# Automatic HTML Generation

# from Formal Hypermedia Specifications

**Rosemeire Shibuya**

NetMaster Consultoria
Av. Brasil, 1885 - Centro
Campinas, SP
Phone: (55)(19) 2437788
Email: rose@netmaster.com.br

**Willie Dresler Leiva**
**Maria Cristina Ferreira de Oliveira**
**Paulo Cesar Masiero**

Departamento de Ciências de Computação e Estatística
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo - Campus de São Carlos
Caixa Postal 668
13560-970 São Carlos, SP, Brazil
Phone: (55) (16) 273-9671
Fax: (55) (16) 273-9751
Email: {wdl,cristina,masiero}@icmc.sc.usp.br

# Automatic HTML Generation

# from Formal Hypermedia Specifications


## Abstract

*HMBS (the Hypermedia Model Based on Statecharts) is a model suitable for specifying highly structured hyperdocuments. HySCharts is an environment that supports the authoring of hyperdocuments based on the HMBS model. It also supports hyperdocument navigation according to a well-defined browsing semantics. In this paper we propose three strategies for automatically deriving an HTML implementation from an HMBS hyperdocument model defined in HySCharts. The resulting implementation can thus be viewed from any standard web browser. The strategies described for HTML code generation may be useful in other situations in which a highly strutured hyperdocument needs to be published in the Web.*

## 1. Introduction

The authoring of hyperdocuments is a highly complex and creative activity, and material produced ranges from completely unstructured to fully structured content. In any case, there is now general agreement that content development can greatly benefit from design methods and development tools. Particularly, there is a demand for tools that can (at least partly) automate the application of design methods. Methods such as HDM - Hypermedia Design Model [Gar93], OOHDM - Object-Oriented HDM [Sch96, Ros97], RMM - Relationship Management Methodology [Isa95, Isa97] and HMBS/M [Car99] provide a sequence of steps and a set of representation models to support the conceptual modeling, design and implementation of hypermedia applications and documents.

Environments that provide computational support for employing design methods in practical situations are valuable tools that allow certain tasks/stages of the methods to be automated, as with the automatic link generation feature proposed in HDM. *Toolbook* and *Director*, for example, are commercial environments for authoring hypermedia applications (they are not supported by any specific design approach). Applications may be created to be accessed from within the environments themselves, or they may be created as independent standalone applications with an embedded browsing semantics, or yet as applications for external publishing in some default media, for example, as a set of HTML files to be viewed in a standard browser. Automated generation of HTML implementations may be particularly critical, as commonly not all the resources and features of the original environment/models may be directly translated into HTML code.

*HySCharts* [Tur99] is an in-house environment developed to support hypermedia authoring with the HMBS model (*Hypermedia Model Based on Statecharts*) [Tur97, 98]. It is a standalone tool that embeds a browsing module for analyzing the specifications created. One of the major goals of the environment, however, is to automatically generate Web-based (HTML) applications directly from the HMBS models created. However, a major feature of HMBS is its ability to model hierarchically structured hyperdocuments. This poses a major problem because HTML documents are essentially planar entities and, therefore, there is a structural incompatibility between both approaches for modeling hyperdocuments. Frames provide a limited solution to introduce hierarchy into HTML documents, and links are also often used to introduce a notion of hierarchy into hyperdocuments.

This paper describes how the HTML code-generation problem has been dealt with, so that a Web implementation of a hyperdocument may be produced automatically from its HMBS specification created within *HySCharts*. The solutions discussed may be adopted in other situations where highly structured hyperdocuments are to be transformed into a set of HTML documents. Three approaches are described for tackling the problem, and their benefits and shortcomings are discussed. An overview of the code generator implemented for automatically generating HTML files from the HMBS specifications in the *HySCharts* environment is provided.

This paper is organized as follows: Section 2 provides the required background on the HMBS model and its supporting environment *HySCharts*. Section 3 describes a case study that is explored in Section 4 to illustrate the possible approaches for generating HTML code from HMBS specification models. The advantages and limitations of each approach are also discussed in Section 4. Concluding remarks are presented in Section 5.

## 2. Background: the HMBS Model and the *HySCharts* Environment

### 2.1 HMBS Model

Statecharts are a visual formalism that extend finite state machines and state transition diagrams by incorporating the notions of hierarchy, orthogonality (concurrency), a broadcast mechanism for communication between concurrent components, composition, aggregation and refinement of states [Har87, Har87b]. A HMBS model [Tur97, Oli00] represents a hyperdocument using three classes of objects, namely structural, navigational and presentation objects. The organizational structure is defined by an underlying statechart comprising states, transitions and events, which are the structural objects of the model. The hyperdocument's navigational structure is provided by pages, links (although these are not explicitly represented in the model) and anchors, which are the model's navigational objects. Presentation objects consist of channels invoked during browsing to interpret and exhibit information contained in pages. A set of statechart states is mapped into pages that correspond to the hyperdocument nodes containing chunks of information, and statechart events are associated to anchors representing link sources. Anchor activation results in the triggering of an associated event in the underlying statechart, thus firing the corresponding transition from a source state (mapped to the link's source page) to a destination state (mapped to the link's target page). Thus, the

operational semantics of statecharts [Har87] is used to define a browsing semantics over an HMBS model.

A HMBS hyperdocument model *H* is a 7-tuple H = <*ST*, *P*, *m*, L, *pl*, *ae*, N>, in which:

a) ST represents the hyperdocument's underlying statechart, and is defined by ST = <S, ρ, ψ, γ, δ, V, C, E, A, R, T> denoting, respectively: the set of states, the hierarchy function, the decomposition type function, history function, default function, set of expressions, set of conditions, set of event expressions, set of actions, set of labels and, finally, the set of transitions;

b) P is a finite set of pages that define the contents of the hyperdocument. Each page *p* ∈ P is conceptually defined by the triple <c, t, $Anc_p$>, such that "c" is the information content expressed in any static (such as text, graphics or image) or dynamic (video, audio, or animation) media; "t" represents a title that uniquely identifies the page; and "$Anc_p$" defines a collection of anchors contained in the page. The set P may also include a special *null page* with no associated contents, titles or anchors;

c) m: $S_S \rightarrow$ P denotes a value function mapping states of set $S_S$ ($S_S \subset$ S) into hyperdocument pages p ∈ P. The subset "$S_S$" is defined by $S_S$: {x ∈ S | ψ(x)=OR ∨ ρ(x) = ϕ}, i.e., $S_S$ is the subset of S comprising basic states and OR states. AND states are not mapped into pages, being used solely to specify concurrent presentation of information;

d) L defines the set of presentation channels, which are abstract devices used to interpret and exhibit the information contained in pages. Channels allow authors to specify spatial coordination parameters and control global presentation attributes;

e) pl: P → L defines a visualization relationship that associates each hyperdocument page p ∈ P with a single channel l ∈ L that is able to interpret it;

f) ae: $Anc_p \rightarrow$ E defines a function that associates anchor elements "$a_p$" ($a_p \in Anc_p$) to statechart events controlling the firing of transitions. An imposed restriction is that an event in a transition label must be unique to have an associated anchor. Moreover, the page containing the anchor must be mapped to a state that is in the source set of the transition or, alternatively, it may be mapped to a sub-state of a state in this source set. Transitions and their corresponding event labels may be reused to define anchors in different navigational contexts; and

g) N ($N \geq 0$) defines the hyperdocument visibility level. The author may use the value of N to define the hierarchy depth when displaying pages during navigation, as described in the following section.

