Стив Бурбек, Ph. D.

Программирование Приложений в Smalltalk-80™: Как использовать Model-View-Controller (MVC)

Примечание автора: Эта статья первоначально описывала среду MVC так, как она существовала в Smalltalk-80 v2.0. Статья была пересмотрена в 1992, чтобы принять во внимание изменения, сделанные для Smalltalk-80 v2.5. ParcPlace сделала существенные изменения в этих механизмах для версий 4.x, которые не отражены в этой статье.

Примечание переводчика: Хотя классический Smalltalk-80 и вышел из широкого употребления, основные идеи, положенные в основу его пользовательских интерфейсов продолжают использоваться в системах VisualWorks и Squeak. Технические детали, описанные в данной статье, уже не применимы полностью к современным системам, хотя адаптация примеров к Squeak не должна быть сложной.

Оглавление

Введение

Один из вкладов Xerox PARC в искусство программирования — многооконный высоко интерактивный интерфейс Smalltalk-80. Этот тип интерфейса с тех пор был заимствован разработчиками Apple Lisa и Macintosh и, в свою очередь, многочисленными подражателями. В таком интерфейсе, ввод осуществляется прежде всего мышью, а вывод — смесь графических и текстовых компонентов как соответствующий. Центральное понятие стоящее за интерфейсом пользователя Smalltalk-80 — парадигма Model-View-Controller (MVC). Она изящна и проста, но совершенно отлична от подхода, применяемого в традиционных прикладных программах. Из-за своей новизны, она требует некоторых пояснений — пояснений, которое трудно получить в изданных источниках по Smalltalk-80.

Если Вы запускали графический пример в классе Pen, Вы могли задуматься, почему это "приложение" рисует непосредственно на экране, а не в окне подобно окнам просмотра, рабочим окнам, или транскрипту, с которыми Вы хорошо знакомы. Конечно Вы хотели бы, чтобы ваши собственные приложения совместно использовали пространство на дисплее с простым апломбом рабочего окна, а не просто рисовали поверх экрана. Но, чем же они отличаются? Наиболее простой ответ: плохо ведущие себя приложения не соответствуют парадигме MVC, в то время как хорошо ведущиеся себя приложения, знакомые Вам, ей соответствуют.

Эта статья предназначена, чтобы обеспечить новых программистов на Smalltalk-80 существенной информацией для того, чтобы начать использовать технику MVC в своих собственных программах. Здесь мы представим механизмы MVC. Если только Вы переварили это введение, Вы сможете использовать их в своих программах. Вы будете “одеть плотью” информацию, данную здесь, рассматривая способ, которым созданы знакомые виды панелей и контроллеров — типа рабочих окон, окон просмотра и списков файлов. Сразу и часто используйте браузер. Помните, это — Smalltalk-80. Вы должны копировать. Начните создавать ваше собственное окно, копируя то, которое является похожим на окно, которое Вы хотите создать. Затем измените его. Не будьте застенчивы. Не стесняйтесь стоять на плечах многих программистов, внесших свой вклад в образ Smalltalk-80 V2.5. В действительности, это их дар Вам.

Основные Понятия

В парадигме MVC ввод пользователя, моделирование внешнего мира, и визуальная обратная связь с пользователем явно отделяются друг от друга и обрабатывается тремя типами объектов, специализированных для выполнения своей задачи. Панель управляет графическим и/или текстовым выводом в той части растрового дисплея, которая распределена приложению. Контроллер интерпретирует ввод мышью и с клавиатуры от пользователя, управляя моделью и/или просмотром, заставляя их изменяться соответствующим образом. И наконец, модель управляет поведением и данными прикладной области, отвечает на просьбы предоставить информацию о состоянии (обычно из панели), и отвечает на команды, требующие изменить состояние (обычно из контроллера). Формальное разделение из этих трех задач — важная концепция, которая особенно подходит для Smalltalk-80, где базисное поведение может быть воплощено в абстрактных объектах: View, Controller, Model и Object. Поведение MVC затем наследуется, добавляется и изменяется по мере необходимости, чтобы обеспечить гибкую и мощную систему.

Чтобы эффективно использовать парадигму MVC, Вы должны понять разделение труда внутри триады MVC. Вы также должны понять, как три части триады взаимодействуют друг с другом и с другими активными панелями и контроллерами; совместное использование одной мыши, клавиатуры и экрана дисплея несколькими отдельными приложениями требует взаимодействия и сотрудничества. Чтобы лучше использовать парадигму MVC, Вы должны также узнать о доступных подклассах классов View и Controller, которые обеспечивают готовые отправные точки для ваших приложений.

