EPKML: Eine Spezifikationssprache für elektronische Produktkataloge*

Alexander Knapp, Nora Koch

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

{knapp,kochn}@informatik.uni-muenchen.de

15. Dezember 1997

Zusammenfassung

Elektronische Produktkataloge sind Softwaresysteme, die eine geeignete Entwicklungsmethodik erfordern. Als Grundlage eines solchen gesamten Entwicklungsprozesses wurde die Spezifikationssprache EpkMl definiert, die eine formale Beschreibung des multimedialen Layouts und der temporalen und navigatorischen Aspekte elektronischer Produktkataloge erlaubt. Produkte werden in einer hierarchischen Struktur angeordnet, innerhalb derer es möglich ist, einheitliche Darstellungen zuzuweisen. Anhand dieser Strukturierung werden die Navigationsmöglichkeiten automatisch bereitgestellt.

Abstract

Electronic product catalogues are software systems that require an appropriate development methodology. The specification language EpkMl was defined as basis for the whole development process. EpkMl permits a formal description of the multimedia layout and the temporal and navigational aspects of electronic product catalogues. Products are organised in hierarchical structures allowing automatic generation of navigation through the structure and an unified presentation of the products.

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

*Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung, Wissenschaft, Forschung und Technologie unter dem Förderkennzeichen 01 IS 520D/7 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

Inhaltsverzeichnis

1 Einleitung

Bei der Entwicklung von elektronischen Produktkatalogen (EPKen) hat die kreative Gestaltung neben der Erstellung der Software einen hohen Stellenwert. Das kreative Design ist ein hochgradig iterativer Arbeitsprozeß, in dem unterschiedliche Varianten bewußt zur audiovisuellen Vermittlung von Arbeitsergebnissen in den verschiedenen Stadien des Enwicklungsprozesses eingesetzt werden. Selektion, Elaboration und Kombination von Gestaltungsvarianten sind dabei typische Arbeitsschritte.

Das Ziel des EPK-fix-Projektes ist die effiziente Erstellung und Pflege von EPKen. Das Projekt setzt an der Wurzel der Probleme – der fehlenden formalen, methodischen und werkzeugtechnischen Unterstützung des gesamten Entwicklungsprozesses von EPKen – an.

Hierzu sind folgende Arbeiten geleistet worden: Erarbeitung und Validierung einer Entwicklungsmethodik für EPKe, Entwicklung einer formalen Spezifikationssprache (EpkMl) für EPK-Software, sowie die Entwicklung von fünf Assistenzsystemen (RASSI, SASSI, GASSI, TASSI und OASSI), die einen verteilten Entwicklungsprozeß eines EPKs in allen Phasen unterstützen [5].

Grundlage des gesamten Entwicklungsprozesses und verbindendes Element der Werkzeuge ist die formale Spezifikationssprache EpkMl (EPK Markup Language) [4], die die multimedialen, temporalen und navigatorischen Aspekte elektronischer Produktkataloge berücksichtigt und eine Ablaufautomatisierung erlaubt.

Die Ergebnisse einer ausführlichen Analysephase, die in [7, 8] im Detail beschrieben wird, dienten als Basis zur Festlegung der folgenden Anforderungen, die eine Entwicklungssprache für elektronische Produktkataloge erfüllen muß:

- Unterstützung multimedialer Elemente,
- Einführung einer Darstellungssprache, um das Layout des Kataloges zu beschreiben,
- Inkorporation von Kontrollkonstrukten, um die Navigation durch den Katalog zu ermöglichen,
- einfache Handhabung von Standardsituationen, die in fast jedem Produktkatalog vorkommen,
- einfache Erlernbarkeit,
- gute Implementierbarkeit und
- gute Testbarkeit.

Ausschlaggebend bei der Entscheidung des Spezifikationssprachtyps war die simultane Unterstützung der Erstellung von Produktkatalogen in Papierversion sowie in elektronischer Form auf CD-ROM und on-line ([9, 10]). Die im World Wide Web (WWW) verfügbaren Produktkataloge sind ähnlich den CD-ROM–Produktkatalogen strukturiert, allerdings bieten sie geringere Interaktionsmöglichkeiten und müssen auf einige multimedialen Effekte des Zeitaufwands wegen verzichten. Information für das WWW wird als

Hypertext-Dokument verfaßt, geschrieben in HtMl (HyperText Markup Language) [1]. Diese Technologie wurde für EpkMl übernommen, aber hier um die fehlenden multimedialen und temporalen Aspekte erweitert.

Unsere Entwurfssprache EpkMl hat eine deklarative Syntax und ist, wie HtMl, eine Instanz der Spezifikationssprache SgMl (Standard Generalized Markup Language)[12]. Außerdem bezieht EpkMl Ideen aus Java, FrameMaker und TEX ein und ist fenster- und nicht seitenorientiert. Der wichtigen Aufgabenstellung der Automatisierbarkeit liegt ein Produktgruppenkonzept zugrunde, das zur automatischen Navigation ausgenutzt wird. Die Produkte werden dabei in Themen und Unterthemen eingeteilt; Produkten desselben Themas wird eine einheitliche Darstellung zugewiesen, wobei Ausnahmen möglich sind und explizit unterstützt werden. Anhand dieser Strukturierung werden die Navigationsmöglichkeiten – zu Unterthemen innerhalb eines Themas – automatisch bereitgestellt. Diese automatisch erzeugte Hierarchie kann manuell verfeinert und verbessert werden.

Überblick: Im Abschnitt 2 werden die allgemeinen Spracheigenschaften beschrieben. Im Abschnitt 3 wird die Syntax der Spezifikationssprache EpkMl vorgestellt und an Beispielen erläutert während der Abschnitt 4 die ausgearbeitete Semantik beschreibt. Abschnitt 5 gibt einen Ausblick.

2 Eigenschaften der Sprache EpkMl

Die Entscheidung für einen Sprachtyp wurde vom Bedarf einer leicht zu erlernenden und erweiterbaren sowie für on-line Kataloge verwendbaren Sprache geprägt. Eine zusätzliche Bedingung war die Integration von Datenbankzugang und Navigationsaspekten. Die Sprache EpkMl stellt alle notwendigen Konstrukte bereit, um elektronische Produktkataloge in einer einheitlichen Umgebung rasch und effizient entwickeln zu können [5, 11].

