A semantic model for specifying data-intensive Web applications using WebML

Sara Comai Piero Fraternali
Politecnico di Milano
Dipartimento di Elettronica e Informazione
Piazza L. Da Vinci, 32
I-20133 Milano, Italy
comai,fraterna@elet.polimi.it

Abstract. WebML (Web Modelling Language) is a language for the design of data-intensive Web sites. It is supported by visual tools allowing the definition of the conceptual data organization and of the pages and links of the actual hypertext(s) which constitute a Web application. In this paper we describe a semantic model for WebML hypertexts by means of Statecharts. Statecharts provide a formal description of the clicking behavior and page data fill of WebML applications. The proposed semantic model has guided the implementation of the WebML runtime and the construction of advanced specification checking functions embedded in the WebML design tools. In particular, developers are supported in the identification of design and runtime problems caused by non-determinism, racing conditions and deadlocks.

1 Introduction

Web applications have spread in every sector of the human activity, well beyond the boundaries of document-oriented systems, for which the Web has been initially conceived.

The enormous demand for Web-enabled applications, both novel or resulting from the re-engineering of existing systems, coupled to the chronic lack of skilled IT personnel, puts forth a dramatic request for better software engineering practices, similar to those adopted in more mature software fields, like database and object-oriented development.

For improving productivity, a broader coverage of the tasks of Web site development is imperative, because the vast majority of Web development tools available on the market still concentrate only on design and implementation, paying little attention to requirement analysis and conceptual modelling [7]. Therefore, implementing and maintaining a large Web site is still a very human-intensive and error-prone activity, which does not benefit from the availability of a formal development process, supported by modelling notations and CASE tools.

To cope with these requirements, the research community has proposed several approaches for the so-called model-driven design of Web sites [1, 5, 8, 9, 11], which share the idea of leveraging semi-formal notations to express the data structure and hypertext topology of a Web site and of using conceptual-level specifications to drive the design and implementation.

WebML [3] is one of the proposals for the conceptual specification and automatic implementation of Web sites. A WebML specification is directed labelled graph, internally represented as an XML document written according to the WebML DTD, which describes the topology of one or more hypertexts conceived to publish information on a set of application objects. The WebML language is backed by a suite of software tools, which transforms visual WebML specifications into server-side page templates and database queries, which implement the desired Web site.

Differently from previous proposals, which were mostly introduced informally and by examples, WebML anchors Web site specifications to a sound formal basis, by associating a formal semantics to the semi-formal visual notation. This paper introduces WebML's formal semantics, which is based on the use of STATECHARTS [10] to express the dynamic behavior of a Web site.

As a consequence of establishing a formal model, Web site specifications acquire an unambiguous meaning and lend themselves to automatic checking for correctness or desired properties. Moreover, the formal semantics can be used as a yardstick to evaluate the correctness of CASE tools generating running Web sites from WebML specifications, because the runtime behavior of the generated site must obey the expected behavior expressed by the formal semantics.

2 Overview of WebML

A WebML specification consists of two major components:

- The *structure model*, describing the conceptual organization of the application data;
- One or more *hypertexts* (*site views* in the WebML jargon) defined on top of the structure model, which express the organization and linking of pages used to publish the application data.

The approach adopted by WebML is data-driven: first the structure of the data is described, then, on the basis of such structure, the hypertext is defined, as explained in the following subsections. For further details about the syntax of WebML the reader may refer to [3] and to the Web site `http://webml.org`.

2.1 Structure model

The structure model describes the conceptual data organization, and is compatible with the Entity-Relationship data model, used in conceptual database design, and with UML class diagrams, used in object-oriented modelling. The fundamental elements of the structure model are *entities*, defined as containers of data elements, and *binary relationships*[1], defined as semantic connections between pairs of entities. Entities have attributes representing the properties of the real world objects and relationships are characterized by named relationship roles (i.e., the two directions in which a binary relationship can be traversed) and cardinality constraints associated to each role.

[1]WebML presently supports only binary relationship without attributes; work on supporting content units defined over generalized n-ary relationships and relationship attributes is ongoing.

Example I: Figure 1 shows a simple structure schema for the publication of an hypertext describing data about books: the rectangles in the graph represent entities, while edges represent relationships (for brevity, relationship roles names are omitted). In the example, each book is written by one or more authors and has a unique publisher; moreover, each book may be associated with zero or more reviews.

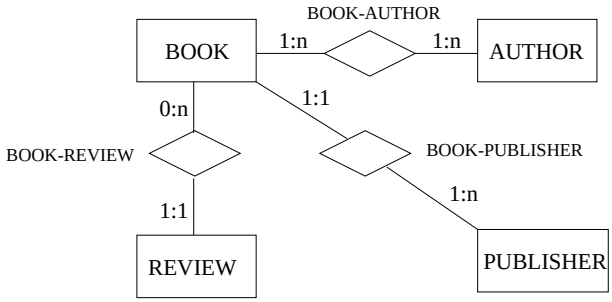

Figure 1: Example of structure schema

2.2 Hypertext

A WebML *hypertext* consists of a set of *pages*, depicted as rectangles, connected by *non-contextual links*, represented by oriented arcs. The content of a page is expressed by means of *content units*. Different kinds of unit are provided by WebML, denoted by different symbols. Units may be connected by *contextual links*, also graphically depicted by means of oriented arcs (See Figure 3). We describe first content units; then, we clarify the use of contextual and non-contextual links.

Units publish information about the objects of the structure schema: each unit is defined over a *master object*, an entity or a relationship role[2], which gives content to the unit.

WebML offers six predefined content units to assemble read-only hypertexts (additional units are available for content management applications):

- *Data units*: they are used to publish a set of attributes of a single object (e.g. the data of a single book). The graphical representation of WebML data units is shown in Figure 2.a.
- *Index units*: they are used to represent sorted lists of objects, where each object is denoted by some representative attributes (e.g. an index of authors may show the first name and last name of each author). Index units are typically linked to a data unit, which shows the details of the object selected from the index (e.g. the data of the selected author). The graphical representation of index units is shown in Figure 2.b.
- *Multidata units*: they show multiple objects together, by repeating the presentation of several, identical data units[3] (e.g., all the books written by an author). See Figure 2.c for the graphical representation.

[2] A relationship role univocally determines a source entity and a destination entity, based on the direction in which the relationship is considered.

[3] In the following sections, multidata units will be treated as a finite set of data units, and therefore will not be considered explicitly.

- *Scroller units*: they provide the commands to scroll over an ordered set of objects. They are generally connected to a data unit showing the current item of the sequence. The graphical representation is shown in Figure 2.d.
- *Filter units*: they allow the user to specify search criteria by means of a search form. Typically, a filter unit is connected to an index unit showing the result of the search (e.g. the user inserts the category of a book, and the list of books belonging to this category is shown). The graphical representation of filter units is shown in Figure 2.e.
- *Direct units*: they associate one object to a single other object along a one-to-one or many-to-one relationship (possibly the identity relationship). They are generally connected to a data unit showing the unique target of the one-to-one or many-to-one relationship (e.g., the data of a book may be connected through a direct unit to the data of its unique publisher). The graphical representation is shown in Figure 2.f.

