ПРОГРАММИРОВАНИЕ, 2004, № 1, с. 64–70

СЕТЕВЫЕ ТЕХНОЛОГИИ

УДК 681.32

ИСПОЛЬЗОВАНИЕ ДИАГРАММ СОСТОЯНИЙ И ПЕРЕХОДОВ ДЛЯ МОДЕЛИРОВАНИЯ ГИПЕРТЕКСТА*

© 2004 г. **Е. А. Горшкова, Б. А. Новиков**

Санкт-Петербургский государственный университет

198904 Санкт-Петербург, Библиотечная пл., 2

E-mail: borisnov@acm.org

Поступила в редакцию 26.06.2003 г.

Предложенное ранее авторами расширение UML, позволяющее описывать пользовательские интерфейсы Web-приложений, дает возможность применения однотипных средств при проектировании и разработке всех уровней и компонент приложения.

В этой статье детально описывается центральная часть этого расширения UML – навигационные диаграммы, используемые для спецификации поведения системы, видимого пользователем.

1. ВВЕДЕНИЕ

Web-технологии развиваются все быстрее и быстрее. Еще несколько лет назад большинство Web-страниц были статическими. Теперь все более и более популярными становятся сложные Web-приложения, обрабатывающие большие объемы данных и доступные только пользователям корпоративной сети. Web-приложения, работающие 24 часа в сутки, призваны удешевить бизнес-процессы и повысить их качество. Это приводит к тому, что все больше и больше компаний заинтересованы в скорейшем внедрении качественных Web-приложений и готовы вкладывать в это деньги.

Основными отличиями Web-приложений от обычных являются повышенные требования к безопасности, отсутствие конфигурации на клиенте, масштабируемость и использование гипертекстовой технолсгии. Обычный пользователь, думая о Web-приложении, представляет себе прежде всего набор HTML-страниц, переход между которыми осуществляется щелканьем на гиперссылку. Хорошо спроектированное приложение должно иметь четкую и ясную гипертекстовую структуру, потому что ничто не утомляет больше, чем поиск информации на плохо структурированном сайте.

По мнению авторов, разработка качественного программного обеспечения невозможна без предварительного проектирования. Однако существующие методологии проектирования не вполне подходят для моделирования Web-приложений. Например, они не охватывают моделирование Web-интерфейса. В этой работе мы предлагаем расширение Unified Modeling Language (UML) [1] для моделирования специфичных аспектов Web-приложений. Выбор стандартного языка моделирования позволяет использовать инструменты, поддерживающие UML.

В этой статье мы рассмотрим вопрос моделирования гипертекста. Для решения этой задачи мы предлагаем создать диаграмму навигации. Эта диаграмма моделирует поведение пользователя в терминах страниц и переходов между ними. В процессе создания этой диаграммы мы пытаемся определить, какие страницы будут в нашем приложении и каким образом пользователь будет попадать с одной страницы на другую. При этом нас интересует структура всего приложения, а не самих страниц. Содержание отдельных страниц определяется довольно абстрактно. Например, содержание страницы "Счета" определяется как "страница, на кото-

*Работа выполнена при частичной поддержке РФФИ, грант 01-01-00935.

рой будет список счетов, доступных пользователю". При этом совершенно не важно, ни какие HTML-элементы будут использованы, ни их расположение на странице.

Исходными данными для создания такой диаграммы являются требования к приложению. Конечно, формального преобразования между требованиями и диаграммой навигации не существует. Однако наш опыт показывает, что даже технически неподготовленному заказчику легко понять предложенную нами нотацию. Искусство же проектирования состоит в том, чтобы, последовательно уточняя требования, привести их в соответствие с навигационной моделью.