В Smalltalk-80, ввод и вывод в значительной степени выполнятся в особой манере. Панели должны управлять реальным экраном и отображать текстовые или графические формы на этом реальном устройстве. Контроллеры должны сотрудничать, чтобы гарантировать, что соответствующий контроллер интерпретирует клавиатуру и ввод мыши (обычно, тот над панелью которого находится курсор). Поэтому поведение ввода и вывода большинства приложений — подчиняется специальным соглашениям, многое из этого поведения унаследовано от обобщенных классов — View и Controller. Эти два класса, вместе с их подклассами, обеспечивают такое богатое разнообразие поведения, что ваши приложения будут обычно требовать небольшого расширения протокола, чтобы выполнить ввод команд и интерактивное вывод. Напротив, модель не может быть ограничена какой-либо особой манерой. Ограничения на тип объектов, которые могут функционировать как модели, ограничили бы полезный диапазон приложений, возможных внутри парадигмы MVC. Требуется, чтобы любой объект мог быть моделью. Число с плавающей точкой мог бы быть модель для панели указателя скорости, который мог бы быть подпанелью более сложного панели приборной доски имитатора полета. Строка (String) дает совершенно пригодную модель для приложения-редактора (хотя немного более сложный объект называемый StringHolder обычно используется для таких целей). Потому что любой объект может играть роль модели, базисное поведение, требуемое для моделей, чтобы участвовать приложении созданном в соответствии с MVC-парадигмой, унаследовано от класса Object, который является суперклассом всех возможных моделей.

Взаимодействие Внутри Триады MVC

Модель, панель и контроллер, включаемый в MVC-триаду должна взаимодействовать между собой, чтобы приложение могло управлять логически последовательным взаимодействием с пользователем. Связь между панелью и связанным с нею контроллером прямая(простая), потому что классы View и Controller специально разработаны, чтобы работать вместе. Модели, с другой стороны, связываются более тонким способом.

Пассивная Модель

В самом простом случае, это для модели нет необходимости делать что-либо для участия в MVC-триаде. Простой WYSIWYG редактор текста — хороший пример. Центральная особенность такого редактора — то, что Вы должны всегда видеть текст, так как он появился бы на бумаге. Так что панель, очевидно, должна быть информирована о каждом изменении текста так, чтобы она могла модифицировать дисплей. Однако модель (за которую мы примем экземпляр класса String) не должны брать ответственность за сообщения об изменениях для панели, потому что эти изменения происходят только по запросами от пользователя. Контроллер может принимать ответственность за уведомление панели о любых изменениях, потому что он интерпретирует запросы пользователя. Он мог бы просто сообщать просмотру, что что-то изменилось — панель могла бы затем запрашивать текущее(актуальное) состояние строки из модели — или контроллер мог бы указывать панели, что изменилось. В любом случае, строковая модель — полностью пассивный держатель строковых данных, управляемых панелью и контроллером. Она добавляет, удаляет, или заменяет подстроки по запросам из контроллера и отрыгивать соответствующие подстроки после запроса из панели. Модель полностью "неосведомленна" о существовании и просмотра, и контроллера, и своего участия в MVC-триаде. Та изоляция - не артефакт простоты модели, но предложения что модельные изменения только по воле одного из других элементов триады.

Связь Модели с Триадой

Но все модели не могут быть настолько пассивны. Предположите, что объект данных — строка в вышеупомянутом примере — изменяется в результате сообщений из объектов, отличных от панели или контроллера. Например, подстроки могли бы быть присоединены к концу строки, как имеет место с SystemTranscript. В этом случае объекту, который зависит от состояния модели — панели — нужно сообщить, что модель изменилась. Так как только модель может отслеживать все изменения своего состояния, модель должна иметь некоторую связь с просмотром. Чтобы удовлетворить эту потребность, в классе Object поддерживается глобальный механизм слежения за зависимостями, типа тех что возникают между моделью и панелью. Этот механизм использует IdentityDictionary по имени DependentFields (переменная класса Object) который просто записывает все существующие зависимости. Ключи в этом словаре — все объекты, для которые зарегистрированы зависимости; значение, связанное с каждым ключом — список объектов, которые зависят от ключа. В дополнение к этому общему

механизму, класс Model обеспечивает более эффективный механизм для управления зависимостями. Когда Вы создаете новые классы, которые предназначены, чтобы функционировать как активные модели в MVC триаде, Вы должны делать их подклассами класса Model. Модели в этой иерархии сохраняют свои зависимые объекты в переменной экземпляра (dependents), который содержит или nil, одиночный зависимый объект, или экземпляр DependentsCollection. Панели полагаются на эти механизмы зависимости, для получения сообщений об изменениях в модели. Когда для модели создается новая панель, она регистрирует себя как объект, зависимый от этой модели. Когда панель уничтожается, она удаляет себя из зависимых объектов.

Методы, которые обеспечивают косвенную связь с зависимыми объектами, находятся в протоколе "изменения" (updating) класса Object. Откройте окно просмотра, и исследуйте эти методы. Сообщение changed инициализирует объявление всем зависимым от данного объекта, что в нем произошло изменение. Приемник сообщения changed посылает сообщение update: self каждому из своих зависимых объектов. Таким образом модель может сообщать любым зависимым панелям, что она изменилась, просто посылая себе сообщение changed. Панель (и любые другие объекты, которые зарегистрированы как зависимые от модели) получают сообщение update: с объектом модели в качестве параметра. [Обратите внимание: имеется также changed:with: сообщение, которое позволяет Вам передавать параметр зависимому объекту.] Заданный по умолчанию метод для сообщения update:, которое унаследовано от класса Object, не делает ничего. Но большинство панелей имеет протокол, чтобы перерисовать свое изображение после получения сообщение update:. Этот механизм изменения/обновления был выбран как канал связи, через который панели могут получать сообщения об изменениях внутри своей модели, потому что он задает наименьшее количество ограничений на структуру моделей.