A browsing section implies in the execution of the hyperdocument's underlying statechart model by a statechart engine that takes it from an initial configuration to a final one. The hyperdocument pages displayed at a certain stage during browsing are those associated to the basic states in its underlying statecharts' current state configuration, plus those pages associated to non-basic states that satisfy the author-specified visibility level. Thus, when an anchor within a page is activated the hypermedia engine takes the following actions:

a) The event associated to the anchor (mapping "ae") is generated, thus triggering the transition associated to the corresponding link;

b) All states in the target sets of the fired transitions are activated, thus generating the next state configuration and disabling the previous one. In sequence, the logical windows associated to pages corresponding to states in the previous configuration and also to their ancestral states (according to the given level N) are removed from the display; and

c) The hyperdocument's new context configuration is generated. It is composed by the pages associated to the states in the new current state configuration and to those that satisfy the visibility level N. For a hyperdocument with "N" set to zero, all the pages associated to the atomic states within the current active state configuration are displayed. If "N" were set to one, the pages exhibited include the same ones defined for N = 0, plus those associated to the immediate OR ancestor states of the states in the current state configuration, and so on for greater values of N. The presentation channels (mapping function "pl") for each page in the new context configuration are invoked for these pages to be exhibited.

### 2.2 The *HySCharts* Environment

*HySCharts* supports the creation and interpreted execution of HMBS specifications, and consists of an application layer, a structuring and a storage layers [Tur99]. The application layer comprises an authoring and a browsing modules, in addition to a module that implements statechart edition and simulation tools. The structuring layer defines the hyperdocuments' structure in terms of the HMBS structural, navigation and presentation objects, stored in databases managed through functions in the storage layer. To manage the structural elements it uses *StatSim* — *Stat*echart *Sim*ulator [Mas91] —, a system with a

graphical interface that allows the automated definition and validation of statechart software specifications.

In the authoring module functionality is provided for authors to specify the structural, navigational and presentation objects and to analyze the navigational structure of the proposed hyperdocument. A graphical statechart editor may be used to create the underlying statechart, as illustrated in Figure 1, and a graphical editor for HMBS objects provides tools for managing pages, anchors and channels. In the Pages menu a "*Create Page*" option prompts a window for the specification of a page by defining its attributes *title*, *presentation channel*, *media type* and *content*. The page content is provided by a file ("*File*") identified by its name. Each page must be associated to a single (author specified) presentation channel that defines its media type (text, image, audio or video) and some visualization properties. Once such information is provided, the author may associate the page to its corresponding structural object (state), thus creating an instance of the "m" mapping between states and pages. Also available are operations for querying pages associated to a given state, and for verifying properties of the pages created for the instantiated document, such as title, media type and name of its associated state.

Having created the pages and associated them to states, the author may specify the anchors for enabling the hyperdocument links. Anchors belonging to textual pages are visually displayed as text strings with underlined and bold typeface and those in pages containing image, video or audio anchors are presented as buttons. To define an anchor object within a page the author chooses its identifying keyword and then selects its associated event, which must belong to a transition label in the statechart. In this way an instance of the "ae" relationship is created, and the navigational objects anchor and link are defined.

The structuring layer also stores the channel objects. They support the definition of high-level presentation attributes for pages, such as font style and size, plus window background color in the case of text media channels, and spatial coordination attributes such as window position, height and width. Default channel classes are defined which may be instantiated, according to the authors' needs, using the mechanisms of inheritance and specialization/generalization. For instance, one may employ two different channels for presenting text, one using a font "*Times New Roman*" and the other using a font "*Arial*". The use of channels allows a single document to be presented in different manners by redefining its presentation attributes rather than its navigational structure.

HMBS supports the analysis of some properties and allows the identification of structural inconsistencies and problems related to presentation and navigation. The authoring module includes functions for analysis and verification of the HMBS hyperdocument specifications, which operate based on the generation of a reachability tree that provides subsidies for detecting specification anomalies. Some dynamic properties of statecharts have been defined in the context of hyperdocuments: page reachability from any given context configuration, reinitiability, deadlock during navigation, vivacity of navigational links and valid sequences of anchors [Tur97, Oli00].

To make the hyperdocument available for readers, HMBS specifications are interpreted and "executed" in the browsing module of *HySCharts*. This module embeds a statechart engine that receives external events (generated by user actions) and interprets them according to the statechart semantics, generating a new statechart active configuration and a corresponding presentation. Navigation may be initiated from an initial default configuration; alternatively the reader may choose the title of a specific page of interest from an index that provides direct access to pages from its titles. In this way, the author can "execute" the hyperdocument specification and observe the effect in its graphical structure representation.

This is useful to support analysis of the hyperdocument's behavior during navigation, allowing identification of problems in the specification. The graphical representation available during browsing also helps improving reader comprehension and reducing disorientation. Figure 1 depicts a snapshot of a browsing section in *HySCharts*, where the contents of a page associated to the basic state named 'Info', assuming a visibility level equal to zero (N = 0), is shown. This state corresponds to a valid configuration for this statechart that defines the structural elements of an example application (see Section 3), whose graphical representation is also shown in the background window. State 'Info' is shaded because it is a currently active state. The *Show-Hview* window allows the reader to browse the contents of pages associated to states up or down in the statechart hierarchy.