2. ДИАГРАММЫ СОСТОЯНИЙ И ПЕРЕХОДОВ

Для моделирования навигации мы будем использовать диаграмму состояний и переходов (statechart diagram). Такие диаграммы, предложенные Харелом [2], описывают реакцию объекта на события, которые он получает. Диаграммы состояний и переходов в UML несколько отличаются от оригинальной версии, позволяя использовать их для объектно-ориентированных систем. Основными элементами этих диаграмм являются *состояния* и *переходы*. Состояние объекта определяется множеством свойств, при которых он заданным образом реагирует на внешние события. То есть в одном и том же состоянии все объекты данного класса одинаковым образом реагируют на какое-то событие. А находясь в разных состояниях, объект может на одно и то же событие реагировать поразному. Событием называется любое происшествие во времени и пространстве. При моделировании нас интересуют события, в результате которых объект меняет свое состояние. Реагируя на событие, объект выполняет какие-то действия (actions) и осуществляет переход (transition) в другое состояние.

На рис. 1 показан пример диаграммы состояний и переходов. Рассматриваемый объект – это книга в библиотеке. Книга может находиться в двух возможный состояниях: в библиотеке на полке (*On shelf*), либо у кого-то из читателей (*On loan*). Когда читатель сдает книгу (происходит событие *return*), запись о том, что книга отсутствовала, удаляется (выполняется действие

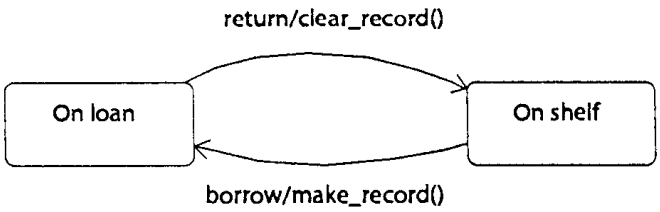

Рис. 1. Диаграмма состояний и переходов.

Рис. 2. Пример простого перехода.

clear_record()), и книга возвращается на полку (в состояние *On shelf*). При этом объект реагирует на событие *return*, только находясь в состоянии *On loan*, а в состоянии *On shelf* его проигнорирует.

Оказывается, что диаграмма состояний и переходов отлично подходит для моделирования гипертекста. В качестве исходной системы рассмотрим Web-приложение в целом. Будем считать, что состояние Web-приложения – это страница, которая отображается в браузере. Когда пользователь щелкает на ссылку, происходит событие, которое вызывает переход с одной страницы на другую, то есть смену состояний. Для обозначения таких состояний используется стереотип <<page>>. Пример простого перехода между состояниями показан на рис. 2. Находясь на странице *List of Books*, пользователь может выбрать книгу и перейти на страницу *Book Details*. Рассмотрим подробнее, что при этом происходит.

1. Пользователь просматривает список книг (находится в состоянии *List of Books*).
2. Пользователь выбирает книгу, щелкая на ссылку. При этом происходит событие *select book*.
3. Клиент посылает HTTP-запрос на сервер.
4. Сервер обрабатывает HTTP-запрос и формирует HTTP-ответ.
5. Сервер посылает HTTP-ответ на клиент.

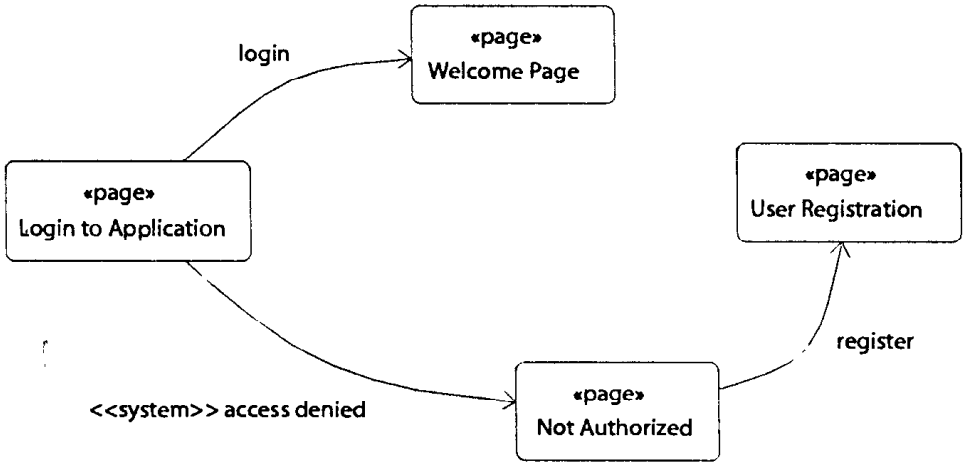

Рис. 3. Разные типы событий.

