# Глава 2. Проектирование редактора документов

В данной главе рассматривается применение паттернов на примере проектирования визуального редактора документов Lexi¹, построенного по принципу «что видишь, то и получаешь» (WYSIWYG). Мы увидим, как с помощью паттернов можно решать проблемы проектирования, характерные для Lexi и аналогичных приложений. Здесь описывается опыт работы с восемью паттернами.

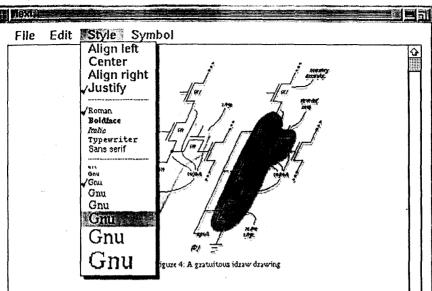
На рис. 2.1 изображен пользовательский интерфейс редактора Lexi. WYSIWYG-представление документа занимает большую прямоугольную область в центре. В документе могут произвольно сочетаться текст и графика, отформатированные разными способами. Вокруг документа — привычные выпадающие меню и полосы прокрутки, а также значки с номерами для перехода на нужную страницу документа.

## 2.1. Задачи проектирования

Рассмотрим семь задач, характерных для дизайна Lexi:

- □ структура документа. Выбор внутреннего представления документа отражается практически на всех аспектах дизайна. Для редактирования, форматирования, отображения и анализа текста необходимо уметь обходить это представление. Способ организации информации играет решающую роль при дальнейшем проектировании;
- □ форматирование. Как в Lexi организованы текст и графика в виде строк и колонок? Какие объекты отвечают за реализацию стратегий форматирования? Взаимодействие данных стратегий с внутренним представлением документа;
- □ создание привлекательного интерфейса пользователя. В состав пользовательского интерфейса Lexi входят полосы прокрутки, рамки и оттененные выпадающие меню. Вполне вероятно, что количество и состав элементов интерфейса будут изменяться по мере его развития. Поэтому важно иметь возможность легко добавлять и удалять элементы оформления, не затрагивая приложение;
- □ поддержка стандартов внешнего облика программы. Lexi должен без серьезной модификации адаптироваться к стандартам внешнего облика программ, например, таким как Motif или Presentation Manager (PM);

<sup>&</sup>lt;sup>1</sup> Дизайн Lexi основан на программе Doc – текстового редактора, разработанного Кальдером [CL92].



the internal representation of the TextView. The draw operation (which is not shown) simply calls draw on the TBBox.

The code that builds a TextView is similar to the original draw code, except that instead of calling functions to draw the characters, we build objects that will draw themselves whenever necessary. Using objects solves the redraw problem because only those objects that lie within the damaged region will get draw calls. The programmer does not have to write the code that decides what objects to redraw—that code is in the toolkit (in this example, in the implementation of the Box draw operation). Indeed, the glyph—based implementation of TextView is even simpler than the original code because the programmer need only declare what objects should interact.

#### ?.? Multiple fonts

Because we built TextView with glyphs, we can easily extend it to add functionality that might otherwise be difficult to implement. For example, Figure 4 shows a screen dump of a version of TextView that displays EUC-encoded Japanese text. Adding this feature to a text view such as the Athena TextWidget would require a complete rewrite. Here we only add two lines of code. Figure 5, shows the change.

Character glyphs take an optional second constructor parameter that specifies the font to use when drawing. For ASCII-encoded text we create Characters that use the 8-bit ASCII-encoded "a14" font; for JIS-encoded

text (kanjii and kana characters) we create Characte that use the 16-bit JIS-encoded "k14" font.

### ?.? Mixing text and graphics

We can put any glyph inside a composite glyph; th

acwikes, and to . Figure 7 shows the modificode that builds the view.

A Stencil is a glyph that displays a bitmap, an HRu draws a horizontal line, and VGlue represents vertic blank space. The constructor parameters for Rule a

Figure 5: Modified TextView that displays Japanese tex

产 12345678913

- □ *поддержка оконных систем*. В оконных системах стандарты внешнего облика обычно различаются. По возможности дизайн Lexi должен быть независимым от оконной системы;
- □ *операции пользователя*. Пользователи управляют работой Lexi с помощью элементов интерфейса, в том числе кнопок и выпадающих меню. Функции, которые вызываются из интерфейса, разбросаны по всей программе. Разработать единообразный механизм для доступа к таким «рассеянным» функциям и для отмены уже выполненных операций довольно трудно;
- □ проверка правописания и расстановка переносов. Поддержка в Lexi таких аналитических операций, как проверка правописания и определение мест переноса. Как минимизировать число классов, которые придется модифицировать при добавлении новой аналитической операции?

Ниже обсуждаются указанные проблемы проектирования. Для каждой из них определены некоторые цели и ограничения на способы их достижения. Прежде чем предлагать решение, мы подробно остановимся на целях и ограничениях. На примере проблемы и ее решения демонстрируется применение одного или нескольких паттернов проектирования. Обсуждение каждой проблемы завершается краткой характеристикой паттерна.

## 2.2. Структура документа

Документ — это всего лишь организованное некоторым способом множество базовых графических элементов: символов, линий, многоугольников и других геометрических фигур. Все они несут в себе полную информацию о содержании документа. И все же автор часто представляет себе эти элементы не в графическом виде, а в терминах физической структуры документа — строк, колонок, рисунков, таблиц и других подструктур. Эти подструктуры, в свою очередь, составлены из более мелких и т.д.

Пользовательский интерфейс Lexi должен позволять пользователям непосредственно манипулировать такими подструктурами. Например, пользователю следует дать возможность обращаться с диаграммой как с неделимой единицей, а не как с набором отдельных графических примитивов, предоставить средства ссылаться на таблицу как на единое целое, а не как на неструктурированное хранилище текста и графики. Это делает интерфейс простым и интуитивно понятным. Чтобы придать реализации Lexi аналогичные свойства, мы выберем такое внутреннее представление, которое в точности соответствует физической структуре документа.

В частности, внутреннее представление должно поддерживать:

□ отслеживание физической структуры документа, то есть разбиение текста и графики на строки, колонки, таблицы и т.д.;

<sup>&</sup>lt;sup>1</sup> Авторы часто рассматривают документы и в терминах их *логической* структуры: предложений, абзацев, разделов, подразделов и глав. Чтобы не слишком усложнять пример, мы не будем явно хранить во внутреннем представлении информацию о логической структуре. Но то проектное решение, которое мы опишем, вполне пригодно для представления и такой информации.