Юрг Гуткнехт

Oberon: перспективы эволюции

Juerg Gutknecht (1994) Oberon — Perspectives of Evolution // Ulm, Germany, p.395-400.
Р. Богатырев, А. Китаев, перевод с англ.

О качестве базового программного обеспечения следует судить по заложенному в него потенциалу дальнейшего развития, а не по тому, что оно дает пользователю сегодня. Поэтому, взявшись за статью о системе Oberon, я решил посвятить ее эволюционным возможностям этой системы. Сформулировать мои цели более точно можно двояко. Задача-минимум данной статьи — просто рассказать о стадиях (отчасти уже пройденных, отчасти остающихся пока перспективой) эволюции исходной версии операционной системы Oberon:

(1) Oberon с объектами и архитектура приспособлений (gadgets) графического интерфейса,
(2) Oberon с поддержкой параллельности в форме активных объектов,
(3) Oberon с доступом к удаленным службам.

Но более всего (и это уже задача-максимум) мне хотелось бы убедить читателя в справедливости следующей мысли: программный продукт родового (generic) характера, построенный на основе универсальных и унифицированных концепций, хорош не просто потому, что так написано в учебниках. Все дело в том, что именно такие идеи закладывают в программное обеспечение мощный потенциал, открывающий широкие возможности для дальнейшего развития.

Введение

Обычно мы считаем, что качество программного обеспечения тем выше, чем больше специальных функций оно нам предоставляет. Пожалуй, такой критерий вполне приемлем, если речь идет о пакетах, предназначенных для конечных пользователей, но он совершенно неуместен в случае со средствами для промежуточных пользователей (middle-user software), т.е. для базовых систем, таких как операционные среды. Достоинством подобных программных продуктов является уже не наличие в них специальных функций, а как раз отсутствие таковых. Причина очевидна: любая специальная функция неизбежно понижает степень универсальности системы и ограничивает потенциал развития на базе положенных в ее основу идей.

Если бы ценность программного продукта определялась исключительно отсутствием специальных функций, то в абсолютные чемпионы выходили бы пустые программы. И поскольку это недопустимо, нам следует незамедлительно пополнить список еще одним критерием. Скорее всего, в качестве такового лучше всего подходит пригодность к многоразовому использованию. Таким образом, нам требуются универсальные концепции, которые можно использовать многократно и в разных вариациях.

Цель этой короткой статьи как раз и состоит в том, чтобы показать на примере Oberon, как универсальные концепции облегчают процесс эволюции и даже стимулируют его. Итак, мы начинаем с краткого обзора движения исходной версии системы Oberon от платформы, ориентированной исключительно на тексты, к полностью интегрированной и вполне завершенной объектно-ориентированной среде. Затем мы покажем некоторые возможные перспективы на будущее, как они видятся нам в свете накопленного опыта.

Интеграция объектов в Oberon,
или как с помощью эволюции добиться революции

В 1985 г., когда мы начинали работу по проекту Oberon, хорошо интегрированные текстовые интерфейсы (textual user interfaces — TUI) были еще в новинку и редко встречались в операционных системах. К тому же они оптимально подходили к ориентированному на пользователя аппаратному обеспечению, т.е. к обладающим высокой разрешающей способностью черно-белым экранам сравнительно низкого быстродействия. Принципиально ситуация стала меняться только в 1990-е годы, и главным образом — благодаря мощному прорыву в технологии производства дисплейных аппаратов, которая сегодня дает возможность применять высокоэффективную цветную графику при весьма низких затратах.

Графические интерфейсы ныне используются повсеместно, и игнорировать это проявление духа времени не может себе позволить ни одна серьезная операционная система. Поэтому летом 1991 г. мы приступили ко второй стадии проекта Oberon. Однако, в ту пору мы хотели не просто облачить по сути "голую" систему Oberon в нарядные GUI-одежды, но и сделать важный шаг вперед. Несколько раздвигая границы устоявшихся представлений, мы сразу стали рассматривать графический интерфейс как набор истинных визуальных объектов.