6. Пользователь попадает на страницу с детальным описанием книги (переходит в состояние *Book Details*).

Так как мы моделируем исключительно навигацию, нас не интересует, что именно происходит в пункте 3. На нашей диаграмме с переходами не связаны никакие действия. Заметим, что соответствие между состоянием и страницей определяется исключительно поведением страницы, а не ее содержанием. Например, странице, которая показывает результат поиска книг, соответствует одно состояние *Found books*, независмо от того, какие книги были найдены.

Мы выделяем два типа событий: происходящие по инициативе клиента (клиентские) и происходящие по инициативе системы (системные). Большинство клиентских событий совершает пользователь, щелкая на ссылку или отправляя на сервер HTML-форму. Другой пример клиентского события – это автоматическое перенаправление между страницами. В этом случае событие совершает браузер. Все клиентские события имеют стереотип `<<client>>`. Так как таких событий большинство, на диаграмме они никак специально не обозначаются. Однако нормальный переход со страницы на страницу может быть нарушен. На рис. 3, если пользователь не имеет прав на доступ к приложению, на сервере происходит событие *access denied*, и пользователь попадает на страницу *Not Authorized*. События, происходящие по инициативе системы, имеют стереотип `<<system>>`.

Заметим, что здесь мы отходим от UML-спецификации, предполагающей атомарное выполнение переходов. Согласно UML, переход не может быть прерван, и, начавшись, должен завершиться. В нашей модели системные события прерывают нормальное выполнение перехода. Так как с переходами никакие действия не связаны, при прерывании побочных эффектов не происходит. Это допущение, хотя и расходящееся со спецификацией, облегчает определение переходов, происходящих в результате исключений.

Таким образом, при моделировании навигации нас интересуют следующие моменты:

- Какие страницы может увидеть пользователь?
- По каким событиям происходит переход между страницами?
- На какую страницу переходит пользователь, после того как событие произойдет?

Возможна ситуация, когда при одном и том же событии пользователь попадает на разные страницы. В этом случае имеет место условный переход (guard condition). Например, на рис. 4 пользователь переходит на разные страницы, в зависимости от того, сколько книг найдено. Условие представляет собой логическое выражение, являющееся частью спецификации перехода. После того как происходит событие, проверяется условие. Переход выполняется только в том случае, если условие истинно. UML не

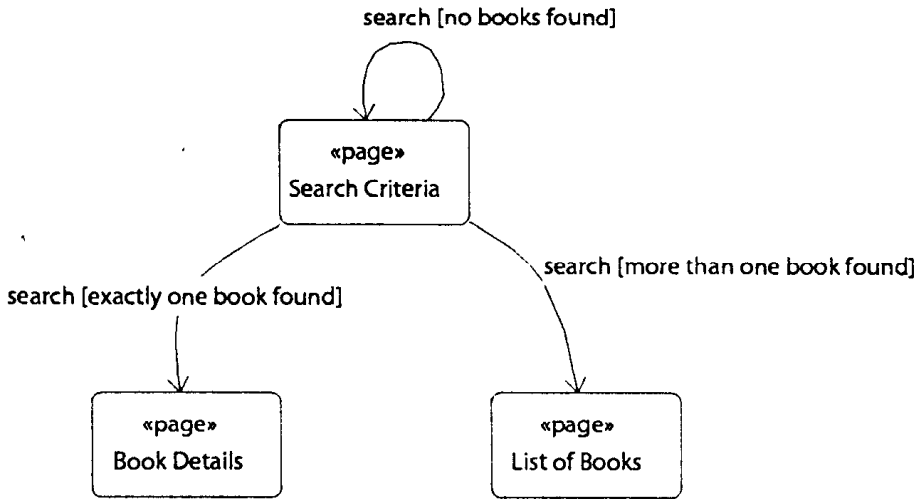

Рис. 4. Пример условного перехода.