## 3. Case Study

To illustrate the process of code generation we shall use as a hypothetical application the site of a public telephony company. Figure 1 provides the visual statechart representation of a partial HMBS model for such an application. The reader is first presented an introductory page containing a general description of the company and its logo (associated to sub-state

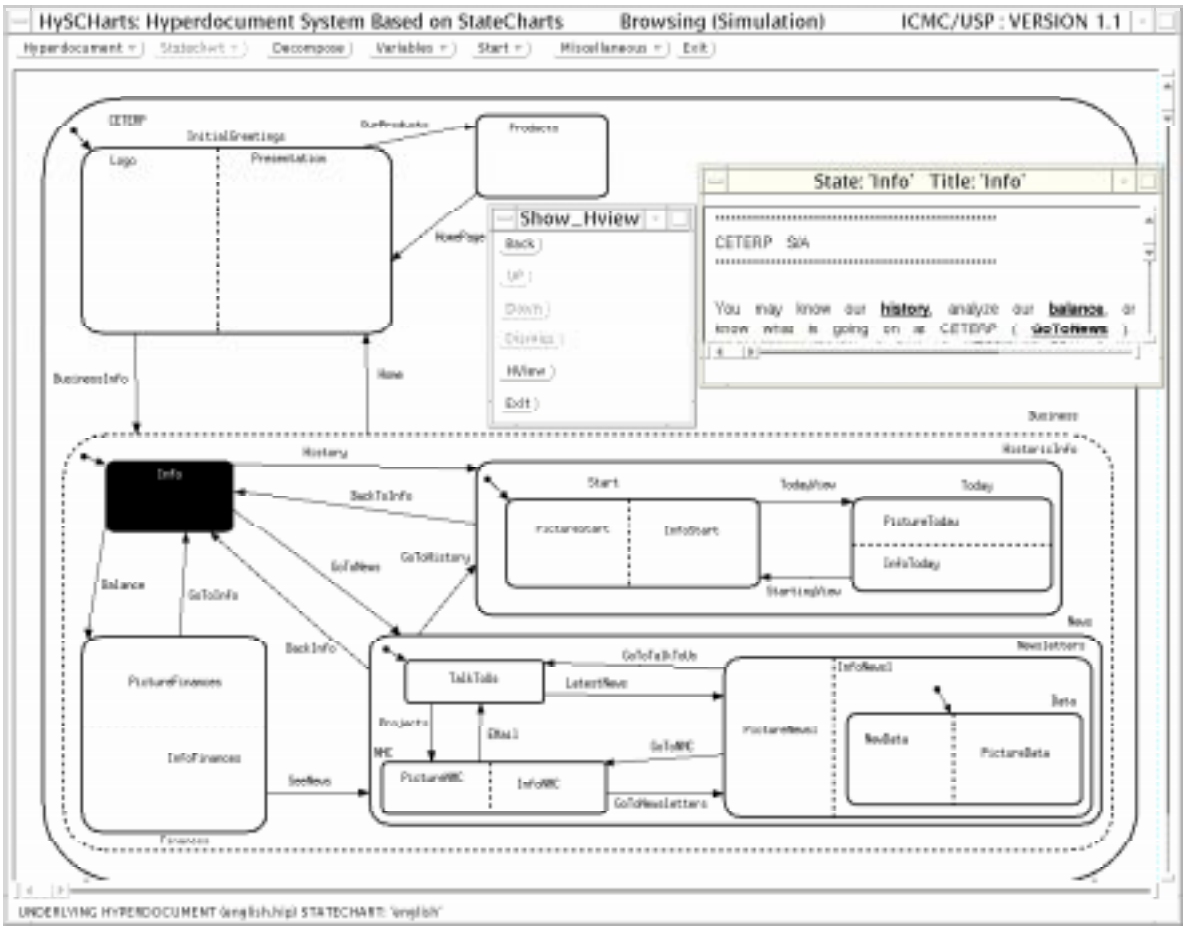


Figure 1 - A snapshop of a browsing section in the *HySCharts* environment containing the statechat representation of the CETERP application discussed in Section 3.

'InitialGreetings', of the root 'CETERP'). From that page she may follow a link to see additional information on the company (transition labeled 'BusinessInfo').

After an initial page with general information (sub-state 'Info' of state 'Business'), the reader may follow links taking to three different navigational contexts: one displays information on financial matters (transition labeled 'Balance'), another containing a history of the company (transition labeled 'History'), and a third one presenting news for the public (transition 'GoToNews'). If the reader chooses to see the news, she is first presented an introductory page (associated to state 'TalkToUs'), from which she may follow links to the news (transition labeled 'LatestNews' taking to state 'Newsletters') or to a description of ongoing projects (transition labeled 'Projects' to state 'NMC').

Table I provides the *m* mapping for this HMBS model, that associates basic and OR states with page contents, provided as external files in *HySCharts*. In this example we adopted as page titles the same names used for the states. Shaded areas indicate AND states that are

not associated to pages. For convenience and latter reference, a numerical identifier is also associated with each state.

Table I - Association of content files and identifiers to OR and basic states. Shaded lines indicate AND states, which are not associated to pages.

| Identifier | State | Page content (file) |
|---|---|---|
| 0 | CETERP | ceterp.txt |
| 1 | InitialGreetings | None (no associated page) |
| 2 | Presentation | presentation.txt |
| 3 | Logo | logo.gif |
| 4 | Business | Business.txt |
| 5 | Info | info.txt |
| 6 | Finances | none |
| 7 | TxtFinanc | finances.txt |
| 8 | TitlePictFinances | finances.jpg |
| 9 | HistoricInfo | histInfo.txt |
| 10 | Start | none |
| 11 | InfoStart | start.txt |
| 12 | PictureStart | start.jpg |
| 13 | Today | none |
| 14 | InfoToday | today.txt |
| 15 | PictureToday | today.jpg |
| 16 | News | news.txt |
| 17 | Newsletters | none |
| 18 | InfoNewsL | infonewsl.txt |
| 19 | NewsletterPicture | newsletter.jpg |
| 20 | Data | none |
| 21 | PictureData | PictData.jpg |
| 22 | NewData | newData.txt |
| 23 | TalkToUs | talkToUs.txt |
| 24 | NMC | none |
| 25 | InfoFromNMC | NMC.txt |
| 26 | PictureFromNMC | NMC.jpg |
| 27 | Products | products.txt |

## 4. HTML Code Generation from HMBS

### 4.1 Code Generation Strategies

As described in Section 2, each hyperdocument node corresponds to a page with an associated state as defined by the *m* mapping. Each page has a title, content and defined anchors. Considering pages in isolation, the code generation process has to recover page content and anchors, that will be converted into HTML files with defined content and links. However, different navigation strategies may be used to exihibit the page contents. Three navigation approaches were considered in this work and are described in the following: one that provides direct access to individual pages of the application from an initial index; one that provides a planar view of the application, disregarding the statechart structure; and one that

