StateWebCharts: A Formal Description Technique Dedicated to Navigation Modelling of Web Applications

Marco Winckler, Philippe Palanque

StateWebCharts: A Formal Description Technique Dedicated to Navigation Modelling of Web Applications

Marco Winckler & Philippe Palanque

LIIHS – IRIT, University of Toulouse III
118, route de Narbonne
31062 Toulouse Cedex 4 France
{winckler, palanque}@irit.fr

Abstract. This paper presents StateWebCharts (SWC), a formal description technique based on statecharts for describing navigation on web applications. This notation extends the classical statecharts notation by adding more necessary concepts such as an appropriate semantics for states and transitions in a Web context, including notions like dialog initiative control and client and server activities. As well as statecharts do, this formal description technique features a graphical representation thus making it easier to use for web designers and formal enough to allow to rigorously reason about properties of navigation models. In order to show the applicability of the notation, we show, in the paper, its use on two real-size web applications.

1. Introduction

"Web applications[1]" is a widely-used and fuzzy term for web sites including informational-centric sites, e-commerce sites, portal sites, etc. Despite the apparent facility to create web pages (HTML pages) the successful development of large web applications is a complex activity that requires appropriate methods and tools [13]. This inherent complexity is not only due to the huge number of pages that must be managed or the diversity of technologies employed (JavaScript, Java, Active-X, etc) but also to dynamic aspects such as on-the-fly page generation. In addition, web applications require regular maintenance in order to update pages content, to follow a particular business workflow, to include new features for supporting new task and/or users, and so on.

To deal with such complex development of web applications, modelling support is essential to provide an abstract view of the application. Modelling can help designers during design phases by defining formally the requirements, providing multi-level of details as well as providing support for testing prior implementation. Support from modelling can also be obtained in later phases via, for instance, support for verification prior to implementation [1].

[1] Some authors [3, 6] call *web applications* only data intensive application which presents dynamic content generation and the term *websites* is applied only for applications based on static content. This distinction is not relevant in this paper and thus, these terms are used as synonyms here.

Statecharts [8, 10] and statecharts-like notations have already been widely used for modelling various aspects of web applications. For instance, they have been previously used for navigation modelling of hypertext/hypermedia applications [14, 18, 20], web applications [12] and even WIMP interfaces [10]. This previous work shows some limitations in the expressive power of statecharts for handling specific and critical aspects of web applications' modelling. For instance, it is not possible, using statecharts, to represent who (the user or the system) is at the source of an event to be received by the application. Even previous works that focused on navigation modelling for web applications, such as [14], do not clearly explain how statecharts can be effectively used to model other web applications features such as dynamic content generation.

For that reason, we have extended statecharts to the StateWebCharts notation (SWC) that provides dedicated constructs for modelling specificities of states and transitions in web applications. Our aim is to provide a visual notation easy-to-apply for web designers and formal enough to be subject of automatic verification, thus supporting designer's activity throughout the design process. Most elements included in SWC notation aim at providing explicit feedback about the interaction between users and the system.

As for now, SWC is mainly used to describe the navigation between documents rather than interaction between objects. We distinguish navigation (communication links between information units) from interaction (e.g. manipulation of interface widgets such as scrollbars, and windows interactors). SWC is powerful enough for handling these two aspects, but such concerns are beyond the scope of this paper.

This paper aims at presenting SWC notation in detail and at showing how this notation can be used for modelling navigation in Web applications. Next section (section 2) presents a formal definition of statecharts. This formal definition is used as a basis for the introduction of the extensions at the core of SWC (section 3). Section 4 presents an exhaustive list of the various key elements of web navigation and for each of these elements how SWC can be used for modelling them. In section 5, this paper brings a detailed discussion about the related work on navigation modelling including statecharts-like notation as well as other approaches for navigation modelling. Conclusion and future work are presented in section 6.

2. The Statecharts Notation

Statecharts [8, 9] is a visual formalism that extends state diagrams for modelling complex/reactive systems. Statecharts can be defined as a set of the *states, transitions, events, conditions* and *variables* and their inter-relations. There are numerous extensions to Statecharts to support different modelling needs and with different semantics [9]. Hereafter we introduce the basics of statecharts that are relevant for this paper.