Figure 2: Graphical representation of WebML units

Example II: Consider for example the hypertext depicted in Figure 3. It contains three pages: the home page, the books' index page and the book page. The home page is empty (we suppose that it contains only unmodeled, presentation-oriented content) and is connected by a link to the books' index page, which contains two units: a filter defined over books (BookFilter) allows one to search all books based on some keywords (e.g. with respect to their category), and is linked to an index unit (BookIndex), which represents the list of books matching the search criteria expressed in the filter unit. The books' index is connected to a data unit in a separate page (BookPage). This page contains several pieces of information, which are shown when the user clicks on an entry in the index of books: the data of the selected book (BookData), the data of its publisher (PublisherData), the index of its authors (AuthorIndex), and a scroller unit defined over the book's reviews (ReviewScroller), which allows the user to orderly browse the book's reviews, displayed one by one in a data unit (ReviewData). A direct unit (Book2Publisher) is interposed between the BookData unit and the PublisherData unit, to associate the book to its unique publisher.

An important difference exists between non-contextual links connecting pages and contextual links between units: the former are a mere navigational device used to change page, the latter imply the transportation of *navigation context* from the source to the destination unit. Navigation context is information passed from one unit to another one in order to make the second unit computable from the data in the structure layer. For example, in Figure 3 the link exiting the home page is a non-contextual link and does not carry any information; instead, the link between the BookIndex unit listing a set of books and the BookData unit showing the data of a particular book is contextual: it must carry the identifier of the book selected in the index, for the data unit to be computable. Note that, as shown in this example,

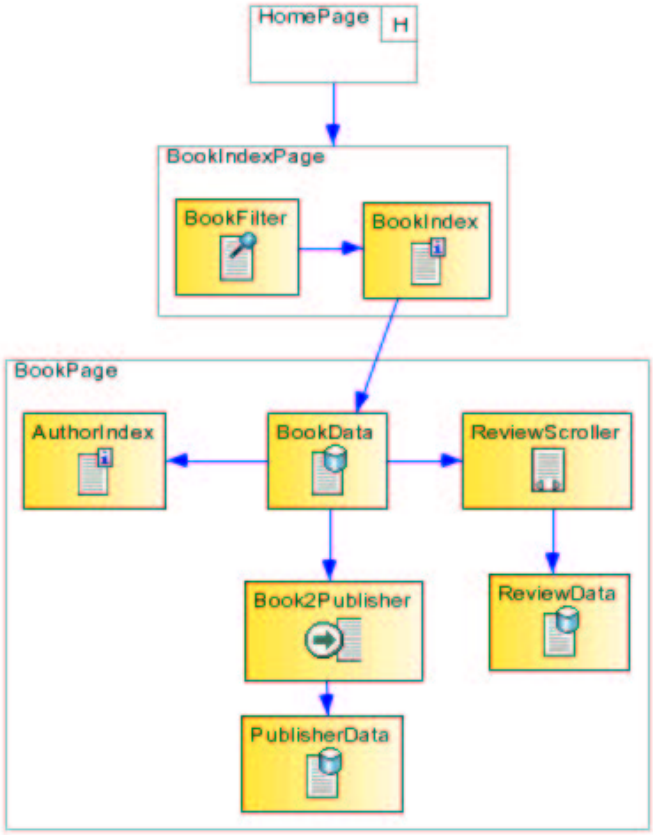

Figure 3: Example of WebML pages

when the source and the destination units of a contextual link belong to different pages, also navigation between pages is performed.

WebML units are both producers and consumers of navigation context. For example, an index unit typically produces the identifier of the object selected from the user; however, it may also consume context, e.g., to display a list of objects connected by a relationship to an input object. For example, in Figure 3 the AuthorIndex unit, listing the authors of a particular book, needs the OID of the current book to be computed: indeed, according to the schema of Figure 1, given the current book, the target objects of the relationship between book and author can be identified.

The following table illustrates the input and output context of the different WebML units.

Unit	Input parameters	Output parameters
Data unit	Selected instance (OID of the current instance)	Current instance
Index unit	Owner of the relationship[1], Optional predicate[2]	Selected item, Owner of the relationship[1]
Multidata unit	Owner of the relationship[1], Optional predicate[2]	Selected item (possibly all the items), Owner of the relationship[1]
Filter unit	Owner of the relationship[1], Optional predicate[2]	New Predicate, Owner of the relationship[1]
Scroller unit	Owner of the relationship[1], Optional predicate[2]	Selected item, Owner of the relationship[1]
Direct unit	Owner of the relationship	Target of the relationship [3]

[1] When a unit is defined over a relationship role, the OID of an instance of the source entity participating to the relationship (called the relationship's owner, in the WebML jargon) is required.

[2] When the unit is preceded by a filter unit, a predicate is passed to compute the result set of the search.

[3] The target of a one-to-one or many-to-one relationship is the unique object associated to the owner of the relationship.

As shown in the example of Figure 3, a WebML page typically contains several units linked in a network topology to produce the desired communication effect. In order to specify how the context is propagated along the chains of linked units, WebML permits the designer to declare links (both contextual and non-contextual) as *automatic* or *clickable*. The former are "automatically clicked" by the WebML runtime system, to propagate context from the source to the destination unit of the link even in absence of user's action. The latter do not exhibit such behavior, but the user must explicitly activate the link for context propagation to occur.

When links are automatic the output parameters of the unit wherefrom the link exits may need proper initialization: the output of an index or scroller unit is initialized to the first instance of the underlying entity or relationship; the output predicate of a filter unit is initialized to "true", to select all objects of the underlying entity or relationship.

For example, in Figure 3, when the BookPage is accessed, the OID of the book to be displayed is passed to the book data unit by its incoming contextual link. Then, propagation of context occurs inside the BookPage page. If all the links between units in BookPage are automatic, context information flows from unit to unit without the user's intervention: the OID of the selected book flows to the subsequent units, thus showing also the index of authors, the publisher's data, the first review and the scroller commands to access the other reviews. The first review is chosen by default by the system, which initializes the output parameter of the scroller unit.

Conversely, if the links exiting the book data unit are defined as clickable, when the BookPage is accessed only the data of the selected book are shown; then the user must click on the provided anchors (one for each link) to transfer the output context and see also the other data in the page. Notice the importance of the automatic links in practical applications: they allow to automatically display information bound to the current data. A more sophisticated example showing the use of automatic and clickable links will be presented in the next section.

3 Semantics of WebML

In the previous section we introduced the syntax and the main characteristics of WebML; now we describe its semantics.

Before introducing a formal description of the behavior of a dynamic Web site, we extend the hypertext of Figure 3 in order to show some particular behaviors and problems that highlight the benefits of having a semantic model.

The WebML specification of Figure 4 extends the previous hypertext with the authors' index page.

