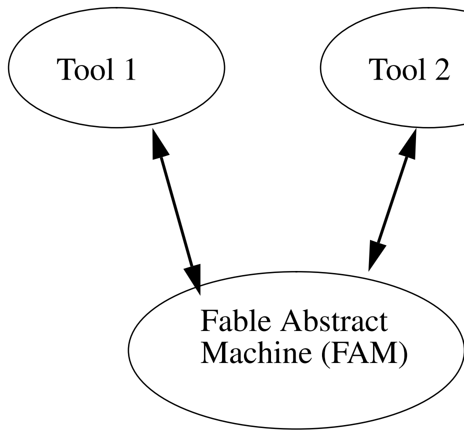
Internal Architecture of the FAM Library

Ralph Melton

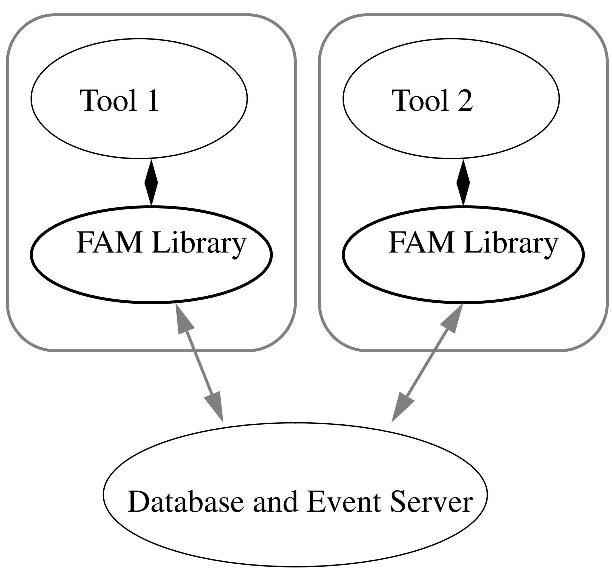
1 Introduction

At a high level, the architecture of the Aesop system looks like this:



Where the connectors are combinations of method invocations from the tool to the FAM, ACME import and export facilities, requests for confirmation from the FAM to the tool, and events broadcast by the FAM.

This logical architecture is realized by the following implementation architecture:

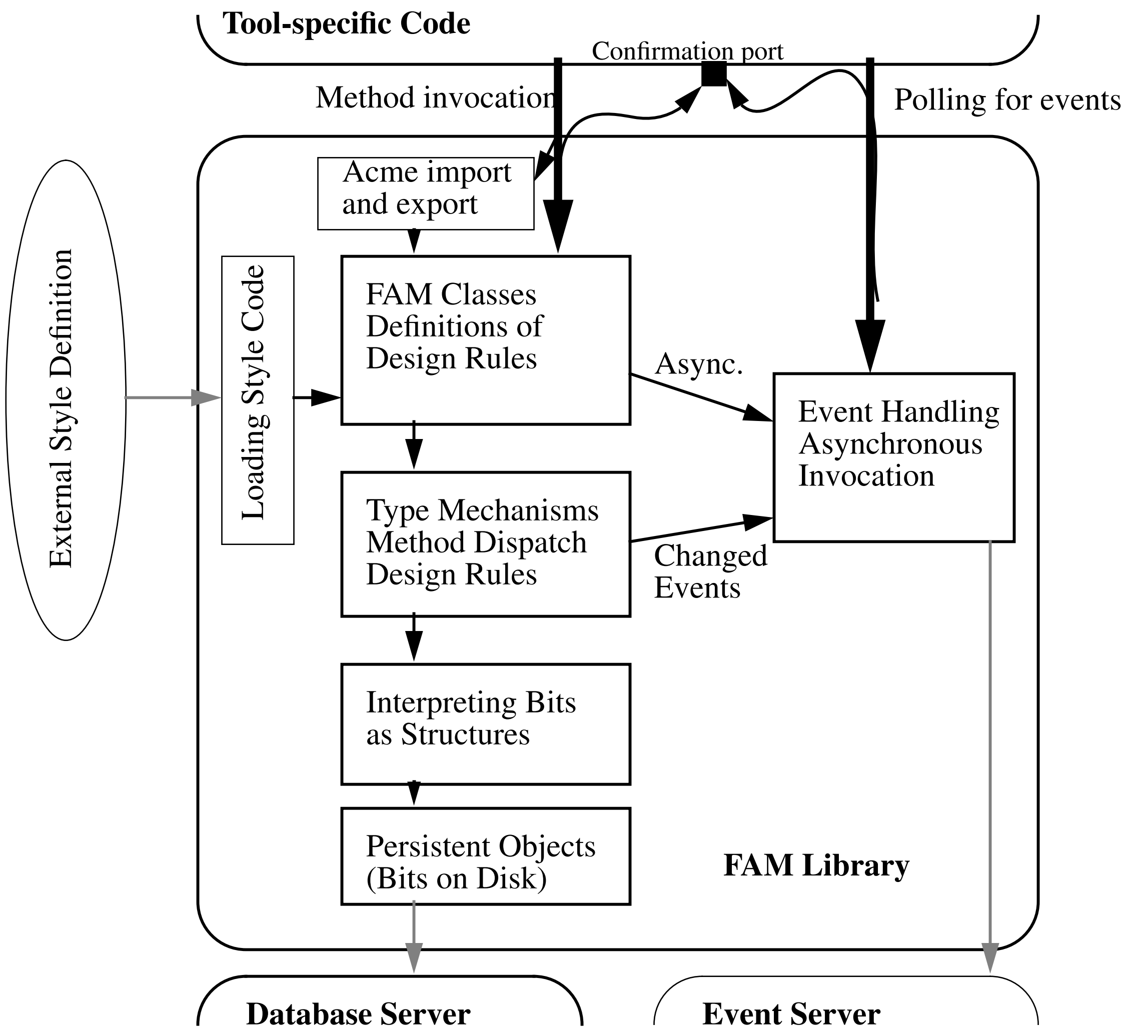


Each tool process has a FAM library of its own, that implements the methods. The database and event server are used to share data between the different tools. We chose to distribute the FAM library, so that each tool has a FAM library of its own, so that we could avoid the concurrency issues raised by having many tools access a single server, and use the database to manage those concurrency issues for us. In this picture, the gray boxes represent single processes, and the gray arrows represent the database connection (implemented in Aesop II by the sm_client library) and the connection to the event server used to distribute events. The solid arrows represent the same Tool-FAM connector as in the previous picture.

This document presents the internal structure of the FAM Library.

2 The FAM Library

The following picture represents the FAM library; a legend follows the picture.



2.1 Major Components

There are four major components identified in this picture.

- **Tool-specific code** is the code that implements the tool's own functionality. It must of necessity be different for each tool.
- The **FAM Library** is the library this document addresses. Each tool is linked with its own copy of the FAM library; all instances of the FAM library use the same code.
- The **Database Server** is the server for the database that allows the many different tools of an environment to see the same design. In Aesop II, this is the Exodus server. The greyed connector to the Database Server is provided by the sm_client library in Exodus.
- The **Event Server** routes and dispatches events and asynchronous messages. The greyed connector is probably a TCP/IP connection or something like that--it must certainly span process boundaries.
- The **External Style Description** is a description of the architectural style that is dynamically loaded by the style. This is probably implemented as a file or set of files.

2.2 Tool-FAM Connectors

There are several parts to the Tool-FAM interface.

- The **Method Invocation** connector is perhaps the primary connector from a tool to the FAM. Through this connector, the tool invokes methods on design elements and receives the results. This connector is essentially a procedure call connector. If the methods need to request confirmation from the user for some operation, they will make a call to the procedure defined in the **Confirmation Port** of the tool.
- The **Acme Import/Export** connector is a restricted set of services accessible through the same procedure-call mechanism as the Method Invocation connector. The ACME Import/Export connector may be limited to just two services:
    - Convert a piece of ACME text into new objects in the database
    - Export an object in the database into ACME text.
- Logically, events must propagate from the FAM into the tool. In terms of implementation, though, the tool-specific code must have an event loop, and when it is willing to be notified of changes, it must **Poll for Events** from the FAM. When it is polling for events, the FAM's handling of events before passing them to the tool may also include calling into the Confirmation Port.

2.3 The Pillar of Method Calls

Five components of the FAM library form a layered architecture. The connectors between these are probably procedure calls; I also tend to think of them as representing logical theories built out of lower-level theories. From bottom to top, these are the layers in that portion of the FAM library:

- **Persistent Objects (Bits on disk)** This is the layer that represents identifying and retrieving binary data from the database. In Aesop II, much of this is handled by the sm_client library.
- **Interpreting Bits as Structures**. This layer is responsible for interpreting the raw data provided by the Persistent Objects layer as structured data, i.e. as objects with slots and their values.
- **Type Mechanisms, Method Dispatch, and Design Rules**. It seems likely that we will want some special mechanism for defining classes, defining design rules, and dispatching methods. That mechanism lives in this layer.
- **FAM Classes and Definitions of Design Rules** At this level, we implement the specific design rules and architectural classes we use in our system. This level is accessible from Tools; the code invoked by methods is defined at this level.
- **Acme Import and Export** These specialized methods are built upon the other FAM method definitions.

2.4 Other Parts of the FAM Library

There are several other parts of the FAM library that interact with the primary pillar that supports method calls.

- **Loading Style Code**. This component is responsible for dynamically loading type definitions and design rules as they are required. It loads definitions from external files, and modifies the FAM Classes and Definitions of Design Rules component by adding new definitions.
- **Event Handling and Asynchronous Invocation**. Communication between tool processes happens in two ways:
    - When a tool modifies the database by invoking a method, a **changed event** is generated, which is broadcast to all tools.
    - Some methods may be implemented by external tools. For these, the method definition will **asynchronously invoke** another tool, which will send its response as an asynchronous message.

  Both of these forms of communication are handled through the Event Handling component, which communicates with the Event Server, which is an external process.