Понять того, как это механизм изменения/обновления используется в MVC, откройте окно просмотра для отправителей сообщения changed (Smalltalk browseAllCallsOn: #changed) и другое для реализаторов сообщения update: (Smalltalk browseAllImplementorsOf: #update:). Обратите внимание что почти все реализаторы update: являются различными Панелями, и что их поведение должно модифицировать дисплей. Ваши панели будут делать что-то подобное. Отправители сообщений changed и changed: находятся в методах, которые изменяют некоторые свойства объекта, которые является важным для панели. И снова, использование Вами сообщения changed будет очень похоже на них.

Объект может действовать как модель для более чем одной триады MVC одновременно. Рассмотрите архитектурную модель здания. Позвольте нам игнорировать структуру самой модели. Важная вещь — чтобы имелась бы панель, показывающая компоновочный план, другая — для внешнего перспективного вида, и возможно еще одна панель для обзора внешних потерь тепла (для оценки энергетической эффективности). Каждая панель имела бы сотрудничающий контроллер. Когда модель изменена, все зависимые просмотры должны быть уведомлены. Если только подмножество этих панелей должно отвечать на данное изменение, модель может передавать параметр, который указывает для зависимых объектов, какого вида изменения произошли так, чтобы только заинтересованные объекты отвечали на изменение. Это выполнимо с сообщением changed:. Каждый приемник этого сообщения может проверять значение параметра, чтобы определить соответствующий ответ.

Связь Панель — Контроллер

В отличие от модели, которая может быть свободно соединена с многими MVC-триадами, каждая панель связана с уникальным контроллером и наоборот. Переменные экземпляра в каждом случае поддерживают эту плотную связь. Переменная экземпляра панели controller указывает на ее контроллер, и переменная экземпляра контроллера view указывает на ассоциированную панель. И, потому что оба должны быть связаны с моделью, каждый имеет переменную экземпляра model, которая указывает на объект модели. Так, хотя модель ограничена посылкой self changed:, и панель и контроллер могут посылать сообщения непосредственно друг другу и своей модели.

Панель берет на себя ответственность за установление этой многосторонней связи внутри данной MVC-триада. Когда Панель получает сообщение model:controller:, она регистрирует себя как зависимый от модели объект, устанавливает переменную экземпляра controller, чтобы указать на контроллер, и посылает сообщение view: self контроллеру, чтобы контроллер мог устанавливать переменную экземпляра view. Панель также берет на себя ответственность за отмену этих соединений. Сообщение release заставляет панель удалить себя из множества зависимых от модели объектов, послать сообщение release контроллеру, и затем послать release всем подпанелям.

Панели

Иерархия Панель/Подпанель

Панели разработаны, чтобы быть вложенными. Большинство окон фактически включает по крайней мере две панели, одна вложена внутрь другой. Наиболее внешняя панель, известная как topView — экземпляр класса StandardSystemView или одного из его подклассов. StandardSystemView управляет хорошо знакомым Вам ярлыком(заголовком) окна. Ассоциированный контроллер, который является образцом StandardSystemController, управляет перемещением, изменением размера, свертыванием, и закрытием — операциями, доступными для окон верхнего уровня. Внутри topView содержатся одна или более подпанелей. Их ассоциированные контроллеры, управляют опциями, доступными в этих панелях. Рабочее окно например имеет StandardSystemView как topView, и StringHolderView как его единственную подпанель. Подпанель может, в свою очередь, иметь дополнительные подпанели, хотя это не требуется в большинстве приложений. Связи подпанель/суперпанель записаны в переменных экземпляра, унаследованных из класса View. Каждая панель имеет переменную экземпляра, superView, который указывает на просмотр, который содержит ее, и другую переменную, subViews, которая является упорядоченным набором (OrderedCollection) подпанелей. Таким образом topView каждого окна — вершина иерархии панелей, прослеживаемой через переменные экземпляра суперпанели/подпанели. Обратите внимание однако, что некоторые классы панелей (например, BinaryChoiceView, и FillInTheBlankView) не имеют ярлыка (заголовка), и не могут изменять размер или перемещаться по экрану. Эти классы не используют StandardSystemView для topView; взамен они используют простой View.