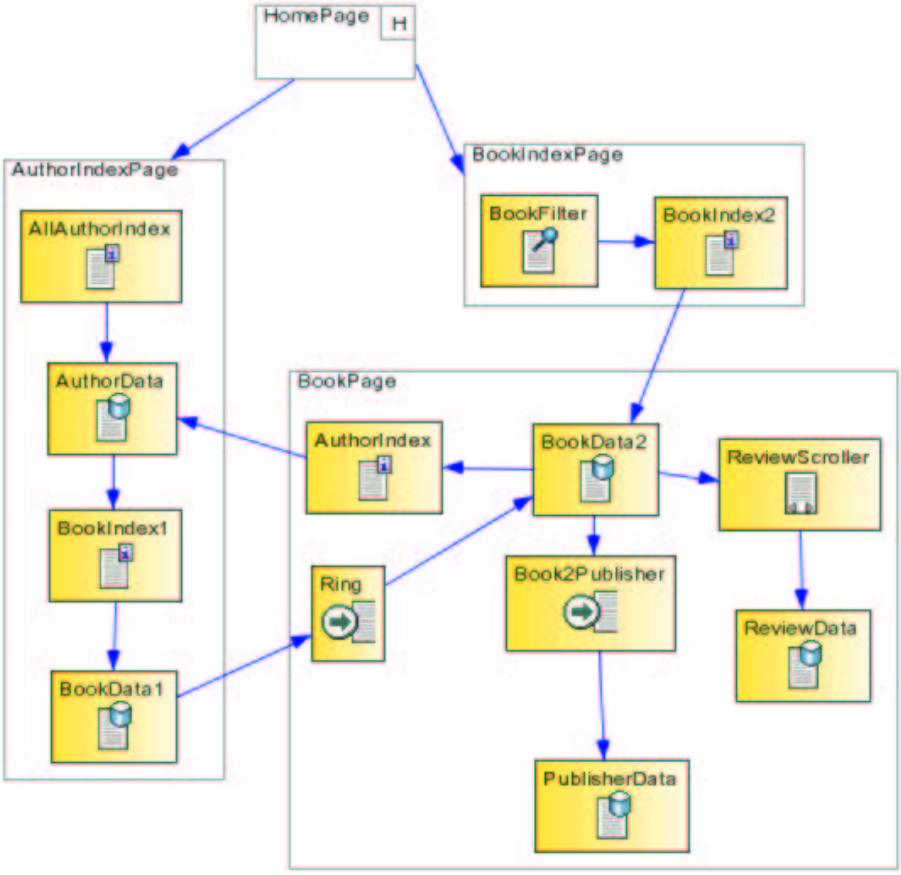

Figure 4: Example of a Web site

Let us carefully analyze such page. It contains several kinds of information to be shown: the index of all the authors (AllAuthorIndex), the data of a selected author (AuthorData), the list of the books of such author (BookIndex1) and the data of one book selected from such list (BookData1). Note that the two first links transport the identifier of the selected author, while the third link transports the identifier of the selected book. Depending on how the links between units are specified, i.e. automatic or clickable, this page behaves differently. Suppose that all the links between the units be automatic, i.e. the first click of every link is automatically done by the system without any intervention of the user: when this page is accessed it displays the list of all the authors, the data of the *first* author of the index (chosen as default by the system), the list of the books of such author and, finally, the data of the *first* book of the book index (chosen as default by the system). All these data are automatically shown. Then, the user may select a different author from the author index or a different book from the book index: in both cases all the data related to the units following the considered index change accordingly. That is, if a new author is selected from AllAuthorIndex the data of the new author are shown, together with his/her books and the data of the first of such

books; if a new book of the author is selected from BookIndex1 the data of the selected book are displayed. Notice how the automatic links allow to define default choices without leaving empty parts in the page.

Consider now the case where the links exiting the index units are defined as clickable and the link exiting the author data unit is still automatic: the behavior of the same page would be different, since the first choice of each index is not performed by the system, but for each index the system waits until the user clicks on an item. This means that, when the page is entered the first time it contains only the list of the authors; then, when the user selects one of such authors, his/her data together with the list of his/her books are shown; finally, when the user selects one of the books of the author also the data of the books are shown. Then, the user may e.g. select a new book from BookIndex1: as a consequence the data of the new selected book are displayed as in the previous case. Instead, if the user selects a new author from AllAuthorIndex, the data of the new author are displayed together with his/her books' list but no data about a particular book are shown until the user explicitly selects one item. Notice that in this case the page content changes, i.e., initially only one unit is populated, then after user's clicking two other units are filled and so on.

So, depending on the kinds of link (automatic or clickable) the behavior of the page is different, since information may be automatically displayed or not displayed at all, and page composition may change after user clicking. For complex pages containing several units the same page may have different configurations at runtime depending on the kinds of links and on user's behavior; such configurations can be properly described by a semantic model.

Let us consider another important aspect that need to be considered. From the authors' index page, it is possible to reach the page displaying further information about the selected book (a direct unit is used to represent the identity relationship, i.e., the current book itself) and from this page it is possible to go back to the authors' index page, by selecting one of the book's authors. When a page may be reached from different pages, the page must be correctly computed for every single access. When designing a Web site several pages could in fact be reused for displaying the same kind of information.

Let us focus on the authors' index page again: when it is reached from the book page the contextual link enters the second unit (AuthorData) of the chain. In this case the first unit in the chain (AllAuthorIndex) may cause some problems. The list of all the authors can always be displayed, independently of how the page is accessed, since this unit does not receive any input context. But what about its outgoing link? If it is clickable, the system waits until the user selects a new item: so, if the page is accessed from the book page the remaining part of the page is computed for the author selected in the book page and it is not changed until a new author is selected from the AllAuthorIndex. Instead, what does it happen if the outgoing link is automatic? Does the system automatically display the data of the first author of such index or the data of the author selected in the book page?

To answer questions like this we need a semantic model, describing the precise behavior of the hypertext, whose interpretation may become difficult for complex sites. Then, on the basis of the semantics the system can be actually implemented and the correctness of the specification can be automatically checked. To formalize the semantics we adopt Statecharts, which allow to easily describe any dynamic system behavior. Indeed, each page of the site can be represented as a state. Intuitively, when we navigate through the different pages we change state. We can change page by clicking on the anchors provided by the current page: the action of clicking represents the event which makes the system change its current state. In a similar

way, also the content of the pages may be represented by concurrent states, each representing the behavior of a single unit: the content of a unit is shown depending on the possible events automatically generated by the system (e.g. when there are automatic links to be followed) or by possible selections performed by the user. In the sequel we formally describe how to map a generic WebML specification into a Statechart describing its semantics. We first provide some preliminary definitions. Then, we define how to map pages into states and how to map units contained inside a page into concurrent states. Finally, we will see that this model allows to analyze the behavior of the system in critical cases, where for example non-determinism or racing conditions arise.

