Wave Model Code Walk

This document provides an informal overview of the code providing the wave model. The wave model describes the basic building blocks of a wave application: what a wave is and how it can change. It also provides application models which implement more abstract data structures on top of waves. The most familiar is the conversation model which implements the conversational structure of waves presented in, for example, the Google Wave client.

This walkthrough describes the major modules comprising the wave model in a bottom-up fashion.

The wave model code lives under the src/org/waveprotocol/wave/model directory

Under construction!

The wave model code is under active development. The code structure may differ from that described here, though we'll try to keep it up to date. The code as initially released also contains a few pieces of legacy from Google Wave's infancy, pieces of code we've learnt should be done differently. We intend to clean these up over time.

Let's explore the code

Operations: model.operation.*

These modules implement operations, which are the fundamental unit of state change. The model.operation package describes purely generic operations simply as things which apply to some target. Sub-packages contain specific operation implementations and their transformation.

- model.operation.core: operations which apply to the data objects in model.wave.data.core.
- model.operation.wave: operations applying to the extended data interfaces in model.wave.data.

Legacy note: The operations in model.operation.core represent our ideal minimum set of operations. The model.operation.wave package contains some legacy operations and concepts. In particular, the concept of a "blip" (a conversational message) and the operations targeting it do not belong at this low level of abstraction.

Starting points:

- model.operation.Operation
- model.operation.core.WaveletOperation

Documents: model.document.*

These modules implement wave documents, which contain almost all the state of a wavelet. A wave document is an XML-like sequence of elements and text nodes plus a set of stand-off key/value annotations.

- model.document.bootstrap: a very simple demonstration document implementation.
- model.document.operation: operations applying to documents and their transformation, along with utilities including composition and normalization. Also an automaton for generating random operations for testing.
- model.document.indexed: an indexed tree structure providing fast random access to nodes in a document. An indexed document is mutated by accepting operations.
- model.document: mutable and observable document implementations based on a indexed documents.

Starting points:

- model.document.Document: the high-level mutable document interface
- model.document.ObservableDocument: extends Document to provide events

Waves: model.wave.*

This modules implement waves and wavelets. A wavelet comprises a set of documents and a set of participant identifiers.

- model.wave.data: Simple, "dumb" ADTs, which are the targets of operations. The core sub-package contains pure, metadata-free implementations sufficient for servers. Also a number of document factories for use with the data objects.
- model.wave: the abstract wave view, wavelet, blip and participant id interfaces which define the semantics of these types.
- model.wave.opbased: operation-based implementation of a wavelet which performs mutations by generating and applying operations to an underlying wavelet data
- model.wave.undo: Utilities for reversing wavelet operations

Note that there is a clear separation between the semantic free data types in model.wave.data, which implement pure "dumb" data containers, and the abstract objects in {{model.wave}}. This separation of data from semantics is repeated in other high-level packages.

Legacy note: The interfaces in these packages conflate wave metadata, such as timestamps and version numbers, with the pure wave data. The model.wave.data.core contains our ideals of the data interfaces. The metadata does not behave in the same way as the data and we intend to separate it to reduce confusion. The inclusion of the blip concept here also is a legacy which doesn't belong.

Starting points:
Identifiers: `model.id` and `model.waveref`

These modules contain implementations of wave identifiers and references plus utility classes for working with them. Wave and wavelet ids uniquely identify their objects.

Waverefs extend wave and wavelet ids to refer to points within a wave. At present this extends just to documents but will soon include locations and versions within those documents.

Legacy note: The code in `model.id` does not yet implement the draft specification for identifiers and includes an inelegant serialization scheme. Waverefs implement a superior serialization.

Starting points:

- `model.id.WaveId` and `model.id.WaveletId`
- `model.waveref.WaveRef`

Abstract data types: `model.adt.*`

These modules implement abstract concurrent data types on wave documents, including lists, sets, maps and monotonic values. These data types are safe for use with wave's optimistic concurrency model, guaranteeing convergence when used correctly. These data types are the foundation for all concurrent application models built on waves.

Starting points:

- `model.adt.BasicValue` and `model.adt.docbased.DocumentBasedBasicValue`
- `model.adt.BasicSet` and `model.adt.docbased.DocumentBasedBasicSet`

Document schemas: `model.schema.*`

This module implements schema constraints for wave documents. Schemas constrain the content of wave documents so they may be safely interpreted by clients, including as abstract data types. Document schemas are compatible with operational transform such that a change which is valid at a client remains valid at the server and all other clients, regardless of the concurrent operations against which the change is transformed.

Legacy note: The current hard-coded schema implementation has a number of shortcomings and there are plans to move to a more powerful and flexible implementation.

The conversation model: `model.conversation`

This package implements the abstract conversation model which describes the threaded conversational waves you commonly interact with in the Google Wave client.

Starting points:

- `model.conversation.Conversation`
- `model.conversation.WaveletBasedConversation`

Testing support: `model.testing`

This package contains support classes for testing the wave model and code interacting with the wave model.