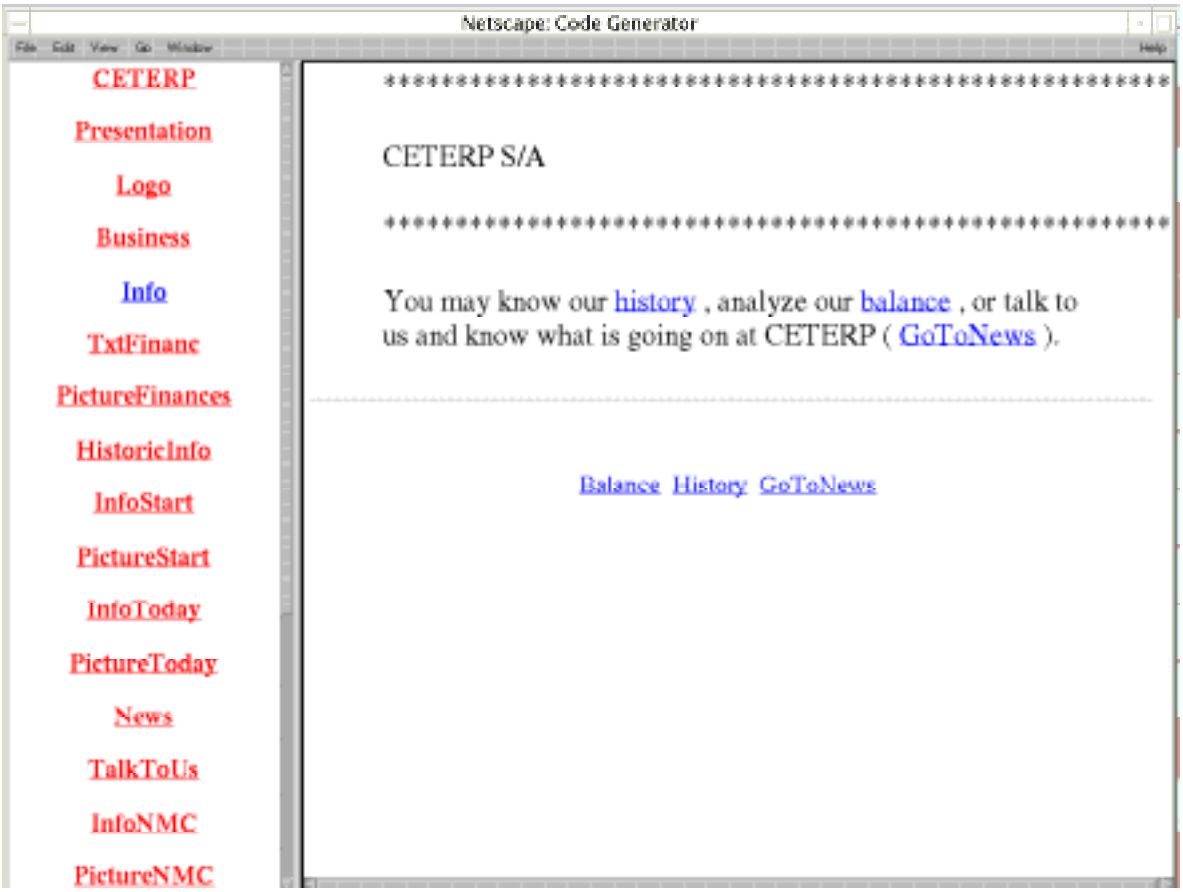


Figure 2 – Overall index for the CETERP application (left frame) and a page with links generated from the *ae* mapping of HMBS (rigth frame), that associates anchors to events.

provides a hierarchical view that preserves the hierarchical structure defined by its underlying statechart and the browsing semantics defined by HMBS.

The first approach consists simply of providing an overview index to all the pages defined by set P recovered from the underlying HMBS model. Two types of display configurations are possible. In the first one, once activating a link in the index the reader sees the content of the corresponding page, from which she may follow the application links embedded in the page. Alternatively, the reader may see, in addition to the content of the corresponding page with its embedded links, structural links that correspond to anchors associated to the statechart events defined for that page by the *ae* mapping. This navigational view is useful to help in the process of verifying page contents.

For the case study described in Section 3, such a view of the application would appear as depicted in Figure 2. It shows the index on the left frame, and the contents of the page associated to state ‘Info’ on the right frame. The links generated by association of anchors to

events ‘History’, ‘Balance’, and ‘GoToNews’, responsible for triggering outgoing transitions from state ‘Info’, are also shown. They appear in the page associated to ‘Info’ as the textual anchors named after the events ‘History’, ‘Balance’ and ‘GoToNews’, respectively. Another possibility would be to display the contents of the page associated to a state showing no links associated to statechart events. The index solution warrants direct access to the contents of all defined pages without preserving any context embedded in the underlying statechart structure, and it is not of much practical value to end users if hierarchical context is important.

An alternative navigational view of the application is based on an analysis of the hierarchical structure of the underlying statechart in the HMBS model. The statechart hierarchy is recovered to create a planarized hierarchical view: the reader may navigate from a given page to its father, child and sister pages, as established by the statechart hierarchy. The overall organization architecture for navigation provides access to all pages, starting from the top most one associated to the root state (state ‘CETERP’, in this case). Figure 3 shows the planar view generated for the CETERP application. The numbers in the figure refer to the numerical identifiers established in Table I, and the structural links starting from the page associated to the root state ‘CETERP’ are depicted. The arrows indicate the structural links