3.1 Preliminary Definitions

The concepts of a WebML hypertext introduced in the previous sections can be formally described as follows:

Definition 1: (**WebML hypertext**): a WebML hypertext is a triple (U, P, L) where U is a set of units, P is a set of pages, and L is a set of links. U, P and L are such that: 1) links in L connect either two pages in P or two units in U; 2) units in U are contained in pages in P; 3) one page in P is defined as the home page.

In the sequel we represent links between units with the pair (u_i, u_j) and links between pages with the pair (p_i, p_j).

Units of a page are classified based on the topology of the links that connect them:

Definition 2: (**Access, depending, and stand-alone units**) Let H=(U, P, L) be a WebML hypertext. Let $u \in U$ be a unit contained in page $p \in P$. Then, u is an *access unit* if it has incoming contextual links originating from outside of p; it is a *depending unit* if it has incoming contextual links originating from units inside p; it is a *stand-alone unit* if it has no incoming contextual links.[4]

We now introduce the variables and alphabets for events (E), conditions (C) and actions (A) needed for mapping WebML concepts to Statecharts:

Definition 3: (**Variables and E[C]/A alphabets**) Let H=(U, P, L) be a WebML hypertext. Let u_i (i=1 . . . n) be the units in U, l_i (i=1 . . . m) be the links in L and p_i (i=1 . . . q) be the pages in P. Then, we define the following variables, events, conditions and actions:

[4]Note that a unit may have multiple incoming links, and thus be both an access and a depending unit. The actual link used at runtime to access a page determines the role of the unit.

Type	Name	Description	NULL value
Variable	access_link_p_i	It refers to the contextual link through which page p_i has been accessed.	yes
Variable	recomputable_u_i	It is a boolean variable stating if the content of unit u_i can be re-calculated for display or not.	yes
Variable	input_context_u_i	It contains the input context of unit u_i.	yes
Variable	output_context_u_i	It contains the output context of unit u_i.	yes
Event	output_context_u_i_available (i=1...n)	It denotes that the content of unit u_i has been calculated and its output context is available in variable output_context_u_i.	—
Event	clicked_on_anchor_l_j (j=1...m)	It denotes that the user has clicked on the anchor corresponding to link l_j.	—
Condition	access_link_$p_k(u_i)$	It checks if there exists an access link entering unit u_i in page p_k.	—
Action	initialize_output_u_i (i=1...n)	It initializes all the output parameters of unit u_i.	—

As customary in Statecharts we use the polymorphic symbol ϵ to denote both the empty event, used to specify automatic transitions, and the empty action.

3.2 Page configuration

We first define how to map the pages of a generic WebML hypertext into a Statechart: given a WebML hypertext all the pages are mapped into states and all the links (both non-contextual and contextual) are mapped into transitions among such states as follows:

Definition 4: (**WebML hypertext Statecharts**) Let H=(U, P, L) be a WebML hypertext. Then, the corresponding *WebML hypertext Statecharts* is obtained as follows:

- For each $p_i \in P$ a top-level state S_p_i is created;
- For each non-contextual link $l_i = (p_j, p_k) \in L$ a transition from S_p_j to S_p_k is created with
  - E[C]/A=ϵ[$true$]/access_link_p_k:=NULL if the link is automatic,
  - E[C]/A=clicked_on_anchor_l_i[$true$]/access_link_p_k:=NULL if the link is clickable.
- For each contextual link $l_i = (u_t, u_v) \in L$ with $u_t \subset p_j$ and $u_v \subset p_k$, a transition from S_p_j to S_p_k is created with
  - E[C]/A=clicked_on_anchor_l_i[$true$]/output_context_u_t_available; access_link_p_k:=l_i if the link is clickable,
  - E[C]/A=output_context_u_t_available[$true$]/access_link_p_k:=l_i if the link is automatic.
- Page_p_{home} is the initial state.

Example III: The WebML hypertext of Figure 4 is mapped into the hypertext statechart shown in Figure 5: the four pages are mapped into four states and all the links among such pages are mapped into transitions. In particular, from the home page two non-contextual links

(transitions 1 and 2) depart, which are activated when the event of clicking on the corresponding anchors occurs. Since the links are non-contextual no access link is set for the following pages. From the authors' index page a link departs toward the book page: it is activated when the user clicks on the anchor of the link (transition 3) provided in correspondence of the book data unit (BookData1), setting the current link as active for the book page. This last operation is necessary when the page can be accessed through different links in order to consider the correct incoming link. This transition notifies also that the output context of BookData1 has been computed and is available to be used to compute the new page. Analogously, two contextual links, one from the books' index page to the book page (transition 4) and one from the book page to the authors index page (transition 5) are obtained.

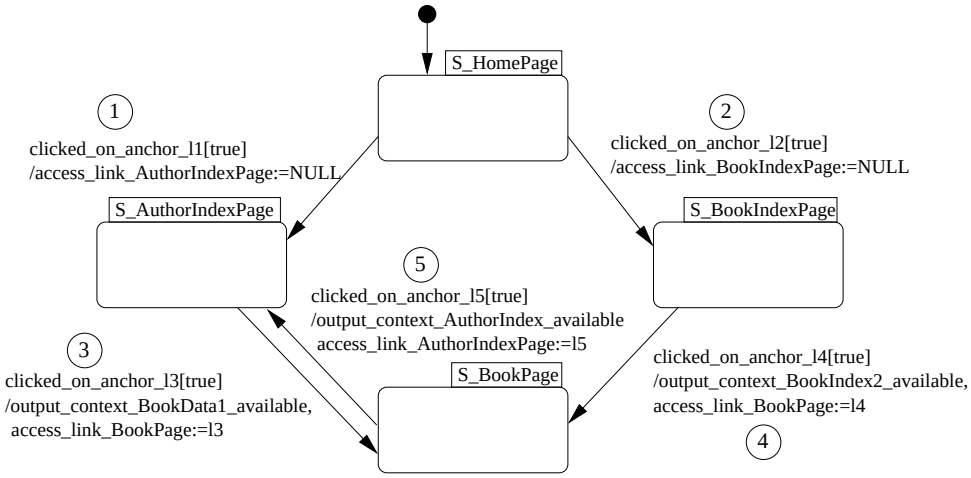

Figure 5: WebML hypertext statechart

3.3 Unit Configurations

Once the pages have been mapped into states, the content of each page can be described. Given a page of the WebML hypertext, the units in it are mapped into a set of concurrent states, each describing the behavior of a single unit.

Intuitively, each unit can be either in a *disabled* state, where no data are shown, or in an *enabled* state where its content is displayed according to the input context. At page entry, a unit is disabled by default. Then, one ore more transitions may lead to the enabled state. From this state one or more transitions are defined, either to go back to the disabled state or to re-enter the enabled state, possibly changing the unit content (see Figure 6). The kinds of events, conditions and actions of the transitions depend on on the fact that the unit is an access, a standalone or a depending unit.

