a state. This amounts to a small step semantics.
We do not provide a full semantics here. Only the control part is considered.

### 4.2.1 Syntactical Structure

The sequential composition is reflected by the usual structural rule.

$$\frac{(s, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (\gamma', \varepsilon', \delta', \lambda', \mu')}{(s\ s', (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (s', (\gamma', \varepsilon', \delta', \lambda', \mu'))}$$

### 4.2.2 Layout

Closed layout elements (i.e. `status=closed`) are treated much the same as variables, but opening their own binding scope via an artificial `<block>` tag. Here *layout* may have the values `window`, `frame`, `button` and so on.

$$(\texttt{<}\textit{layout}\ \texttt{name=}n\ \texttt{status=closed>}\ s\ \texttt{</}\textit{layout}\texttt{>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow$$
$$(\gamma, \varepsilon \cup \{n \mapsto \text{new}(\mu)\}, \delta, \lambda, \mu \cup \{\text{new}(\mu) \mapsto (\texttt{<block>}\ s\ \texttt{</block>}, \varepsilon)\})$$

$$(\texttt{<block>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (\gamma, [\emptyset|\varepsilon], \delta, \lambda, \mu)$$

$$(\texttt{</block>}, (\gamma, [\alpha|\varepsilon], \delta, \lambda, \mu)) \Longrightarrow (\gamma, \varepsilon, \delta, \lambda, \mu)$$

### 4.2.3 Database

The semantics of the database tag relies on an appropriate semantics of the corresponding SQL statements.

$$(\texttt{<sql result=}r\texttt{>}\ \texttt{SELECT}\ s\ \texttt{</sql>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow$$
$$(\gamma, \varepsilon, \delta, \lambda, \mu \cup \{r \mapsto [\![\texttt{SELECT}\ s]\!]_{\text{sql}}(\delta)\})$$

$$(\texttt{<sql> INSERT } s \texttt{ </sql>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (\gamma, \varepsilon, [\![\texttt{INSERT } s]\!]_{\text{sql}}(\delta), \lambda, \mu)$$

$$(\texttt{<sql> DELETE } s \texttt{ </sql>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (\gamma, \varepsilon, [\![\texttt{DELETE } s]\!]_{\text{sql}}(\delta), \lambda, \mu)$$

### 4.2.4 Themes

The theme structure graph is built up from bottom to top taking into account the scope of the theme names.

$$\frac{((\texttt{<theme name=}n_i\texttt{> } s_i \texttt{ </theme>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Rightarrow (\gamma_i, \varepsilon_i, \delta, \lambda, \mu_i))_{1 \le i \le k}}{\begin{array}{l}(\texttt{<theme name=}n\texttt{> } s \\ \quad \texttt{<theme name=}n_i\texttt{> } s_i \texttt{ </theme>})_{1 \le i \le k} \\ \texttt{</theme>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow \\ (\text{bld}(\gamma, n, s, \theta, (\gamma_i)_{1 \le i \le k}), \varepsilon \cup \{n \mapsto \text{new}(\mu)\}, \delta, \lambda, \mu \cup \{\text{new}(\mu) \mapsto (s, \varepsilon)\})\end{array}},$$

where $\theta$ is the set of all theme names in the theme structure graph. Note that every theme structure subgraph carries its own relevant environment information such that the environment changes in the antecedens of the rule need not be taken into account.

### 4.2.5 Control

$$(\texttt{<var name=}n\texttt{ value=}v\texttt{> </var>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (\gamma, \varepsilon \cup \{n \mapsto \text{new}(\mu)\}, \delta, \lambda, \mu \cup \{\text{new}(\mu) \mapsto (v, \varepsilon)\})$$

The contents of a variable may be treated as a statement with its own environment:

$$\frac{(\text{stm}(\varepsilon(n)), (\gamma, [\text{env}(\varepsilon(n))|\varepsilon], \delta, \lambda, \mu)) \Longrightarrow (\gamma', \varepsilon', \delta', \lambda', \mu')}{(\$n\$, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (\gamma', \varepsilon, \delta', \lambda', \mu')}$$

Thus implicitly a hiding block is opened.

$$(\texttt{<set name=}n\texttt{ value=}v\texttt{> </set>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (\gamma, \varepsilon, \delta, \lambda, \mu \,\ddot{\cup}\, \{n \mapsto (v, \varepsilon)\})$$

$$(\texttt{<set name=}n\texttt{> } s \texttt{ </set>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (\gamma, \varepsilon, \delta, \lambda, \mu \,\ddot{\cup}\, \{n \mapsto (s, \varepsilon)\})$$

The conditionals are dealt with conventionally:

$$(\texttt{<empty which=}n\texttt{> } s \texttt{ </empty>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (s, (\gamma, \varepsilon, \delta, \lambda, \mu)), \qquad \text{if } \varepsilon(n) = \emptyset$$

$$(\texttt{<empty which=}n\texttt{> } s \texttt{ </empty>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (\gamma, \varepsilon, \delta, \lambda, \mu), \qquad \text{if } \varepsilon(n) \neq \emptyset$$

$$(\texttt{<non-empty which=}n\texttt{> } s \texttt{ </non-empty>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (s, (\gamma, \varepsilon, \delta, \lambda, \mu)), \qquad \text{if } \varepsilon(n) \neq \emptyset$$

$$(\texttt{<non-empty which=}n\texttt{> } s \texttt{ </non-empty>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (\gamma, \varepsilon, \delta, \lambda, \mu), \qquad \text{if } \varepsilon(n) = \emptyset$$

For an <open> statement its attributes are evaluated first, especially its status attribute will be set to opened; we omit the details.

$$(\texttt{<open name=}n\ \{a_i \mapsto (s_i, \varepsilon)\}\texttt{>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (\text{stm}(\varepsilon(n)), (\gamma, [\{a_i \mapsto \text{new}(\mu)_i\} | \text{ env}(\varepsilon(n))], \delta, \lambda \cup \{\varepsilon(n)\}, \mu \cup \{\text{new}(\mu)_i \mapsto (s_i, \varepsilon)\}), \qquad \text{if } n \text{ is not a theme}$$

$$(\texttt{</open>}, (\gamma, [\alpha|\varepsilon], \delta, \lambda, \mu)) \Longrightarrow (\gamma, \varepsilon, \delta, \lambda, \mu)$$

$$(\texttt{<close name=}n\texttt{> </close>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (\gamma, \varepsilon, \delta, \lambda \setminus \{\varepsilon(n)\}, \mu \cup \{\mu(\varepsilon(n))\texttt{.status} \leftarrow \texttt{closed}\})$$