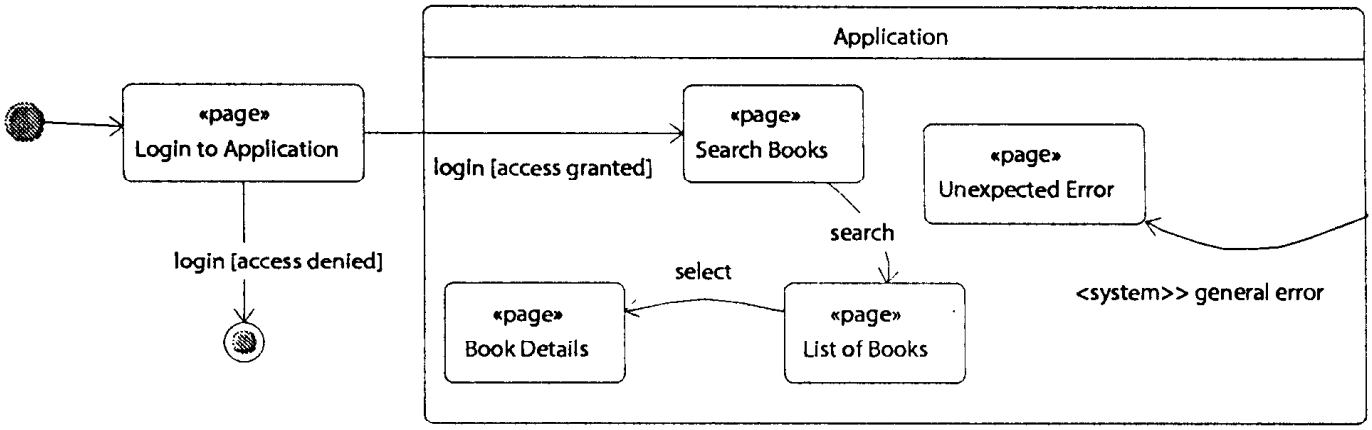

Рис. 5. Пример последовательных вложенных состояний.

накладывает каких-либо ограничений на запись условий, и на навигационной диаграмме они могут быть записаны в произвольной форме.

Кроме переходов между страницами, существуют переходы, выполняющиеся внутри одной страницы. Такое поведение можно моделировать внутренними переходами (internal transitions). По-другому обстоит дело, когда состояние имеет переход само в себя. В этом случае, в отличие от внутреннего перехода, пользователь выходит из текущего состояния и заходит в него вновь. Во время этого перехода происходит взаимодействие с сервером.

3. ИСПОЛЬЗОВАНИЕ ВЛОЖЕННЫХ СОСТОЯНИЙ

Часто бывает, что одни и те же ссылки присутствуют на разных страницах. Например, страницы могут иметь общее меню. В этом случае несколько состояний имеют одинаковые переходы, т.е. переходы, различающиеся только исходным состоянием. Такая ситуация моделируется при помощи последовательных вложенных состояний. Объемлющие состояния называются суперсостояниями, а вложенные – подсостояниями. В каждый момент времени система может находиться только в одном из состояний одного уровня. Вложенные состояния представляют собой специализацию суперсостояний, то есть наследуют все переходы суперсостояний. Если при моделировании навигации простые состояния (не имеющие вложенных) имеют семантику страниц, то суперсостояния как раз и нужны для группировки общих переходов. На рис. 5 с каждой страницы приложения при событии *general error* пользователь попадает на страницу *Unexpected Error*.