Die Syntax ist intuitiv, zur weiteren Vereinfachung und den Automatisierungsprozeß unterstützend gedacht und gestaltet. Als automatisierbar werden dabei alle Layout–Teile angesehen, ein kleiner Teil der Regie und speziell die Dienste (services); die Datenbank wird als (eventuell remodelliert) gegeben betrachtet, auf sie wird durch SQL–Anweisungen zugegriffen. Die wichtigsten Merkmale (auch in [4, 3] ausführlich beschrieben) der entworfenen Spezikationssprache können wir folgendermaßen zusammenfassen:

- EpkMl *ist* HtMl-*ähnlich.* Viele Elemente der Sprache HtMl zur Layoutbeschreibung wurden übernommen. Die Spezifikationssprache wurde als eine Instanz von SgMl (ISO-standard 8879) definiert.
- EpkMl *bettet SQL-Statements ein.* Die Standard Query Language (SQL) ist in der Sprache integriert, um den Datenbankzugriff zu gewährleisten.
- EpkMl *beinhaltet Basiselemente für den Kontrollfluß.* Zwar sind nur ein paar Elemente zur Steuerung des Kontrollflusses vorgesehen, sie sind aber für das spezielle Verhalten eines elektronisches Produktkataloges ausreichend.

- EpkMl *erlaubt Anbindung an externe Programmiersprachen.* Die Anbindung ist mittels Applets wie in HtMl vorgesehen.

Die Sprache erfüllt die gestellten Anforderungen insbesondere auf eine deklarative Beschreibung eines elektronischen Produktkataloges wie folgt:

- EpkMl *läßt eine hierarchische Organisation von Themen zu.* Die Produkte werden in Hierarchien organisiert, die Themen genannt werden. Für jedes Thema kann der Katalogentwickler die Produkte angeben, die zu diesem Thema gehören, sowie die visuelle Präsentation und Unterthemen beschreiben.
- EpkMl *erlaubt automatisches Navigieren.* Die Themen werden zur automatischen Generierung der Navigation verwendet. Das bedeutet, daß der Benutzer auch durch den Katalog geführt werden kann.
- EpkMl *bietet spezielle Dienste an.* Die Hauptfunktionalitäten eines EPKs, wie Bestellung, Anmeldung oder Warenkorb- und Bestellungsverwaltung sind als vordefinierte Elemente in die Sprache integriert.
- EpkMl *liefert Multimedia-Elemente.* Multimediale Objekte sind in der Sprache so integriert, daß der Benutzer sich nur auf ihren deklarativen Aspekt konzentrieren muß.

3 EpkMl-Syntax

Nur eine kurze informelle Beschreibung der wichtigsten Elemente ist im Rahmen dieses Berichtes möglich. Wir konzentrieren uns auf die Aspekte, die EpkMl von HtMl unterscheiden. Außerdem beschränken wir uns in der Beschreibung auf die Syntax und Semantik der Elemente der Sprache, ohne auf die Attribute der Elemente näher einzugehen.

Großer Wert wurde auf die Strukturierungsmöglichkeiten des Kataloges gelegt. Gemäß dem eingangs erwähnten Automatisierungskonzept wird eine Gliederung in Themen (entspricht `Theme`) ermöglicht, denen einerseits Unterthemen und andererseits Produkte in einheitlicher Darstellung — über sogenannte Vorlagen oder `Templates` zugewiesen werden. Innerhalb dieser Struktur kann über Standardfunktionen (zum nächsten, zum vorherigen Produkt; zum nächsten, zum vorherigen Thema, etc.) navigiert werden.

Im Vergleich zu einem objekt-orientierten Ansatz erweist sich die fehlende Subklassenbildung und Objektkreierung neben dem Fehlen operationeller Möglichkeiten als gewisser Mangel des SgMl–Ansatzes. Wir imitieren Aggregationen in EpkMl durch Makros (die, technisch gesprochen, eine Erweiterung der *Document type definition* innerhalb von SgMl ermöglichen), bieten variable Inhalte von Rahmen, Fenstern, usw., sowie dynamische Formulare und Einbindung externer, d. h. nicht in EpkMl formulierbarer Funktionalität.

3.1 Document Type Definition

EpkMl als eine Instanz von SgMl ist in einer *Document type definition* (DTD) beschrieben und verwendet *mark-up tags*. Jeder Block beginnt mit `<Name-des-Tags>` und ended

mit </Name-des-Tags>. Für einige Elemente ist der abschließende Tag eines Blockes optional (wird mit - O in der DTD spezifiziert). Für jedes Element wird der Inhalt definiert, (wobei der Inhalt optional (mit ? gekennzeichnet), mindestens einmal (+) oder beliebig oft (*) vorkommen kann). Altenativen für Elemente werden durch |gekennzeichnet, durch Kommas wird die Reihenfolge der Elemente festgelegt.

Einige Definitionen der DTD für die Sprache EpkMl werden unten exemplarisch aufgelistet.

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

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

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

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

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

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

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

<!ENTITY % open "open | redraw">
<!ENTITY % close "suspend | close ">
<!ENTITY % database "sql">
<!ENTITY % navigation "next | previous | up | down | back |
                       additional | exit">
<!ENTITY % contents "%flow; | %interactive; | %toplevel; |
%services; | %action;">

<!ELEMENT (%open;) - O (attribute | element)*>
<!ATTLIST (%open;)
```

```
  %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>

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

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

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

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

<!ELEMENT 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>

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

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

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

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

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

<!ELEMENT exceptions - - ((sql, template)+,exceptions*)>
<!ATTLIST exceptions
  %oid;>

<!ELEMENT styles - O (stylesheet)*>

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

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

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

3.2 Grundlegendes

Die Repräsentation und die Bedeutung der meisten in der *Document type definition* definierten *Tag*s und Attribute sind auf Grund der Namensgebung klar. Auf einige Besonderheiten für Variablen, Anweisungen und den Kontrollfluß muß aber hingewiesen werden.

Variablen. Die SGML–Unterscheidung von Attributwerten und dem Inhalt von Elementen wird bei der EPKML durch die Variablen durchbrochen. Wertzuweisungen (mit `<set>`) können sowohl im Inhalt als auch über ein Attribut erfolgen. Variablennamen werden als $*name*$ geklammert. Sie dürfen überall als Elementinhalt und auf rechten Seiten von Attributzuweisungen auftreten.

Variablen werden entweder über `<var>`, über Parameter oder die Benennung von Elementen eingeführt (s. u. Beispiel). Der Sichtbarkeitsbereich ist durch die natürliche Blockstruktur vorgegeben; Variablen können nur in dem Block verwendet werden, in dem die Deklaration erfolgte. Es gilt statische Bindung.

```
<var name=months value=12>
<var name=pic>
  <img src=pic.gif>
</var>
<var name=money value=500\$>
<var name=foo>
  $pic$
  $pic$
</var>
```

Folgende Namen sind reserviert:

- `$author$`, `$title$`, `$date$`, `$last-modified$` zum Speichern der im `<header>` angegebene Daten;
- `$curdate$` und `$curtime$` für das aktuelle Datum und die aktuelle Zeit;
- `$dimension-unit$` eine Grundeinheit für Längenangaben (`mm`, `cm`, `in`, `pt`, `%`);
- `$time-unit$` eine Grundeinheit für Zeitangaben (`ms`, `s`) und
- $\$name$`.selected$`, wird beim Selektieren oder Deselektieren einer Option in `<pop-up>`, `<browser>`, `<checkbox>`, `<radio-button>`, `<pull-down>` verwendet. Diese Variable mit dem Namen des Gesamtelementes, qualifiziert durch `selected`, enthält die Nummer der Option innerhalb des Gesamtelementes.

