ПРОГРАММИРОВАНИЕ, 2005, № 1, с. 44–51

СЕТЕВЫЕ ТЕХНОЛОГИИ

УДК 681.32

МОДЕЛИРОВАНИЕ КОНТРОЛЛЕРА WEB-ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ UML*

© 2005 г. **Е. А. Горшкова[1], Б. А. Новиков[1], Д. Д. Белов[2], В. С. Гуров[3], С. В. Спиридонов[4]**

[1]*Санкт-Петербургский государственный университет*
198504 Санкт-Петербург, Университетский просп., 28
[2]*Балтийский государственный технический университет*
Санкт-Петербург, 1-я Красноармейская ул.
[3]*Санкт-Петербургский государственный институт точной механики и оптики*
Санкт-Петербург, Саблинская ул., 14
[4]*Санкт-Петербургский государственный морской технический университет*
Санкт-Петербург, Лоцманская ул., 9
E-mail: borisnov@acm.org

Поступила в редакцию 26.06.2003 г.

В этой статье мы расширяем предложенный ранее метод проектирования [1], моделируя логику контроллера web-приложения. Мы детально описываем UML-диаграмму конфигурации контроллера и объясняем ее применение в разработке приложений.

1. ВВЕДЕНИЕ

Основным архитектурным шаблоном при разработке web-приложений является шаблон Модель–Представление–Контроллер (Model–View–Controller, далее – MVC). Этот шаблон разбивает приложение на три модуля. К модели относятся данные и бизнес-логика, к контроллеру – обработка запросов и поток управления, а представление отвечает за отображение состояния приложения.

Контроллер – это модуль, который принимает запросы от клиента и определяет, какому обработчику их передать. Выбранный обработчик производит операции с моделью и через контроллер передает управление обратно соответствующему представлению. Представления, в свою очередь, отвечают за непосредственное создание HTML-страницы. Такой шаблон обеспечивает слабое связывание между представлением и моделью, что делает разработку и поддержку приложения значительно более гибкой.

В этой статье мы рассматриваем web-приложения, использующие MVC, и предлагаем метод моделирования конфигурации контроллера приложения, основанный на универсальном языке моделирования (Unified Modeling Language – UML) [2].

Так как web-приложения – сравнительно новая область, еще не существует проверенных методологий для их разработки и проектирования. В основном для создания web-приложений используется опыт, позаимствованный из разработки другого программного обеспечения: клиентско-ориентированных систем и объектно-ориентированных приложений. За последнее время был предложен ряд различных методологий проектирования, и наибольшее распространение получили UML-методологии.

Однако все эти методологии редко используются на практике. Обычно при разработке web-приложений единственным повторно используемым решением является применение готовых каркасов приложения, таких как [4, 5]. Мы используем термин *каркас* для обозначения понятия, описываемого в англоязычной литературе термином framework. Каркас представляет собой набор взаимосвязанных понятий, языко-

*Работа выполнена при частичной поддержке РФФИ, грант 04-01-00173.

вых конструкций и компонент, ориентированных на некоторую специальную область применения (например, разработку приложений определенного класса). В этой работе, однако, используется намного более узкое понимание термина каркас, определенное в разделе 3.3. Существенной частью MVC-каркаса web-приложения является реализация модуля контроллера. Модуль контроллера – общий для всех приложений, а информация о том, как осуществлять перенаправления, задается в конфигурационном файле, который у каждого приложения свой.

В наших работах мы предлагаем UML-методологию моделирования web-приложений. Предложенная методология состоит из последовательности шагов, на каждом из которых создается модель какого-нибудь аспекта web-приложения. Например, нам потребуется построить модель структуры данных или модель пользовательского интерфейса. Для каждой модели мы предлагаем набор расширенных UML-диаграмм со специальной нотацией. Моделирование клиентской части web-приложений было рассмотрено в наших предыдущих работах [1, 5]. Теперь мы детально рассматриваем моделирование логики приложения. Мы объясняем, как смоделировать поведение приложения с момента получения сервером HTTP-запроса до момента отправки на клиент HTTP-ответа.

