Доступ к распределенной информации

Для многих видов анализа необходимо рассматривать текст посимвольно. Но анализируемый текст рассеян по иерархии структур, состоящих из объектов-глифов. Чтобы исследовать текст, представленный в таком виде, нужен механизм доступа, знающий о структурах данных, в которых хранится текст. Для некоторых глифов потомки могут храниться в связанных списках, для других – в массивах, а для третьих и вовсе используются какие-то экзотические структуры. Наш механизм доступа должен справляться со всем этим.

К сожалению, для разных видов анализа методы доступа к информации могут различаться. Обычно текст сканируется от начала к концу. Но иногда требуется сделать прямо противоположное. Например, для реверсивного поиска нужно проходить по тексту в обратном, а не в прямом направлении. А при вычислении алгебраических выражений необходим внутренний порядок обхода.

Итак, наш механизм доступа должен уметь приспосабливаться к разным структурам данных и поддерживать разные способы обхода.

Инкапсуляция доступа и порядка обхода

Пока что в нашем интерфейсе глифов для доступа к потомкам со стороны клиентов используется целочисленный индекс. Хотя для тех классов глифов, которые содержат потомков в массиве, это, может быть, и разумно, но совершенно неэффективно для глифов, пользующихся связанным списком. Роль абстракции глифов в том, чтобы скрыть структуру данных, в которой содержатся потомки. Тогда мы сможем изменить данную структуру, не затрагивая другие классы.

Поэтому только глифу разрешено знать, какую структуру он использует. Отсюда следует, что интерфейс глифов не должен отдавать предпочтение какой-то одной структуре данных. Например, не следует оптимизировать его в пользу массивов, а не связанных списков, как это делалось до сих пор.

Мы можем решить проблему и одновременно поддержать несколько разных способов обхода. Разумно включить возможности множественного доступа и обхода прямо в классы глифов и предоставить способ выбирать между ними, возможно, передавая константу, являющуюся элементом некоторого перечисления. Выполняя обход, классы передают этот параметр друг другу, чтобы гарантировать, что все они обходят структуру в одном и том же порядке. Так же должна передаваться любая информация, собранная во время обхода.

Для поддержки данного подхода мы могли бы добавить в интерфейс класса `Glyph` следующие абстрактные операции:

```
void First(Traversal kind)
void Next()
bool IsDone()
Glyph* GetCurrent()
void Insert(Glyph*)
```

Операции `First`, `Next` и `IsDone` управляют обходом. `First` производит инициализацию. В качестве параметра передается вид обхода в виде перечисляемой константы типа `Traversal`, которая может принимать такие значения, как