Этот новый взгляд на вещи, а также желание добиться максимально возможной интеграции побудили нас первым делом оснастить ядро исходной версии Oberon [1] функцией общего управления объектами и затем, уже на основе этого нового ядра (которое получило название System 3), реализовать в высшей степени гибкую GUI-архитектуру, которую мы назвали Gadgets [2, 3]. Хотя эта модернизация и по масштабам, и в плане функциональности была весьма радикальной, все прошло на удивление гладко. Когда сегодня вспоминаешь об этом, кажется, будто в то время мы просто использовали некую изначально заложенную возможность.

Из нашего опыта можно извлечь важный урок: универсальные идеи, как правило, обеспечивают системе широкие эволюционные возможности. Хорошей тому иллюстрацией служит универсальная схема модель/вид (model/view), суть которой состоит в четком разделении модели и ее (подчас многочисленных и различных) визуальных представлений. В исходной версии системы Oberon мы находим строгую реализацию схемы модель/вид, состоящую из:

(1) абстрактной текстовой модели;
(2) представлений отформатированного текста;
(3) механизма, оповещающего текстовые представления об изменениях в модели текста, на которой они построены.

Эта схема, которая естественным образом обобщает тексты, превращая их в произвольные категории, оказалась одной из важных концептуальных основ нашего проекта, имевшего целью модернизацию системы и выведение ее на объектно-ориентированный уровень.

Достоинства схемы модель/вид удачнее всего иллюстрируют два типичных, хотя и весьма непохожих друг на друга примера. Рассмотрим "рабочий стол" пользователя системы Gadgets с размещенным на нем инструментом для раскраски. Инструмент являет собой панель, содержащую (помимо прочих) следующие элементы: цветовую карту, два "бегунка" и три текстовых поля. Данные элементы, представляющие соответственно некоторый выбранный цвет и его RGB-распределение, семантически связаны. Это дает нам возможность весьма изящно интерпретировать семь имеющихся элементов как различные представления одной и той же модели типа Color, на которой они построены.

Второй пример иллюстрирует применение схемы модель/вид в более широком контексте. Мы можем отнести его к категории отделения прикладной программы от ее пользовательского интерфейса. Подключив имитатор ждущего сигнала (реализован в виде структуры модель/вид) не к видеотерминалу, а к некоей абстрактной модели, мы добьемся полной независимости приложения (имитатора) от интерфейса (видеотерминала). Разумеется, впоследствии мы можем разработать какой угодно удобный пользовательский интерфейс и рассматривать его как еще одно представление абстрактной модели — и все это без малейших изменений хотя бы одного оператора прикладной программы.

Здесь нам уместно чуть подробнее коснуться схемы модель/вид и кратко остановиться на ряде технических подробностей, касающихся ее реализации. Вспомним для начала древовидную

структуру пространства отображения, для которой характерно иерархическое размещение данных на экране монитора. С технической точки зрения это дерево представляет собой неоднородную структуру данных с базовым типом узла, который есть не что иное, как (прямоугольная) рамка в зоне экрана. Одновременно это дерево задает структуру с расширенными (специализированными) типами узлов, которые соответствуют следующим структурным категориям: зона экрана (корень), видеодорожка (первый уровень), устройство отображения (второй уровень) и представление данных (третий уровень). По идее, уведомление о каком-либо изменении в любой модели передается в пространство отображения, т.е. сообщение об изменении сначала направляется в зону экрана, при этом оно должно быть передано на все видеодорожки (а при этом, в свою очередь, такое сообщение должно быть передано на все устройства отображения и т.д.), так что в конце концов возникает возможность вновь согласовать все затронутые этим представления данных.