For themes, <open> has to be treated specially:

$$(\texttt{<open name=}n\texttt{> </open>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (\text{presentation}(\text{stm}(\varepsilon(n))), \text{pos}(\gamma, n), \text{env}(\gamma), \delta, \lambda, \mu)), \qquad \text{if } n \text{ is a theme}$$

$$(\texttt{<next theme> </next>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \Longrightarrow (\text{stm}(\varepsilon(\text{nxt}(\gamma))), \text{pos}(\gamma, \text{nxt}(\gamma), \text{env}(\varepsilon(\text{nxt}(\gamma))), \delta, \lambda, \mu))$$

The other navigation commands are evaluated alike.

When the user interacts with a layout element its action statement is executed in its own environment. Here the permanence of changes of the memory space (not those of the environment) is important. Note that for these external stimuli a new transition symbol is used.

$$\frac{(\mathrm{act}(\mathrm{stm}(l)), (\gamma, [\mathrm{env}(l)|\varepsilon], \delta, \lambda, \mu)) \Longrightarrow (\gamma', \varepsilon', \delta', \lambda', \mu')}{(\texttt{<wait> </wait>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \stackrel{\iota:l}{\Longrightarrow} (\texttt{<wait> </wait>}, (\gamma', \varepsilon, \delta', \lambda', \mu'))}$$

$$(\texttt{<wait end\_of=}t\texttt{> </wait>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \stackrel{\tau:t}{\Longrightarrow} (\gamma, \varepsilon, \delta, \lambda, \mu)$$

$$(\texttt{<wait end\_of=}p\texttt{> </wait>}, (\gamma, \varepsilon, \delta, \lambda, \mu)) \stackrel{\eta:p}{\Longrightarrow} (\gamma, \varepsilon, \delta, \lambda, \mu)$$

# 5 Conclusions and Further Steps

What we have achieved in this project is a comfortable specification language for both electronic product catalogues on CD-ROM and, in general, multimedia information systems. With the features and built-in services of the language `epkml` the development of a catalogue becomes an easy and inexpensive task for companies of any kind and size. The additional tools of the EPK-fix–project, namely RASSI, SASSI, GASSI, and TASSI, will improve further on this.

As specification of EPCs is not only of academic interest, technology transfer between universities and the industry is a declared aim of the EPK-fix–project. Industrial knowledge in the catalogue domain influenced this language design, since it provided, besides many other hints, standard features and layout considerations and our knowledge in specification shall influence catalogue design because we formalized and standardized the whole production process. Again, the collaboration between academic and non-academic institutions has proven invaluable for the development of innovative systems that can find their way to the market.

In the future, a lot of challenging goals in the field of EPC-development are to be worked on. First of all, the distribution of catalogues on the World Wide Web must not only be made possible but also as easy as the specification of a catalogue in `epkml` now is. For this purpose, a sound basis must be given through the definition of a client/server–model for database accesses, especially including security aspects. A translation to the—perhaps changing—standards of the Internet must be supported.

A further extension of `epkml` that we are currently working on is a simplified description and declaration of processes, in analogy to movie scripts. That could look like this:

```
<time-line end-of=12s periodical slice=3s>
  <slice>
    <par>
```

```
      <open name=m>
    <par>
      <open name=n>
  <slice>
    <par>
      <close name=m>
    <par>
      <close name=n>
  <slice end-of=1s>
    <open name=o>
  <slice end-of=5s>
    <close name=o>
</time-line>
```

Product and information integration seems to be another desirable aim. The production of a traditional paper catalogue out of an `epkml`–specification, e.g. by compiling the EPC to a word processor, will play a major rôle in the future developments.

# Acknowledgments

We wish to thank the EPK-fix developer groups of TH-Darmstadt, TU-Dresden and FORWISS-Erlangen as well as the people of the Mediatec Company for their helpful comments on the language. Also Earl Hood's `dtd2html`- and `MHonArc`-scripts were a great help.

# References

[AG96] Ken Arnold and James Gosling. *The Java Programming Language*. Addison–Wesley, Mountain View, May 1996.

[Bal95] Heide Balzert. *Methoden der objektorientierten Systemanalyse*. BI–Wissenschaftsverlag, Mannheim–Leipzig–Wien–Zürich, 1995.

[Bol94] Dietrich Boles. Das IMRA–modell. Diplomarbeit, Carl von Ossietzky Univerät Oldenburg, September 1994.

[Bos95] Bosch. Wir bewegen Ihre Welt. Bosch–Pneumatik — Das Komplettprogramm auf CD–ROM katalog nr. 15, May 1995. Made by telemedia interactive software (Bertelsmann).

[Dav90] Alan M. Davis. *Software Requirements*. Prentice Hall, Englewood Cliffs, N. J., 1990.

[Fla96] David Flanagan. *Java in a Nutshell*. O'Really & Associates, February 1996.

[GM95] James Gosling and Henry McGilton. *The Java Language Environment: A White Paper*. Sun Microsystems, Mountain View, October 1995.

[Gol94] Charles Goldfarb. *The SGML Handbook*. Clarendon Press, Oxford, 1994.

[Gra95] Ian S. Graham. *HTML Sourcebook*. John Wiley & Sons, New York–etc., 1995.

[KKMW96] Alexander Knapp, Nora Koch, Luis Mandel, and Martin Wirsing. Die Sprache EPKML. Interner Bericht 1:96, LMU München, March 1996.

[KM96] Nora Koch and Luis Mandel. Catalogues on CD-ROM: The State of the Art. to appear, Ludwig–Maximilians–Universität München, 1996.

[Mac95] Macromedia. Macromedia Showcase CD 4.0. CD-ROM, 1995. Made with Macromedia.