Definition 5: (**WebML unit Statechart**) Let H=(U, P, L) be a WebML hypertext. Let $p \in P$ be a page containing one or more units and S_p be its corresponding state. Then, for each unit u_k contained in p the following states are introduced:

- A concurrent state S_u_k nested at the first level of S_p is created;
- A state $S_{disabled}\_u_k$ nested inside S_u_k is created and set as initial state;

- An state $S_{enabled}\_u_k$ nested inside S_u_k is created containing the entry action recomputable_u_k:=false;

A set of transitions between $S_{disabled}\_u_k$ and $S_{enabled}\_u_k$ are introduced as follows:

- If u_k is an *access unit*, then, for each contextual link $l_j = (u_j, u_k) \in L$ with $u_j \not\subset p$ a transition from $S_{disabled}\_u_k$ to $S_{enabled}\_u_k$ is added with

  E[C]/A=ϵ[output_context_u_j<>NULL AND access_link_p(u_k)]
  /input_context_u_k:=output_context_u_j;
  access_link_p:=NULL.

  The transition states that at page entry the output context of the source unit of the incoming link becomes the input context of the access unit.

- If u_k is a *standalone unit*, then a transition from $S_{disabled}\_u_k$ to $S_{enabled}\_u_k$ is added with E[C]/A=ϵ[$true$]/ϵ. The transition states that a standalone unit is automatically enabled at page entry.

- If u_k is a *depending unit*, then for each contextual link $l_j = (u_j, u_k) \in L$ with $u_j \subset p$

  – The following transitions or actions are added to the state of the *source* unit u_j, which feeds navigation context to the depending unit u_k:

    * A new ring transition on $S_{enabled}\_u_j$ is added with:
      E[C]/A=clicked_on_anchor_l_j[$true$]/ recomputable_u_j:=true, $j = 1, \ldots, n$.
      This transitions expresses that a unit feeding another unit inside the same page may need re-computation (this happens if there is a cycle of links leading back to the unit).
      If the outgoing link l_j is clickable action output_context_u_j:=NULL is added to all the other transitions entering $S_{enabled}\_u_j$; this expresses that for clickable links there is the need of cleaning the output context, when the destination unit is enabled.
      If the outgoing link l_j is automatic, the action initialize_output_u_j is added to all the other transitions entering $S_{enabled}\_u_j$. This expresses that for automatic links there is the need of properly initializing the output context, when the destination unit is enabled.
    * Action output_context_u_j_available is added as entry action in $S_{enabled}\_u_j$ to activate the depending units.
    * Actions output_context_u_j:=NULL; output_context_u_j_available are added to the transitions from $S_{enabled}\_u_j$ to $S_{disabled}\_u_j$. These transitions indicate that the unit cannot be computed and therefore also their depending units must be inhibited by setting the passed context to NULL.

  – The following transitions are defined for the depending unit u_k:

    * A transition from $S_{disabled}\_u_k$ to $S_{enabled}\_u_k$ is created with:
      E[C]/A=output_context_u_j_available[output_context_u_j<>NULL AND NOT access_link_p(u_k) AND recomputable_u_k] /input_context_u_k:=output_context_u_j.
      This transitions expresses that the depending unit is enabled when the output context of its feeding unit becomes available and is not null, the unit has not

been directly accessed from outside the page and is re-computable (i.e., its content has not already computed in the context propagation, e.g., due to link cycles).

* A ring transition on $S_{enabled}\_u_k$ is added having:
E[C]/A=output_context_u_j_available[output_context_u_j<>NULL AND recomputable_u_k]
/input_context_u_k:=output_context_u_j. This transition ensures that if the input context changes also the output context is re-calculated.
* A transition from $S_{enabled}\_u_k$ to $S_{disabled}\_u_k$ is created with:
E[C]/A=output_context_u_j_available[output_context_u_j=NULL]/ϵ. This transition states that the unit is disabled if due to some event (e.g., a user click on a preceding index in the same page) and to the context propagation rules the input context of the depending unit becomes null.

Note that access and standalone units have no transition from the enabled to the disabled state, because for such units it is not possible to change their content once they have been calculated. For example, in the page containing the index of all the books' authors, such index is immediately shown at page entry and no event can change its content.

Example IV: Consider the authors' index page of Figure 4: Figure 6 expresses the hypertext of its first two units, i.e. AllAuthorIndex and AuthorData. For the other units the mapping is applied in an analogous way. Here we suppose that the link between the units be automatic.

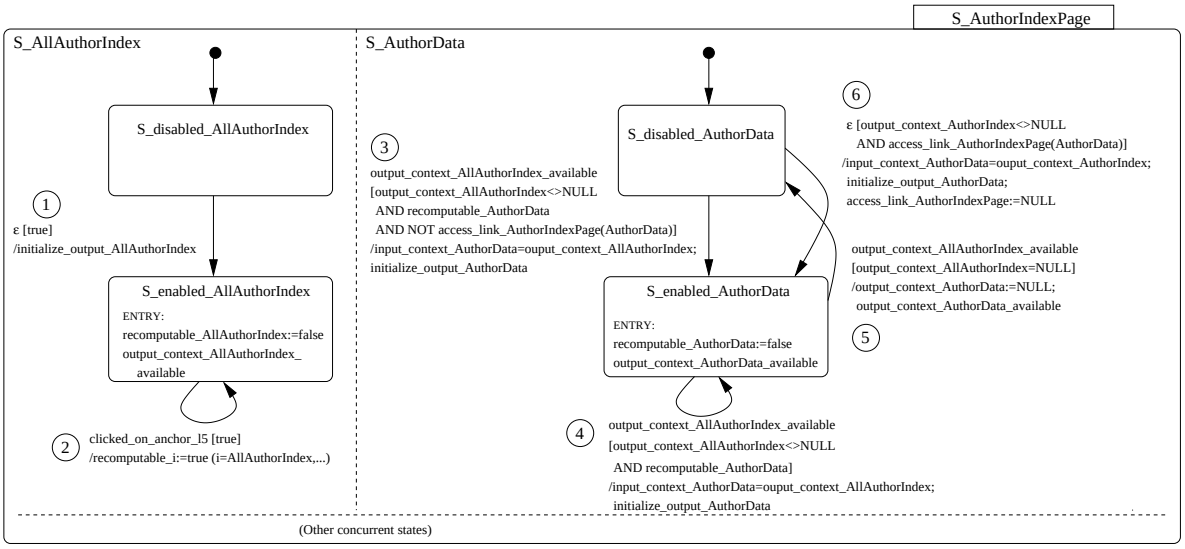

Figure 6: WebML unit configuration statechart

For each unit a concurrent state is created, having two internal states (disabled and enabled).

The first unit (AllAuthorIndex) is a standalone unit and therefore it is automatically enabled (see transition 1). Since its outgoing link is automatic, it also initializes its output with default values: in our system it automatically selects as output the identifier of the first author listed in the index. When the enabled state is entered the unit is set as non-recomputable, i.e.

its content cannot be automatically recomputed[5], and its output context is rendered available to the depending units. The ring transition on the enabled state (transition 2) is triggered when the user selects a new item from the index unit: all the units can then be recomputed according to the new choice (for the index unit we may for example highlight the current choice, for the data unit connected to the index we must show the new selected author, and so on).