A formal definition of statecharts (also called state machine) is [8]:

S is defined as the set of states;
ρ : S → 2^S, is the map function that associates each state to its sub-states, where **ρ(s)=0** means *s* is a *basic state* with no children inside;
ψ : S → {AND / XOR} is the function that defines whether $s \in S$ is a *composed AND / XOR* state or not
H is the set of *history symbols*
γ : H → S is the function that match *history symbols* to *states*[2]
δ : S → $2^{S \cup H}$ is the *default function* that defines the initial states in *S*
Φ is the set of *final state symbols*
V is the set of variables
C is the set of *conditions*
E is the set of *events*
A is the set of *actions*, where each action is a term of a language **£$_a$**, which defines the allowed operations in a SWC machine
L = E x C x A is the set of *labels* on transitions
T ⊂ 2^S x L x $2^{S \cup H \cup \Phi}$, is the set of transitions represented by a source state (2^S), a label (L) and a target state ($2^{S \cup H \cup \Phi}$)

States are graphically represented by rounded rectangles and transitions are represented by unidirectional arrows going out from a source state to a target state (see Figure 1). Transitions are usually represented by arrows that are labelled by the expression *event/condition:action* (see figure 1a). Optionally, the label could be just a generic identification (*t1*, as in Figure 1b). Guard conditions and actions are optional. If the guard condition is not given it is assumed to be *true*. When actions are not given, control is given to the arrival state. Parameters can be passed-on from a state to another. Only events are explicitly required on transitions.

Figure 1. Graphical representation of states and transitions

By opposition with state diagrams that are "flat" and inherently sequential in nature, statecharts propose three basic structuring mechanisms: hierarchy of states, orthogonality and broadcasting communication. The first two ones, that are critical for web applications navigation modelling, are presented more in detail hereafter.

[2] The difference between *History states* types (*shallow* and *deep* history) and *end-states* is not relevant for this paper.

2.1. Hierarchy

The hierarchy is represented by composition of nested states (XOR-states) thus allowing an efficient use of transition arrows. XOR-states can have exclusively one active sub-state at a time. Figures 2a, 2b and 2c are equivalent representations

- On Figure 2b, states *A1* and *A2* are nested into the XOR-state *A*. All transitions going out from a composite state are propagated to sub-states;
- Figure 2c hides details for the XOR-state *A*. This is a useful abstraction mechanism in statecharts.
-

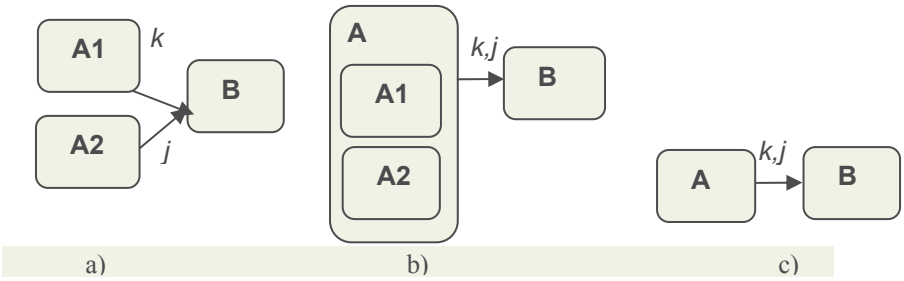


Figure 2. Hierarchy of states in statecharts with XOR-states.

2.2. Orthogonality

Orthogonality is the decomposition of composite states into concurrent regions representing independent modules in a system. Each concurrent region in an AND-state is delimited by a dashed row. Figure 3 shows 3 concurrent states: D, C and A. Like a XOR-state, each concurrent region can have at most one active state at a time.

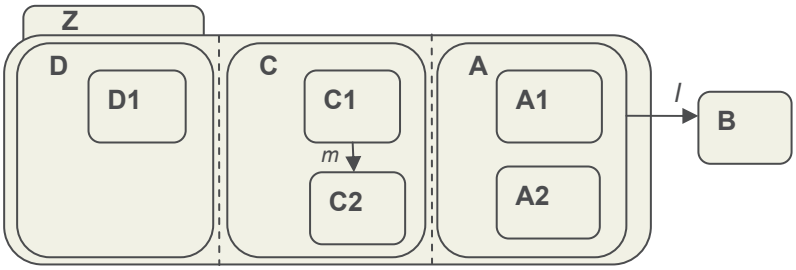


Figure 3. Concurrent states in statecharts with AND-states.

Like a state diagram, a statecharts model starts in an initial state represented by an arrow with a black circle at its starting end (see figure 4). It is also possible to define the initial state in a XOR-state, as shown by figure 4a (state A1). In figure 4a, the execution starts by state *B*. If transition *p* is activated, the system enters the state A1 (the initial state in the composite state *A*). Figure 4b uses a history state, which is

represented by an *H* inside a circle. The history state sets the active state to the most recent state in the set (*A1* or *A2*).