Vektoren und Felder können benutzerseitig nicht angelegt werden. Systemseitig werden einige Vektoren und Felder zur Verfügung gestellt (etwa aus der Datenbank). Auf die einzelnen Elemente kann dann mit $\$name[m]\ \ldots\ [n]\$$ zugegriffen werden. Ist die Zugriffsnummer eine Variable, so muß diese nicht in `$` eingeklammert werden. Auf Attribute benannter Elemente wird durch Qualifikation mit dem Attributnamen zugegriffen.

Anweisungen. In der EpkMl selbst steht nur eine kleine Menge von Operationen zur Verfügung. Diese betreffen

- die Variablenmodifikation (`<set>`),
- den Kontrollfluß (`<open>`, `<close>`, `<previous>`, `<next>`, `<down>`, `<up>`, `<back>`, `<empty>`, `<non-empty>`),
- Benutzerinteraktions- und Zeitaspekte (`<wait>`, `<suspend>`) und
- den Anschluß externer Funktionalität (`<sql>`, `<applet>`).

Das Kommando `<set>` setzt Variableninhalte entweder durch Zuweisung des im Attribut `value` angegebenen Wertes oder seines Inhalts (dieser hat höhere Priorität).

Mit `<open>` und `<close>` können alle Elemente (im Attribut `name` zu spezifizieren) geöffnet (d. h. gestartet) bzw. geschlossen (d. h. beendet) werden, die das Attribut `status` (mit den Werten `opened`, `closed` und `suspended`) besitzen. Ebenso kann ein solches Element mit `<suspend>` vorübergehend suspendiert werden.

Der Befehl `<wait>` wartet auf eine Benutzereingabe, das Verstreichen einer Zeitspanne oder die Beendigung eines Videos, einer Diashow, einer Audiosequenz oder eines `<applet>`s.

Weiters greift ein in `<sql>` ... `</sql>` geklammertes SQL–Anweisung auf die Datenbank zu; das Ergebnis dieses Zugriffs wird in der im Attribut `result` angegebenen Variable gespeichert. Mit `<empty>` und `<non-empty>` kann ein leeres oder nichtleeres Ergebnis abgefragt und unterschieden werden.

Ein `<applet>`–Befehl schließlich ruft die in seinem Attribut `function` angegebene Funktion (Methode) der verwendeten Hintergrundsprache auf, wobei die Parameterübergabe wie für Java gestaltet ist.

Kontrollfluß. Der Kontrollfluß innerhalb der EpkMl ist sequentiell. Er führt nach der Ausführung einer Anweisung, wenn nicht explizit anders angegeben, zum nächsten in Aufschreibungsreihenfolge. Der Kontrollfluß beginnt bei der ersten Anweisung in `<main>` (s. u.).

Einige Anweisungen werden dabei nebenläufig ausgeführt (sie starten einen neuen `Thread` und kehren sofort zum Aufrufort zurück). Es sind dies `<open>` für `<audio>`, `<video>` und `<slide-show>` (bzw. diese Elemente selbst, falls ihr Attribut `status` den Wert `opened` hat) und `<applet>` (falls dadurch tatsächlich ein neuer Prozeß erzeugt wird).

3.3 Die Gliederung eines Katalogs

Ein Katalog in EpkMl teilt sich in

- allgemeine Verwaltungsinformation (`<header>`),
- die Deklaration der Anbindung zur Außenwelt (`<externals>`),
- die Deklaration von Standardformatinformationen (`<styles>`),
- die Erklärung der Katalogstruktur (`<definitions>`) und
- das Hauptprogramm (`<main>`).

Als Beispiel präsentieren wir den kürzesten in EpkMl beschreibbaren Katalog:

```
<epkml>
  <header
        title="The Shortest EPKML Catalogue"
        author="LMU"
        date="01/08/97"
        last-modified="01/08/97">
```

```
  <externals>
  <styles>
  <definitions>
  <main>
    <exit>
</epkml>
```

Die Verwaltungsinformation wird durch ihre Beschreibung gleichzeitig in den Variablen `$title$`, `$author$`, `$date$` und `$last-modified$` zum späteren Gebrauch zur Verfügung gestellt.

Anbindung externer Funktionalität. In `<externals>` werden alle externen Klassen (`<class>`) und Datenbankschemata (`<scheme>`) dem Katalog bekanntgemacht. Es ist ein Fehler, in `<applet>` oder `<sql>` Klassen bzw. Datenbankschemata zu verwenden, die nicht in `<externals>` deklariert wurden.

Standardvorlagen. Im Stilteil werden für die Layoutobjekte Voreinstellungen mittels `<stylesheet>` vorgenommen. Dies erfolgt in Anlehnung an die *Cascading style sheets*. Verschiedene Einzelvoreinstellungen (für Paragraphen, Rahmen, Fenster, etc.) können mit `<stylesheet>` zusammengefaßt und über das Attribut `name` insgesamt benannt werden.

Stylesheets können Erweiterungen anderer sein. Das Attribut `extends` enthält eine Liste von Stylesheetnamen (s.u.), deren Voreinstellungen übernommen werden. Dabei priorisieren zunächst Voreinstellungen in zuerst genannten Stylesheets solche in später genannten. Zusätzlich überladen Voreinstellungen im erweiternden Stylesheet die übernommenen Voreinstellungen der in der Liste genannten Stylesheets. Ebenso können die einzelnen Voreinstellungen modifiziert werden.

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

Deklarationen und Definitionen. Im `<definitions>`–Teil werden Variablen (s. o.), Makros, beliebige Elemente und die Themenstruktur (s. u.) definiert. Makros dienen vor

allem einer kürzeren Schreibung und einer klareren Übersicht. Wie bei allen Elementen werden Parameter in den Attributen `attribs` und `elems` deklariert. Makros werden mit `<expand>` aufgerufen. Im folgenden Beispiel wird text durch *Hello World!* und das Bild durch *hello.gif* ersetzt.

```
<macro name=my-image
  attribs="image"
  elems="text">
  <img src=$my-image.image$>
  <em>$text$</em>
</macro>
...
<expand name=my-image>
  <attribute name=image value=hello.gif>
  <element name=text>
    Hello world!
  </element>
</expand>
```

Die in `<definitions>` angegebenen Elemente werden ebenfalls nur deklariert: Das `status`–Attribut hat den Wert `closed`, das `invisible`–Attribut ist gesetzt.