Некоторые исследования [6–11] расширяют классические методы моделирования данных, добавляя к ним конструкции для моделирования гипертекста. Все эти методологии предлагают свою собственную нотацию и поэтому требуют специального инструмента. Выбор стандартного языка моделирования позволяет использовать инструменты, поддерживающие UML.

Моделирование web-приложений с использованием UML рассматривается в [12]. Предложенный подход покрывает различные аспекты моделирования данного класса приложений, однако он использует только статические диаграммы и плохо подходит для описания гипертекстовой структуры. С некоторыми изменениями методология, описанная в этой работе, реализована в Rational Rose [13].

Наши предыдущие работы [1, 5, 14] были посвящены моделированию клиентской части web-приложения. Мы разработали UML-нотацию для определения структуры страниц, переходов между страницами и описания связи между пользовательским интерфейсом и бизнес-данными.

Эта статья организована следующим образом. Сначала мы кратко излагаем весь процесс моделирования, а затем детально описываем диаграмму, моделирующую конфигурацию контроллера приложения и объясняем, как эта диаграмма связана с каркасом web-приложения.

2. ПРОЦЕСС ПРОЕКТИРОВАНИЯ WEB-ПРИЛОЖЕНИЯ

В этом разделе мы кратко опишем процесс проектирования web-приложений. Процесс состоит из последовательности шагов, на каждом из которых строится модель некоторого аспекта приложения. Некоторые из рассматриваемых моделей были подробно описаны в [1], в этой работе предлагается модель конфигурации контроллера приложения.

1. *Варианты использования* (*Use Cases*). Это первый шаг нашего процесса проектирования. Варианты использования широко применяются для определения функциональности приложения, видимой внешним пользователям. Это представление моделирует взаимодействие системы и идеализированных пользователей (actors).

2. *Навигационная модель.* На этом шаге мы проектируем гипертекстовую структуру web-приложения. Эта модель описывает, какие страницы входят в приложение и как между ними осуществляется переход. Навигационная диаграмма, визуализирующая эту модель, описана в следующем разделе.

3. *Концептуальная модель.* Эта модель определяет все бизнес-сущности приложения и взаимосвязи между ними. Концептуальная модель описывается при помощи диаграммы классов. Заметим, что модель должна быть описана абстрактно, то есть она не должна зависеть от выбранных технологий. Также в этой модели описываются все ограничения целостности, которым должны удовлетворять данные приложения.

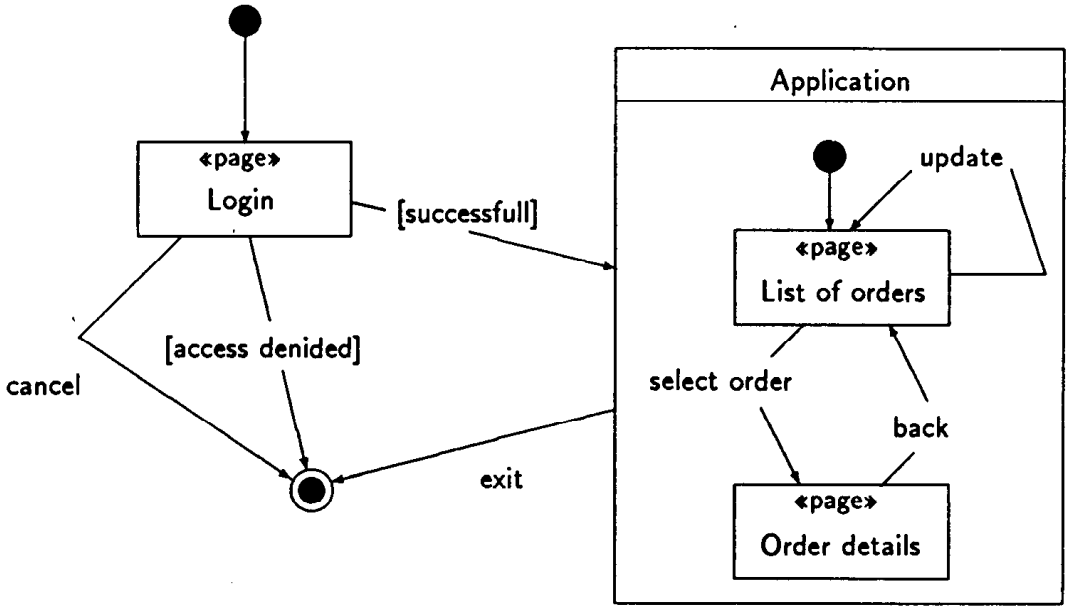

Рис. 1. Пример навигационной диаграммы.