Следует подчеркнуть, что описанный выше механизм оповещения является в высшей степени объектно-ориентированным и строится на том, что адресаты принимают и, по крайней мере, передают сообщения, даже когда конкретная семантика последних им неизвестна. Иными словами, данный механизм оповещения требует объектов с некоторым новым типом средств взаимодействия при передаче сообщений, который мы называем родовым (generic) интерфейсом. В Oberon мы можем легко выразить осуществляющие проверку типов родовые средства взаимодействия при передаче сообщений, добавляя к сообщениям расширения типов или, выражаясь более точно, используя связанные (instance-bound) процедуры с формальным параметром записи сообщения, который может быть расширен во время вызова.

Все дело в том, что на основе изложенной выше схемы модель/вид можно легко выходить на желаемые произвольные объекты и GUI-приспособления (именно поэтому, кстати говоря, наши технические исследования вполне окупились). Мы просто по-новому интерпретируем прямоугольные области (рассматривая их как визуальные объекты) и допускаем в пространстве отображения вложения произвольной глубины. Рассмотрим структуру нового пространства отображения, состоящего из визуальных объектов. Она продолжается снизу объектами модели (ссылками на них) и в действительности новое пространство уже не является деревом. Это последнее качество представляет собой логическое следствие существования так называемых аспектов (camera-views), которые разделяют общую структуру данных наблюдаемых нами объектов.

Важно отметить также, что мы расширили вышеописанную стратегию оповещения, распространив ее не только на широковещательные типы сообщений, но и на нешироковещательные. Сообщения визуальным объектам согласованно направляются в корень пространства отображения с неявным запросом передать сообщение реальным целевым объектам. У этой стратегии несколько интересных особенностей. Во-первых, она подчеркивает функцию контроля над объектами более низкого уровня или даже управления ими со стороны объектов более высокого уровня (подобно тому, как панельные GUI-приспособления управляют своими элементами). Во-вторых, она делает возможной контекстнозависимую обработку сообщений.

Идея предпринятого нами технического экскурса в глубины универсальной схемы модель/вид состоит не столько в ее подробном техническом разъяснении, сколько в демонстрации потенциала развития, заложенного в универсальных концепциях. Памятуя об этом, прервем здесь наши рассуждения и обратимся ко второму примеру, совсем не похожему на первый. Начнем с обстоятельства на первый взгляд весьма незначительного — с того, как в исходной версии системы Oberon определяется понятие "текст".

Обычно текст представляет собой не более чем последовательность знаковых кодов. Однако, учитывая прогресс в области книгопечатания, мы благоразумно взглянули на проблему в более широком контексте и пришли (говоря несколько упрощенно) к определению Oberon-текста как последовательности пар (знаковый код, фонт). В то время мы еще не знали, насколько полезным в концептуальном плане окажется это определение позднее, при решении проблемы "чистой" интеграции введенных нами в текст общих объектов.

И вновь для решения проблемы оказалось достаточно лишь несколько сместить акценты в интерпретации. Стоит взглянуть на фонты не как на атрибут, а как на набор знаковых схем (patterns), и мы сразу приходим к определению обобщенного текста как набора пар (библиотека, предметный указатель), где библиотека — это индексированное множество произвольных объектов. Несмотря на то, что понятие библиотеки было введено из чисто прагматических соображений, оно стало в конечном итоге незаменимым, естественным и унифицированным инструментом обработки ключевых элементов вроде сохраняемых (persistent) объектов, а также

связывания (linking) и встраивания (embedding) объектов. Обратите внимание, что термин "общедоступные (public) библиотеки" мы используем для обозначения связывания объектов, а "частные (private) библиотеки" — для обозначения встраивания объектов. Частные библиотеки обычно инкапсулируются документом, например, обобщенным текстом или панельным приспособлением.