3.4 Die Themenstruktur

Produktgruppen dienen benutzerseitig der Gliederung des Produktkataloges, auf der Generierungsseite aber auch der automatischen Erstellung von Navigationsmöglichkeiten. In EpkMl werden solche Gruppierungen als sogenannte Themen (`<theme>`) wiedergegeben.

Ein Thema besteht aus

- Präsentationsseiten (`<page>`) oder -fenster (`<window>`),
- einer Deklaration der Datenbankeinträge, die es umfaßt (`<extension>`), und einer Formatvorlage für diese Datenbankeinträge (`<template>`),
- einer Liste von Ausnahmen für diese Formatvorlage (`<exceptions>`) und
- einer Liste von Unterthemen.

Ein Thema wird mit `<open>` geöffnet und mit `<close>` geschlossen. Zur Navigation innerhalb des Themenbaumes stehen `<next>`, `<previous>`, `<up>`, `<down>`, `<back>` und `<back hierarchical>` zur Verfügung. Im folgendem Beispiel ist ein Thema für Angebote definiert. Der Datenbankanfrage folgt die Definition der Präsentationsschablone.

```
<theme name=products>
  <extension result=on-sale>
    <sql>
      select name,id,price,image,audio
      from database
```

```
      where database.price<100
    </sql>
    <template name=group-template>
      <window name=group-window style=on-sale-products>
        <button name=shopping disabled>
          <on-click>
            <open name="shopping-bag">
          </on-click>
        </button>
        <frame name=product-description>
          <img src="which-picture">
          <audio src="which-audio">
        </frame>
        <browser>
          <make-options from=on-sale>
            <p> "on-sale.name"
            <on-selected>
              <set name=shopping.disabled value="">
              <redraw name=shopping>
              <set name=which-picture value="on-sale.image">
              <set name=which-audio value="on-sale.audio">
              <open name=product-description>
            </on-selected>
            <on-deselected>
              <set name=shopping.disabled value='disabled'>
              <redraw name=shopping>
              <close name=product-description>
            </on-deselected>
        ...
</theme>
```

In der Übersichtsdarstellung eines Themas sollten Knöpfe zur Weiterverzweigung in eventuelle Unterthemen und Datenbankeinträge (s. u.) vorhanden sein. Implizit wird ein `<wait>`–Anweisung zum Abschluß eingefügt, um die Benutzerreaktion abzuwarten. Die Datenbankeinträge, die von einem `<theme>` behandelt werden sollen, sind in `<extension>` durch eine `<sql>`–Anweisung zu beschreiben. Dem Ergebnis der Datenbanksuche kann im Attribut `result` ein Name gegeben werden. Es wird als Liste von Einträgen aufgefaßt; die Sortierung spielt damit eine Rolle (s. `<previous>`, `<next>`). Eine Formatvorlage (`<template>`) wird durch Variablen mit Inhalten aus der in `<extension>` erzeugten Datenbanktabelle gefüllt.

Für einige Einträge von `<extension>`, für die zum Beispiel das definierte `<template>` unbefriedigende Ergebnisse liefert, können mit `<exceptions>` Ausnahmen spezifiziert werden. In einer `<sql>`-Anweisung werden die Ausnahmen angegeben (das Ergebnis muß eine Untermenge von `<extension>` sein), in `<template>` eine neue, alternative Formatvorlage.

3.5 Services

In EpkMl lassen sich Standardsituationen wie Inhaltsverzeichnis, Benutzeranmeldungsformular, Fragebogen, Produktsuche, Warenkorb und Einkaufsliste ohne Aufwand definieren. Hierfür sind in EpkMl die folgenden Elemente definiert worden:

- `<table-of-contents>` macht die Definition eines Einleitungsfensters oder -rahmens möglich. Es dient als Inhaltsverzeichnis von den gebotenen Alternativen den Kataloges: eine Firmenpäsentation, eine Führung durch den Katalog, verschiedene Sichten der Produkte, Bestellungen, zusätzliche Informationen oder Funktionalitäten.
- `<registration-form>` erlaubt die persönliche Anmeldung des Katalogbenutzers. Die eingegebenen Daten werden im Bestellungsformular verwendet.
- `<search-form>` erlaubt die Spezifikation des Suchverfahrens in der Datenbank.
- `<shopping-bag>` ist eine Schablone für die Liste der zur Bestellung ausgewählten Produkte.
- `<shopping-list>` dient zur Selektion einer Liste von Produkten, dessen Seiten oder Fenster mittels der Navigation besucht werden sollen. Diese Liste wird zu Beginn festgelegt.
- `<order>` beinhaltet das konkrete Bestellungsverfahren. Die Semantik dieser Funktion kann je nach Konfiguration bei der Installation des Kataloges festgelegt werden. Eine Bestellung kann per E-mail, Internet oder per Fax gesendet, oder gedruckt werden.
- `<question-form>` kann vom Benutzer ausgefüllt werden, um dem Kataloghersteller und -designer seine Kommentare, Fragen und/oder Anmerkungen mitzuteilen.

3.6 Layout

Die Layout-Möglichkeiten von EpkMl sind Erweiterungen derer von HtMl. Unterschiedliche Schriftarten und Stile werden für Texte, Paragraphen (`<p>`), Bilder (`<image>`), Rahmen (`<frame>`), etc. angeboten. Auch die üblichen interaktiven Elemente wie `<browser>`, `<checkbox>`, `<input>` stehen zu Verfügung.

Ein Element zur Definition von Fenstern (`<window>`) wurde hinzugefügt; folglich ist EpkMl fenster-orientiert und nicht seiten-orientiert. Zusätzlich wurde die Sprache um `<flowbox>` für mit Text umgebene Bilder und andere Elemente wie `<pulldown-menu>` oder `<button>` erweitert. Multimediale Objekte werden mittels der zeitabhängigen Elemente `<video>`, `<slide-show>` und `<audio>` integriert.

Für alle Layout-Elemente können Voreinstellungen mittels der Definition eines Stils vorgenommen werden: mit `stylesheet` ist es möglich, jedem Element z. B. eine Farbe, einen Rand, eine Größe zuzuweisen, die bei einer individuellen Einstellung überladen werden kann. Layoutelemente, wie Knöpfe, Rahmen, Fenster, Browser, Audio- und Videosequenzen zeigen die Beispiele 3.4 und in [3].