The second unit (AuthorData) is both a depending unit, since it depends from the AllAuthorIndex unit, and an access unit, when the page is accessed from the book page. Due to the two access methods, its corresponding state embodies two different transitions (transitions 3 and 6) for passing from the disabled to the enabled state.

As a depending unit the unit is enabled when the output context of the AllAuthorIndex unit is available (transition 3), i.e., every time a new selection in the author index is made by the user or possibly by the system itself. It is fired only if the output context is valid, if the unit has not been already computed, and if there are not any active access links which must be calculated first. Then, it sets its input context to the value of the output context of the unit from which it depends and initializes its output context. Once in the enabled state, it shows the data of the current author and, as in the previous case, it is set as non-recomputable and renders its output available to the following depending unit. From the enabled state two transitions exit: if the user selects a new author from the index a new output context is available: if the context is valid the current state is re-entered and recomputed for the new context (transition 4), i.e., the new current author is shown, otherwise the unit must not be displayed and the unit returns to the disabled state (transition 5). In this latter case the output context of the unit is set to NULL and becomes available to its depending units which cannot be displayed.

As a depending unit the AuthorData unit is enabled when the page variable *access_link* is active for its incoming link (transition 6). Notice that if the page is entered from the book page such variable has been set as active by its incoming link (see transition 4 in Figure 5). If the context is valid the output context is properly initialized and variable *access_link* is unset. Notice that when the page is accessed from the book page only transition 6 is enabled, while transition 3 cannot be triggered. Now we are able to answer the question we issued in Section 3 about the behavior of the page when the outgoing link of the AllAuthorIndex unit is automatic: when the page is accessed from the book page, according to our semantics, the system displays the data of the author selected in the book page and not the first author of the AllAuthorIndex unit.

3.4 Checking the Consistency of WebML Specifications

The specification of WebML semantics through Statecharts allows the designer to better grasp the application behavior and to predict potential critical cases, e.g., non-deterministic and deterministic conflicting transitions, racing conditions, deadlocks and so on. Here, we only show a simple case by means of an example.

Example V: Consider the WebML page of Figure 7: two indexes allow the reader to display information on a certain book. The user may select either a bestselling book from the first index or a recent book using the second one, and the data about the selected book are shown in the data unit. If the two links between the index units and the data unit are defined as

[5]The problem of recomputing the content of a unit becomes relevant in case of cycles among units where all the links are defined as automatic.

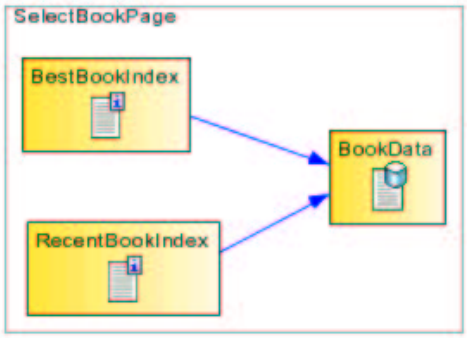

Figure 7: Example of WebML page with racing condition

automatic, the selected item for both indexes is initialized, which results in an unpredictable navigation context to be passed to the data unit (see statechart in Figure 8).

In a typical implementation, propagation of context along links takes places according to some implementation-dependent order, and thus, since the result of applying the two transitions depends on their execution order, a racing condition arises [10].

Notice that, although automatic links are very useful for the specification of automatic behaviors, their use must be carefully controlled to avoid unpredictable behaviors as shown in this example.

The Statechart semantic model has been applied to other, more complex, WebML primitives, including AND-OR nested pages, data entry forms, and update operations. Many subtle behavioral issues have been clarified before implementation with the aid of the illustrated approach.

4 Implementation

The proposed semantic model has set the basis for the implementation of the WebML tool suite.

Figure 9 represents the architecture of WebML, which can be divided into three layers:

1. The *Design Layer*: it includes WebML Control Center, which is the core software element of the WebML architecture, supporting the visual specification of Web sites. Designers use Control Center to input the data structure, the hypertexts diagrams, and presentation directives[6]. WebML specifications are stored as XML documents, which feed the WebML code generator. The output of the code generator is a set of page templates and unit descriptors, which enable the execution of the application in the runtime layer. A page template is a template file (e.g., a JSP file), which expresses the content and mark-up of a page in the mark-up language of choice (e.g. in HTML, WML, etc.). A unit descriptor is an XML file, which expresses the dependencies of a WebML unit from the data layer (e.g., the name of the database and the code of the SQL query from which the population of an index must be computed). Both the templates and the unit descriptors are produced

[6]Presentation directives are expressed as XSL style sheets, which apply to XML documents conforming to the WebML DTD.

by a set of translators coded in XSL and executed by a standard XSL processor. WebML Control Center provides also an interface to the Data Layer to assist the designer in mapping an abstract WebML structure schema to an existing data repository (e.g. a relational database).

2. The *Runtime Layer*: it includes a stack of software components, which produce the actual pages of the application from page templates. Presently, WebML runs on top of any existing JSP 1.1 execution engine, enriched with a thin layer of Java classes decoupling the processing of WebML units from the access API of the data layer. This layer is responsible of extracting the data from the data repository (WebML RunTime) and of formatting it to compose the actual page (WebML TagLib).

3. The *Data Layer*: it includes the repository of data necessary to instantiate the page templates. The inputs to the Data Layer are requests from the runtime layer for data access. The output is the requested content. Presently, WebML can access data stored in any JDBC-compliant relational database and in XML documents.

Three components of the architecture illustrated above have been influenced most by the work on WebML semantics:

- WebML Control Center: the WebML design tool has been extended with a module responsible of checking the consistency of the WebML specifications and of producing warnings and error reports. Checking rules are coded in XSL and enforced by a standard XSL processor. They embody several conservative correctness checks for alerting the designer of potentially dangerous hypertext configurations, e.g., unit and link mismatches, racing conditions and deadlocks.

- The WebML template generator: it embodies the page, unit, and link behavior specified in the Statecharts semantic model. E.g., the sequence of operations needed to correctly generate the passage of context among units and the navigational logics (automatic and clickable links) are encoded at this level.

- The WebML runtime, which insulates the WebML templates from the data source. It has been revised in several aspects to adhere to the described formal semantics. Indeed, this module actually executes the operations specified in the page templates and unit descriptors, by querying the needed data and checking the actual presence of the data in the data source.

The presence of the Statecharts formal semantics has permitted WebML developers to examine on the paper alternative execution options for WebML constructs, and to compare the behavior of the implementation with the expected hypertext execution semantics.

5 Related Work

WebML [3] is one of a family of proposals for the model-driven development of Web sites, which includes also other approaches, e.g., Araneus [11], and Strudel [5]. Like Araneus and Strudel, WebML allows to define the site's structure and content: in the former, the Entity-Relationship model is used to describe the data structure and a conceptual model is used to define the site's hypertext; the latter relies on a data model for semi-structured information and

sites are specified through queries expressed in the StruQL language over the semi-structured data model.