[Mer95] Mercedes-Benz AG. Die neuen E–Klasse Limousinen von Mercedes–Benz auf CD–ROM, 1995. Made with Macromedia.

[Mic95] Microsoft Corporation. Microsoft technet, Technical Information Network. CD-ROM, May 1995. Vol 3, Issue 5.

[MS93] Jim Melton and Alan R. Simon. *Understanding the new SQL*. Morgan Kaufmann, San Mateo, California, 1993.

[NN92] Hanne Riis Nielson and Flemming Nielson. *Semantics with Applications*. John Wiley & Sons, Chichester, 1992.

[OTT95] OTTO. Shopping Interactive. CD-ROM, 1995. Made by Feldmann.

[Pfl91] Shari Lawrence Pfleeger. *Software Engineering*. Macmillan, New York, 2nd edition, 1991.

[Que95] Quelle Schickedanz AG & Co. Easy Shopping per CD–ROM, 1995. Made with Macromedia.

[RB95] Steven A. Rogers and Mark A. Breland. Hypermedia Authoring - An Experiment, January 1995.

[RBP+91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object–Oriented Moodelling and Design*. Prentice Hall, Englewood Cliffs, N. J., 1991.

[RS ] RS Components GmbH. RS: Der Katalog. Das Original (september 95 – februar 96). CD-ROM. Made with Virtual Page.

[Spr95] Springer Verlag. Springer in Print 95/96. CD-ROM, 1995. Made with Adobe Acrobat.

[vH94] Arthur van Herwijnen. *Practical SGML*. Kluwer Academic, Boston–Dordrecht–London, 1994.

[vHSS96] Arthur van Hoff, Sami Shaio, and Orca Starbuck. *Hooked on Java*. Addison–Wesley, Reading, Massachusetts, 1996.

# A Example

In this section we present an example of the specification of an EPC. The functionality of the catalogue is graphically given by the figures 6 and 7.
It is a very simple EPC, which begins with a window of the form table-of-contents, from this window the end-user can choose between the following actions: navigate to the company's presentation or to the product selection, access to the help window, or exit the catalogue. The presentation is the most trivial one, because it only includes a video. The selection window gives a list of all the products available. If one product is clicked, a product window will be opened with an image and a brief description of the product. The "buy" button of this page allows to add the product in the shopping bag, which can be seen in a browser including name, description, quantity, and price. Notice that every window includes the possibility to return to the previous page with the "back" button. The help window is not graphically represented and the object identifications (oid)s are not included in the example, although they are required according to the DTD document because they will be automatically set by the SASSI tool.

```
<!DOCTYPE EPKML SYSTEM "epkml.dtd">
  <!-- Example Catalogue -->
  <epkml>
    <header
       title = Example Catalogue
       author = The LMU Software Corporation Ltd.
       date = 25/2/96
       last-modified = 29/2/96>
    <externals>
    <styles>
      <stylesheet name = example-catalogue-style
                extends = catalogue-style>
        <default>
          <window>
            <img src = logo.gif    align = "top left">
            <img src = example.gif align = "top right">
            <button name = help-button
                   align = "bottom right">
              <img src = help.gif>
              <on-click>
                   <open  name = help-window>
              </on-click>
            </button>
          </window>
        </default>
      </stylesheet>
```