3.7 Regie

Ein großer Teil von EpkMl betrifft interaktive Elemente. HtMl wurde erweitert bei Menüs (`pulldown-menu`) und Knöpfe (`button`). Zusätzlich wird jedes interaktive Element mit einer spezifizierbaren Methode versehen z.B. `on-click` für `button`, die beim Eintreten der Interaktion aufgerufen wird. Für die meisten interaktiven Situationen, wie bei der Navigation durch die Themenstruktur, sind Elemente mit vordefiniertem Verhalten gegeben. Dieses Verhalten kann selbstverständlich erweitert oder geändert werden.

Der Kontrollfluß eines Kataloges ist sequentiell, kann aber durch das Verwenden von speziellen Elemente der Sprache geändert werden. Die Elemente `<empty>` und `<non-empty>` wurden zur Verzweigung basierend auf dem Ergebnis eine Datenbankanfrage eingeführt. Bedingungslose Verzweigung ist mittels der Navigationselemente (`<next>`, `<previous>`, etc.) möglich oder durch ein direktes Öffnen (`<open>`), das für alle Layoutelemente und Themen verwendet werden kann. Elemente können auch mit der Anweisung `<close>` geschlossen werden, aber dies hat keinen Einfluß auf den Kontrollfluß. Die EpkMl bietet zusätzliche Möglichkeiten, um ihre fehlenden Ablaufstrukturen zu kompensieren. Neben der Leerheitsabfrage für Datenbanken (`<empty>`) sind dies vor allem `<make-options>` und `<make-items>`, die Datenbanktabellen in Browser- und Listeneinträge verwandeln.

Hierzu zunächst ein größeres Beispiel, in dem die Definition eines statischen und eines dynamischen Browsers zu beobachten ist. Der erste Browser greift auf feste Elemente einer Datenbanktabelle (`$out$`) zu und fügt ein weiteres Element in die Auswahl ein. Der zweite Browser generiert aus den Datenbankeinträgen dynamisch eine Liste seiner `<option>`s, die bei ihrem Aufruf ein entsprechend angepaßtes Verhalten zeigen. Ebenso verhält sich `<make-items>`.

```
<sql result=out>
  SELECT code price prod-name
  FROM products
  WHERE saison=winter
</sql>
<browser name=static-browser>
  <option>$out.prod-name[1]$ $out.price[1]$
  <option>$out.prod-name[2]$ $out.price[2]$
  <option>Product A  Price 100.-
  <option>$out.prod-name[4]$ $out.price[4]$
  <option>$out.prod-name[3]$ $out.price[3]$
</browser>
<frame name=abc
  elems="prod-name prod-price">
  The product $prod-name$ has a price of $prod-price$.
</frame>
<window name=abc elems="prod-name prod-price">
  The product $prod-name$ has a price of $prod-price$.
</window>
<browser name=dynamic-browser>
```

```
  <make-options from=out>
    $out.prod-name$ $out.price$
    <on-selected>
      <open name=abc>
        <element name=prod-name>
          $out.prod-name$
        <element name=prod-price>
          $out.price$
      </open>
    </on-selected>
  </make-options>
</browser>
```

4 Eine strukturelle operationelle Semantik für EpkMl

Zur Präzisierung der bei der Syntax und der Pragmatik der Sprache EpkMl getroffenen Entscheidungen wurde eine formale Semantik entwickelt, die statische Bindung für Variablen und Prozeduren (Layoutelemente), eine *Call by value*–Parameterübergabe und eine Blockstrukturierung für Variablensichtbarkeitsbereiche realisiert. Daneben wurden Interaktionsmöglichkeiten des Benutzers und der zeitliche Aspekt berücksichtigt. Die Zustände enthalten alle notwendigen Informationen für den Kontrollfluß eines EpkMl-Programms. Vom tatsächlichen Layout und Datenbankinterna wird abstrahiert. Die möglichen Übergänge zwischen Zuständen werden durch Regeln definiert, die zusätzlich die jeweilige Zustandsänderung angeben.

Ein EpkMl-Programm wird sequentiell ausgeführt; Ausnahmen bilden audio, video, slide-shows. Dieses Konstrukt ermöglicht die nebenläufige Ausführung und Kontrolle mehrerer Prozesse. Nebenläufigkeit wird durch Interleaving modelliert. Es können schwache Echtzeitbedingungen zur Begrenzung der Dauer von Prozessen (bzw. allgemeiner Timelines) oder der periodischen Ausführung von Prozessen spezifiziert werden. Dazu haben wir die Sprache EpkMl um Hilfskonstrukte angereichert, in die einige ursprüngliche EpkMl-Anweisungen übersetzt werden. Für diese Hilfskonstrukte wird eine operationelle Semantik angegeben.

Wir stellen aber im folgenden keine vollständige Semantik für die gesamte Sprache EpkMl vor, sondern konzentrieren uns auf einige wesentlichen Ausschnitte. Ausführlich ist die Semantik in [6] beschrieben.

4.1 Der formale Hintergrund

Die Menge aller möglichen expandierten EpkMl Anweisungen (die Programme) werden S genannt. Jede Anweisung, die keine freien Variablen beinhaltet, ist für unsere Semantik ein Wert. Die Menge aller EpkMl Variablenamen bezeichnen wir mit N. Einige Anweisungen

enthalten Aktionen (z.B. `<on-click>` in `<button>`); wir definieren eine Funktion

$$\text{act} : S \rightharpoonup S$$

die die lexikographische erste Aktion für eine Anweisung zurückgibt, wenn immer dieses möglich ist. Es wird eine globale Zeit vorausgesetzt, die nur mit der vordefinierten Operation *now* zugänglich ist, und die den Wert der aktuellen Zeit ermittelt.

Das Zustandsmodell. Die Semantik arbeitet auf Zuständen aus einer Menge Σ. Diese sind Sechstupel, bestehend aus dem Themenstrukturgraphen (eine Menge ,), aus der Variablenbelegung (eine Menge E), der Datenbankinstanz (Δ), aus den dargestellten Elementen (Λ), aus dem Speicher (M) und einer Menge von Signalen (Π):

$$\Sigma = , \times E \times \Delta \times \Lambda \times M \times \Pi.$$

Der Speicherzustand M wird modelliert als Menge von Speicherzellen C, in denen Paare von EpkMl Anweisungen (aus der Menge S) und Variablenbelegungen (aus E) untergebracht werden können; formal ist M eine partielle Abbildung von C nach $S \times E$:

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

Eine Operation

$$\text{new} : M \to C$$

liefert für einen Speicherzustand eine neue, noch unbelegte, d. h. nicht im Definitionsbereich des Speicherzustandes liegende Speicherzelle. Die Operationen

$$\text{stm} : C \to S \qquad \text{und} \qquad \text{env} : C \to E$$

liefern für eine Speicherzelle die darin enthaltene Anweisung bzw. die darin enthaltene Variablenbelegung.