Давайте рассмотрим пример, который формирует и запускает MVC-триаду. Этот пример — упрощенная версия кода, который открывает methodListBrowser — окно просмотра, которое Вы видите, когда выбираете из меню подпанели списка методов в системном окне просмотра пункт “реализаторы” или “отправителей”. Верхняя подпанель этого окна просмотра отображает список методов. Когда один из этих методов выбран, его код появляется в нижней подпанели. Мы пронумеровали строки кода для упрощения ссылок.

```
openListBrowserOn: aCollection label: labelString initialSelection: sel
    "Создают и планируют окно просмотра Списка Метода для методов в aCollection."
    | topView aBrowser |
1.  aBrowser := MethodListBrowser new on: aCollection.
2.  topView := BrowserView new.
3.  topView model: aBrowser; controller: StandardSystemController new;
4.          label: labelString asString; minimumSize: 300@100.
5.  topView addSubView:
6.      (SelectionInListView on: aBrowser printItems: false oneItem: false
7.      aspect: #methodName  change: #methodName:  list: #methodList
8.      menu: #methodMenu  initialSelection: #methodName)
```

```
9.          in: (0@0 extent: 1.0@0.25) borderWidth: 1.
10.     topView addSubView:
11.         (CodeView on: aBrowser  aspect: #text  change: #acceptText:from:
12.             menu: #textMenu  initialSelection: sel)
13.             in: (0@0.25 extent: 1@0.75) borderWidth: 1.
14.     topView controller open
```

Давайте рассматривать этот текст программы строка за строкой. После создания [1] модели, мы создаем topView [2]. Обычно это будет StandardSystemView, но здесь мы используем BrowserView, который является подклассом StandardSystemView. Строка [3] определяет модель и контроллер. [Обратите внимание: Если контроллер явно не обеспечивается, метод панели defaultController будет обеспечивать контроллер, когда контроллер панели будет запрошен впервые. Многие приложения определяют контроллер косвенно в этом заданном по умолчанию методе, а не явно задают контроллер, когда открывается панель.] Следующая строка задает текст ярлыка и минимальный размер topView [4]. Строки [5-9] устанавливают верхнюю подпанель, которая является SelectionInListView. Нижняя панель (экземпляр CodeView) устанавливается строками [10-13]. Оба из эти типа панелей известны как "подключаемые панели" Они будут обсуждаться более подробно ниже. Рассмотрим подробнее в строки [9] и [13], в которых указывается размещение подпанелей внутри прямоугольника, занятого topView. Имеется ряд способов указать панели, как она должна разместить подпанели, например с помощью addSubView:below: или insertSubView:above:. Они находятся в протоколе вставки подпанелей класса View. Здесь положение панелей даны относительно канонического прямоугольника 1.0@1.0. Ваш код не должен зависеть от окончательного размера и формы topView окна. Верхняя панель помещена в верхний левый угол (то есть, в 0@0) и может занять полную ширину, но только верхние 25% высоты topView (то есть протяженность: 1@0.25). Нижняя подпанель помещена в 0@0.25 и может занять оставшуюся часть окна (1@0.75). Каждая панель имеет границу шириной (borderWidth) 1 пиксел. В заключение, открывается контроллер [14], что заставляет окно инициализировать процесс определения своего размера — курсор становится курсором верхнего левого угла так, чтобы пользователь мог задать положение и размер окна. Ваши собственные MVC приложения будут обычно открываться также.

Изображение панелей

Ваш просмотр может нуждаться в собственном протоколе изображения. Этот протокол будет использоваться, и для начального изображения вашего просмотра, и для перерисовки изображения, когда модель сообщает об изменении (и возможно для перерисовки изображения вызываемой контроллером). Точнее, метод update: в классе View посылает self display. Метод display из класса View, в свою очередь посылает:

```
self displayBorder.
self displayView.
self displaySubviews.
```

Если ваша панель требует какого-либо специального поведения при ее изображении, поведения, другого, чем унаследованное, оно будет определяться в одном из этих трех методов. Вы можете просмотреть список реализаторов метода displayView чтобы найти примеры различных способов отрисовки. Если Вы это сделаете, Вы обратите внимание, что некоторые методы изображения используют преобразования изображения.

Преобразования изображения — экземпляры WindowingTransformation. Они обрабатывают масштабирование и перенос (сдвиг), чтобы связать окна (aWindow) и области просмотра (aViewport). Окно — ввод к преобразованию. Это — прямоугольник в абстрактном пространстве изображения с любой произвольной системой координат, которую Вы находите наиболее соответствующим для вашего приложения. О области просмотра можно думать как специфической прямоугольной области экрана дисплея, в которую это абстрактное окно должно быть отображено. Класс WindowingTransformation вычисляет и применяет к изображаемым объектам, таким как точки и прямоугольники, параметры масштабирования и переноса так, чтобы aWindow, когда преобразовано, соответствовало aViewport. Однако преобразование просто масштабирует и переносит один набор координат в другой, и следовательно, нет никакой необходимости связывать его с экраном дисплея; преобразования могут быть использованы и для других целей.