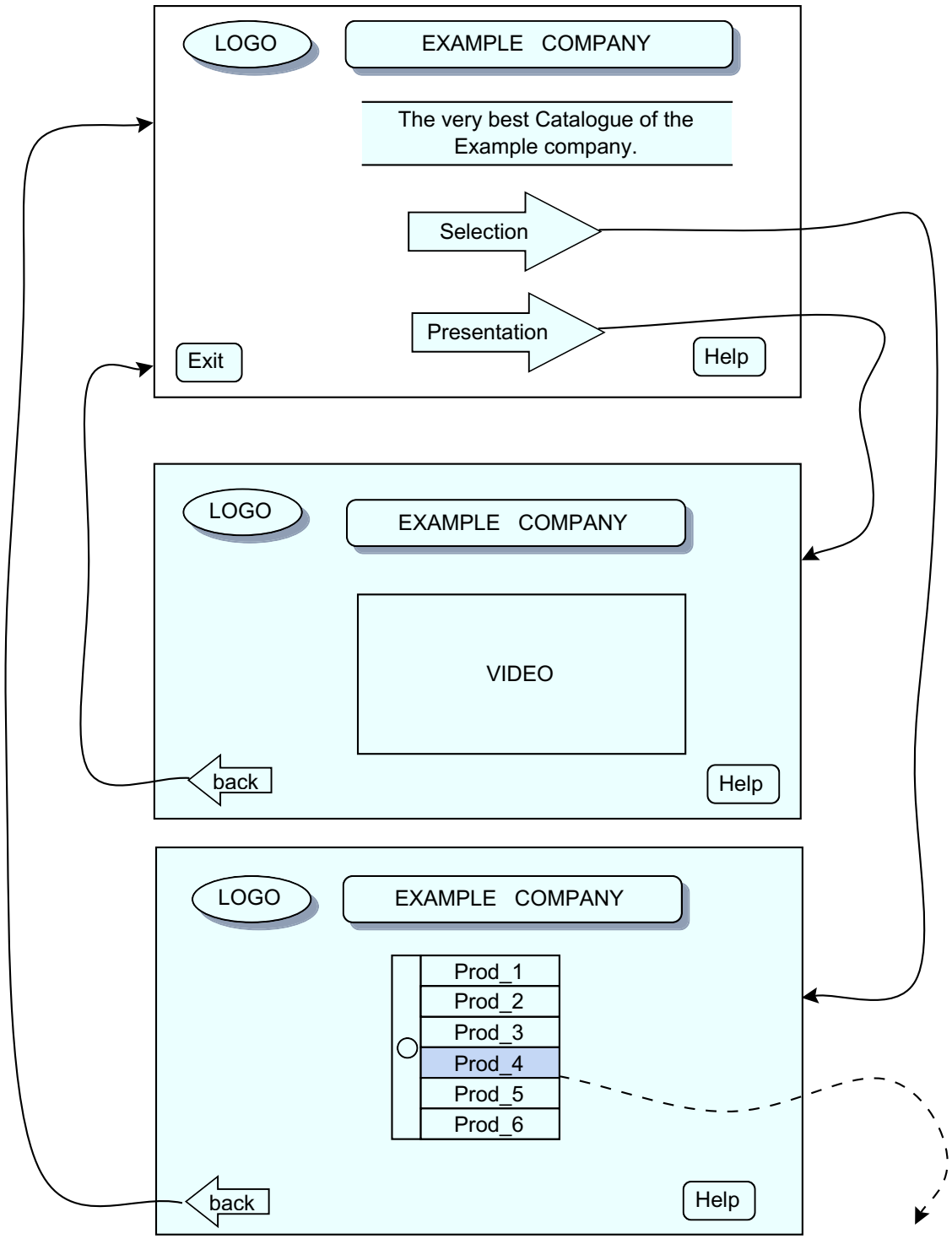


Figure 6: Example catalogue part I

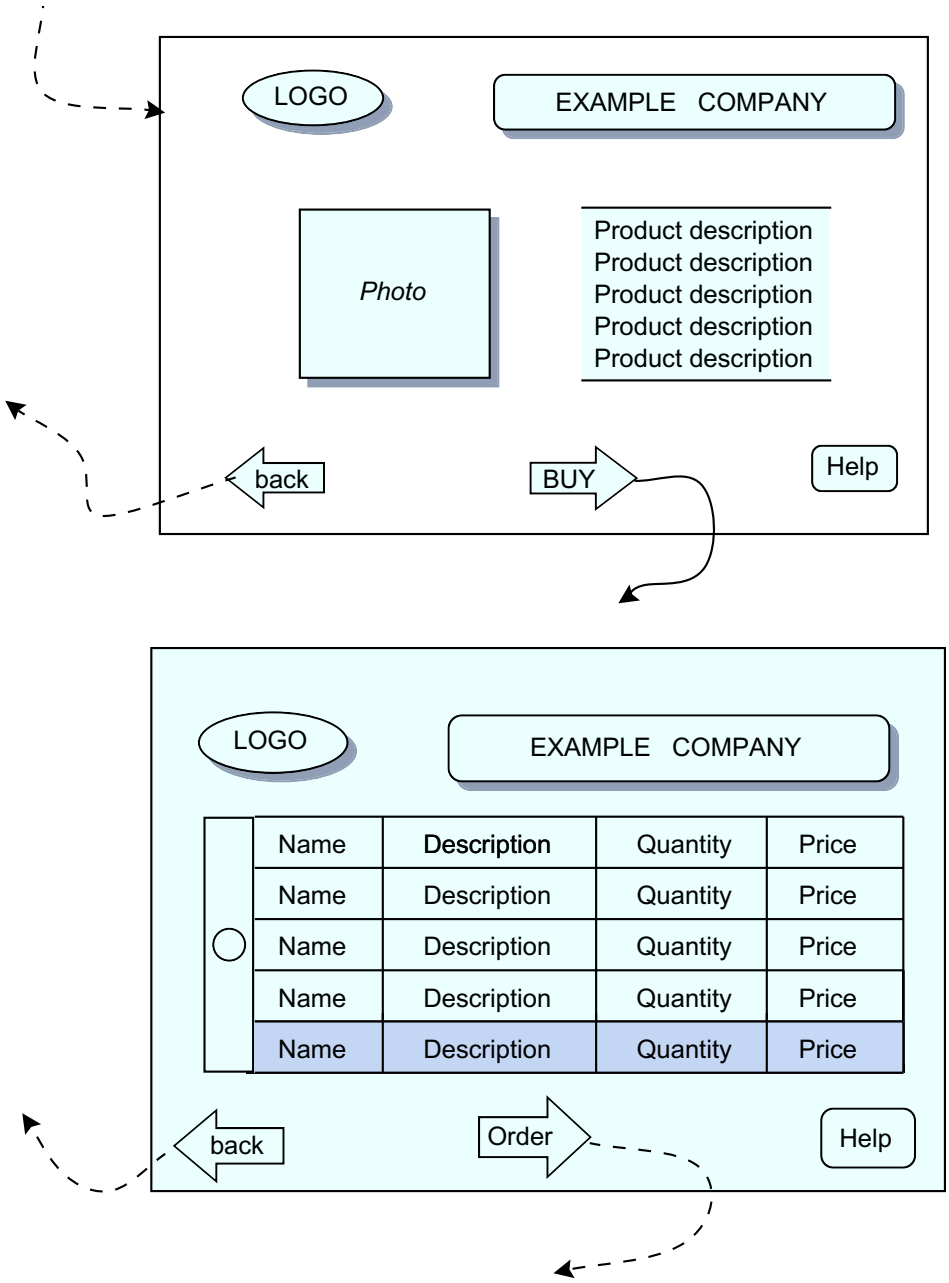


Figure 7: Example catalogue part II

```
  </styles>
  <definitions>
    <window name = main style = example-catalogue-style>
      <p align = left baselineskip = 10pt>
        The very best catalogue of the Example Company
      </p>
      <button name = selectbutton
              align = center>
        <img src = select.gif>
        <on-click>
           <close name = main>
           <open  name = select>
        </on-click>
      </button>
      <button name = presentationbutton
              align = center>
        <img src = presentation.gif
        <on-click>
          <close name = main>
          <open  name = presentation>
        </on-click>
      </button>
      <button name = exitbutton
              align = "bottom left">
        <img src = exit.gif
        <on-click>
          <close name = main>
         </on-click>
      </button>
    </window>
  </definitions>
  <include file = help.epk>
  <include file = presentation.epk>
  <include file = select.epk>
  <include file = product.epk>
  <include file = buy.epk>
  <main>
    <open name = main>
  </main>
</epkml>

<!-- Help File -->
<window name = help-window style = example-catalogue-style>
  <p>
    This is a help window
    This is a help window
    This is a help window
    This is a help window
    This is a help window
    <img src = help.gif>
    This is a help window
  </p>
  <button name = back align = "bottom left">
    <img src = back.gif>
    <on-click>
       <close name = help-window>
```

```
    </on-click>
  </button>
</window>

<!-- Presentation File -->
<window name = presentation
       style = example-catalogue-style>
  <video name = pres
         src = "/videos/summer-video.mov"
       format = mov
       status = closed>
    <stop-button>
      <frame name = video invisible>
      </frame>
      <on-click>
         <close name = pres>
      </on-click>
  </video>
  <button name = back align = "bottom left">
    <img src = back.gif>
    <on-click>
       <close name = presentation>
       <open  name = main>
    </on-click>
  </button>
  <open  name = pres>
  <on-end-of  name= pres>
       <close name = presentation>
       <open  name = main>
   </on-end-of>
</window>

<!-- Select File -->
<theme name = general>
  <extension result = general-result>
    <sql>
      SELECT name price description photo
      FROM   products description
      WHERE  products.code = description.code
    </sql>
  </extension>
  <page name = all-products>
    <window name = select style = example-catalogue-style>
      <browser name = dynamic-browser>
        <make-options from = general-result>
           <p>
             $general-result.name$
           </p>
          <on-selected>
             <close name = select>
             <open  name = prod-desc>
          </on-selected>
        </make-options>
      </browser>
      <button name = back align = "bottom left">
        <img src = back.gif>
```

```
        <on-click>
           <close name = select>
           <open  name = main>
        </on-click>
      </button>
    </window>
  </page>
  <exceptions>
    <sql>
      SELECT name price description photo
      FROM   products description
      WHERE  products.code = description.code and
              price < 100
    </sql>
    <page name = products-on-sale>
      <window name = on-sale style = example-catalogue-style>
        <p>
          Products ON-SALE!
          <img src=on-sale.gif>
        </p>
        <browser name = dynamic-browser>
          <make-options from = general-result>
           <p>
             $general-result.name$
           </p>
           <on-selected>
              <close name = select>
              <open  name = prod-desc>
            </on-selected>
          </make-options>
        </browser>
        <button name = back align = "bottom left">
          <img src = back.gif>
          <on-click>
            <close name = select>
            <open  name = main>
          </on-click>
        </button>
      </window>
    </page>
  </exceptions>
</theme>

<!--Product File-->
<window name = prod-desc
       style = example-catalogue-style