4. *Композиционная модель.* Эта модель, для построения которой используется диаграмма классов, отвечает за проектирование структуры страниц. Каждой странице соответствует какой-то UML-класс. Все элементы страницы, такие как ссылки, кнопки, картинки, моделируются как атрибуты этой страницы. Мы также показываем, как элементы на композиционной диаграмме связаны с данными, которые они представляют. Эти связи используются для генерации кода, проверяющего правильность данных, вводимых пользователем.

5. *Модель конфигурации контроллера приложения.* Данная модель используется для моделирования логики контроллера приложения и рассматривается подробно в этой статье.

2.1. *Навигационная диаграмма*

Один из наиболее важных шагов в понимании функциональности web-приложения – это проектирование навигации. Навигационная диаграмма, являющаяся частным случаем диаграммы состояний и переходов, моделирует поведение пользователя в терминах страниц и переходов между ними. Моделируя навигацию, мы пытаемся определить, какие страницы будут в нашем приложении, и каким образом пользователь будет попадать с одной страницы на другую.

Навигационная диаграмма дает полное представление о гипертекстовой структуре web-приложения. Она показывает страницы приложения, условия, по которым происходит переход между страницами, и различные исключительные ситуации. 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. Пример навигационной диаграммы показан на рис. 1.

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. Если при моделировании навигации простые состояния (не имеющие вложенных) имеют семантику страниц, то суперсостояния как раз и нужны для группировки общих переходов. Подробно навигационная диаграмма описана в работе [5].