Можно составлять суперпозиции экземпляров WindowingTransformation, и обращать их. Панели используют преобразования, чтобы управлять размещением подпанели. Вы также можете использовать их, если Вы должны рисовать непосредственно в панели. Посылка сообщения displayTransform: anObject применяет преобразование изображения приемника к anObject и возвращает возникающий в результате отмасштабированный и перемещенный объект. Он обычно применяется к прямоугольникам (Rectange), точкам (Points), и другим объектам с координатами, определенными в локальной системе координат панели, чтобы получить отмасштабированный и перенесенный объект в аппаратных координатах устройства дисплея. Посылка сообщения displayTransformation панели возвращает экземпляр WindowingTransformation, который является результатом суперпозиции всех локальных преобразований в цепочке суперпанелей приемника с собственным локальным преобразованием приемника. Посылка панели сообщения inverseDisplayTransformation: aPoint используется контроллерами при ответе на сообщение redButtonActivity, чтобы преобразовать aPoint (например, Sensor cursorPoint) от аппаратных координат устройства к координатам окна панели.

Примечания относительно Существующих Просмотров

Ваши первые панели приложений начнутся с существующих панелей, аннотированный список которых находится в конце этого раздела. Некоторые из них, типа инспекторов, окон просмотра и отладчиков — полные приложения, которые Вы можете использовать как примеры. Другие — универсальные панели, которые Вы используете целиком как подпанели в ваших приложениях. Некоторые из них Вы, без сомнения, захотите усовершенствовать, создавая подклассы с более специализированным поведением. Можно узнать много полезного об использования данной панели, просто просматривая методы создания панелей, которые используют эту панель. Например, выполнение Smalltalk browseAllCallsOn: (Smalltalk associationAt: #SwitchView) предоставит Вам окно просмотра методов для всех методов, которые посылают сообщения классу SwitchView. Среди них будет методы создания экземпляров других панелей, которые используют данную панель как подпанель. Вы можете использовать их

как примеры того, как это делается.

Четыре из универсальных существующих просмотров — BooleanView, SelectionInListView, TextView и CodeView — являются особенно гибкими. Их вызывают "подключаемые панели." Их дополнительная гибкость разработана, чтобы уменьшить потребность в создании многих подклассов, которые отличаются только по селекторам метода, используемым, чтобы выполнять общие задачи типа получения данных из модели или использования различных yellowButtonMenu. Подключаемые панели и ассоциированные с ними контроллеры выполняют эти задачи, вызывая селекторы "адаптера", переданные к ним во время создания экземпляра. SelectionInListView и CodeView, используемое как подпанели в методе openListBrowserOn:, показанном в предыдущем разделе — примеры. Обратите внимание, что параметры, переданные методам создания этих панелей — имена селекторов. Комментарий класса каждой подключаемой панели определяет селекторы, которые должны передаваться этой панели.

Существующая Иерархия Панелей

View — используется как нестандартный topView для BinaryChoiceView и FillInTneBlankView

- **BinaryChoiceView** — бинарный переключатель
- **SwitchView** — используется в BitEditor и FormEditor как кнопки инструментов
  - **BooleanView** — [подключаемая] панель используемая для переключателя экземпляр/класс окна просмотра
- **DisplayTextView** — используется для сообщения в верхней подпанели да\нет подсказчик
- **TextView** — [подключаемый] не используемый в обычном образе V2.5
  - **CodeView** — [подключаемая] панель используемая для подпанели окна просмотра, которая показывает код
    - **OnlyWhenSelectedCodeView** — используется в нижней подпанели FileList
- **StringHolderView** — используется рабочими окнами
  - **FillInTheBlankView** — известный промптер вопроса с текстовым ответом
  - **ProjectView** — панель описания проекта
  - **TextCollectorView** — используется транскриптом
- **FormMenuView** — используемый BitEditor и FormEditor для кнопок
- **FormView** — используемый BitEditor и фоновым экранным серым InfiniteForm
  - **FormHolderView** — используемый BitEditor и FormEditor для редактируемой формы
- **ListView** — не используется в обычном образе V2.5
  - **ChangeListView** — полное приложение
  - **SelectionInListView** — [подключаемая] панель, используемая для списковой подпанели окна просмотра
- **StandardSystemView** — представляет общую функциональность topView
  - **BrowserView** — полное приложение
  - **InspectorView** — полное приложение
  - **NotifierView** — окно сообщения об ошибке, например, "Объект не понимает"

Контроллеры

Smalltalk-80 представляет воззрение, что управление постоянно осуществляется мышью. Когда пользователь двигает и щелкает мышью, объекты на экране Smalltalk-80 е действуют почти как оркестр, подчиняющийся своему дирижеру. Но здесь, фактически, нет никакой единой деспотичной силы. Одиночная нить управления поддерживается сотрудничеством контроллеров, присоединенных к различным активным панелям. Только один контроллер фактически имеет управление в любой момент времени. Так что вся сложность в том, чтобы делать это соответствующим образом!

Связь Между Контроллерами

Первичный организующий принцип, который делает этот прием возможный — то, что активные контроллеры для каждого проекта формируют иерархическое дерево. В корне этого дерева — глобальная переменная ScheduledControllers, который является экземпляром класса ControlManager, присоединенным к активному проекту. Ветви, растущие из ScheduledControllers — контроллеры верхнего уровня каждого активного окна, плюс дополнительный контроллер, который управляет основной системой yellowButtonMenu доступной на сером экранном фоне. Так как каждая панель связана с уникальным контроллером, дерево панель/подпанель наводит параллельное дерево контроллеров внутри каждого topView. Дальнейшие ветви из каждого контроллера верхнего уровня следуют этому наведенному деревом. Управление переходит от контроллера к контроллеру по ветвям этого дерева.