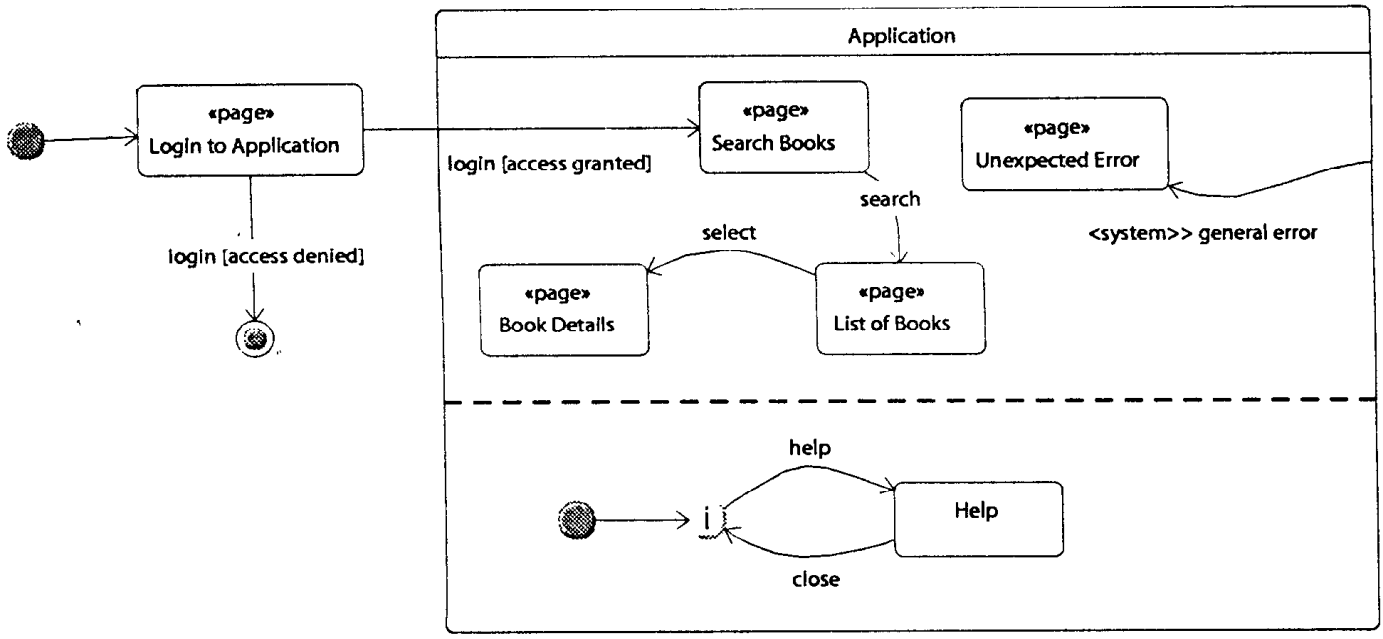

Рис. 6. Пример параллельных вложенных состояний.

Рассмотрим некоторые состояния, имеющие специальную семантику и псевдосостояния. Псевдосостоянием называется узел, в котором выполнение действий не закончено, и события не обрабатываются. Псевдосостояния придуманы для создания сложных переходов и для автоматических переходов, не требующих какого-либо события.

- *initial* – стартовое псевдосостояние. Обозначает входную точку для переходов в суперсостояние и обозначается закрашенным кружком. То есть переход в суперсостояние означает переход в его вложенное стартовое псевдосостояние. В нашем случае стартовое псевдосостояние верхнего уровня соответствует входу в Web-приложение. Например, ему может соответствовать страница index.html, с которой управление передается на первую страницу.

- *final* – это специальное состояние, которое указывает, что действия, связанные с суперсостоянием, выполнены. На диаграмме такое состояние изображается кружком с закрашенной серединой. Для состояния верхнего уровня может означать окончание работы приложения, переход на какую-то специальную страницу, сообщающую, что работа закончена, или закрытие окна браузера.

- *history* – изображается значком H внутри кружка и соответсвует предыдущему активному состоянию. Возврат в это псевдосостояние означает возврат на предыдущую страницу.

- *idle* – это состояние введено нами для случая, когда окно браузера не показывается клиенту. Обозначается буквой I внутри кружка. Используется для параллельных вложенных состояний.