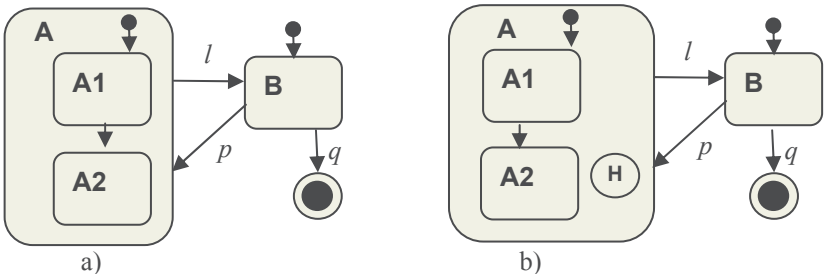


Figure 4. Initial states, history states and final states in statecharts.

Transition *q* in both Figure 4a and Figure 4b is going from state *B* to a *final state* which is represented by a black dot inside a circle. *Final states* mean that the execution is completed. When the enclosing state is on the top state, then it means that the entire state machine has completed.

The operational semantics for statecharts is given by a sequence of steps. At each step the state machine evaluates a single transition and may assume a new state configuration (the set of currently active states). When an event happens, the system transmits it to the transition associated to the triggered event. Then, the corresponding guard condition is evaluated, and if it is true the statemachine sets the target state as active.

An optional activity can be added to the transition label, indicating which activity will take place when the transition happens. The triggered activity can in turn be received by the system as another event to trigger other transitions creating compounding transitions. The broadcasting mechanism in statecharts is represented by events that are associated to more than one transition. In that case, when an event happens, all transitions associated to the triggered event are evaluated and executed if the guarding conditions are true. In classical statecharts, activities and events are considered to be instantaneous (they take no time to perform).

3. StateWebCharts Formal Description

In SWC, states are abstractions of containers for objects (graphic or executable objects). For web applications such containers are usually HTML pages. States in SWC are represented according their function in the modelling. In a similar way, a SWC transition explicitly represents the agent activating it. The basis of SWC modelling is a state machine, as described in previous section, plus the following elements:

P is the set of containers storing information content that will be exhibited as a whole in a web application (generally web pages). **P** is defined by the set (**o**, **k**) where

o is the set of objects (text, sound, image, etc) it contains and **k** is the set of anchors in the set. The set **P** include an empty container.

Ω : S → P, is the map function that associates each state to a container

M : k → E, is map function that associates a anchor to an event in the state machine

Σ : S → AC is the mapping function that associates a state to its activities

AC is the set of optional actions associated to a state. $\mathbf{AC=AC_{entry} \cup AC_{do} \cup AC_{exit}}$, where $\forall$ ac $\in$ A, AC_{entry} is the action executed when entering a state, AC_{do} is the main action executed by it, AC_{exit} is the action executed before the state is left.

Y = {static/transient/dynamic/external} is the set of sub-types for a basic state

ϖ : S → Y is the function that maps a sub-type to a basic state in the state machine $\forall$ s $\in$ S **.** ρ(s) = 0, ϖ≠∅, $\exists$ y $\in$ Y, ϖ (s)=y

W = {user/system/completion} is the set of events sub-types where each event type indicates an agent triggering the event in the system

$\mathbf{E = W_{user} \cup W_{system} \cup W_{completion}}$ is the redefined set of event in a system

A container *P* is considered as a compound document according to W3C DOM[3] definition, which may contain objects (text, images, etc) as well as other documents.

By the function $\varpi : S \rightarrow Y$ we make each basic state $s \in S$ assume an appropriate sub-type *static, transient, dynamic* or *external*. Each sub-type describes a special function performed by the state in the SWC state machine. Figure 4 shows the graphic representation of these sub-types.

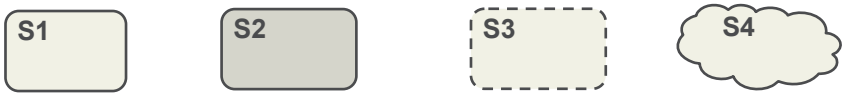


Figure 5. Basic state sub-types in SWC notation.

Static states (figure 5a) are the most basic structures to represent information in SWC. A *static state* refers to a container with a static set of objects; once in a static state the same set of objects is always presented. However, the objects it contains are not necessarily static by themselves; they could have dynamic behaviour as we usually find, for example, in applets, JavaScript or animated images. Static is the default type for *States*.

Transient states (figure 5b) describe a non-deterministic behaviour in the state machine. *Transient* states are not part of the original statecharts notation, but they are needed when a single transition cannot determine the next state of the state machine (see figure 8 for an example). The formal definition for a transient state says that only *completion* or *system* events are accepted as outgoing transitions. Frequently they refer to server-side parts of web applications, such as CGI[4] and Java Servelts

[3] http://www.w3.org/DOM/

[4] CGI - *Common Gateway Interface*

programs. *Transient states* only process instructions and they do not have a visual representation towards users.