**Интеграция параллельности в Oberon,
или как активизировать объекты**

В предыдущем разделе мы рассмотрели объектно-ориентированную версию базовой системы Oberon и показали на примере схемы модель/вид, как универсальные концепции стимулировали ее развитие. В данном разделе мы сделаем еще один шаг на этом пути и поговорим о перспективах. Отметив, что представленная схема модель/вид является чисто статической, мы начинаем с вопроса о том, как в нее вписываются динамические объекты. В учебниках пишут — с помощью контроллера. Контроллеры расширяют схему модель/вид и превращают ее в схему "модель/вид/контроллер" (MVC, model/view/controller), впервые предложенную разработчиками языка Smalltalk [4].

Мы же лишь проанализировали эту схему, в особенности — роль контроллера, и сделали при этом несколько важных выводов.

Отметим для начала, что интерактивные приложения обычно контролируются посредством взаимодействия в связи с представлениями, так что в этом случае мы имеем естественные переплетения типа "вид/контроллер". Так, в первоначальной версии Oberon в каждом фрейме текста (форматированные представления текста) встроен редактор/интерпретатор текста и, по логике, в каждое визуальное приспособление в системе Gadgets встроен дизайнер/редактор/интерпретатор. Теперь уместно напомнить, что и здесь только эффективная поддержка переплетений вид/контроллер обеспечивается универсальной концепцией динамической активации команд (вызов вида "Модуль.Процедура").

Впрочем, существуют и принципиально иные виды прикладных программ. Вернемся для примера к нашему имитатору ждущего сигнала, и к его структурной декомпозиции. В этом случае контроллер (имитатор) представляет собой отдельный логический процесс, который дистанционно воздействует на моделируемый объект. Мы полагаем, что такая структурная декомпозиция довольно неудачна и искусственна, и у нас опять-таки возникает искушение применить свой волшебный инструмент унификации необходимых нам артефактов "объект-процесс", которые мы просто будем называть активными объектами. Обратите внимание, что объекты в традиционном смысле пассивны, ибо они, как правило, просто ждут сообщений, которые появляются время от времени, и лишь затем реагируют на них.

Мы утверждаем, что активные объекты представляют собой идеальные абстракции многочисленных типов приложений, среди которых средства моделирования, анимации, все виды задач по мониторингу, серверы и агенты. Чтобы дать представление о том, сколь широкое воздействие этого нового понятия активного объекта, мы представим два небольших примера, совершенно различных как по своей природе, так и по степени детализации.

Первый пример — это движущиеся частицы. Допустим, что наше приложение предназначено для анимации множества частиц, которые движутся в данном поле вектора. Тогда, используя активные объекты, мы сможем действовать примерно так:

(1) Определяем класс абстрактной модели Particle (частица), реализующий общий алгоритм для решения дифференциального уравнения вида: x' = f(x,t), т.е. процесс, который при определенном состоянии итеративно рассчитывает последующие состояния: (t+dt, x(t+dt)), (t+2dt, x(t+2dt)) и т.д. Обратите внимание, что соответствующий алгоритм (скажем, алгоритм Рунге-Кутта) будет работать с абстрактной функцией f, реализованной как процедурная переменная (или, по терминологии C++, как виртуальная функция).

(2) Определяем конкретный подкласс класса Particle, который специфицировал бы

  (a) некую конкретную f и

  (b) дополнительные параметры для f.

(3) Определяем класс представлений, показывающих

   (a) вектор поля f и

   (b) текущую позицию моделируемой частицы (или частиц).

(4) Создаем желаемое число моделируемых объектов (в различных начальных состояниях), создаем видимый объект и связываем представление с моделями.

Отметим, что моделируемые частицы, ведущие себя как параллельные объекты, затем независимым образом инициируют анимацию.

Второй пример применения активных объектов являют собой серверы. Система Oberon и в особенности ее однопоточная парадигма задач (one-thread tasking) разработаны специально для рабочих станций, рассчитанных на одного пользователя. И хотя наши эксперименты с серверами в среде Oberon прошли на удивление успешно (они были связаны с решением фоновых задач без применения процедуры вытеснения), мы, разумеется, хотели бы иметь множественные потоки для обслуживания многочисленных клиентов.