Проще говоря, поток управления требует совместного действия хорошо воспитанных контроллеров, каждый из которых вежливо отказывается от управления, если курсор не находится в его панели. И после получения управления хорошо воспитанный контроллер пытается полагаться на контроллер некоторой подпанели. ControlManager на верхнем уровне спрашивает каждый из контроллеров активных topView, требуется ли им управление. Только тот, чей просмотр содержит курсор, отвечает утвердительно и получает управление. Он, в свою очередь, делает опрос контроллеров подпанелей. Снова тот, который содержит курсор получает управление. Этот процесс находит самую внутреннюю вложенную панель, содержащую курсор и, вообще, контроллер панели сохраняет управление, пока курсор остается в этой панели. (Более детализированное описание этого потока управления содержится в Приложении А.) В этой схеме, передача управления вовлекает сотрудничество всех активных панелей и контроллеров в запутанно координированный менуэт. Панели требуются, чтобы опросить контроллеры их подпанелей. Контроллеры спрашивают свои панели, содержат ли они курсор. По этой причине, это не принято — и опасно — делать изменения в ваших панелях или контроллерах, которые вызывают нестандартный поток управления. Запомните это твердо, когда Вы впервые попытаетесь установить новый контроллер приложения, потому что неосторожный сбой потока управления приведет к разрушению системы. Благоразумный программист сохраняет резервную копию системы перед такой попыткой!

Жизненно важная роль, играемая контроллерами подразумевает, что Вы не можете иметь пару модель-панель без контроллера. Если этому позволяли, поток управления исчезнет в промежутке, оставленном отсутствующим контроллером. Однако

имеются некоторые случаи, где Вы могли бы хотеть набор подпанелей, управляемых коллективно из контроллера содержащей их панели, а не из индивидуальных контроллеров подпанелей. Специальный контроллер для таких подпанелей обеспечивается классом NoController, который специально создан, чтобы отказаться от управления.

Танец полос прокрутки среди подпанелей окна просмотра по мере того как курсор движется между ними — наиболее видимое следствие потока управления. При пересечении границы подпанелей курсором внутри окна просмотра полоса прокрутки покидаемой подпанели исчезает, а полоса прокрутки для подпанели, в которую переместился курсор, появляется. Это выполняется методами controlInitialize и controlTerminate тех подпанелей с контроллерами, которые наследуются из класса ScrollController. Когда курсор выходит из данной подпанели, viewHasCursor возвращает ложь, и контроллер выполняет метод controlTerminate, который восстанавливает изображение области, ранее закрытой полосой прокрутки. Затем соответствующая часть дерева панелей/контроллеров пересматривается, чтобы найти контроллер, который теперь должен получить управление. Если эта панель имеет полосу прокрутки, метод controlInitialize нового контроллера сохраняет область дисплея, которая должна быть покрыта полосой прокрутки, а затем изображает полосу прокрутки.

Вход и оставление потока управления

Не забудьте, что этот манерный менуэт постоянно продолжается. Как же ваша недавно созданная MVC-триада должна войти в процесс, и как выйти из него, когда она выполнилась?

Сначала, контроллер вашего topViews один ответственен за вхождение в этот процесс. Он затем передает управление на контроллеры подпанелей (который фактически делают большую часть работы в типичном приложении). Контроллеры верхнего уровня должны быть потомками класса StandardSystemController, который разработан, чтобы быть контроллером просмотра верхнего уровня. Сообщение open к StandardSystemController заставляет ваш новый MVC стать ветвью верхнего уровня дерева управления. Сообщение open должно быть последним сообщением в методе, который создает новый MVC, потому что управление не возвращается из этого сообщения. Код, появляющийся после сообщение open не будет выполнен. Метод controlTerminate класса StandardSystemController берет ответственность за исключение контроллера из планирования процессов, когда окно закрыто.

MouseMenuController

Большинство приложений использует мышь для указания и вызова меню. Большинство контроллеров следовательно установлено где-нибудь в иерархии классов ниже MouseMenuController, который обеспечивает основные переменные и методы экземпляра. Соответствующие переменные экземпляра — red-, yellow-, blueButtonMenu и red-, yellow-, blueButtonMessages, в который Вы заносите ваши меню и связанные с ними сообщения. Важные методы — redButtonActivity, yellowButtonActivity, и blueButtonActivity. Метод контроллера controlLoop, как подразумевает его имя, — основной цикл управления. Каждый раз через этот цикл, это посылает себе сообщение controlActivity. Этот метод переопределен в MouseMenuController, чтобы проверить каждую из кнопок мыши и, например, посылать себе redButtonActivity, если красная кнопка нажата, и панель содержит курсор.

