Contents

Proposed Aesop System Architecture

Andrew Kompanek
February 21, 1996

Introduction

This document is currently an explanation of the basic connectors in the new Aesop architecture and does not describe the decomposition of the basic components in any detail. It does not include any discussion of the mechanisms required by the Shelf. The communication protocols are not yet completely described in some cases. This document is still pretty rough.

The rest of the text refers to the following diagram:

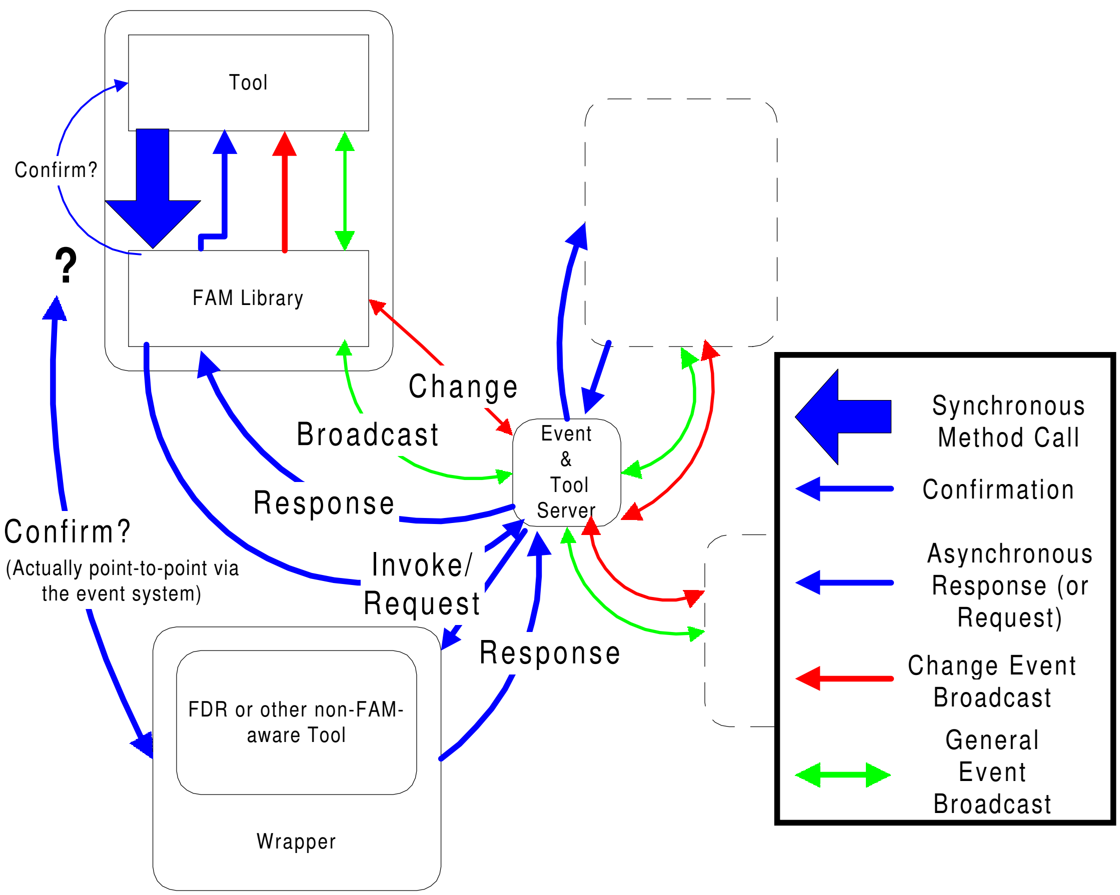

Figure 1: A big fuzzy informal architectural model of Aesop

Summary of Basic Components

The Tool/FAM interface is described as one-to-one with multiple FAMs: each "FAM-aware" tool has a FAM library in it. Conceptually, we can view the system as containing a single FAM to which tools are connected, but I've chosen the multiple FAM view, so that all communication is explicit. In either view, the FAM has a connection to the shared architectural Database (not pictured).

In addition to FAM-aware tools, there are also tools which are less tightly coupled to the system and do not have a direct knowledge of objects in the design database. Requests to these tools are made with asynchronous events and data is shared using the external file system. Each tool has a "wrapper" (which could be implemented as part of the tool process or within the Tool Server) which interprets requests and translates responses so that they can be returned to the system but otherwise is not connected to the system.