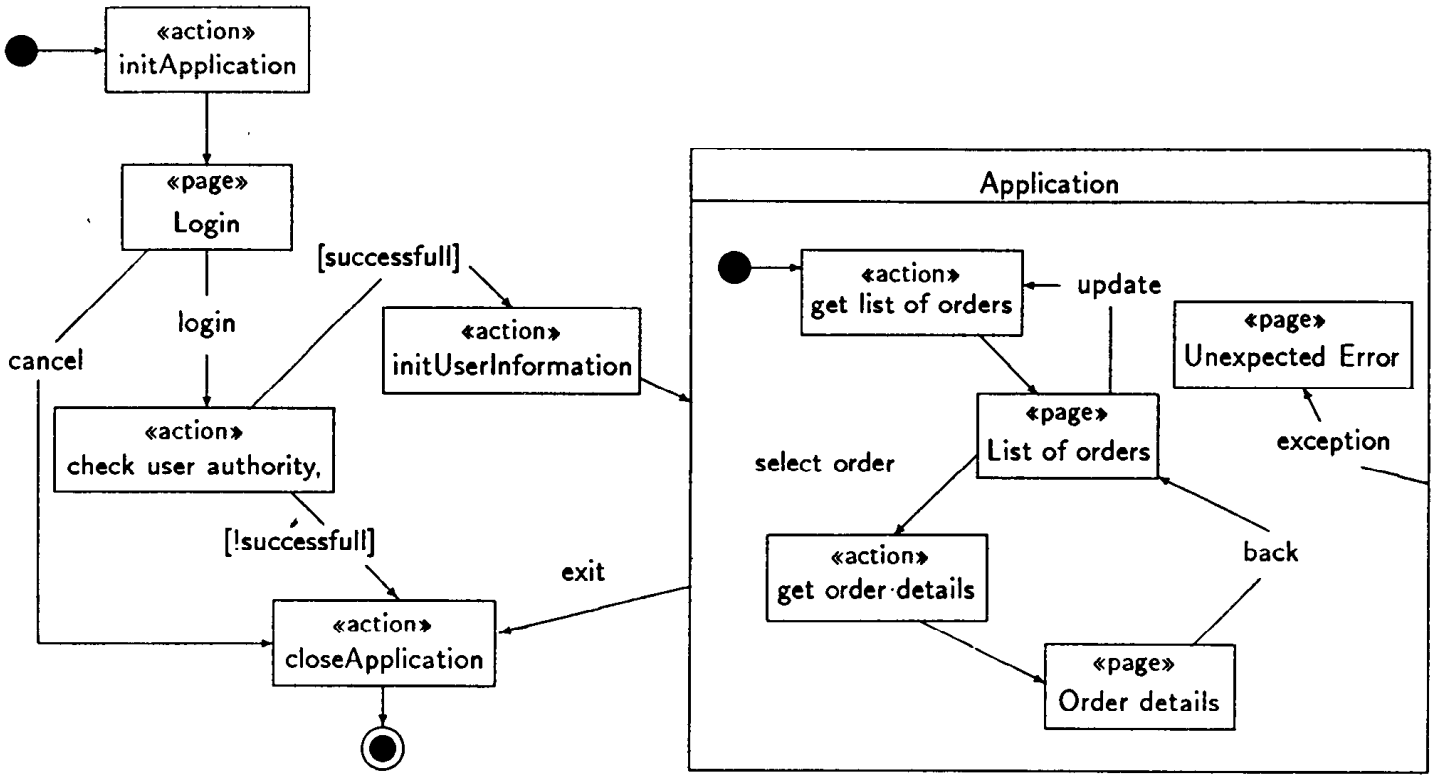

Рис. 2. Пример диаграммы конфигурации контроллера.

3. МОДЕЛИРОВАНИЕ КОНФИГУРАЦИИ КОНТРОЛЛЕРА ПРИЛОЖЕНИЯ

3.1. Диаграмма конфигурации контроллера приложения

Основной обязанностью контроллера является управление приложением. Контроллер получает событие от клиента и передает его некоторому обработчику, который взаимодействует с моделью. Для отображения результатов взаимодействия с моделью контроллер передает управление соответствующему представлению.

Диаграмма конфигурации контроллера приложений представляет собой частный случай диаграммы состояний и переходов. Она уточняет навигационную диаграмму, оставляя неизменными все пользовательские состояния, соответствующие страницам, и топологию переходов. Каждый переход навигационной диаграммы уточняется действиями, которые выполняются во время этого перехода. Действия могут быть привязаны к переходам (как в автоматах Мили) или к состояниям (как в автоматах Мура). Пример диаграммы конфигурации контроллера приложения показан на рис. 2.

Часто во время уточнения создаются новые пользовательские состояния. Обычно эти состояния отражают информацию об исключительных ситуациях, таких как страница "Unexpected Error" на рис. 2. Эти состояния должны быть добавлены в навигационную диаграмму. Таким образом, создание диаграммы конфигурации контроллера помогает обнаружить состояния, забытые при моделировании навигации.

3.2. Правила уточнения

Наложим некоторые ограничения на рассматриваемые диаграммы. Будем предполагать, что события, происходящие по инициативе пользователя, не могут быть отложены (deferred events), так как браузер принимает HTTP-ответ только на последний HTTP-запрос. Мы также предполагаем, что такие события не наследуются. Сформулируем правила уточнения, используя формальный синтаксис описания диаграмм, описанный в работе [15].

- $States_N$, $States_C$ – множества состояний навигационной диаграммы и диаграммы конфигурации контроллера соответственно.

- E_N, E_C - множества всех событий Transitions;
- $Transitions_N$, $Transitions_C$ - множества кортежей $t = (source(t), event(t), guard(t), action(t), target(t))$, где
  - $source(t) \in States$ – *начальное состояние* перехода t;
  - $event(t) \in E$ – событие, вызывающее переход t;
  - $guard(t) \in BoolExp$ – булевское выражение;
  - $action(t) \in Actions$ – действие, выполняемое при переходе между страницами;
  - $target(t) \in States$ – *целевое состояние* перехода t.

Мы также допускаем переходы по завершению (completion transition) $ComplEv \in E$. Такой переход происходит сразу после завершения действия.

Введем также следующие определения множеств:

- $parent(e) \in E, e \in E$ – множество, в которое входит e и его суперклассы;
- $parent(s) \in States, s \in States$ – множество, в которое входит s и его суперсостояния.