Dynamic states (figure 5c) represent content that is dynamically generated at runtime. Usually they are the result of a *transient state* processing. The associated container of a *dynamic state* is empty. The semantics for this state is that in the modelling phase designers are not able to determine which content (transitions and objects) will be made available at run time. However, designers can include static objects and transitions inside dynamic states; in such case transitions are represented, but designer must keep in mind that missing transitions might appear at run time and change the navigation behaviour.

External states (figure 5d) represent information that is accessible through relationships (transitions) but are not part of the current design. For example, consider two states A and B. While creating a transition from A to B, the content of B is not accessible and cannot be modified. Thus B is considered external to the current design. Usually they represent connections to external sites. *External states* avoid representing transitions with no target state, however all activities (whatever it is *entry, do, exit*) in *external states* are *null*.

Events are classified in SWC notation according to the agent triggering them: **user** (e.g. a mouse click), **system** (e.g. a method invocation that affects the activity in a state) or **completion** (e.g. execute the next activity). A *completion* event is a fictional event that is associated to transitions without triggers, e.g. change the system state after a timestamp. Fictional *completion* events allow us to give the same representation for all transitions in SWC machines. This classification of event sources is propagated to the representation of transitions. Transitions whose event is triggered by a user are graphically drawn as continuous arrows (figure 6a.) while transitions triggered by *system* or *completion* events are drawn as dashed arrows (figure 6b).

Figure 6. Graphical representation of SWC transitions.

Even though figure 6 only shows transitions ids, we can promptly identify who owns the control on the activation of a transition, whether the system (transition *t2*) or a user (transition *t1*). In order to be able to use the SWC models to perform usability evaluation, the fact that a transition is related to a user event or not is critical. Thus, SWC puts in a single graphic representation those transitions (*completion* and *system* transitions) that are not triggered by the users. If explicit representation is required for distinguishing between completion and system events, a full label for transition (as presented by figure 1a) such as *t2=completion/true:action1*.

4. Web Navigation Modelling with SWC

Web applications have some similarities with hypermedia and hypertext systems such as the occurrence of multimedia content, linking information units, etc., but many other features are specific to the web environment such as the mandatory use of browser, client/server architecture, and so on. This section describes the most important features related to navigation design for web applications and their corresponding representation with SWC notation, when it is applicable.

4.1. Browser Effects

Web applications can only be accessed through dedicated client applications called web browsers. Browsers interpret every single page sent back by the web server before to display it according proprietary (browser vendors') directives for technology, client-side system platform (e.g. PC Windows, Palm, etc) and additional preferences for display set by users. In addition, from a user interface perspective, the browser itself proposes functions (e.g., cut, copy, save) that could compete with the ones proposed by the application. Recent works [7, 17] have analysed the non-uniform implementation of functions such as history mechanisms (back button and history list, for instance) of web browsers. The worst is that most users rely on such mechanisms to navigation because web applications provide poor navigation [5].

As several browsers with different capabilities are available, it is almost impossible for the designer of the web application to know precisely the software environment of the user. Moreover, it impossible to predict when users will make use of back of the application, so, it is not an advisable strategy to represent browser controls (such as *back button* and other history mechanisms) as part of application design. Such controls are considered as interaction mechanisms such as scroll bars and windows selection that are not represented by SWC notation.

4.2. Link Types Support

When analysing how pages are related to each other on web applications we can observe three different types of links: a) internal-pages links, b) inter-pages links and c) external links. *Internal-pages* links related different parts of a same web page, which can be very helpful for long documents. These links present the same semantic behaviour of scroll bars on windows browser, so at first sight we can consider irrelevant to include the specification of such elements into navigation design. If required, *internal-links* can be easily represented with SWC by decomposing the page in a composite state and create links between the sub-parts of the document as presented by figure 7a. As we can see in figure 7a is a *spaghetti*-link interconnection between all sections of a same document.

Inter-page links is the most classical example; it means a simple connection of two pages belonging to the same web site. We can see in figure 7b how two pages can be connected by an *inter-page* link with passage of a parameter that indicates the subsection in the document to be displayed. Figure 7b is a preferable representation

for the same problem described above (*internal-pages*) because it increases the legibility of diagrams.

External links are links connecting the web application with foreign web sites or non-relevant parts of the web application for the current design. Even though its name makes a reference to a link, this concept is treated as a state because it is not possible to represent targetless transitions in SWC, even though the transition makes references to an external site.