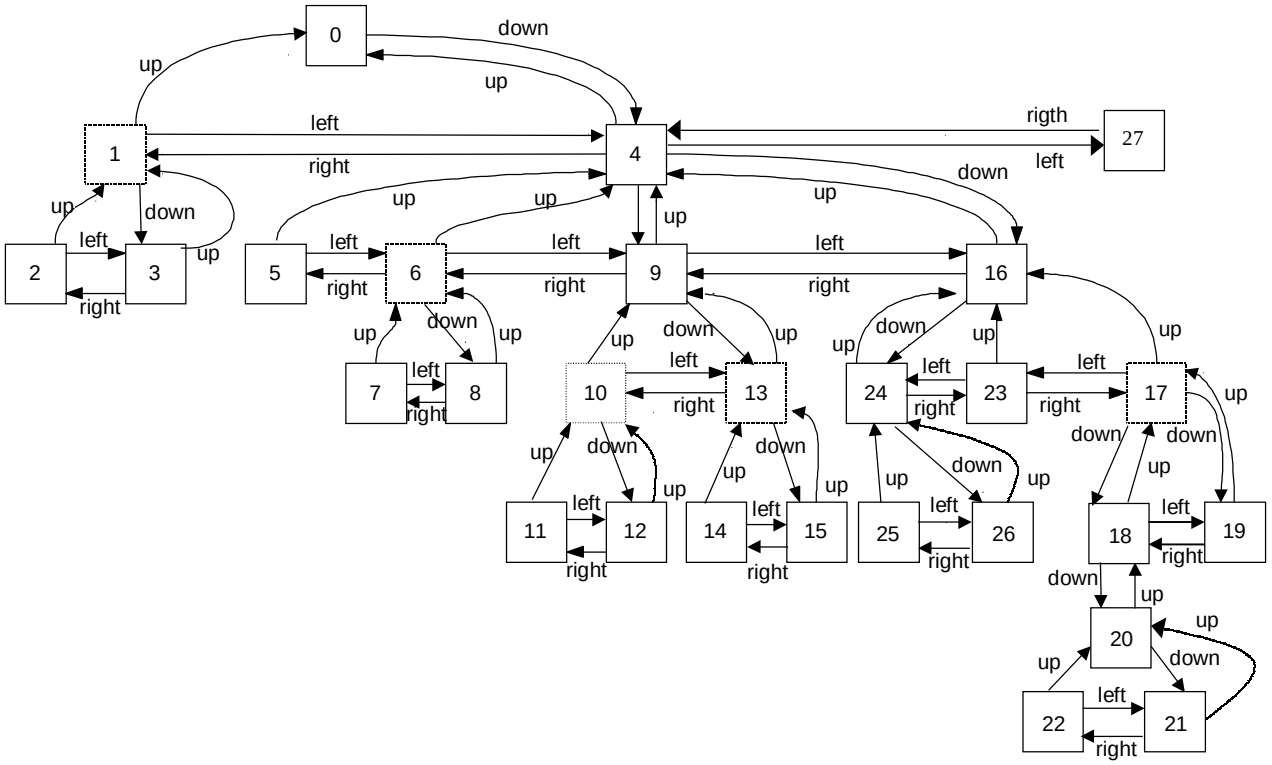


Figure 3 – Planar view of the CETERP example. Boxes represent pages, and the numbers are the page identifiers introduced in Table I.

generated automatically, with the *up*, *down*, *left* and *right* labels meaning that access is being provided to the page associated to the parent state in the statechart, or to the ones associated to first child, first sister to the left, and first sister to the right, respectively (if they exist).

Although this solution does provide information regarding the hierarchy of a page within the overall structure, it does not provide all the contextual information of the hyperdocument hierarchical organization, as the browsing semantics of HMBS is not preserved. Unlike it is done in *HySCharts* pages are not shown as part of a configuration defined by a valid statechart configuration.

Figure 4 illustrates the code generated for the planar view approach as seen from a standard browser. It shows the content of the page associated to state 'Info', that corresponds to identifier 5 in Table I. Observe that the generator created the anchors *up*, *down*, *right* and *left*, that provide access to the pages associated to the father, child and sister states of the state associated to the current page. For the page associated to state 'Info'only anchors *up* and *left* are enabled, as it has no child or right sister state (see Figure 3). Because the browsing semantics is not preserved, features such as control of the visibility level for the application, allowed in *HySCharts*, are lost.

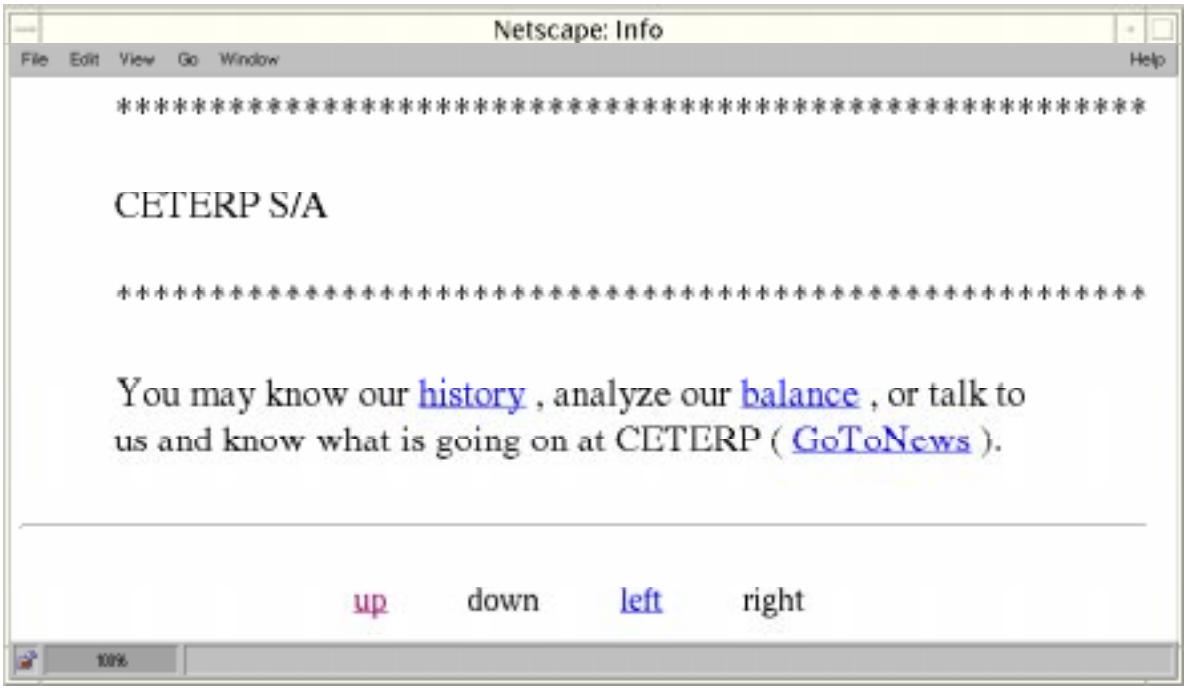


Figure 4 – Code generated for the page associated to state 'Info' in a planar view of the CETERP application.