     attribs = "photo"
       elems = "desc">
  <img src = "$photo$" align = left>
  <p>
    $desc$
  </p>
  <button name = back align = "bottom left">
    <img src = back.gif>
    <on-click>
      <close name = prod-desc>
```

```
      <open  name = general>
    </on-click>
  </button>
  <button name = buy align = "bottom middle">
    <img src = buy.gif>
    <on-click>
      <close name = prod-desc>
      <sql>
        INSERT INTO shopping-bag
        VALUES
             $general-result[$dynamic-browser.selected$].name
             $general-result[$dynamic-browser.selected$].price
      </sql>
      <open name = buy>
    </on-click>
  </button>
</window>

<!--Buy File-->
   <window name = buy style = example-catalogue-style>
      <multiple-browser name = shopping-bag-browser>
        <make-options from = shopping-bag>
           <p>
             $general-result.name$
           </p>
         </make-options>
      </multiple-browser>
      <button name = order align = "bottom middle">
        <img src = order.gif>
        <on-click>
           <close name = buy>
           <open  name = order>
        </on-click>
      </button>
      <button name = back align = "bottom left">
        <img src = back.gif>
        <on-click>
          <close name = buy>
          <open  name = general>
       </on-click>
      </button>
    </window>
```

# B Document Type Definition

The Standard Generalization Markup Language (`sgml`) is the the ISO standard for document description. It is designed to enable text interchange and is used mainly in the publishing field. `sgml` documents have a rigorously described structure separating content from logical structure. Every `sgml` document has three parts, two formal parts: the `sgml` declaration and the Document Type Definition, called DTD, and the third part is the document instance.

The sgml declaration usually is common to all documents in an sgml installation and gives the details on how sgml will be applied to the document, defining which character set will be choosen or which characters should be used as delimiters. Examples of sgml declarations can be found in [vH94], [Gol94].
The DTD is written in sgml and defines the structure of the document. The instance contains the data and the mark up. An example for a document instance corresponding to the DTD defined in this appendix is shown in Appendix A.
In a DTD we can distinguish elements, attributes and entities. An element is marked up with symbols called start-tag and end-tag and it has associated a name called general identifier (GI). To define an sgml element its content, its attribute list, and the mark-up minimization rules must be given. The attribute list related to an element specifies further information and can be compared to the specification of parameters. Entities are used as a short form for text strings, to code special characters, to include external files, or as variables in the DTD.
The two books already mentioned [vH94] [Gol94] describe the complete sgml.

```
<!-- Document Type Definition for the EPKML -->
<!-- 04.12.1996 -->

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

<!-- Some definitions -->

<!ENTITY % IDENT "CDATA ''">
<!ENTITY % IDENTS "CDATA ''">
<!ENTITY % REF "NAME">
<!ENTITY % BOOL "NUMBER 0">
<!ENTITY % ALIGN "CDATA ''">
<!ENTITY % DIMEN "NUTOKEN 0">
<!ENTITY % TIME "NUTOKEN 0">
<!ENTITY % COLOR "NMTOKEN 'black'">
<!ENTITY % SHAPE "CDATA">
<!ENTITY % SOURCE "CDATA './'">
<!ENTITY % HEADING "NUMBER 1">

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

<!ENTITY % attributes "name %IDENT;
                       style %IDENTS;
                       invisible (invisible) #IMPLIED
                       layer NUMBER  0
                       xpos %DIMEN;
                       ypos %DIMEN;
                       width %DIMEN;
                       height %DIMEN;
                       lftpad %DIMEN;
                       rgtpad %DIMEN;
                       toppad %DIMEN;
                       botpad %DIMEN;
                       lftmrg %DIMEN;
                       rgtmrg %DIMEN;
                       topmrg %DIMEN;
                       botmrg %DIMEN;  ">

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