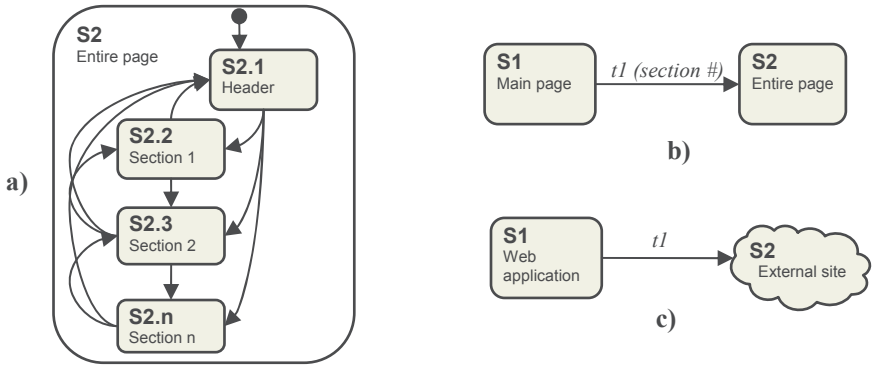


Figure 7. Links types: a) internal-page, b) inter-page and c) external.

4.3. System-Driven Navigation (The Use of Transient States)

In many cases, the combination of event plus condition determines the next state. However, it not true for all cases. Consider the case of user authentication in figure 8. In this example, the event *press button* (to send user name and password) in transition *t1* does not count to determine whether user will get access to the system or not. But it is the result processing of the transient state *S2*. Notice transitions *t2* and *t3* going out from *S2* presenting system events.

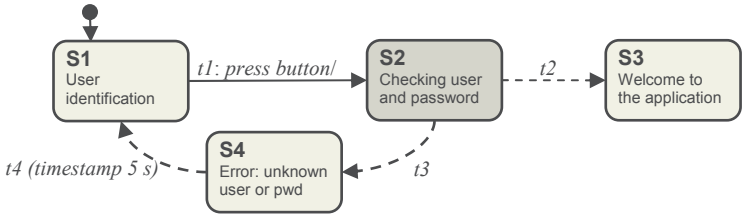


Figure 8. Example of a simple user authentication .

In most cases, the user will send additional information filling in forms or following parameterized links to a server-side application (represented by a transient state) that will execute some processing and then send back user the appropriate answer.

4.4. Dynamic Content Generation

A particular feature of web application is the dynamic generation of pages on an application by server-side applications. Dynamic pages does not exist on the web server and that is why the function $\mathbf{\Omega : S \rightarrow p}$ (see section 3) maps an empty set to dynamic states. Dynamic states represent such unpredictable content for the page but it does not exclude the possibility to represent required transitions for the design.

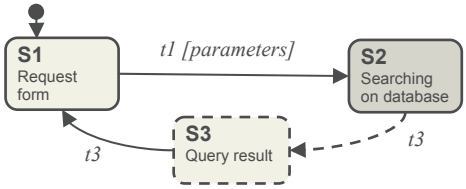


Figure 9. Query search.

Figure 9 shows a classical example of dynamic content generation as result of a database query. Notice that the dynamic state *S3* has a user transition that allows user return to the request form (state *SI*). It important to note that, at run time, the page resulting by the database searching can include links that are not represented in the modelling and may alter the navigation on the web application.

4.5. Frames

Frames are elements that split the browser's windows into two or more concurrent visual areas were each region can load and display a different document. *Frames* were introduced as a standard in HTML 4.0[5]. Links in a frame region can alter the exhibition of documents in another frame region. *Frames* are modelled in SWC with AND-states where each orthogonal region represents an individual frame, as shown in figure 10. When entering the state *A*, two concurrent regions are activated *A'* and *A''* which pass the control to their initial states *B* and *C*, respectively. When the transition *t2* is fired, the configuration in region *A''* changes arbitrarily to states *D*, the region *A'* maintains its configuration.