Определим теперь правила уточнения.

1. Все переходы на навигационной диаграмме должны быть уточнены на диаграмме конфигурации контроллера приложения.

$$\begin{array}{l} \forall t \in Transitions_N \\ \exists t_0, \ldots, t_n \in Transitions_C : \\ event(t_0) = parent(event(t)) \bigwedge \\ \bigwedge source(t_0) = source(t) \\ source(t_i) = target(t_{i-1}), i = 1, \ldots, n \\ target(t_n) = target(t) \\ target(t_i) \notin Transitions_N, \\ i = 1, \ldots, n-1. \end{array} \quad (1)$$

2. Все переходы на диаграмме конфигурации контроллера должны быть уточнениями переходов на навигационной диаграмме.

$$\begin{array}{l} \forall t \in Transitions_C \\ \exists t_0, \ldots, t_n \in Transitions_C, \exists i : t = t_i, \\ \exists t' \in Transitions_N \\ event(t_0) = parent(event(t')) \bigwedge \\ \bigwedge source(t_0) = source(t') \\ source(t_i) = target(t_{i-1}), i = 1, \ldots, n \\ target(t_n) = target(t') \\ target(t_i) \notin Transitions_N, \\ i = 1, \ldots, n-1. \end{array} \quad (2)$$

3. Конфигурация контроллера должна быть непротиворечива. Условия переходов, имеющих общее начальное состояние и инициируемых одним и тем же событием должны быть попарно дизъюнктивны. Это предотвращает одновременное выполнение нескольких переходов. Данное ограничение распространяется только на диаграмму конфигурации контроллера, так как на навигационной диаграмме условия определяются неформально.

$$\begin{array}{l} (\forall t_i, t_j \in Transitions_C, i \neq j) \\ (source(t_i) = source(t_j) \bigwedge \\ \bigwedge event(t_i) = event(t_j)) \Rightarrow \\ \Rightarrow (guard(t_i) \bigwedge guard(t_j) \equiv false). \end{array} \quad (3)$$

4. Конфигурация контроллера должна быть полной. Дизъюнкция условий переходов, имеющих общее начальное состояние и инициируемых одним и тем же событием, должна быть тождественно истинной. Это ограничение гарантирует, что хотя бы один переход обязательно произойдет. (То есть приложение гарантирует формирование HTTP-ответа после каждого HTTP-запроса.)

$$\begin{array}{l} (\forall s \in States_C, e \in E_C) \\ (T = \{t \in Transitions_C : \\ source(t) \in parent(s), \\ event(t) \in parent(e)\}) \Rightarrow \\ \Rightarrow (\bigcup_{t \in T} guard(t) \equiv true). \end{array} \quad (4)$$

3.3. *Интеграция модели с каркасом web-приложения*

Термин каркас (framework) не имеет четкого определения. Под каркасом мы будем понимать множество классов и интерфейсов, кото-