```
                        elems %IDENTS; ">

<!ENTITY % halign "(left | center | right | justify | decimal) left" >
<!ENTITY % valign "(top | middle | bottom | baseline) baseline">

<!ENTITY % img-format   "format CDATA 'rgb'">
<!ENTITY % audio-format "format CDATA 'au'">
<!ENTITY % video-format "format CDATA 'avi'">
<!ENTITY % slide-format "format CDATA 'avi'">
<!ENTITY % end-of       "end-of CDATA ''">

<!ENTITY % font        "font       CDATA 'helvetica'">
<!ENTITY % fontstyle   "fontstyle CDATA 'normal'         ">

<!ENTITY % def-lang        "'german'">
<!ENTITY % def-background  "'grey'">
<!ENTITY % def-bulletstyle "'*'">
<!ENTITY % def-enumstyle   "'arabic'">
<!ENTITY % begin-number "1">
<!ENTITY % def-maxlength    "1">
<!ENTITY % def-colspec "'right'">
<!ENTITY % def-numofpics "1">
<!ENTITY % def-picpersec "16">

<!ENTITY % void "''">

<!ENTITY % defaults  "display   CDATA  '640x480'
                      units     CDATA  'pixel'
                      user-mode NAME   'client'">

<!ENTITY % ISOlat1 PUBLIC
  "ISO 8879-1986//ENTITIES Added Latin 1//EN//HTML"
  "added-iso-latin-1.dtd">
%ISOlat1;


<!-- Inclusion of files and macros -->

<!ELEMENT include - O EMPTY>
<!ATTLIST include
  %oid;
  file CDATA #REQUIRED>

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

<!ELEMENT  variant  - -  (and | or | element | variant)*>
<!ATTLIST  variant
  %oid;
  name      CDATA %void
  object    CDATA #REQUIRED
  attribute CDATA %void
  value     CDATA %void>

<!ENTITY  % operator "and | or">

<!ELEMENT (%operator;) - O (and | or | element)*>
<!ATTLIST  (%operator;)
  %oid;
  name      CDATA %void
  object    CDATA %void
  attribute CDATA %void
```

```
  value     CDATA %void>

<!ELEMENT attribute - O EMPTY>
<!ATTLIST attribute
  %oid;
  name CDATA #REQUIRED
  value CDATA #REQUIRED>

<!ELEMENT element - O (#PCDATA)>
<!ATTLIST element
  %oid;
  name CDATA #REQUIRED>

<!-- Declaration of layout elements -->

<!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 | demo |
                     registration-form | question-form |
                     search-form | shopping-bag | shopping-list |
                     table-of-contents">

<!-- Declaration of actions -->

<!ENTITY % open "open">
<!ENTITY % close "suspend | close">
<!ENTITY % database "sql">
<!ENTITY % navigation "next | previous | up | down | back |
                       additional | exit">
<!ENTITY % action "%open; | %close; | %database; | %navigation; |
                   audio | applet | set | wait | empty | non-empty |
                   foreach | order | time-line | on-end-of">

<!-- Declaration of contents of elements -->

<!ENTITY % contents "%flow; | %interactive; | %toplevel; | %action;">

<!-- Actions -->

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

<!ELEMENT (%close;) - O EMPTY>
<!ATTLIST (%close;)
  %oid;
  name CDATA #REQUIRED>

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

<!ELEMENT (%navigation;) - O EMPTY>
```

```
<!ATTLIST (previous | next)
  %oid;
  theme (theme) #IMPLIED
  circular (circular) #IMPLIED>

<!ATTLIST back
  %oid;
  hierarchical (hierarchical) #IMPLIED>

<!ATTLIST exit
  %oid;
  demo (demo) #IMPLIED>

<!ELEMENT applet - - (param)*>
<!ATTLIST applet
  %oid;
  %attributes;
  %properties;
  function CDATA #REQUIRED
  result %IDENT; >

<!ELEMENT param - O EMPTY>
<!ATTLIST param
  %oid;
  name CDATA   #REQUIRED
  value CDATA #REQUIRED>

<!ELEMENT set - O ANY>
<!ATTLIST set
  %oid;
  name CDATA #REQUIRED
  value CDATA %void;>

<!ELEMENT wait - O EMPTY>
<!ATTLIST wait
  %oid;
  %end-of;>

<!ELEMENT (empty | non-empty) - O (%contents;)*>
<!ATTLIST empty
  %oid;
  which CDATA #REQUIRED>

<!ELEMENT foreach - O (%contents;)+>
<!ATTLIST foreach
  %oid;
  in CDATA #REQUIRED>

<!ELEMENT order - O EMPTY>
<!ATTLIST order
  %oid;
  orders CDATA %void;>

<!ELEMENT time-line - - (slice)+>
<!ATTLIST time-line
  %oid;
  %end-of;
  periodical (periodical) #IMPLIED
  slice %TIME;                  >

<!ELEMENT slice - O ((%contents;)+ | (par)+)>
<!ATTLIST slice
  %end-of;
  periodical (periodical) #IMPLIED>
```

```
<!ELEMENT par - O (%contents;)*>

<!ELEMENT on-end-of - - (%action;)*>
<!ATTLIST on-end-of
  %oid;
  name CDATA #REQUIRED>


<!-- Text elements -->

<!ENTITY % fonts "u | b | s | i | tt | big | small">
<!ENTITY % phrases "em | strong">
<!ENTITY % positionings "sub | sup">
<!ENTITY % specials "br">
<!ENTITY % misc "q | lang">
<!ENTITY % text "#PCDATA | %fonts; | font | %positionings; | %phrases; |
                 %specials; | %misc;">
<!ENTITY % fontprops "%font;
                      fontsize %DIMEN;
                      %fontstyle;
                      fontcolor %COLOR; ">

<!ELEMENT (%fonts; | %phrases;) - - (%flow;)+>
<!ATTLIST (%fonts; | %phrases;)
  %oid;
  %properties;>

<!ELEMENT (%positionings;) - - (%flow;)+>
<!ATTLIST (%positionings;)
  %oid;
  %properties;
  distance %DIMEN; >