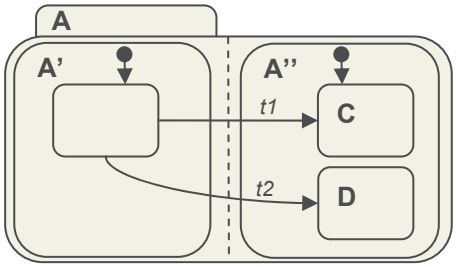
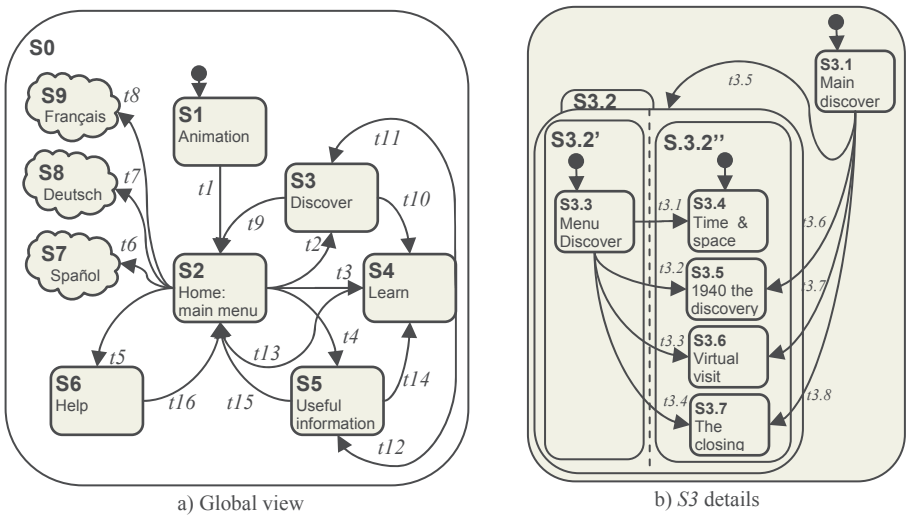


Figure 10. Concurrent visual area representations (frames).

[5] http://www.w3.org/TR/1998/REC-html40-19980424/

4.6. Modularization

The number of pages on most web applications increases very quickly and the representation of documents and links became a problem in flat-notation such as automata [4]. In addition, large projects must be cut in small part and splat among the member of development team. The modularization is also required to deal with the complexity during the development. SWC takes benefits from the multi-level hierarchy from classical statecharts to better manage large web applications. Figure 11 presents a partial modelling for the web site *The Cave of Lascaux*[6].



a) Global view b) *S3* details

Figure 11. Hierarchical view for the Web Site '*The Cave of Lascaux'*

Figure 11a present the global view for the application, which contains 9 states, some of them are composite whose details are not represented in higher level. For example, *S3* is a composite state whose details are shown in figure 11b. In such approach, composite states represent classes of pages which share the same structure. Sub-states inherit relationships from their parents. For example, in Figure 1a, the transitions (*t9 a*nd *t10*) going from state *S3-Discover* to states *S2-Home-main menu* and *S4-Learn*, respectively, are shared by all *S3* sub-states (*S3.1, S3.2, S3.2', S3.2'', S3.3, S3.4, S3.5, S3.6* and *S3.7*). The states at the left are instances of classes of pages that have their own navigation. For reasons of space only state *S3* is detailed in this modelling, even though the S3 some of its sub-states (*S3.4, S3.5, S3.6* and *S3.7*) are at their turn suitable to be decomposed in modules.

[6] http://www.culture.fr/culture/arcnat/lascaux/en/

4.7. Dialog Control Modelling

Modelling dialog control means to identify who (system or user) causes events changing the interface. As before mentioned in section 3, SWC explicitly represents system interaction (by *system* and *completion* events) and user control (by *user events*). A typical example of system event is timed transitions used to redirect Web pages. In the figure 12, users start at the state *S1*, which contains two associated transitions: *e2* and *e3*. The transition *e2* represents a system event that, once activate, will change system state to *S2* five seconds [5s] after users have been entered in *S1*. Users can also cause a transition by selecting a link associated to user event *e3*.

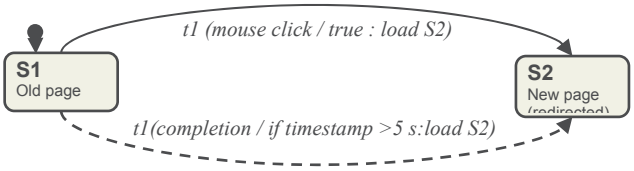


Figure 12. User x System dialog control.

4.8. Client-side and Server-side Execution

On its origins Web applications were built over a client/server architecture where the server-side is responsible for all processing leaving to the client-side (the web browser) just the display of the content information. The advent of new technologies such as JavaScript, Java and Active-X, for example, put on the client more interactive than just display functions. We define *client-side execution* as any processing changing the state of the application without communication with a web server. *Server-side execution*, at its turn, is defined as any instruction processed on a web server following a client's request.

Transient states and *system/completion* transitions in SWC are suitable to describe executable states and system initiative on web applications but they say nothing about where (on the client- or server-sides) it occurs. SWC do not impose a particular architecture for the design and a modelling can be quite easily implemented using as thin-client architecture (no processing in the client-side) or a robust-client (full client-side functionality). However, at this time we have not included a description about how to model objects in a container, so we could consider that *transient* states are always on the server-side.

5. Discussion and Related Work

Research work in navigation modelling has a long history in hypertext and multimedia domain [16, 20 and 22]. Web applications are directly originated from

this research field and much of the web technology related to construction of web pages find its main contributions in hypertext and hypermedia research work.