Wegen der statischen Bindung und auch wegen der möglichen Blockbildung ist eine Variablenbelegung eine Liste (ein Keller) von Abbildungen von den Variablennamen N auf Speicherzellen:

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

geschrieben als $\varepsilon = [e_1, \ldots, e_n|\varepsilon']$ (wo $e_1, \ldots, e_n$ eine partielle Abbildung und ε' eine andere Variablenbelegung sind). Eine Variablenbelegung kann mit

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

erweitert werden (die erste Abbildung der Liste wird entsprechend manipuliert) oder der Wert einer Variable für eine Belegung kann durch rekursive Suche in der Kellerstruktur verändert werden:

$$\leftarrow\!\!+ : E \times N \times C \to E.$$

Hier wird für gegebenes (ε, n, l) mit $\varepsilon = [e_1, \ldots, e_k]$ die erste Komponente e_i in ε gesucht, die $n \in \operatorname{dom} e_i$ erfüllt und der Wert von n wird auf l gesetzt. Wir schreiben $\varepsilon \nleftarrow \{n \mapsto l\}$ für diese Operation.

Die Menge der Themenstrukturgraphen , umfaßt alle Listen von Bäumen, die in EpkMl beschreibbar sind, deren Knoten mit einem Namen, einer EpkMl–Anweisung (dem Inhalt der Themenanweisung ohne die Unterthemen), einer Variablenbelegung und einem Aktualitätsvermerk markiert sind. Diese Menge , ist eine Untermenge von

$$, _0 = (N \times (S \times E) \times \mathbf{B} + (N \times (S \times E) \times \mathbf{B}) \times , _0)^*$$

Schließlich erstellt die partielle Funktion

$$\text{bld} : , \times N \times (S \times E) \times , ^* \rightharpoonup ,$$

aus einem Graphen, einer Markierung und einer Liste von Graphen einen neuen Graphen, der den ersten Graphen erweitert (um einen Baum aus der Markierung und der Liste). Ebenso wie für Variablennamen liefert

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

für einen Themengraphen die an der aktuellen Position abgelegte Variablenbelegung.

Die Menge von Datenbankeninstanzen Δ beinhaltet Listen von Tupeln, die dem Datenbankschema entsprechen. Wir modellieren diese Schemata nicht und gehen von gegebenen SQL-Anweisungen aus.

Die Layoutelemente Λ sind Mengen von Speicherzellen, deren Inhalte auf dem Bildschirm visualisierbar sind.

$$L = \wp(C)$$

Interaktionen und Synchronisation werden durch die Menge Π modelliert, die alle verfügbaren Signale enthält. Jedes Signal hat einen zeitlichen Stempel und den Namen des Elementes, durch das es erzeugt wurde.

$$\Pi = \bigcup_{t,n} (\wp(\{\, \mathit{clicked}(t,n), \mathit{end}(t,n), \mathit{close}(t,n), \\ \mathit{play}(t,n), \mathit{stop}(t,n), \mathit{pause}(t,n), \mathit{rewind}(t,n), \mathit{forward}(t,n)\}))$$

Diese Signale informieren das System, daß eine bestimmte Aktion für ein bestimmtes Layoutelement zu einem bestimmten Zeitpunkt stattgefunden hat.

Voraussetzungen. Wir geben hier nur die Regeln für normalisierte EpkMl–Texte an. Von diesen wird vorausgesetzt, daß sie syntaktisch fehlerfrei sind; außerdem ist der Wert des Attributs `status` für jedes Layoutelement `closed`. Dieses Attribut wird erst nach der Deklaration des Elements durch die entsprechenden Kommandos (`<open>`, etc.) manipuliert. Dadurch wird die Menge der zulässigen EpkMl–Texte offensichtlich nicht eingeschränkt.

Um Seiteneffekten für die interaktiven Elemente (z. B. `<button>`) Rechnung zu tragen, werden Änderungen in deren Variablenbelegung als permanent betrachtet (die zur passenden Speicherzelle gehörige Variablenbelegung wird auch im Speicher verändert).

Jede Regel geht von einer EpkMl-Anweisung und einem Zustand entweder nur zur einem Zustand, und zwar im Falle, daß die Anweisung vollständig ausgeführt worden ist, z. B.

$$\langle s, (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\rangle \Longrightarrow (\gamma', \varepsilon', \delta', \lambda', \mu', \pi') \ ,$$

oder wieder zu einem Paar bestehend aus einer Anweisung und einem Zustand, z. B.

$$\langle s, (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\rangle \Longrightarrow \langle s', (\gamma', \varepsilon', \delta', \lambda', \mu', \pi')\rangle \ .$$

Hilfsangaben. Zur Beschreibung der Semantik wird die Sprache EpkMl (dementsprechend die Menge S) durch einige Hilfskonstrukte erweitert. Zeitschranken werden durch das Hilfskonstrukt

$$\text{do } \{\ s\ \} \ \text{watching}(c, t)$$

angegeben, wobei s eine Anweisung, c eine Bedingung und t eine Zeitangabe ist. Der Ausdruck soll die Anweisung s solange ausführen, wie die Bedingung c erfüllt ist. Die Zeit t gibt den Startzeitpunkt des Ausdrucks an; sie dient zur Unterdrückung älterer Unterbrechungen. Wie in SDL [2] gehen wir davon aus, daß die aktuelle Systemzeit in einem Operator *now* zur Verfügung gestellt wird.

$$\frac{\langle p, (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\rangle \Longrightarrow \langle p', (\gamma', \varepsilon', \delta', \lambda', \mu', \pi')\rangle}{\begin{array}{l}\langle \text{do } \{\ p\ \} \ \text{watching}(c, t), (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\rangle \Longrightarrow \\ \quad \langle \text{do } \{\ p'\ \} \ \text{watching}(c, t), (\gamma', \varepsilon', \delta', \lambda', \mu', \pi')\rangle \\ \quad \text{if } c \text{ holds in } (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\end{array}}$$

Wenn c nicht im Zustand bleibt, wird dieser Befehl abgebrochen:

$$\begin{array}{l}\langle \text{do } \{\ p\ \} \ \text{watching}(c, t), (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\rangle \Longrightarrow (\gamma, \varepsilon, \delta, \lambda, \mu, \pi) \\ \quad \text{if } c \text{ does not hold in } (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\end{array}$$

Wenn keine Anweisung auszuführen ist, wird der Befehl auch abgebrochen:

$$\langle \text{do } \{\ \emptyset\ \} \ \text{watching}(c, t), (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\rangle \Longrightarrow (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)$$