<!ELEMENT br - O EMPTY>
<!ATTLIST br
  %oid;
  %properties;>

<!ELEMENT q - O (%flow; | %text;)+>
<!ATTLIST q
  %oid;
  %properties;>

<!ELEMENT lang - O (%flow; | %text;)+>
<!ATTLIST lang
  %oid;
  %properties;
  name NAME %def-lang;>

<!ELEMENT font - O (%flow; | %text;)+>
<!ATTLIST font
  %oid;
  %properties;
  %fontprops;>

<!ELEMENT p - O (%interactive; | %flow; | %text;)*>
<!ATTLIST p
  %oid;
  %attributes;
  %properties;
  %fontprops;
  baselineskip %DIMEN;
```

```
  indent %DIMEN;
  align %halign;>

<!ELEMENT heading - - (%interactive; | %flow; | %text;)*>
<!ATTLIST heading
  %oid;
  %attributes;
  %properties;
  %fontprops;
  number %HEADING;
  baselineskip %DIMEN;
  leftmargin %DIMEN;
  align %halign;>

<!ENTITY % listprops "leftmargin %DIMEN;
                      itemsep    %DIMEN;
                      align      %halign;">

<!ELEMENT listing - - ((term?, item) | make-items)*>
<!ATTLIST listing
  %oid;
  %attributes;
  %properties;
  %listprops;>

<!ELEMENT itemize - - (term?, (item | make-items)*)>
<!ATTLIST itemize
  %oid;
  %attributes;
  %properties;
  %listprops;
  bulletstyle CDATA %def-bulletstyle>

<!ELEMENT enumerate - - (item | make-items)*>
<!ATTLIST enumerate
  %oid;
  %attributes;
  %properties;
  %listprops;
  enumstyle CDATA %def-enumstyle
  number CDATA %begin-number;>

<!ELEMENT term - O (%flow;)+>
<!ATTLIST term
  %oid;
  %attributes;
  %properties;>

<!ELEMENT item - O (%flow;)+>
<!ATTLIST item
  %oid;
  %attributes;
  %properties;>

<!ELEMENT make-items - O (%flow;)+>
<!ATTLIST make-items
  from CDATA #REQUIRED>

<!ELEMENT tabular - - (row)*>
<!ATTLIST tabular
  %oid;
  %attributes;
  %properties;
  colspec CDATA %def-colspec;
```

```
  align %halign;>

<!ELEMENT row - O (cell)*>
<!ATTLIST row
  %attributes;
  %properties;
  line (line) #IMPLIED>

<!ELEMENT cell - O (%flow;)*>
<!ATTLIST cell
  %attributes;
  %properties;
  colspan NUMBER 1
  rowspan NUMBER 1
  align %ALIGN; >

<!-- Images, Video, Audio and Effects -->

<!ELEMENT img - O EMPTY>
<!ATTLIST img
  %oid;
  %attributes;
  %properties;
  %img-format;
  align %ALIGN;
  src CDATA #REQUIRED>

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

<!ELEMENT video - O (%cntrlbtns;)*>
<!ATTLIST video
  %oid;
  %attributes;
  %properties;
  %video-format;
  numofpics NUMBER %def-numofpics;
  picpersec NUMBER %def-picpersec;
  align %ALIGN;
  src CDATA #REQUIRED>

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

<!ELEMENT slide-show - O (%cntrlbtns;)*>
<!ATTLIST slide-show
  %oid;
  %attributes;
  %properties;
  %slide-format;
  interval %TIME;
  align %ALIGN;
  src CDATA #REQUIRED>

<!ELEMENT demo - - (%action;)* +(click)>
<!ATTLIST demo
  %oid;>
```

```
<!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>

<!-- Buttons and Hyperlinks -->

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

<!ELEMENT (next-button | previous-button)
                 - - (disabled?, clicked?, (%flow;)*, on-click?)?>
<!ATTLIST (next-button | previous-button)
  %oid;
  %attributes;
  %properties;
  disabled (disabled) #IMPLIED
  align %ALIGN;
  circular (circular) #IMPLIED
  theme (theme) #IMPLIED>

<!ELEMENT back-button - - (disabled?, clicked?, (%flow;)*, on-click?)?>
<!ATTLIST back-button
  %oid;
  %attributes;
  %properties;
  hierarchical (hierarchical) #IMPLIED
  disabled (disabled) #IMPLIED
  align %ALIGN;>

<!ELEMENT (disabled | clicked) - - (%flow;)+>

<!ELEMENT on-click - - (%action;)*>

<!ELEMENT hyperlink - - (%flow;)*>
<!ATTLIST hyperlink
  %oid;
  %attributes;
  %properties;
  ref CDATA #REQUIRED
  disabled (disabled) #IMPLIED
  align %ALIGN;>

<!-- Sliders -->

<!ENTITY % slider "vertical-slider | horizontal-slider">

<!ELEMENT (%slider;) - O (slider-previous?, slider-next?,
                          slider-box?, on-reposition?)>
<!ATTLIST (%slider;)
  %oid;
  %attributes;
```

```
  %properties;
  position NUMBER 0
  step NUMBER 1
  disabled (disabled) #IMPLIED
  align %ALIGN;>

<!ELEMENT (slider-previous | slider-next | slider-box) O O (%flow;)*>

<!ELEMENT on-reposition - - (%action;)*>

<!-- Form Elements -->

<!ELEMENT input - O EMPTY>
<!ATTLIST input
  %oid;
  %attributes;
  %properties;
  maxlength NUMBER %def-maxlength;
  disabled (disabled) #IMPLIED
  value         CDATA           #REQUIRED
  align %ALIGN;>

<!ELEMENT scribble - O EMPTY>
<!ATTLIST scribble
  %oid;
  %attributes;
  %properties;
  src %SOURCE;
  disabled (disabled) #IMPLIED
  align %ALIGN;>

<!ELEMENT browser - - (option | make-options)* +(on-no-option)>
<!ATTLIST browser
  %oid;
  %attributes;
  %properties;
  disabled (disabled) #IMPLIED
  align %ALIGN;>

<!ELEMENT multiple-browser - - (browser-row | make-options)* +(on-no-option)>
<!ATTLIST multiple-browser
  %oid;
  %attributes;
  %properties;
  colspec CDATA %def-colspec;
  align %ALIGN;>