Мы утверждаем, что активные объекты дают еще более удачное решение, поскольку с их помощью мы можем отвечать на каждый запрос клиента с учетом его конкретной специфики. К тому же такие ответы, как правило, являлись бы моделями дистанционных представлений на станции клиента (скажем, отображение содержания почтового ящика).

Подведем итоги этого раздела: будущую архитектуру системы Oberon мы видим как организованную систему весьма различных в функциональном отношении параллельных объектов. Эти объекты или пассивны (т.е. управляются дистанционно), или активны (т.е. управляются встроенным процессом). Связь "модель/вид" определяет важное отношение на данном множестве объектов.

Интеграция в Oberon системы служб,
или как повысить функциональность

Среди всех критических вопросов, которые нам задавали в связи с проектом Oberon, один встречался особенно часто. Это язвительный вопрос о коммерческих перспективах Oberon, т.е. о его шансах против таких общепризнанных сред, как Windows, Macintosh, NextStep и др. Причем те, кто считает, что шансов у Oberon абсолютно никаких, обычно выдвигают весьма прямолинейные и столь же сомнительные доводы. Они строятся на том, что Oberon не предусматривает совместимости с популярными пакетами прикладных программ. Разумеется, кто же захочет отказаться от уникальной возможности легко оснащать систему новыми функциями. И все же тактику "брать и запускать" коммерческие пакеты трудно квалифицировать иначе, как преждевременную попытку использовать гибкость, внутренне присущую программному обеспечению.

Но вернемся к перспективам Oberon в коммерческом мире. Здесь нужно учитывать два обстоятельства:

(1) Oberon — это мощная среда разработки программного обеспечения

(2) В настоящее время быстро развивается т.н. "открытое" обслуживание, т.е. обслуживание, предоставляемое как на местном, так и на глобальном уровне и предполагающее точную спецификацию протокола доступа.

Сопоставив эти два наблюдения, мы легко придем к выводу о том, что одно из наиболее перспективных направлений дальнейшего развития Oberon — превращение его в усовершенствованную общую платформу для предоставления услуг удаленным пользователям; слово "усовершенствованную" мы понимаем в данном случае как гибкую, высоко интегрированную и индивидуализированную.

К настоящему времени мы провели эксперименты по организации следующих услуг удаленным пользователям (и реализовали доступ к ним через Ethernet и TCP/IP):

- электронный телефонный справочник;
- электронный справочник пассажира железной дороги;
- электронный словарь;
- фотосервис Digital Kodak;
- информационная система по географии Швейцарии;
- служба символьных вычислений Maple;
- служба TrueType-шрифтов;
- FTP;
- электронная почта;
- управляющая служба Telnet;
- сеть World Wide Web;
- Teletext и Telenews.

Подчеркнем, что реализация услуг удаленным пользователям (или доступа к ним) в усовершенствованном варианте, как мы его понимаем, включает в себя:

(a) проектирование пользовательского интерфейса (обычно это одно-два панельных приспособления) и

(b) интеграция в среду пользователя.

Проблема интеграции хорошо иллюстрируется на примере двух служб — World Wide Web и Telenews. World Wide Web — это всемирная информационная служба, которая предоставляет пользователю глобальный гипертекст, т.е. глобальный текстовый документ, содержащий особые элементы любого типа (графика, рисунки, видео, звук) и ассоциативные узлы (links). Нажатие кнопки мыши на такой узел обычно переводит документ в новый контекст, логически связанный с этим узлом (и, возможно, физически расположенный совсем в другом уголке мира). В принципе, узлы носят более общий характер, так как они могут сочетаться с произвольными действиями. Предполагается, что клиенты сами участвуют в бесконечном процессе создания документа, добавляя к нему все новые и новые части.