Das zweite Hilfskonstrukt ist

$$\text{repeat } \{\ p\ \}$$

das eine beliebige Wiederholung einer Anweisung erlaubt:

$$\langle \text{repeat } \{\ p\ \}, (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\rangle \Longrightarrow \langle p \ \text{repeat } \{\ p\ \}, (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\rangle$$

Zur Unterbrechung der Wiederholungsschleife, wird eine Anweisung der folgenden Form verwendet: do { repeat { p } } watching(c, t), weil anderenfalls der Prozeß p unendlich oft wiederholt wird. Außerdem wird die Schleife abgebrochen, wenn keine Anweisungen zur Wiederholung vorliegen:

$$\langle \text{repeat } \{ \emptyset \}, (\gamma, \varepsilon, \delta, \lambda, \mu, \pi) \rangle \Longrightarrow (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)$$

Um den leeren Prozeß zu beschreiben, haben wir die Anweisung skip hinzugefügt:

$$\langle \text{skip}, (\gamma, \varepsilon, \delta, \lambda, \mu, \pi) \rangle \Longrightarrow (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)$$

Zusätzlich wurden die Operationen replace und restore definiert. Das replace-Konstrukt ersetzt ε durch ε' in der Variablenumgebung, das restore-Konstrukt stellt die alte Variablenumbegung wieder her. Hier wird semantische Information in die Syntax übernommen.

$$\langle \text{replace}(\varepsilon'), (\gamma, \varepsilon, \delta, \lambda, \mu, \pi) \rangle \Longrightarrow (\gamma, [\emptyset | \varepsilon'], \delta, \lambda, \mu, \pi)$$

$$\langle \text{restore}(\varepsilon'), (\gamma, \varepsilon, \delta, \lambda, \mu, \pi) \rangle \Longrightarrow (\gamma, \varepsilon', \delta, \lambda, \mu, \pi)$$

4.2 Die Regeln

Die sequentielle Komposition von Anweisungen wird mit der üblichen strukturellen Regel wiedergegeben.

$$\frac{\langle s_1, (\gamma, \varepsilon, \delta, \lambda, \mu, \pi) \rangle \Longrightarrow \langle s_1', (\gamma', \varepsilon', \delta', \lambda', \mu', \pi') \rangle}{\langle s_1\ s_2, (\gamma, \varepsilon, \delta, \lambda, \mu, \pi) \rangle \Longrightarrow \langle s_1'\ s_2, (\gamma', \varepsilon', \delta', \lambda', \mu', \pi') \rangle}$$

Layout. Geschlossene Elemente (i.e. `status=closed`) werden wie Variablen behandelt. *layout* den Wert `window`, `frame`, `button` usw. annehmen.

$$\begin{aligned}&\langle \texttt{<}\mathit{layout}\ \texttt{name=}n\ \texttt{status=closed>}\ s\ \texttt{</}\mathit{layout}\texttt{>}, (\gamma, \varepsilon, \delta, \lambda, \mu, \pi) \rangle \Longrightarrow \\ &\quad (\gamma, \varepsilon + \{n \mapsto \text{new}(\mu)\}, \delta, \lambda, \\ &\quad\ \mu + \{\text{new}(\mu) \mapsto (\texttt{<}\mathit{layout}\ \texttt{name=}n\ \texttt{status=closed>}\ s\ \texttt{</}\mathit{layout}\texttt{>}, \varepsilon)\}, \pi)\end{aligned}$$

Multimedia. Die Elemente `<video>`, `<slide-show>` und `<audio>` produzieren Prozesse, die nur im Speicher sichtbar sind. Wir beschreiben hier nur `<video>`, die anderen multimedialen Elemente werden analog behandelt.

Wir nehmen an, daß, wenn ein Prozeß, sagen wir n, ans Ende kommt, ein Signal $end(t, n)$ zur Menge der Signale hinzugefügt wird.

Ein Video kann im suspendierten Zustand (für $t_0 = now$) deklariert werden:

$$\begin{aligned}&\langle \texttt{<video name=}n\ \texttt{status=suspended>}\ s\ \texttt{</video>}, (\gamma, \varepsilon, \delta, \lambda, \mu, \pi) \rangle \Longrightarrow \\ &\quad (\gamma, \varepsilon + \{n \mapsto \text{new}(\mu)\}, \delta, \lambda + \{\text{new}(\mu)\}, \\ &\quad\ \mu + \{\text{new}(\mu) \mapsto (\texttt{<video name=}n\ \texttt{status=}(t_0, \text{suspended})\texttt{>}\ s \\ &\qquad\qquad \texttt{</video>}, \varepsilon)\}, \pi)\end{aligned}$$

Zur Modellierung der unterschiedlichen Zustände, in denen sich ein Video befinden kann, beim Abspielen, gestoppt, usw., fügen wir diesen Zustand der Syntax hinzu. Zusätzlich gibt dieser Zustand den Zeitpunkt an, zu dem das Video gestartet wurde. Ein EpkMl-Programm kann diesen Zustand nicht beobachten. Aus diesem Grund liefert die Evaluierung des Video-Zustandes `status` einen übersetzten Wert. Videos können auf verschiedene Signale reagieren; wir beschreiben nur das Verhalten für *play*.

$$\begin{array}{l}\langle s, (\gamma, \varepsilon, \delta, \lambda, \mu + \{l \mapsto \texttt{<video name=}n\ \texttt{status=}(t_0, z)\texttt{>}\ s'\ \texttt{</video>}\}, \\ \quad \pi \uplus \{play(t, n)\})\rangle \Longrightarrow \\ \quad \langle s, (\gamma, \varepsilon, \delta, \lambda, \mu + \{l \mapsto \texttt{<video name=}n\ \texttt{status=}(now, \text{playing})\texttt{>}\ s' \\ \qquad\qquad\qquad\qquad \texttt{</video>}\}, \pi)\rangle \\ \qquad \text{for } t_0 < t\end{array}$$

Zu bemerken ist, daß t_0 die Startzeit für das Video n ist.

Themen. Der Themenstrukturgraph wird von unten nach oben konstruiert.

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

wobie θ die um eine partielle Abbildung aller Themennamen des Themenstrukturgraphen auf ihre entsprechenden Speicherzellen erweiterten Liste ε bezeichnet und $\mu_0 = \mu$ ist. Zu bemerken ist, daß jeder Themenstrukturuntergraph seine eigene Variablenumgebungsinformation enthält, so daß die Veränderungen in der Variablenumgebung in der Vorbedingung der Regel nicht berücksichtigt werden müssen.