<!ELEMENT browser-row - O (browser-cell, on-selected?, on-deselected?)*>
<!ATTLIST browser-row
  %attributes;
  %properties;
  line (line) #IMPLIED
  align %ALIGN;>

<!ELEMENT browser-cell - O (%flow;)*>
<!ATTLIST browser-cell
  %attributes;
  %properties;
  input (input) #IMPLIED
  colspan NUMBER 1
  rowspan NUMBER 1
  align %ALIGN;>

<!ELEMENT checkbox - - (option*)>
```

```
<!ATTLIST checkbox
  %oid;
  %attributes;
  %properties;
  disabled (disabled) #IMPLIED
  align %ALIGN;>

<!ELEMENT radio-button - - (option*)>
<!ATTLIST radio-button
  %oid;
  %attributes;
  %properties;
  disabled (disabled) #IMPLIED
  align %ALIGN;>

<!ELEMENT pop-up - - (option*) -(on-deselected)>
<!ATTLIST pop-up
  %oid;
  %attributes;
  %properties;
  disabled (disabled) #IMPLIED
  align %ALIGN;>

<!ELEMENT pull-down - - ((menu-title)*)>
<!ATTLIST pull-down
  %oid;
  %attributes;
  %properties;
  align %ALIGN;>

<!ELEMENT menu-title - - (option)* -(on-deselected)>
<!ATTLIST menu-title
  %attributes;
  %properties;
  disabled (disabled) #IMPLIED
  align %ALIGN;>

<!ELEMENT option - O ((%flow;)+, on-selected?, on-deselected?)>
<!ATTLIST option
  %attributes;
  %properties;
  selected (selected) #IMPLIED
  disabled (disabled) #IMPLIED
  align %ALIGN;>

<!ELEMENT make-options - O ((%flow;)+, on-selected?, on-deselected?)>
<!ATTLIST make-options
  from CDATA #REQUIRED>

<!ELEMENT (on-selected | on-deselected | on-no-option) - - (%action;)*>

<!-- Boxing -->

<!ELEMENT flowbox - - (around?, (%flow; | %interactive; | %action;)*)>
<!ATTLIST flowbox
  %oid;
  %attributes;
  %properties;
  align %ALIGN;
  distribute (distribute) #IMPLIED
  background CDATA %def-background;>

<!ELEMENT around - - (%flow; | %interactive;)+>
<!ATTLIST around
```

```
  align %ALIGN;>

<!-- Frames and Windows -->

<!ELEMENT frame - - (%flow; | %interactive; | %action;)*>
<!ATTLIST frame
  %oid;
  %attributes;
  %properties;
  align %ALIGN;
  background CDATA %def-background;>

<!ELEMENT window - - (%flow; | %interactive; | %action;)*>
<!ATTLIST window
  %oid;
  %attributes;
  %properties;
  title CDATA %void;
  iconized (iconized) #IMPLIED
  background CDATA %def-background;>

<!-- Themes -->

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

<!ELEMENT extension - - (sql)*>
<!ATTLIST extension
  %oid;
  result %IDENT;>

<!ELEMENT page - - (%flow; | window | %interactive; | %action; | page)*>
<!ATTLIST page
  %oid;
  name CDATA #REQUIRED>

<!ELEMENT exceptions - - (sql, page?)+>
<!ATTLIST exceptions
  %oid;>

<!-- Services -->

<!ELEMENT table-of-contents - - (%flow; | %interactive; | %action;)*>
<!ATTLIST table-of-contents
  %oid;
  name %IDENT; >

<!ELEMENT registration-form - - (%flow; | %interactive; | %action;)*>
<!ATTLIST registration-form
  %oid;
  name %IDENT;
  users %SOURCE;>

<!ELEMENT question-form - - (%flow; | %interactive; | %action;)*>
<!ATTLIST question-form
  %oid;
  name %IDENT;
  users %SOURCE;>

<!ELEMENT search-form - - (%flow; | %interactive; | %action;)*>
<!ATTLIST search-form
  %oid;
```

```
  name %IDENT;
  entries %SOURCE;>

<!ELEMENT (shopping-bag | shopping-list) - - (%flow; | %interactive; |
                                              %action;)*>
<!ATTLIST (shopping-bag | shopping-list)
  %oid;
  name %IDENT;
  orders %SOURCE;>

<!-- Header -->

<!ELEMENT header - O EMPTY>
<!ATTLIST header
  title          CDATA          #REQUIRED
  author         CDATA          #REQUIRED
  date           CDATA          #REQUIRED
  last-modified  CDATA          #REQUIRED>

<!-- Externals -->

<!ELEMENT externals - O (class | scheme)*>

<!ELEMENT class - O EMPTY>
<!ATTLIST class
  %oid;
  name CDATA #REQUIRED
  slots CDATA %void;
  methods CDATA %void;>

<!ELEMENT scheme - - (table)+>
<!ATTLIST scheme
  %oid;
  name CDATA #REQUIRED>

<!ELEMENT table - O EMPTY>
<!ATTLIST table
  %oid;
  name CDATA #REQUIRED
  columns CDATA #REQUIRED>

<!-- Styles -->

<!ELEMENT styles - O (stylesheet)*>

<!ELEMENT stylesheet - - (default)*>
<!ATTLIST stylesheet
  %oid;
  name CDATA #REQUIRED
  extends %IDENTS;>

<!ELEMENT default - O (%flow; | %interactive; | %toplevel;)>
<!ATTLIST default
  %oid;
  extends %IDENTS;>

<!-- Definitions -->

<!ELEMENT definitions - O (macro | audio | theme | %flow; |
                           %interactive; | %toplevel;)* +(var)>
<!ATTLIST definitions
  %oid;>

<!ELEMENT var - O ANY>
```

```
<!ATTLIST var
  %oid;
  name CDATA #REQUIRED
  value CDATA %void;>

<!ELEMENT macro - - ANY>
<!ATTLIST macro
  %oid;
  name CDATA #REQUIRED
  attribs %IDENTS;
  elems %IDENTS;>

<!-- The Main Element -->

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