WebML shares several features also with the languages for hypermedia applications, such as HDM - Hypermedia Design Model [9], OOHDM - Object Oriented HDM [12] and RMM - Relationship Management Methodologies [9], from which its basic notations and concepts derive. However, w.r.t. such models WebML has been simplified in order to be effectively supported by CASE tools, and new features specific to data-intensive Web applications have been integrated.

W.r.t. all such approaches (Araneus, Strudel and the hypermedia models), WebML pages and units may be structured in complex ways by means of linking and nesting and exhibit a more sophisticated navigation context semantics, which permits one to define a wide spectrum of page configurations and interactive page-fill behaviors. Indeed, by linking the different kinds of units it is possible to obtain a variety of navigation modes and by defining links as automatic rather than clickable also the content filling of the pages at runtime can be designed. Therefore also our semantics, focusing on the description of such features is quite sophisticated and is not associated only to the simple navigation among pages.

In literature navigation semantics has already been described by means of formal methods: in [13] Petri Nets are used to describe hypertext systems; in [15] and [6, 14] Statecharts are employed to describe navigation browsing. Statecharts are a more powerful mean for the description of reactive systems, since they allow the specification of hierarchical structure; therefore they seem to be suitable for the specification of hypermedia applications requiring synchronization control across different levels of the hierarchical structure.

In [15] Statecharts are used to describe the behavior of hypertext networks: however, the focus is on the specification of system interface behaviors, e.g., related to buttons, frames and so on. Instead, in [6, 14] a model based on Statecharts, called HMSB - Hypermedia Model Based on Statecharts, is used to specify both the structural organization and the browsing semantics of hypermedia applications. Here the focus is on synchronization of multimedia data (i.e. text, audio, animations, images and so on). An environment, called HySChart, supporting the authoring of structured hyperdocuments based on the HMSB model has been proposed, which can be used also as a front-end for Web applications. Compared to the HMSB model, the WebML semantic model addresses structured, data-intensive hypertexts, which do not consist of page instances connected by links, but of page templates, composed by content units which retrieve data from a data layer (e.g. a relational database). For this reason the navigation semantics of WebML results in a more complex specification. Moreover, w.r.t. [6, 14], Statecharts in WebML are not used as a notation for specifying the Web application, but they merely represent the method to formally describe its semantics: in fact, the WebML tools provide a more intuitive graphical notation for the specification of an hypertext, and rely on the formal semantics to provide an efficient specification checker for such hypertext model.

Finally, a different approach based on a generalization of Statecharts is used in OOHDM, which employs ADVcharts [2]. ADVcharts use notations from Petri Nets and statecharts and are used to provide a formal semantics of Abstract Data Views, a concept for designing interactive user interfaces. Diversely from our approach, ADVcharts address the formal specification of dynamic aspects of user interfaces, which are seen as composition of simple visual objects.

6 Conclusions

In this paper we have presented a semantic model for the WebML site design language. The model relies on the mapping of WebML constructs (pages, units, and links) into a Statechart. This mapping caters for all the design primitives of WebML, which are able to formally express the clicking (and automatic) behavior of complex, real-life data-intensive Web applications. The proposed semantics has been extensively used as a reference in the implementation of the WebML design and runtime tools. In the future, further aspects of the WebML language, designed to cope with sophisticated application requirements (e.g., nested pages, update operations, data entry units) will also be given a formal semantics using Statecharts, to formally investigate their properties and runtime behavior and direct their integration in the WebML design and execution environment.

References

[1] A. Bongio, S. Ceri, P. Fraternali, A. Maurino: Modeling Data Entry and Operations in WebML. WebDB (Informal Proceedings) 2000: 87-92

[2] L.M.F. Carneiro, D.D. Cowan, C.J.P. Lucena. ”ADVcharts: a Graphical Specification for Abstract Data Views”. CASCON’93, Toronto, Canada, pp. 84-96 October, 1993.

[3] S. Ceri, P. Fraternali, A. Bongio. ”Web Modeling Language (WebML): a Modeling Language for Designing Web Sites”. Computer Networks, 33, pp. 137-157 (2000).

[4] S. Ceri, P. Fraternali, A. Maurino, S. Paraboschi: One-to-One Personalization of Data-Intensive Web Sites. WebDB (Informal Proceedings) 1999: 1-6

[5] M. F. Fernandez, D. Florescu, J. Kang, A. Y. Levy, D. Suciu. ”Overview of Strudel - A Web-Site Management System”. Networking and Information Systems 1(1): 115-140 (1998).

[6] M.C. Ferreira De Oliveira, M.A.S. Turine, P.C. Masiero. ”A Statechart-based Model for Modeling Hypermedia Applications”. ACM TOIS, April, 2001.

[7] P. Fraternali. ”Tools and Approaches for Developing Data-Intensive Web Applications: A Survey”. ACM Computing Surveys 31(3): 227-263 (1999)

[8] F. Garzotto, P. Paolini, D. Schwabe. ”HDM - a Model-based Approach to Hypertext Application Design”. ACM Transaction on Information Systems 11(1), January, 1-26, 1993.

[9] T. Isakowitz, W. Sthor, P. Balasubramanian. ”RMM: a Methodology for Structured Hypermedia Design”. CACM, 38(8), pp. 34-44 (1995).

[10] D. Harel, A. Naamad. ”The STATEMATE Semantics of Statecharts”. TOSEM 5(4): 293-333 (1996).

[11] G. Mecca, P. Atzeni, A. Masci, P. Merialdo, G. Sindoni. ”The Araneus Web-Base Management System”. SIGMOD Conference 1998: 544-546

[12] D. Schwabe, G. Rossi. ”The Object-Oriented Hypermedia Design Model”. Communications of the ACM 38, 8, 45-46, 1995.

[13] P. Stotts, R. Furuta. ”Petri-Net-Based Hypertext: Document Structure with Browsing Semantics”. TOIS 7(1): 3-29 (1989)

[14] M. A. S. Turine, M. C. Ferreira de Oliveira, P. C. Masiero. ”HySCharts: A Statechart-Based Environment for Hyperdocument Authoring and Browsing”. Multimedia Tools and Applications 8(3): 309-324 (1999).

[15] Y. Zheng, M. Pong. ”Using Statecharts to Model Hypertext”. ECHT 1992: 242-250, 1992.

