OmniMark - Design Principles

The unique advantages of OmniMark stem from a set of key design principles which, when brought together into one tool, deliver a powerful combination of performance and productivity.

- The Streaming Paradigm
- Rules-based Programming
- Hierarchical Markup Parsing Model
- Powerful Pattern Matching
- Referents

The Streaming Paradigm

The streaming paradigm is an approach to programming that concentrates on describing the process to be applied to a piece of data, and on processing data directly as it streams from one location to another. In the streaming model, the use of data structures to model input data is eliminated, and the use of data structures to model output is greatly reduced. This keeps to a minimum the use of system resources when processing large volumes of data. As a side-effect, because the processing requirements are consistent, system performance on larger data sets can be predicted with a great deal of accuracy. A program will run with equal success on a 2 kilobyte file or a 2 gigabyte file.

OmniMark has an abstracted streaming model which allows a stream to be attached to different sources of input and output — files, databases and messages — with a minimum of effort. This abstraction also allows code processing the content itself to be dissociated completely from the problems of managing or even knowing about details of the input or output type, with obvious productivity and code simplification benefits.

An application may have multiple input streams open to permit data integration. Multiple output streams may also be used to feed different targets. For instance a complex application may be taking a stream fed from a file, integrating that input with a stream fed by accessing a database and outputting the data to multiple systems (potentially in different formats).

Rules-based Programming

OmniMark incorporates a declarative scripting language. This means that an application is constructed of rules for dealing with events which are triggered by the recognition of patterns of data coming into the program from a stream. In dealing with content the individual pieces of content are well known, the order in which they occur is not. This arbitrariness of content is one of its basic properties and rules are the best mechanism for dealing with it. OmniMark's rules may be triggered by data events generated by the two types of built-in processors, the pattern processor and the markup processor. The markup processor is tightly coupled with markup parsing.

The two processors may be used in conjunction to process a single piece of content to create powerful hybrid applications; where XML is being processed and complex pattern matching is used upon the content in the markup - the text in the elements. The pattern processor may also be used ahead of the markup processor to prepare content for parsing - converting non-XML into XML for instance. The output stream of the pattern processor is fed in as the input stream of the markup processor.

All of these features are implemented in an elegant framework, which results in applications consisting of well-delineated functional code blocks both in terms of readability and actual functionality, thus producing an easily maintained application.

Hierarchical Markup Parsing Model

Many people concerned with XML will be familiar with, or will at least have heard of, SAX and DOM models for processing. SAX is an event-based model and DOM is tree-based. OmniMark employs a third model - hierarchical. Like SAX, OmniMark leverages an event-based model, but where SAX would generate three events for an element (the start, the content and the end) OmniMark generates only one, treating the occurrence of the whole element as a single event to activate a rule. Since elements can be nested, a hierarchy of activated rules will be created, modeling the structure of the content. This simple model is easy to understand and the resulting process flow is clear, concise and thus easy to maintain.

The event-based parsing model fits neatly with the streaming approach to processing content, with the markup processor receiving a stream of data and triggering events as the elements are found, without needing to buffer or decompose the input. Therefore this model supports the design considerations for OmniMark of remaining scalable and performant when processing massive data sets or receiving high volumes of content.

In conjunction with the triggering of the rules, the markup processor maintains the current element context for the set of elements being processed at any instant. This allows the application to query and make decisions based on data about that context including the attribute values associated with the elements being processed and their parents. This mechanism has been shown to handle the vast majority of content processing requirements. However, by using other features of OmniMark this may be augmented should it be necessary - for instance a tree of all elements accessed may be constructed in the application for later manipulation.

Powerful Pattern Matching

The OmniMark pattern processor implements a pattern matching language that is both powerful and easy to use. Based upon an optimized regular expression mechanism, it has many other features, including:

- Maintaining context. The same pattern may have different meanings in different contexts. Therefore context needs to be maintained to allow different rules to fire in different situations.
- The ability to lookahead for patterns without actually processing the values. This allows program flow to be changed, before the pattern is reached, to allow the pattern to be processed in the right context.
- Complex pattern matching procedures (i.e., independently-called functions). This allows sophisticated pattern matching to be encapsulated and reused.
- Nested pattern matching (matching a pattern within a pattern).

The pattern processor will activate the associated rule when a pattern defined has been matched. These features are encapsulated in a language that is very English-like, making it clear, easy to comprehend application functionality, which simplifies both development and maintenance.

Referents

Often the order in which content is received as input is not the order in which it is required for output. OmniMark's patented referent mechanism allows a placeholder to be inserted in the output stream and its value supplied later when it is available. The streaming mechanism handles the buffering of output containing unresolved placeholders. The whole referent mechanism may be scoped and nested so that buffering is kept to a minimum. Code that is processing content needs no knowledge of the mechanism; a referent is just like any other target. The major benefit of this mechanism is that it maintains the efficiency of the streaming model while enabling powerful re-ordering functionality that would otherwise be severely constrained. Referents are a key innovation within the OmniMark language and it is one reason why OmniMark is so successful at blending power and performance.