Иногда нужно показать пользователю несколько страниц одновременно в разных окнах браузера, которые работают синхронно или асинхронно. Эта ситуация моделируется при помощи параллельных вложенных состояний. Суперсостояние делится на несколько подсостояний, которые являются активными одновременно. Синхронизация параллельных подсостояний происходит через события. Каждое событие приходит одновременно во все активные состояния. На рис. 6 показано, как работает окно *Help* параллельно с другими окнами приложения. Событие *help* приходит одновременно в оба подсостояния, работающие параллельно. Но реагирует на него только одно подсостояние, переходя из состояния `<<idle>>` в состояние *Help*.

После этого пользователь видит 2 окна одновременно, пока *Help* по событию *close* снова не перейдет в состояние <<idle>>.

Заметим, что параллельные состояния могут быть реализованы разными способами. Вместо нескольких окон браузера можно использовать фреймы. Идея фреймов состоит в том, что окно браузера делится на несколько областей, в каждую из которых можно загрузить независимую страницу. С точки зрения сервера нет никакой разницы между фреймами и разными окнами браузера, поэтому все сказанное выше в полной степени относится к фреймам. Решение о том, как именно эта диаграмма будет реализована можно принять позже, так как реализация никак не влияет на поведение.

Часто ставится обратная задача – создания диаграммы навигации по имеющемуся приложению (или его прототипу). В этом случае важно понимать, что применение фреймов не всегда связано с поведением. Например, в отдельном фрейме может находиться логотип. В этом случае область с логотипом присутствует на каждой странице и не влияет на переходы. Такую область вообще не нужно учитывать при моделировании навигации. Другой распространенный пример использования фреймов – это общее для страниц меню. Если на всех страницах меню одинаковое, то нужно использовать последовательные вложеннные состояния.

4. БЛИЗКИЕ РАБОТЫ

Практически любая методология моделирования Web- или гипермедиа-приложений предлагает какие-то решения для моделирования гипертекста. Наиболее известны три следующие методологии: RMM, HDM и OOHDM [3–5], В основе RMM лежит свой язык моделирования, построенный поверх entity-relationship модели. HDM также идет от моделирования данных, но предлагает широкий набор предопределенных навигационных конструкций. OOHDM расширяет HDM, используя объектно-ориентированную парадигму. Все эти методологии предполагают наличие специального инструмента, поддерживающего их нотацию.

Исследовательский проект Strudel [6] предлагает описание Web-сайта при помощи запросов над полуструктурированной моделью данных. Эти запросы, написаные на специальном языке StruQL, определяют содержание страниц сайта, а также ссылки для переходов.

Моделирование абстрактного пользовательского интерфейса для Web-приложений рассматривается в [7]. UIML для описания использует XML и не имеет какой-либо визуальной нотации. Также этот подход не рассматривает отдельное моделирование навигации.

Подход на основе UML, предложенный Коналленом [8], позволяет отдельно моделировать клиентские и серверные аспекты Web-приложения. К сожалению, этот подход использует только статические диаграммы для описания как структуры приложения, так и его поведения. Представление Web-страницы на стороне клиента описывается классом, содержание страницы – атрибутами этого класса, а поведение – методами. Такая модель не позволяет описать многие важные моменты, например, условные переходы и обработку ошибок. Другой недостаток подхода состоит в том, что модель приложения в нем описывается не абстрактно, а включая детали реализации. Например, в модели явно указывается, какие технологии используются: Active Server Pages, Java Server Pages, JavaScript и т. д.

Идея моделирования гипертекста с использованием диаграмм состояний рассмотрена в работе Zheng и Pong'a [9]. Авторы предлагают моделировать именно поведение пользовательского интерфейса. Этот подход наиболее близок к нашему методу. Наша работа отличается выбором другой гранулярности, использованием расширения UML и описанием некоторых особенностей, характерных именно для Web-приложения. Другая работа [10] использует диаграммы состояний для описания как поведения, так и структурной организации гипертекста.

WebML – это методология моделирования Web-приложений при помощи ортогональных представлений [11]. Одно из таких ортогональных представлений отображает гипертекстовую модель. Гипертекст моделируется атомарными блоками(units), которые содержат некоторую информацию на концептуальном уровне. Например, одна страница может состоять из блока списка (index unit). Такие блоки сами по

себе являются шаблонами навигации и содержания Web-страниц. Из-за использования моделирования через шаблоны на WebML невозможно смоделировать сложное, нестандартное поведение.