Метод xxxButtonActivity проверяет не nil ли xxxButtonMenu, и если найдено, посылает сообщение:

self menuMessageReceiver perform: (redButtonMessages at: index). Обратите внимание: menuMessageReciever обычно возвращается self — то есть, контроллер — так, чтобы протокол сообщений меню обычно находился в контроллере.

Все контроллеры верхнего уровня — экземпляры StandardSystemController или его подклассов. StandardSystemController — подкласс MouseMenuController, который специализирован для нахождения в верхнем уровне иерархии контроллера окна. Это управляет обычным blueButtonMenu — frame, close, collapse, ... (размер, закрыть, свернуть, ...) поведением окон — которое применимо к topView. Ваши контроллеры подпанели не должны быть подклассами StandardSystemController. Вместо этого они должны выводится независимо из MouseMenuController. Однако функции меню голубой кнопки должны все еще обрабатываться контроллером верхнего уровня. Чтобы гарантировать это, ваш контроллер может перенаправить нажатие голубой кнопки на верхний уровень следующим методом blueButtonActivity:

```
blueButtonActivity
    view topView controller blueButtonActivity.
```

Это наиболее прозрачный способ чтобы человек, просматривающий ваш код контроллера, немедленно увидел, что голубая кнопка обрабатывается контроллером topView. Более тонкий подход — для контроллера подпанели, чтобы отказаться от управления, если нажата голубая кнопка . Это делается в вашем isControlActive методе:

```
isControlActive
    ^super isControlActive & sensor blueButtonPressed not.
```

В типичном случае, ваш контроллер будет иметь собственное yellowButtonMenu, и возможно использовать красную кнопку для некоторого вида указания или функции выбора. Вы будете обычно делать ваши меню похожими на следующее:

```
PopUpMenu labels:
'foo baz
over there
file out
new gadget'
      lines: #(1 3).
```

Это обеспечивает меню с данными опциями, и с разделяющими линиями после первых и третьих строк. Вы затем устанавливаете параллельный список сообщений, например, #(fooBaz overThere fileOut newGadget). Меню и список сообщений должны находится в переменных экземпляра yellowButtonMenu и yellowButtonMessages. Это может быть выполнено на лету если необходимо, но при более типичном подходе формируют меню, и сообщения в методе инициализации класса и устанавливают их в методе инициализирующем экземпляр. Для примера, рассмотрите метод инициализации класса ChangeListController. В заключение, Вы должны установить методы реализующие сообщения меню. Они традиционно находятся в протоколе сообщений меню вашего контроллера. Если Вы желаете использовать нажатие красной кнопки для действия отличного от выбора из меню, переопределите метод redButtonActivity вашего контроллера, чтобы выполнить желаемое действие. В вашем новом методе redButtonActivity Вы можете проверять другие условия типа Sensor leftShiftDown, чтобы обеспечить большую гибкость. Вы можете видеть примеры таких методов redButtonActivity в FormEditor, ListController, и SelectionInListController.

Меню, несмотря на использование ими экрана дисплея, не обрабатываются MVC-триадами. Они управляются своим собственным экранным устройством отображения и управлением, включая сохранение и замену содержимого экрана в области, скрытой меню. В дополнение к PopUpMenu в вышеупомянутом примере, Вы должны исследовать ActionMenu, который является особенно полезным для подключаемых панелей.

ParagraphEditor

Все контроллеры, которые принимают ввод текста клавиатуры, находятся ниже ParagraphEditor в иерархии класса Controller. ParagraphEditor предшествует полной разработке MVC парадигмы. Он обрабатывает и ввод текста и отображение текста на дисплее, следовательно он находится некоторым образом пересечении между панелью и контроллером. Панели, которые используют его или его подклассы в качестве контроллера, обращают обычные роли при рисовании; они выполняют рисование текста, посылая сообщение display контроллеру. Множественность ролей, играемых классом ParagraphEditor делает его сложным объектом. Он управляет специальными клавишами (например, Ctrl-T отображается в ifTrue:). Он также управляет выбором текста, выбором шрифта и его размера, и форматированием текста (то есть пропорциональным интервалом между символами и разбиением на строки, приспособленные к ширине панели). Поскольку он находится наверху иерархию контроллеров обрабатывающих текст, Вы имеете свободный доступ ко всей его мощи. Но сложность классов обработки текста несет реальный непроизводительный штраф. Эти непроизводительные затраты наиболее видимы в раздражающей задержке между печатью символа и его изображением на экране. Три класса, которые выполняют большую часть обработки текста — ParagraphEditor, Paragraph, и CharacterScanner. Некоторые Smalltalk-80 программисты создали параллели к каждому из этих классов, которые обходятся без возможностей, потребляющих много времени. Они достигли скоростей обработки текста для специализированных приложений в несколько раз больших, чем обеспечиваемые стандартными классами.