Interaktion. Der Benutzer interagiert mit dem Programm über Knöpfe und über die Tastatur. Das Programm wartet auf Signale der Interaktionen mit dem `<wait>` Element. Die Warteschleife kann unendlich lang sein oder solange andauern, bis eine Bedingung erfüllt wird. Unendliches Warten wird wie folgt behandelt:

$$\begin{array}{l}\langle\texttt{<wait> </wait>}, (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\rangle \Longrightarrow \\ \quad \langle\texttt{<var name=}S\ \texttt{value=}now\texttt{> </var>} \\ \quad\ \text{do } \{ \text{ repeat } \{ \text{ skip } \} \} \text{ watching}(\top, S), (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\rangle \\ \qquad \text{für eine neue Zeitvariable } S\end{array}$$

Wenn der Befehl `<wait end-of=`t`>` ausgeführt werden soll, dann wird eine neue Zeitvariable S gewählt und mit einen Wert now belegt. Das System muß prüfen, ob $S + t < now$

gilt.

$$\langle \texttt{<wait end-of=}t\texttt{> </wait>}, (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\rangle \Longrightarrow$$
$$\langle \texttt{<var name=}S\ \texttt{value=}now\texttt{> </var>}$$
$$\text{do } \{ \text{ repeat } \{ \text{ skip } \} \} \text{ watching}(now < S + t, S), (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\rangle,$$
$$\text{für eine neue Zeitvariable } S$$

Wenn ein `<wait>`-Befehl ein Attribut `end-of` hat, wird die Ausführung aufgeschoben, bis ein *end*-Signal für diesen Namen in der globalen Signalmenge vorkommt. Wir beschreiben das mit folgender Bedingung: $end(t, n) \in \pi \rightarrow t < S, S$.

$$\langle \texttt{<wait end=of=}n\texttt{>}, (\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\rangle \Longrightarrow$$
$$\langle \texttt{<var name=}S\ \texttt{value=}now\texttt{> </var>}$$
$$\text{do } \{ \text{ repeat } \{ \text{ skip } \} \} \text{ watching}(end(t, n) \in \pi \rightarrow t < S, S),$$
$$(\gamma, \varepsilon, \delta, \lambda, \mu, \pi)\rangle, \qquad \text{für eine neue Zeitvariable } S$$

Wenn der Benutzer eine Interaktion mit einem Layoutelement ausübt, wird dieser Befehl in seiner eigenen Variablenumgebung ausgeführt. Hier sind die Änderungen im Speicher, nicht in dieser Variablenumgebung wichtig. Das System reagiert auf das früheste Signal.

$$\langle \text{do } \{ \ s \ \} \text{ watching}(c, S), (\gamma, \varepsilon, \delta, \lambda, \mu, \pi \uplus \{clicked(t, n)\})\rangle \Longrightarrow$$
$$\langle \text{replace}(\text{env}(\varepsilon(n))) \ \text{act}(\text{stm}(\varepsilon(n))) \ \text{restore}(\varepsilon)$$
$$\texttt{<set name=}S\ \texttt{value=}now\texttt{> </var>}$$
$$\text{do } \{ \text{ repeat } \{ \text{ skip } \} \} \text{ watching}(c, S), (\gamma, \varepsilon, \gamma, \delta, \lambda, \mu)\rangle$$
$$\text{wenn } t \text{ minimal ist, wie bei } S < t \text{ und } clicked(t, n) \in \pi$$

Signale werden nur berücksichtigt, wenn sie nach dem Starten der Warteschleife gesendet worden sind.

5 Ausblick

Wir haben die Spezifikationssprache (EpkMl) speziell für EPKe entwickelt. Sie wird ergänzt durch Werkzeuge, die eine solche EPK-Spezifikation erzeugen, manipulieren, verarbeiten und testen können. Ein wesentlicher Schwerpunkt lag somit auf der gemeinschaftlichen und koordinierten Entwicklung einer allgemein gültigen Modellierung von EPKen, die alle gängigen Katalogvarianten einbezieht. Dadurch ist EpkMl eine sehr mächtige und umfangreiche Sprache geworden. Für den Entwickler eines EPK liegt der Vorteil auf der Hand: Alle wünschenswerten Eigenschaften und Funktionen eines EPK können auf abstraktem Niveau und ohne Detailkenntnis einer umfangreichen Programmbibliothek spezifiziert werden.

Die Spezifikationssprache wurde als eine Instanz von SGML definiert. Die Syntax erlaubt eine deklarative Beschreibung der EPKe und ist HTML-ähnlich. Die Sprache enthält Elemente zur Themenstrukturierung und zur automatischen Navigation, SQL-Anweisungen, Basiselemente für den Kontrollfluß, Elemente für Multimedia-Objekte und erweitert die Menge der HTML-Layoutelemente.

Wir haben eine formale Semantik für die Sprache EPKML präsentiert. In diesem Bericht wurden nur einige Aspekte erläutert. Die Semantik ist operational und kann zu einer Implementierung ausgebaut werden. Weitere bisher nicht behandelte interessante semantische Fragestellungen sind die Integration dieser Semantik mit der SQL-Semantik, sowie die Definition einer denotationellen Semantik für EPKML.

Literatur

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

[2] ITU-TS. Recommendation Z.100. CCITT Specification and Description Language (SDL). Technical report, ITU–TS, Genova, 1994.

[3] Alexander Knapp and Nora Koch. EPKML: A Specification Language for Electronic Product Catalogues. *IEEE Multimedia*, to appear, 1998.

[4] Alexander Knapp, Nora Koch, and Luis Mandel. The Language EPKML. Technical report 9605, LMU München, November 1996.

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

[6] Alexander Knapp and Piotr Kosiuczenko. Developing Formal Semantics of EPKML. Technical Report 9704, Ludwig–Maximilians–Universität München, May 1997.

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

[8] Nora Koch and Luis Mandel. State of the Art and Classification of Electronic Product Catalogues on CD-ROM. *International Journal of Electronic Commerce, University of St. Gallen*, (3), August 1997.

[9] Nora Koch and Josef Schneeberger. Integrated Assistance for the Development of Electronic Product Catalogues. In *Symposium on Software Technology (SoST)*. SADIO, August 1997.

[10] Nora Koch and Andreas Turk. Towards a Methodical Development of Electronic Catalogues. *International Journal of Electronic Commerce, University of St. Gallen*, (3), August 1997.

[11] Josef Schneeberger, Nora Koch, Aandreas Turk, Rainer Lutze, Martin Wirsing, Hartmut Fritzsche, and Patrick Closhen. EPK-fix: Software-Engineering und Werkzeuge für elektronische Produktkataloge. In M. Jarke, K. Pasedach, and K. Pohl, editors, *Informatik'97, Informatik als Innovationsmotor, 27. Jahrestagung der Gesellschaft für Informatik*, Informatik aktuell. Springer Verlag, September 1997.

[12] Arthur van Herwijnen. *Practical SGML*. Kluwer Academic, Boston, 1994.