Telenews — еще один вид динамического документа, который мы разрабатываем самостоятельно. В структурном и функциональном отношении он напоминает сеть World Wide Web. Однако, в отличие от последней, Telenews генерируется (и корректируется) автоматически из базы данных Teletext. Teletext — это ориентированная на страницы информационная служба; данные передаются TV-станцией одновременно с телевизионной "картинкой". И здесь узлы могут сочетаться с произвольными действиями (скажем, вызовом электронной энциклопедии, которая после этого становится динамической частью документа Telenews).

World Wide Web и Telenews представляют собой два различных примера одного и того же нового типа динамических интерактивных документов. Такие документы иногда называют виртуальными, поскольку в любой момент времени не существует четко определенного статического глобального состояния.

Памятуя о том, что наша задача состоит в реализации усовершенствованного пользовательского интерфейса для двух упомянутых служб, мы сразу же принимаем общую модель интерфейса, ориентированного на документ (document oriented interface — DOI). В обоих случаях устанавливается соответствие между узлами в документе и гиперузлами, т.е. с экземплярами некоторого абстрактного объекта, именуемого Hyperlink, который является расширением типа Object в языке Oberon. Короче говоря, мы интегрировали и унифицировали интерфейсы для обеих служб, хотя их семантика и реализация совершенно различны.

Заключение

Прошло уже 9 лет с тех пор, как были разработаны основы концепций системы Oberon. Эти концепции, в первую очередь благодаря своей универсальности, сохранили свое значение и по сей день. Мало того, они фактически стимулировали и продвигали важный процесс развития системы в строгом соответствии с первоначальным замыслом. Причем надо сказать, что Oberon System 3 с архитектурой Gadgets — это лишь первая ласточка.

Вне всякого сомнения, впереди нас ждут еще более впечатляющие достижения на пути к унифицированной объектно-ориентированной и параллельной системной архитектуре и к

интегрированной и открытой платформе для организации обслуживания удаленных пользователей на самом высоком технологическом уровне. Конца этой дороги пока не видно.

Благодарности

Процесс эволюции системы Oberon не был бы столь плодотворным, если бы рядом со мной не работали мои коллеги, которых отличает широкая компетентность, большой опыт и неиссякаемый энтузиазм. Я многим обязан им, и, в первую очередь — Ханнесу Мараису (Hannes Marais) и Карлу Реге (Karl Rege) — за помощь в разработке Oberon System 3 with Gadgets, а также Андреасу Дистели (Andreas Disteli), Мартину Гитсельсу (Martin Gitsels), Эриху Освальду (Erich Oswald) и Ральфу Зоммереру (Ralph Sommerer) — за реализацию базового программного обеспечения для сетей и интеграцию систем обслуживания удаленных пользователей.

Литература

[1] Wirth N., Gutknecht J. (1992) Project Oberon: The Design of an Operating System and Compiler" // Addison-Wesley, ACM Press.

[2] Gutknecht J. (1994) Oberon System 3: Vision of a Future Software Technology // Software—Concepts and Tools, No.15, p.45-54.

[3] Marais H. (1994) Oberon System 3 // Dr. Dobbs Journal, No.220, October 1994, p.42-50.

[4] Krasner G.E., Pope S.T. (1988) A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80" // Journal of Object-Oriented Programming, Vol.1, No.3, August 1988, p.26-49.

Об авторе. Юрг Гуткнехт (Juerg Gutknecht) — известный педагог, профессор Швейцарского федерального технологического института, принимал активное участие в проекте Oberon. По сути является соавтором Н. Вирта в создании языка Oberon, сыграл большую роль в доведении основополагающей концепции этого языка, концепции расширения типа (type extension), до математической основы в виде проекции пространств. Большой популяризатор языка и системы Oberon. Сферой его профессиональных интересов являются операционные системы и системы пользовательского интерфейса.