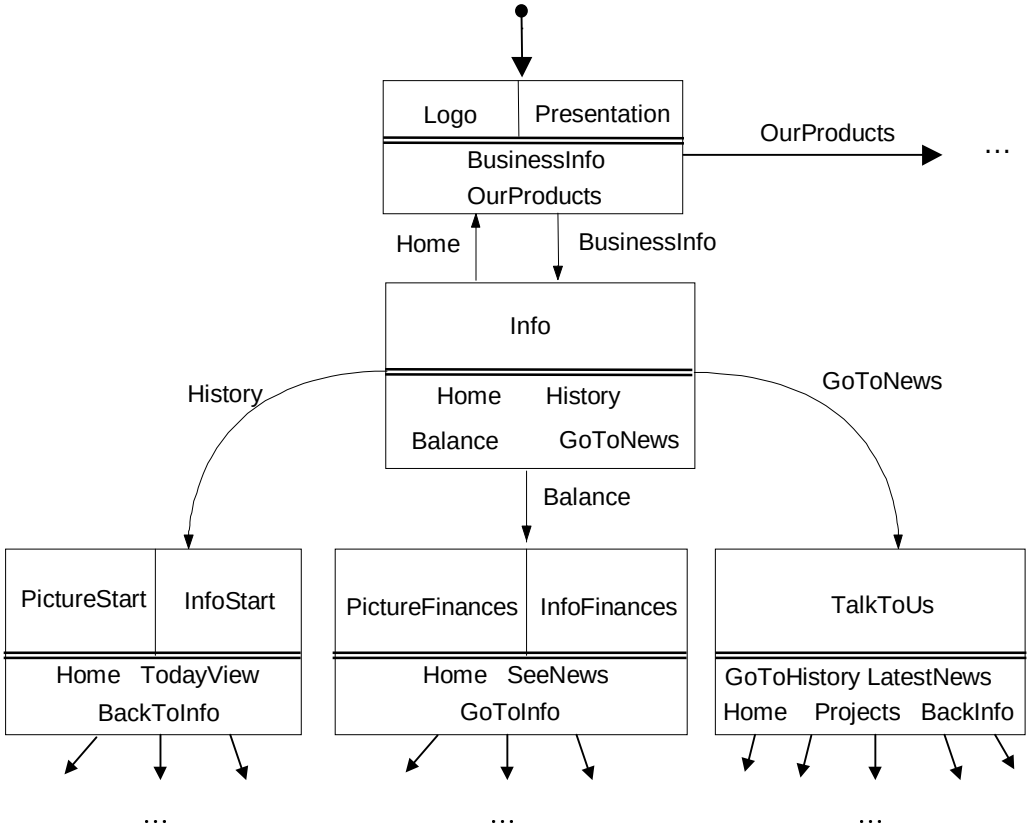


Figure 5 – Hierarchical view of possible configurations of the CETERP application starting from $SC_1$ = {Logo, Presentation, InitialGreetings, CETERP}.

The third solution is the most interesting one, because it keeps the browsing semantics defined for HMBS. It uses the operational semantics of statecharts to provide a hierarchical navigational view of the application in which each page is exhibited as part of a user configuration, the same approach adopted in the *HySCharts* environment. As mentioned in Section 2, navigation in *HySCharts* occurs by simulation of the application's underlying statechart in response to user events, and the set of pages displayed at a given moment is obtained from the current statechart configuration. This concept is preserved in the hierarchical navigation approach: the set of pages observed by the reader at a given moment comprises those pages associated to states in a valid statechart configuration. Each user-generated event produces a new statechart configuration, with a new set of pages being shown. The set of possible configurations is determined after an exhaustive execution of the underlying statechart model that recovers all reachable statechart configurations. User configurations may be derived from each statechart configuration from the HMBS mapping functions. The visibility level adopted for display is also considered when defining user

configurations, but it cannot be changed after the code is generated. Thus, one may generate several HTML applications from the same specification, using different levels of N.

Figure 5 depicts the structure of the hierarchical view of the CETERP application from an initial state configuration given by $SC_1$ = {Logo, Presentation, InitialGreetings, CETERP}. The upper area in the rectangles display the active states (or, equivalently, the pages associated to them), and the lower areas display the anchors contained in the corresponding pages. The structure has been derived considering a visualization level set to zero, that is, only pages associated to the basic states in the statechart configuration are displayed. In this case, it means the contents of the pages associated to states 'Logo' and 'Presentation'. From this one, the reader may reach another configuration $SC_2$ = {Info, Business, CETERP}. From $SC_2$ three other new configurations are possible, namely $SC_3$ = {PictureStart, InfoStart, Start, HistoricInfo, Business, CETERP}, $SC_4$ = {PictureFinances, InfoFinances, Finances, Business, CETERP}, and $SC_5$ = {TalkToUs, News, Business, CETERP}, all depicted in the figure. For this particular example, the reachability tree provides thirty one possible state configurations.

Figures 6 and 7 show examples of pages generated from the HMBS model of the CETERP application with visibility levels set to zero and one, respectively. Configurations involving multiple pages are exhibited using frames. In figure 6 the reader sees the pages associated to the state configuration $SC_1$ (only the pages associated to the basic states 'Logo' and 'Presentation', as the visibility level has been set to zero).


Figure 6 - Code generated for the user configuration associated to statechart configuration $SC_1$ - {Logo, Presentation, InitialGreetings, CETERP}, N = 0.

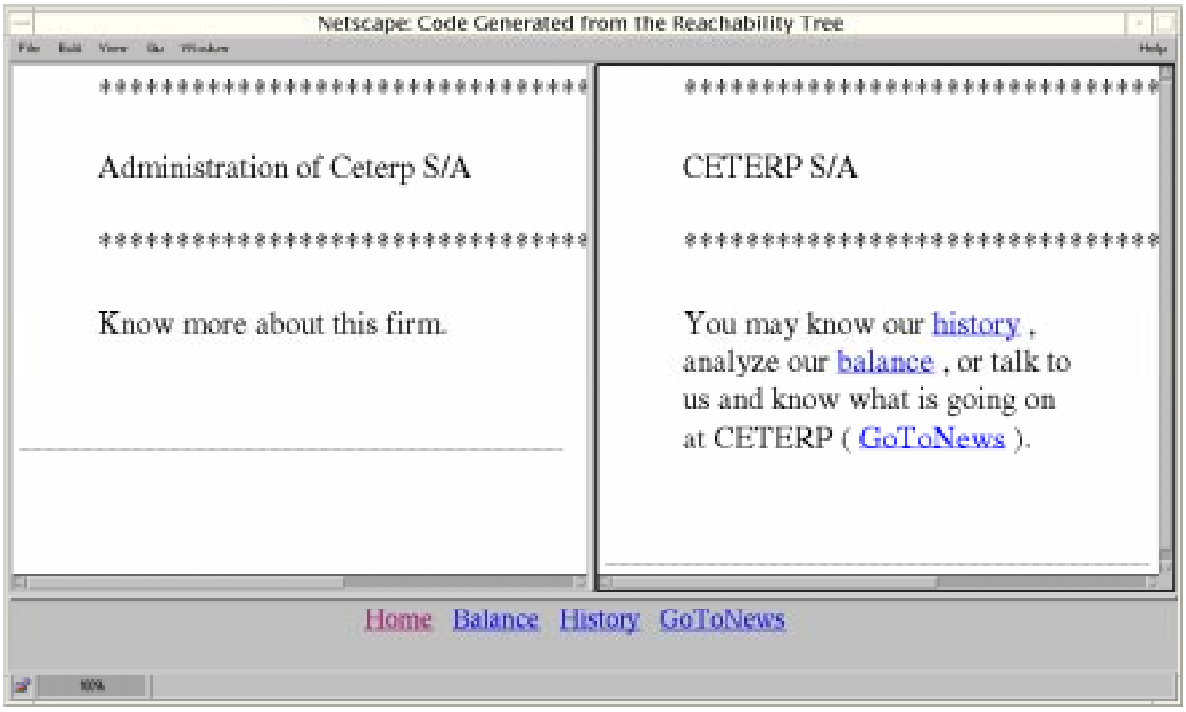


Figure 7 - Code generated for the user configuration associated to statechart configuration $SC_2$ - {Info, Business, CETERP}, N = 1.