State-based notations such as Petri nets [16] and Statecharts [14, 18, 20] have been explored to represented navigation for hypertext systems. However, when trying to represent web applications they do not model dynamic content generation, web link-types support (external states, for examples), client and server-side execution, and other aspects related to web domain. Besides, some of them [16, 20] do not make explicit the separation between interaction and navigation aspects in the models while this is a critical aspect for web application.

More recent work devoted to web applications, propose efficient solutions to describe navigation and architecture in a single representation, as it has been done by Connallen [3] with UML stereotypes and Fraternalli with WebML [2]. These approaches mainly target data-intensive applications and propose even prototyping environments to increase productivity. The main inconvenient is that navigation is described at a very coarse grain (for instance navigation between classes of documents) and it almost impossible to represent detailed navigation on instances of these classes or documents. The same problem appears in KOCH [11]. Other approaches such as UML stereotypes as in [3] and WebML [2] may reduce refrain creativity at design time as they impose the underlying technology and as they do not provide efficient abstraction views of the application under developement.

Other studies such as those presented in [4 and 12] take into account all the navigation aspects of web applications (that have been presented in section 4). They are able to represent dynamic content generation and provide efficient support to link-types. However, they do not allow for explicating who (between the user and the system) is triggering events.

Table 1 presents a summary of several notation dedicated to navigation modelling coming from different domains such as Wimp interfaces, hypertext/multimedia systems and web applications. We compare how these notations are able to deal with web design concerns such as those described in section 4. Each aspect is rated according to the following values:

- **N** (no) means that the notation does not support the modelling of such aspect or if it is possible, no information is available on how to cope with it;
- **C** (cumbersome) i.e. the notation provides some support for modelling this aspect but some limitations exist;
- **P** (primitive) the aspect is fully supported and fully documented in the approach (it can be seen as a primitive of the notation).

6. Conclusions and Future Work

In this paper we have presented a statechart-based formalism, StateWebCharts (SWC), which is able to deal with navigation design of web applications. SWC is a technological-independent notation whose main intention is to enable designer to model all the specific features required for modelling navigation of web applications.

One of the contributions of the SWC notation proposed here is that it makes explicit in the models the points where users interact with the application with respect to those where the system drives and controls the navigation. Moreover, all elements

Table 1. Comparative study of several notations for modelling navigation.

Web Design Features	Methods/Notations											
	WIMP interfaces		*Hypertext/Multimedia systems*				*Web applications*					
	Horrocks' statecharts [10]	UMLi [21]	Petri nets [16]	Zheng&Pong's statecharts [20]	HBMS [14, 18]	OOHDM [15]	Automata [4]	UML class diagrams [11]	UML stereotypes [3]	WebML [2]	Leung's statecharts [12]	StateWebCharts - SWC
Interaction Modelling	P	P	P	P	P	P	P	N	N	N	N	C
Navigation modelling	C	C	P	P	P	P	P	P	P	P	P	P
(Web) Link-types support	N	C	C	C	C	C	P	P	P	N	P	P
System-driven navigation	N	C	N	N	N	N	N	N	P	P	P	P
Dynamic content generation	N	N	N	N	N	P	C	P	P	P	P	P
Frames	N	N	N	P	P	C	P	N	N	N	P	P
Modularization	P	P	C	P	P	P	C	P	P	P	P	P
Dialog control modelling	N	N	N	N	N	N	N	N	N	N	N	P
Client-side execution	N	N	N	N	N	N	N	N	P	P	N	C
Server-side execution	N	N	N	N	N	N	N	N	P	P	N	C

Legend: N no information is provided, **C** cumbersome, **P** primitive.

in SWC have a clear semantic with a corresponding visual representation, which is supposed to increase the legibility of the models. SWC supports *client-side execution* and *server-side execution* with some limitations as explained in section 4.8. However, this is an intended limitation as solving this problem (for instance by including architectural information within the notation) would bind models to implementation/architectural concerns too early on the design process. In the same way, SWC is not the best solution for representing interaction on objects inside states. Here again, the focus of SWC is more on the early design phases where low level interaction modelling is premature. Besides, several notations deal very efficiently with these aspects and our goal is more to integrate SWC with such approaches rather than making it suitable for all purposes.

Relationships between SWC models and other models that has to be built during the development process of web applications has already been studied and can be found in [23]. For instance this paper presents how conformance between task models and SWC can be checked. This is another advantage of using formal description techniques for navigation modelling.

As for future work, we intend to use SWC model as a key component of the evaluation phase. Indeed, this phase is really critical for web application development as they are by nature hard to test and evaluate. The idea is to exploit the models to pilot and drive (possibly remote) evaluations by providing users with structural information about navigation and continuously monitoring coverage of the tests.