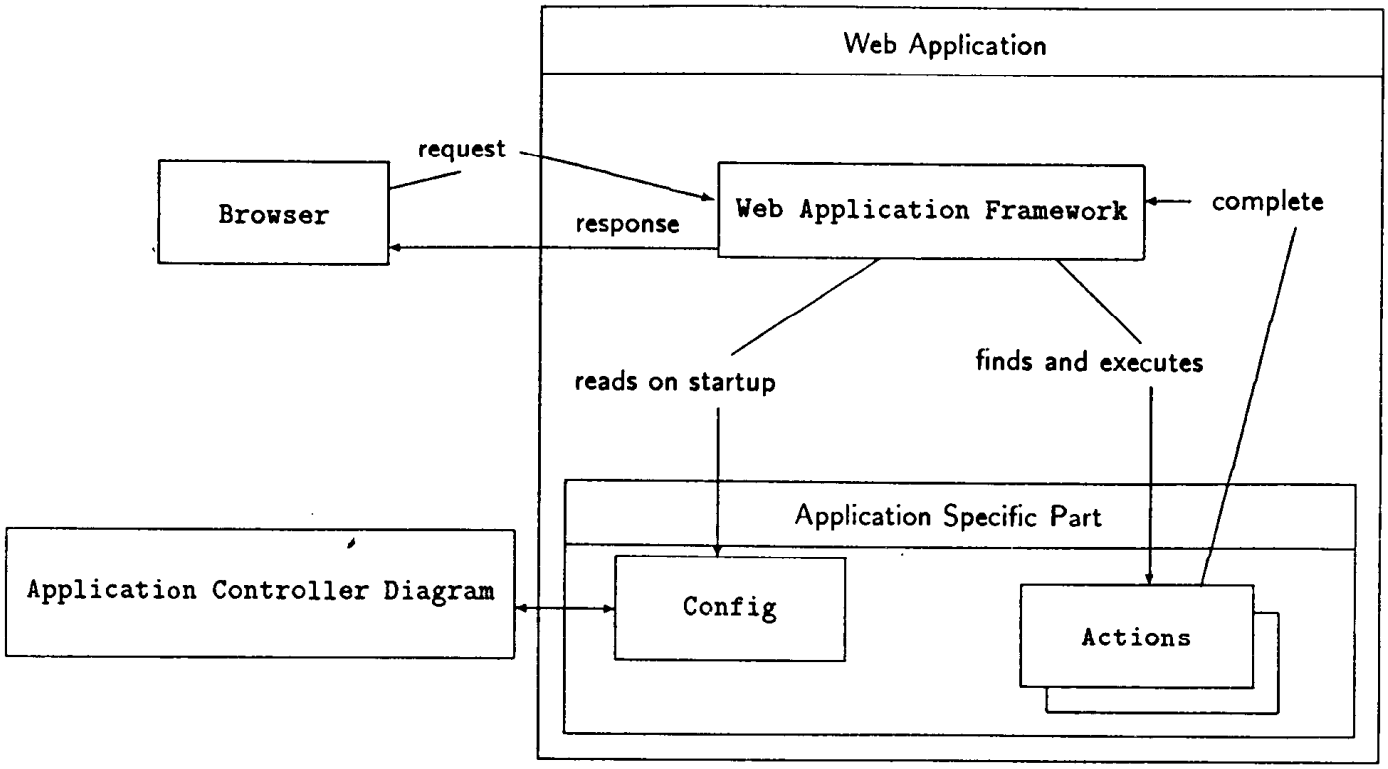

Рис. 3. Структура web-приложения.

рые обеспечивают общее поведение различных приложений и может быть повторно использовано.

Типичный каркас, использующий MVC-парадигму, имеет следующую структуру. Существует несколько конфигурационных файлов, которые ставят в соответствие Java-классам (обработчикам) логические имена. HTTP-запрос, приходящий от клиента, содержит такое логическое имя. Контроллер, являющийся частью каркаса, обрабатывает запрос и, используя конфигурационный файл, определяет, какому обработчику передать этот запрос дальше.

Существуют инструменты, облегчающие работу с популярными каркасами. В основном эти инструменты предоставляют специальные редакторы для редактирования конфигурационных файлов. Однако ни один из известных авторам инструментов не обеспечивает интеграцию каркаса с моделью приложения.

Мы предлагаем сериализовать диаграмму конфигурации контроллера и использовать ее в качестве конфигурационного файла web-приложения. Каркас в этом случае представляет собой двоичный модуль, который читает этот файл в момент инициализации. После этого контроллер работает по следующей схеме:

1. ожидает события от пользователя (HTTP-запрос);
2. определяет текущее состояние клиента и идентифицирует присланное событие;
3. находит все переходы из текущего состояния, происходящие по данному событию;
4. вычисляет все условия рассматриваемых переходов и выбирает переход с истинным условием;
5. вызывает последовательность действий, связанных с выбранным переходом. В случае, когда действия связаны с состоянием, переходы между состояниями происходят по завершению действия;
6. посылает HTTP-ответ клиенту.

На рис. 3 показана структура web-приложения. Сериализованная диаграмма конфигурации контроллера содержит всю необходимую

информацию. Для каждого действия и условного перехода генерируются классы-заглушки, которые разработчик должен реализовать. При этом самой логикой переходов разработчик не управляет.