The lowermost frame shows the links available from this page, determined from the HMBS *ae* mapping that maps anchors to statechart events. For this page, the links are the ones mapped to the statechart events labeled 'BusinessInfo' and 'OurProducts'. The first one takes from configuration $SC_1$ to configuration $SC_2$, which is the one depicted in Figure 7. In this case, the contents of both the page associated to the basic state 'Info' (left frame) and the page associated to its parent state 'Business' (rigth frame) are displayed, due to the visibility level set to one. The links available in the lowermost frame result from the HMBS *ae* mapping that maps anchors to statechart events. For this pages, such anchors have been mapped to statechart events labeled 'Home', 'Balance', 'History', and 'GoToNews' (refer to Figure 1). All such events lead to valid state configurations reachable from $SC_2$.

### 4.2 HTML Generation

The code generator has been implemented in C++ and the *Xview* graphics library for Unix platforms. It may be used as a standalone tool or integrated to the *HySCharts* environment. It presents a graphical interface which supports user intervention in two manners: selecting the *HySCharts* HMBS model to be converted and setting the visibility level to be adopted in the code generation process. The third approach for generating code, that preserves the browsing semantics assigned to HMBS, requires, prior to the code

generation itself, the construction of the database describing the set of reachable configurations. Such information may be obtained from the reachability tree derived from the statechart model [Mas94].

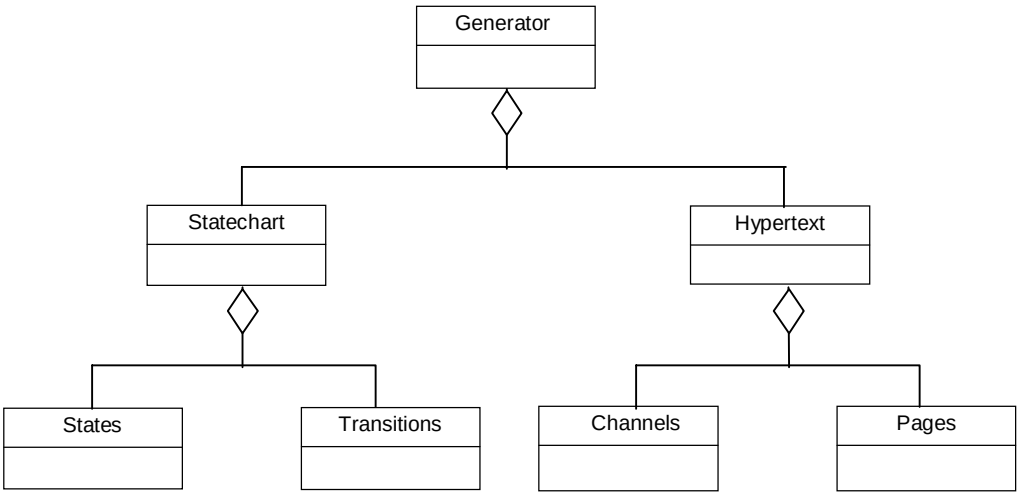


Figure 8 – Class structure of the code generator.

A set of classes has been defined in the code generator, as shown in figure 8, to represent the structural information obtained from *StatSim* and *HySCharts* and also information derived by the code generator itself. These are *states* and *transitions*, which compose the aggregate class *statechart*; and *channels* and *pages*, which compose the aggregate class named *hypertext*.

The HTML code generation process is split into three stages. In the first one, textual pages defined within *HySCharts* are mapped into HTML structures, and anchors and links are identified. In the second stage, code is generated to support the three navigational approaches previously identified, whereas in the third one information is stored for later handling by readers. These stages demand recovering of three kinds of information: the statechart structure kept within the *StatSim* environment; the contextual and navigation structure kept within *HySCharts*; and the set of reachable configurations for the application, which is recovered from the reachability tree. This three stage architecture is depicted in Figure 9. Each of the above mentioned stages is described in the following.

The first stage of the code generation process converts textual pages authored within *HySCharts* into HTML documents. The conversion process retrieves the set {E,H}, where E stands for the set of valid state configurations for the statechart, and H stands for the set of possible user configurations within the associated hyperdocument. A lexical analyzer has

been implemented to recover from the page contents the information required to produce equivalent HTML code, particularly anchoring information which is mapped into HTML links.

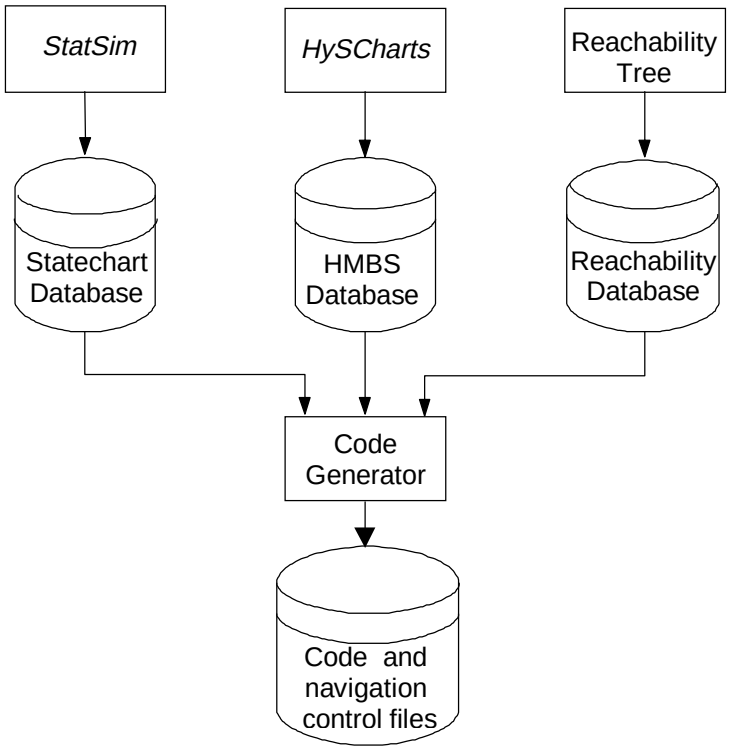


Figure 9 – Architecture of the code generation program.

The codification of a simple textual page created within *HySCharts* is exemplified in Table II. The first line depicts the textual content of a page given by file *info.txt*, and the second one depicts the corresponding HTML code produced by the code generation tool, kept in file *infotb.html*. Symbols '<<' and '>>' delimit anchors within a *HySCharts* page. From information recovered from the application's HMBS model kept within *HySCharts*, the code generator identifies the links associated to the anchors, as well as its exhibition attributes, and produces the appropriate HTML tags. Depending on the coding strategy adopted, additional links will be inserted. For example, to exhibit a planar view of the page content depicted in Table II, the generation will insert additional anchors *up*, *down*, *left* and *right* at the end of the page, linking to the appropriate files.