Acknowledgments

This work has been partially supported by Capes/Cofecub SpiderWeb project. First author is also sponsored by CNPq (Brazilian Council for Research and Development).

References

1. Campos, J. C., Harrison, M. D. (1997) Formally Verifying Interactive Systems: A Review. In Harrison, M.D.&Torres, J.C.(eds.), DSVIS'97, 109-124 pp, Springer.
2. Ceri, S.; Fraternali, P. & Bongio, A. Language (WebML): a modelling language for designing Web sites. In Proc. 9th WWW Conference, Amsterdam, May 2000.
3. Connallen, J. Building Web Applications with UML. Addison-Wesley, 1999.
4. Dimuro, G. P.; Costa, A. C. R. Towards an automata-based navigation model for the specification of web sites. In…: 5th Workshop on Formal Methods, Gramado, 2002. Electronic Notes in Theoretical Computer Science, Amsterdam, 2002.
5. Fleming, J. Web Navigation: Designing the User Experience. O'Reilly. 1998.
6. Fraternali, P. Tools and approaches for developing data-intensive Web applications: a Survey. ACM Computing Surveys, 31(3), 227-263p. 1999.
7. Greenberg, S. and Cockburn, A. Getting back to back: Alternate behaviors for a web browser's back button. In Proceedings: 5th Annual Human Factors and the Web Conference, Maryland, USA, 1999.
8. Harel, D. Statecharts: a visual formalism for computer system. Science of Computer Programming, 8, N. 3:231-271 p., 1987.
9. Harel, D.; Naamad, A. The STATEMATE semantics of statecharts. ACM Trans. Software Engineering Methodology, vol. 5, 4 (Oct. 1996), 293–333 pp.
10. Horrocks, I. Constructing the User Interface with Statecharts. Addison-Wesley, Harlow. 1999, 253 p.
11. Koch, N.; Kraus, A. The expressive Power of UML-based Web Engineering. In 2nd International Workshop on Web-oriented Software Technology (IWWOST02). D. Schwabe, O. Pastor, G. Rossi, and L. Olsina (eds.), June 2002.
12. Leung, K., Hui, L., Yiu, S., Tang, R. Modelling Web Navigation by StateCharts. In Proceedings: 24th Inter. Comp. Software and Applications Conf., 2000, Electronic Edition (IEEE Computer Society DL).
13. Murugesan, S.; Deshpande, Y. (2001). Web Engineering: Managing Diversity and Complexity of Web Applications Development. Berlin: Springer.
14. Oliveira, M.C.F. de; Turine, M. A. S.; Masiero, P.C. A Statechart-Based Model for Modeling Hypermedia Applications. ACM TOIS. April 2001.
15. Schwabe, D.; Esmeraldo, L.; Rossi, G. & Lyardet, F. (2001) Engineering Web Applications for Reuse. IEEE Multimedia, 8(1), 20-31.
16. Stotts, P. D.; Furuta, R. Petri-net-based hypertext: document structure with browsing semantics. ACM Trans. on Inf. Syst. 7, 1 (Jan. 1989), Pages 3 - 29.
17. Tauscher, T and Greenberg, S. How people revisit web pages: Empirical findings and implications for the design of history systems', International Journal of Human Computer Studies 47(1), 97–138. 1997.
18. Turine, M. A. S.; Oliveira, M. C. F.; Masieiro, P. C. A navigation-oriented hypertext model based on statecharts. In Proceeding… 8th ACM Hypertext Conf. April, 1997, Southampton United Kingdom. Pages 102 - 111.
19. Winckler, M.; Farenc, C.; Palanque, P. & Bastide, R. Designing Navigation for Web Interfaces. IHM-HCI2001 Proceedings, Lille France, September 2001.

20. Zheng, Y.; Pong, M. C. 1992. Using statecharts to model hypertext. In Proceedings of the ACM Conference Pankaj K. Gargypertxt (ECHT'92, Milan, Italy). ACM Press, New York, NY, 242-250.
21. Silva, P. P. da, Paton, N. W. UMLi: The Unified Modelling Language for Interactive Applications. In 3rd International Conference on the Unified Modeling Language UML'2000. LNCS V.1939, 117-132 p., Springer, Oct. 2000.
22. Halasz, F., Schwartz, M. The Dexter hypertext reference model, Communications of the ACM, v.37 n.2, p.30-39, Feb. 1994
23. Winckler, M.; Palanque, P.; Farenc, C.; Pimenta, M. Task-Based Assessment of Web Navigation Design. In Proceedings: ACM TAMODIA'02, Bucharest, 2002.