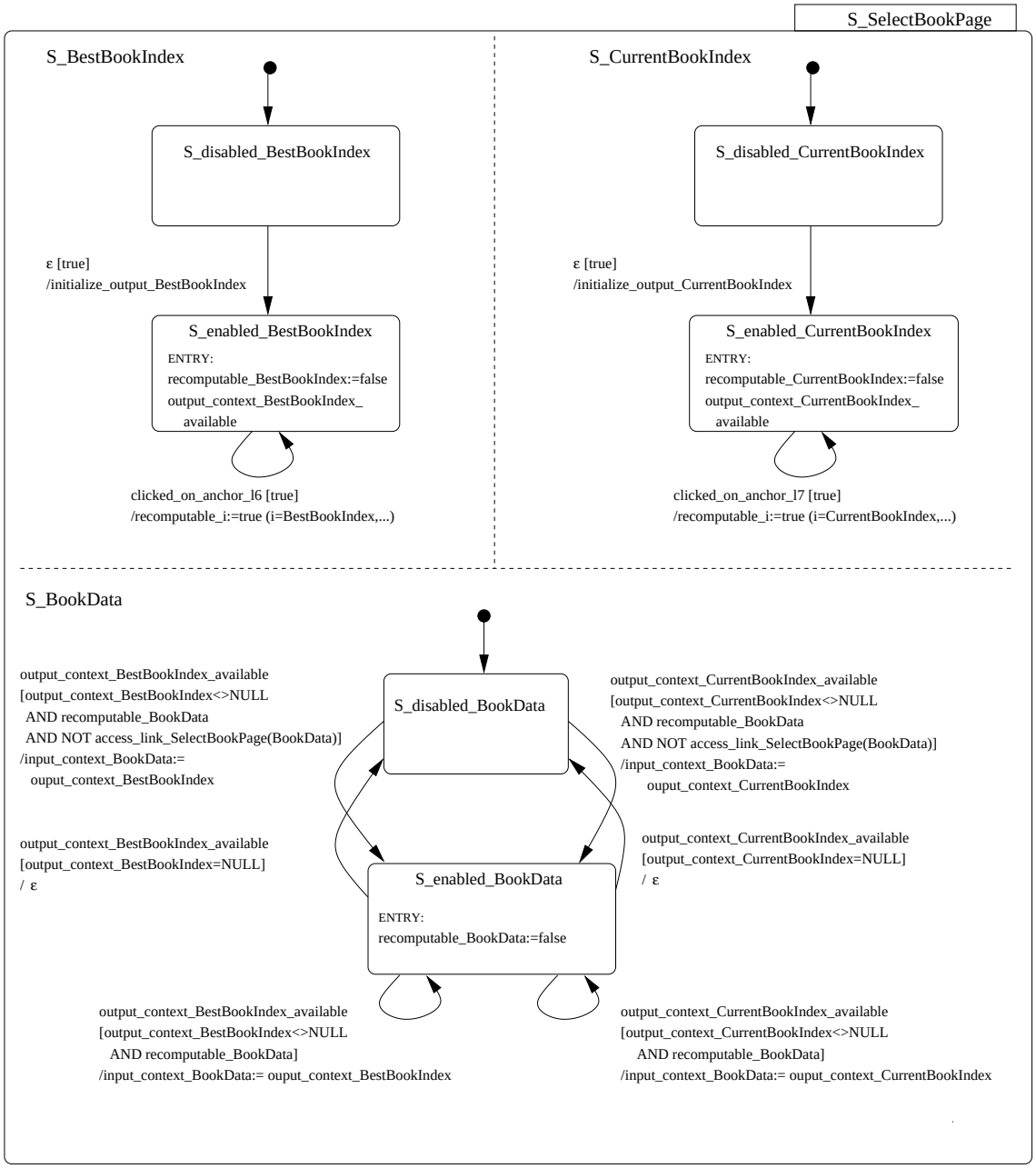

Figure 8: Statechart of a page with racing condition

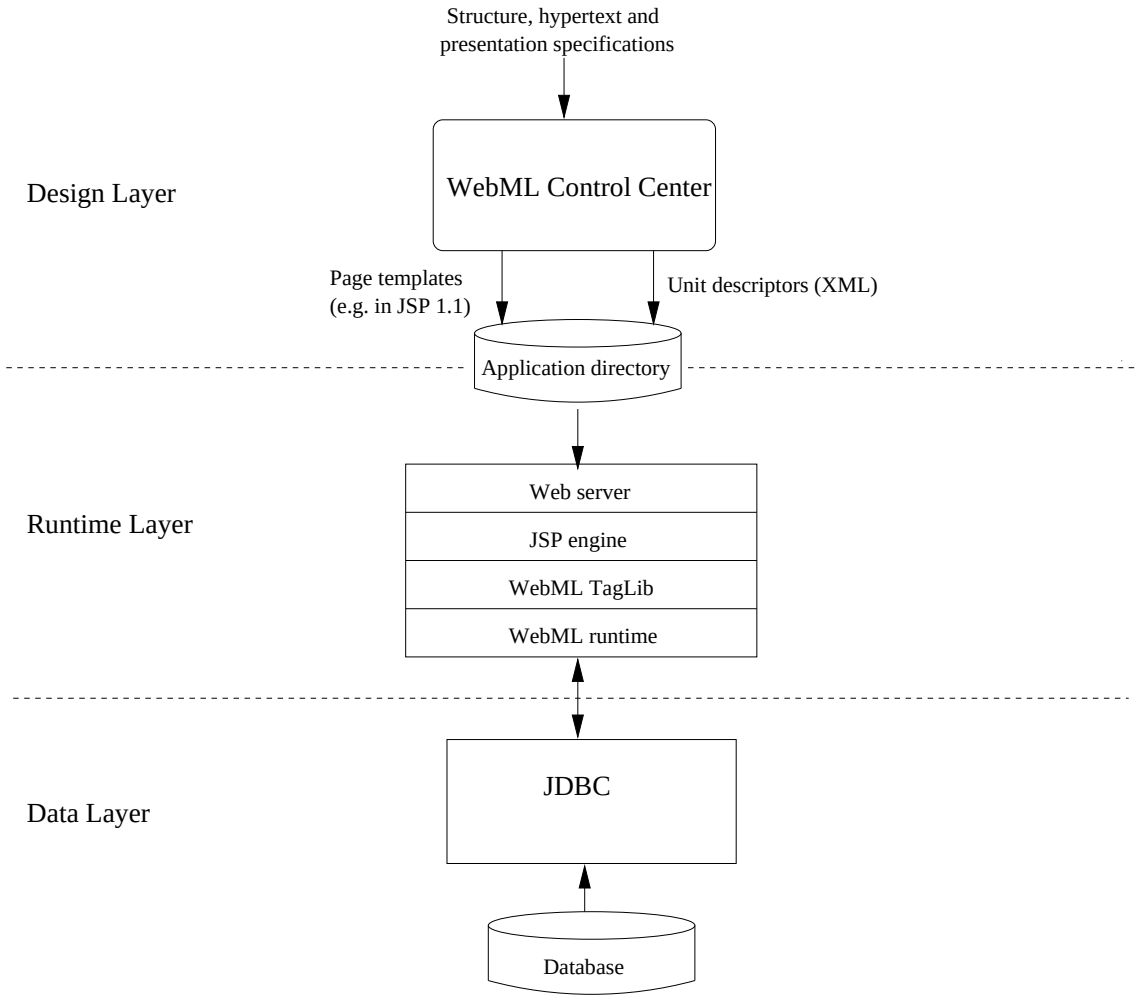

Figure 9: The WebML architecture