ScreenController

Серый экранный фон и yellowButtonMenu доступное на этом фоне — не специальные исключения в MVC парадигме; они также управляются MVC-триадой. Модель — InfiniteForm (colored gray), панель — экземпляр FormView, и контроллер - единственный экземпляр класса ScreenController. Чтобы добавлять опцию к основному экранному меню (возможно опцию printScreen), Вы редактируете метод класса initialize класса ScreenController (в протоколе инициализации класса) и добавляете ваш метод в протокол сообщений меню. Не забудьте выполнить комментарий в конце метода инициализации класса, чтобы установить ваше новое меню. Вы могли бы также желать рассмотреть другие методы в протоколе сообщений меню, чтобы увидеть, как отрываются окна просмотра, списки файлов, или рабочие окна.

MVC Инспектор

Так как MVC-триада настолько важна, Smalltalk-80 обеспечивает специализированного инспектора — MVC-инспектора — чтобы исследовать все три объекта сразу. Вы вероятно будете часто использовать этот инспектор, когда Вы начинаете создавать ваши собственные приложения, и как средство для наблюдения, как работают другие MVC-триады, и как неоценимая помощь в отладки, чтобы увидеть, почему ваш собственные приложения не работают. Вы открываетесь, MVC инспектор, посылая сообщение inspect любой панели. Самый простой способ использовать MVC-инспекторов — загрузить (fileIn) дополнение BLUEINSP.ST входящее в дистрибутив Smalltalk-AT. [Обратите внимание: не все дистрибутивы Smalltalk-80 включают это дополнение.] Оно устанавливает пункт меню синей кнопки "inspect" для всех topViews. Этот пункт меню открывает MVC-инспектора на панели, ее модели и ее контроллере. Если только Вы понимаете немного MVC-триаду, MVC-инспектор позволит Вам, шарить во внутренностях любого окна, которое захватывает ваше воображение. В сложной панели типа окна просмотра, с многими подпанелями, Вы можете следовать за цепочкой подпанелей вниз, открывая далее MVC-инспекторы на выбранных подпанелях.

Как упражнение, откройте MVC-инспектор на окне System Transcript (сначала убедитесь, что SystemTranscript открыт на вашем экране). Начните, открывая инспектора на экземплярах DependentsCollection, выполняя DependentsCollection allInstances inspect. Найдите объект с двумя зависиммыми: "StandardSystemView" и "aTextCollectorView". Выберите TextCollectorView, и скомандуйте inspect. Теперь Вы будете иметь MVC инспектора на TextCollector, просмотре, и контроллере. Например, в верхней подпанели модели, выберите переменную contents: в правом окне Вы увидете тот же самый текст который Вы видите в вашем транскрипте. В разделе контроллера нижней части изучают и осматривают меню кнопок и их сообщения. Они будут знакомы Вам по транскрипту. В среднем разделе просмотра, выберите суперпанель: это будет StandardSystemView. Выберите его, и скомандуйте inspect. Вы получите другой MVC-инспектор для topView транскрипта. Эти два MVC инспекторы вместе дают доступ к всей структуре приложения SystemTransript. Обратите внимание, обе MVC-триады совместно используют ту же самую модель — TextCollector.

[Обратите внимание: В версиях Smalltalk-80 до вложения класса Model, Вы будете должны начать по-другому: начните, открывая инспектор на словаре DependentsFields (словарь, в котором поддерживаются все объектные зависимости) выполняя Object classPool at: #DependentsFields) inspect. Это дает Вам доступ к инспектору на IdentityDictionary. Найдите пункт с ключом "aTextCollector," который является моделью транскрипта. Выберите его, и выберите опцию inspect в yellowButtonMenu. Это откроет инспектор на OrderedCollection объектов, зарегистрированных как зависимые от TextCollector. В этом новом инспекторе, одна из единиц будет TextCollectorView. Выберите TextCollectorView, и команду inspect.]

Приложение А: Подробности о потоке управления

Выше всех контроллеров topViews находится ScheduledControllers — экземпляр ControlManager, присоединенный к текущему проекту. Этот controlManager определяет, кто из контроллеров окон (topView) должен быть активен. Он использует метод searchForActiveController, который по очереди посылает метод isControlWanted всем контролерам из ScheduledControllers. Когда он находит соответствующий контроллер верхнего уровня, он посылает этому контроллеру сообщение startUp, унаследованное от класса Controller. Этот контроллер посылает себе следующие три сообщения:

```
self controlInitialize.
self controlLoop.
self controlTerminate.
```

Метод controlLoop обрабатывает поток управления:

```
[self is ControlActive] whileTrue: [Processor yield.  self controlActivity]
```

Выражение self controlActivity просто говорит: self controlToNextLevel. Здесь поток управления передается прямо панели с помощью кода:

```
aView := view subViewWantingControl.
aView ~~ nil ifTrue: [aView controller startUp]
```

который только просит, чтобы просмотр определил, какая из подпанелей нуждается в управлении, и если такая подпанель нашлась, передает управление контроллеру этой панели. Метод isControlWanted просто возвращает результат сообщения self viewHasCursor, которое в свою очередь просто говорит view containsPoint: Sensor cursorPoint. Таким образом, на дне потока процесса управления — вопрос кокая из панелей содержит курсор. Единственное исключение — NoController, который переопределяет isControlWanted, чтобы этот метод всегда возвращал ложь.