Эволюция OOHDM привела к классификации гипермедиа-шаблонов [12]. В OOHDM определяются шаблоны интерфейса, которые задают как расположение элементов на странице, так и навигацию между узлами.

5. ЗАКЛЮЧЕНИЕ

Предложенный метод описания гипертекста позволяет эффективно описать навигацию большого класса Web-приложений. Использование этой диаграммы имеет следующие преимущества:

- диаграмма описывает все переходы между страницами, создавая четкое представление о структуре приложения;
- данную диагрмму можно уточнить, добавив действия, выполняющиеся на переходах. Таким образом, ее можно рассматривать как первое приближение к описанию бизнес-логики приложения;
- данную диаграмму можно использовать для генерации HTML-прототипа приложения. Многие UML-инструменты умеют сохранять диаграмму в формате XMI. В результате получается XML-документ, который содержит всю информацию о страницах и переходах между ними. Этот документ можно преобразовать в HTML, используя расширяемый язык стилевого оформления (Extensible Style Language, XSL) [13]. Также генерацию HTML-прототипа можно осуществить, разработав плагины к UML-инструментам, которые имеют открытый интерфейс прикладного программирования (API).

Остаются, однако, случаи, для которых этот метод применить не удается. В предложенной схеме пользователь не может настраивать навигацию, создавая закладки на страницы. Другим ограничением является поддержка только статических переходов. Переходы, задаваемые случайным образом или же в зависимости от каких-то сложных условий, навигационная диагрмма не поддерживает. Насколько известно авторам, ни одна из альтернативных методологий не позволяет элегантно справиться с перечисленными ситуациями.

СПИСОК ЛИТЕРАТУРЫ

1. http://www.uml.org/.
2. *Harel D.* Statecharts: A visual formalism for complex systems // Science of Computer Programming. 1987. V. 8. № 3. P. 231–274.
3. *Isakowitz T., Stohr E., Balasubramanian P.* RMM: A methodology for structured hypermedia design // Comm. ACM. 1995. V. 38. № 8. P. 34–44.
4. *Garzotto F., Paolini P., Schwabe D.* HDM – A model-based approach to hypertext application design // ACM TOIS. 1993. V. 11. № 1. P. 1–26.
5. *Rossi G., Schwabe D., Barbosa S.* Systematic hypermedia design with 00hdm. Proc. of ACM International Conference on Hypertext (Hypertext 96). Washington. March 1996.
6. *Fernandez M.F., Florescu D., Kang J., Levy A.Y., Suciu D.* Catching the Boat with Strudel: Experiences with a Web-Site Management System. Proc. ACM SIGMOD Intl. Conf. May 1998. P. 414–425.
7. *Abrams S., Phanouriou C., Batongbacal A.L., Williams S.M., Shuster J.E.* Uiml: An appliance-independent xml user interface language. Proc. of the Eighth International World Wide Web Conference (WWW8). Toronto. May 1999.
8. *Conallen J.* Modeling web application architectures with UML // Comm. ACM. 1999. V. 42. № 10.
9. *Zheng Y., Pong M.* Using statecharts to model hypertexts / Lucarella D. (ed.) Proc. ECHT92. Milan, Italy. December 1992. P. 242–250.
10. *Turine M.A.S., de Oliveira M.C.F., Masiero P.C.* A statechart-base model for modeling hypermedia applications // ACM Transactions on Information Systems. 2001.
11. *Ceri S., Fraternali P., Bongio A.* Web Modeling Language (WebML): a modeling language for designing Web sites // Computer Networks. 2000. V. 33. № 1–6. P. 137–157.
12. *Rossi G., Schwabe D., Carrido A.* Design reuse in hypermedia applications development // ACM Computing Surveys (CSUR). 1999. V. 31. № 4.
13. http://www.w3.org/Style/XSL/.