The code generator is also capable of creating HTML code corresponding to other content types, such as image, audio and video. The name of the HTML file created is the same of the original file, plus some control characters. In the example of Table II, the 't'

indicates textual content (as ‘i’, ‘a’, ‘v’ would indicate image, audio, or video content, respectively), and the ‘b’ indicates that the page corresponds to an index-based navigational view, with the inclusion of page anchors mapped to statechart events into HMBS (different letters are used for conveying other information [Shi98]). For each content type a set of coding attributes is kept. The author may manually modify the content of the HTML pages created.

Table II - Conversion of a *HySCharts* textual page into an equivalent HTML page (with the same contents and links).

| | |
|---|---|
| *File: info.txt* | ********** CETERP S/A ********** You may know our <<history>> , analyze our <<balance>> , or talk to us and know what is going on at CETERP ( <<GoToNews>> ). |
| *File: infotb.htm* | <HTML><br><HEAD><br><TITLE> Info </TITLE><br><META HTTP-EQUIV='Content-Type'<br>CONTENT='text/html; charset=iso-8859-1'><br><META NAME='GENERATOR'<br>CONTENT='Hypertext Code Generator Based on HMBS'><br></HEAD><br><BODY TEXT= #000000 BGCOLOR= #FFFFFF><br><font size=4 face="TIMES" ><br><P style="margin-left:2cm; margin-right:2 cm "><br>********** </P><br><P style="margin-left:2cm; margin-right:2 cm "> CETERP S/A </P><br><P style="margin-left:2cm; margin-right:2 cm "><br>********** </P><br><P style="margin-left:2cm; margin-right:2 cm "><br>You may know our <a href="histInfota.htm">history</a> , analyze our <a href="EstN06a.htm">balance</a> , or talk to us and know what is going on at CETERP ( <a href="newsta.htm">GoToNews</a>). </P><br><P style="margin-left:2cm; margin-right:2 cm "> </P><br></FONT><br><BR><BR><HR><br></BODY><br></HTML> |

The main output of the previous stage is a set of HTML documents with their embedded application links as defined in *HySCharts*. The second stage of the code generation process comprises the definition of the navigation strategies discussed in Section 4.1. Generating the indexed view involves a simple parsing of the set of pages, recovering their content while simultaneously creating the index structure with the page titles. For the planar view the process recovers the statechart hierachical tree structure to establish the proper hierarchical links to be added to the page being shown.

In the hierarchical navigation approach each page is exhibited as part of a user configuration, as in the *HySCharts* environment. The set of possible configurations is determined after an exhaustive execution of the underlying statechart model that recovers all reachable statechart configurations. From the HMBS associations, user configurations may be derived from each statechart configuration. The visibility level adopted for display is also considered when creating user configurations, and HTML pages may be generated for different values of the visibility level N. The user must specify the visibility level for navigation prior to starting the code generation process.

## 5. Concluding Remarks

This paper described an approach for automatic HTML code generation from formal HMBS hyperdocument specifications. The code generation process enables designers of hyperdocuments to benefit from the use of a formal model assisted by the *HySCharts* computational environment and still produce a final application accessible by standard Web browsers. In this way, it illustrates a possible approach for bridging the gap between the design and the implementation stages for developing hypermedia applications in general, and Web applications in particular. Additionally, the HTML application created by the code generator may be taken as a first prototype of a family of applications that may be improved by working directly on the HTML code using appropriate tools.

## Acknowledgments

The authors acknowledge the support of FAPESP and CNPq, Brazil.

## References

[Car99] M.R. Carvalho, M.R., M.C.F. de Oliveira, and P.C. Masiero; “An Object-Oriented Method for Hypermedia Design”, *Proc. V Brazilian Symposium on Multimedia and Hypermedia Systems* (SBMIDIA’99), June 15-18, Instituto de Informática, UFG, Goiânia, pp. 43-62.

[Dru89] D. Drusinsky, D. Harel, “Using Statecharts for Hardware Description and Syntesis”, *IEEE Transactions on Computer-Aided Design*, Vol. 8 (7), pp.798-806, 1989.

[Gar93] F. Garzotto, P. Paolini, and D. Schwabe, “HDM — a model-based approach to hypertext application design”, *ACM Transactions on Information Systems* Vol. 11, No. 1 (January), 1993, pp. 1-26.

[Har87] D. Harel, “Statecharts: a visual formalism for complex systems”, *Science of Computer Programming*, Vol. 8, No. 3, pp. 231-274, July 1987.

[Har87a] D. Harel, A. Pnueli, J.P. Schmidt, and R. Sherman, "On the formal semantics of statecharts", in *Proc. II IEEE Symposium on Logic in Computer Science*, pp.54-64, 1987.

[Isa95] T. Isakowitz, E.A. Stohr, and P. Balasubramanian, "RMM: a methodology for structured hypermedia design", *Communications of the ACM*, Vol. 38, No. 8 (August), 1995, pp. 34-44.

[Isa97] T. Isakowitz, A. Kamis, and M. Koufaris, "Extending the capabilities of RMM: Russian Dolls and Hypertext", in *Proc. XXX Annual Hawaii Int. Conf. on System Sciences*, (HICSS'97), 1997.

[Mas91] P.C. Masiero, R.P.M. Fortes, and J.E.S. Batista Neto, "Editing and simulating behavioral aspects of real-time systems", in *Proc. XVIII Integrated Hardware and Software Seminar (SEMISH)*, Santos, Brazil, August 5-9 1991, pp. 45-61 (in Portuguese).

[Mas94] P.C. Masiero, J.C. Maldonado, and I.G. Boaventura, "A reachability tree for statecharts and analysis of some properties", *Informantion and Software Technology* 36(10), pp. 615-624.

[Oli00] M.C.F. de Oliveira, M.A.S. Turine, and P.C. Masiero, "A statechart-based model for modeling hypermedia applications", accepted for publication in *ACM Transactions on Information Systems*, 2000.

[Ros97] G. Rossi, D. Schwabe, and A. Garrido, "Design reuse in hypermedia applications development", in *Proc. ACM Hypertext'97*, Southampton, UK, April 6-11, pp. 57-66.

[Sch96] D. Schwabe, G. Rossi, and S.D.J. Barbosa, "Systematic hypermedia application design with OOHDM", in *Proc.* ACM *Hypertext'96*, Washington DC, USA, March , pp. 116-128.

[Shi98] R. Shibuya, "*Code Generation from Statecharts*", Master's Dissertation, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, Brazil, 101p., 1998 (in Portuguese).

[Tur97] M.A.S. Turine, M.C.F de Oliveira, and P.C. Masiero, "Hypertext Model Based on Statecharts", in *Proc. ACM Hypertext'97*, Southapton, UK, April 6-11, pp. 102-111, 1997.

[Tur99] M.A.S. Turine, M.C.F. de Oliveira, P.C. Masiero, "HySCharts: A Statechart-Based Environment for Hyperdocument Autoring and Browsing", *Multimedia Tools and Applications*, Vol. 8, pp 309-324, 1999.