Используя расширения UML-инструмента, весь набор диаграмм можно использовать для генерации прототипа приложения. Фактически, построив указанный набор диаграмм и имея необходимый инструмент, мы можем получить скелет web-приложения, в котором будет реализована логическая структура страниц и логика переходов.

Диаграмма конфигурации контроллера представляет собой неотъемлемую часть web-приложения, которая может быть использована без изменений. В предложенном подходе не используется ни прямое, ни обратное проектирование, поэтому синхронизация модели и кода не требуется. Разработчик отвечает только за реализацию действий и условных переходов и не может изменить поведение приложения намеренно или по ошибке. Все проверки корректности переходов производятся еще на этапе проектирования, что уменьшает число ошибок. Например, такие проверки гарантируют отсутствие необработанных исключений и нереализованных вариантов поведения.

4. ЗАКЛЮЧЕНИЕ

В этой работе мы рассматриваем проектирование web-приложений, использующих один из самых распространенных архитектурных шаблонов – MVC. Применение этого шаблона очень актуально для web-приложений, поскольку модель, контроллер и представление, как правило, создаются разными людьми, использующими для этого разные технологии.

Мы предлагаем метод моделирования контроллера web-приложения с помощью UML-диаграмм и объясняем, как построенные диаграммы используются для моделирования конфигурации контроллера web-приложения. Построение модели контроллера ускоряет разработку приложения, улучшает взаимодействие между разработчиками и уменьшает количество кода. Также использование строгих моделей позволяет выполнять работу параллельно.

- Предложенный нами процесс позволяет разработчикам сконцентрироваться на моделировании бизнес-аспектов web-приложений.
- Разработка модели для конфигурации контроллера не является "лишним" усилием. Обычно UML-диаграммы используются для генерации программного кода. После этого код меняется программистом и часто перестает соответствовать диаграмме. Нужны специальные действия для синхронизации диаграммы с кодом при помощи обратного проектирования. В нашем случае диаграмма используется в качестве входных данных для каркаса приложения, и никакой синхронизации между моделью и кодом не требуется.
- Так как проверка корректности контроллера происходит на этапе моделирования, приложение содержит меньше ошибок.

В будущем мы планируем описать наиболее распространенные шаблоны, используемые в web-приложениях и поддержать их в нашей методологии.

СПИСОК ЛИТЕРАТУРЫ

1. *Gorshkova E., Novikov B.* Exploiting uml extensibility in the design of web information systems. Databases and Information Systems. Proc. fifth international conference Baltic DB & IS 2002. Tallinn, June 2002. P. 49–64.
2. http://www.uml.org.
3. http://jakarta.apache.org/struts/.
4. http://jakarta.apache.org/turbine/.
5. *Горшкова Е.А., Новиков Б.А.* Использование диаграмм состояний и переходов для моделирования гипертекста // Программирование. 2004. № 1. С. 64–80.
6. *Atzeni P., Mecca G., Merialdo P.* Design and Maintenance of Data-Intensive Web Sites. Proc. EDBT'98 // Lect. Notes in Comp. Sci. Springer-Verlag, March 1998. V. 1377. P. 436–450.
7. *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 Intnl. Conf. May 1998. P. 414–425.

8. *Fraternali P., Paolini P.* Model-driven development of web applications: the autoweb system // ACM TOIS. 2000. V. 18. № 4. P. 323–382.

9. *Garzotto F., Paolini P., Schwabe D.* HDM – A model-based approach to hypertext application design // ACM TOIS. 1993. V. 11. № 1. P. 1–26.

10. *Isakòwitz T., Stohr E., Balasubramanian P.* RMM: A methodology for structured hypermedia design // Comm. ACM. 1995. V. 38. № 8. P. 34–44.

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. *Conallen J.* Modeling web application architectures with UML // Comm. ACM. 1999. V. 42. № 10.

13. http://www.rational.com.

14. *Gorshkova E., Novikov B.* Exploiting uml extensibility in the design of web applications. Databases and information systems. Posters Proc. the eleventh international World Wide Web conference 2002. Honolulu, Hawaii, USA. May 2002.

15. *Jurjens J.* A uml statecharts semantics with message-passing. Proceedings of the 17th 2002 ACM symposium on applied computing. New York, NY, USA. 2002. P. 1009–1013.