Events are handled and dispatched centrally by an "Event and Tool Server". This server also invokes tools and passes requests (in the form of asynchronous events) onto tools. The Event Server and Tool Server are separate processes in the current system although they're pictured as one component (or connector, if you like) above.

Summary of Basic Connectors

The only synchronous communication is procedure call (including a method dispatch mechanism) between a tool and its FAM library. All other communication in the system is essentially asynchronous although confirmations (see below) might be viewed as synchronous requests.

The following types of system-wide communication is supported by the proposed design:

- Synchronous Procedure Call with Confirmation

  This is the basic Tool/FAM communication – a synchronous call which could include a confirmtaion callback. Valid return values include "not yet available" in which case the Tool can expect a (single) asynchronous response at a later time.
- Asynchronous Request and Response

  Methods invoked in a FAM may actually be implemented by an external tool. A request is made via an asynchronous event by the FAM which the tool server receives. The tool server then passes the request on to an existing tool or invokes a new tool to service the request.
- Confirmation Event

  A separate but related Confirmation connector is needed for faceless tools which provides a communication path between in it and the FAM of user interface tool(s). The discussion of this connector is in the section on Asynchronous Request Reponse.
- Change Events

  This is the mechanism used to communicate changes to objects in the design database to tools which use the information in the event to update any internal state which is dependent on the changed objects.
- General Broadcast Events

  These include any general system events other than change events.

In each case of asynchronous communication, the tool announces an event to anonymous recipient(s). This mechanisms do not address what would be required by tool like the Shelf which requires Tool-to-Tool communication.

Connectors

This is a description of the connectors in the Aesop system. The sections correspond roughly to different categories of communication – more than one connecor is discussed is some of the sections.

Synchronous Procedure Call with Confirmation

In FAM-aware tools, he FAM is accessed through synchronous procedure calls (left-most, fat connector). A particular call may return "result not yet available". In this case, the actual return value is returned asynchronously (next connector) in the form of an event. In some cases an operation may require confirmation from the user, in which case the FAM makes a synchronous confirmation callback to the tool[1] which may then prompt the user or return an answer automatically. Note that in a user interface tool this confirmation port could also be used (indirectly) by another tool which does not have its own user interface (see below) which needs confirmation at some point in its operation.

Protocols

The "synchronous return value" (i.e., returned with "not yet available) for asynchronous requests must also include a *Request Id* which identifies the particular call. The tool which initiates the operation can match this Id to the Id in later asynchronous return values so that it can maintain state about pending operations and notify the user when operations finish.

The confirmation callback takes an argument which describes the message to be displayed to the user and returns either "yes" or "no". *Is this all we need?*

Asynchronous Request and Response

Inside the FAM, a method will either be executed internally or by an external tool. In the case of an external tool the FAM issues a tool *request* event which is serviced by a central server. The server sends the request to an existing tool in the system or starts up a new tool. At this point the FAM can either return to the Tool with a "not yet available" result or wait for a *response* event (in practice, I'm not sure whether we'd ever do the latter). In either case the FAM will eventually receive a response, do any work it needs to do based on the reponse, and pass the event[2] back to the invoking tool. These operations may generate *change* events as objects in the database are altered (see below).

An external tool may also require confirmation from the main user-interface if it does not have a UI. There are two cases corresponding to the two different types of "faceless" tools:

1. A *FAM-aware tool* may call a FAM method which requires confirmation from the user. The FAM must send a Confirmation Event via the Event Server, which will be received by the FAM of some (?) user interface tool.
2. A tool *without a FAM* may also require confirmation – for example, if only because it's about to do "rm *". I'm not sure this is necessary in practice.

In the diagram, I have a response indicated as a point-to-point asynchronous event. It is not broadcast to all tools in the system.

---

[1] Note that if the FAM and Tool were implemented as separate single-threaded processes, that the Tool would not be available to service the confirmation!

[2] Or a corresponding event, anyway. The FAM/Tool response connector may thought of as different at least in some sense from the FAM/Event System connector.

Protocols

A response event could include the following:

- The *Tool Class* – for example, "Text Editor"
- The *Operation* requested
- An argument list consisting of type/value pairs where types could include: URL, External File, ACME, ACME File and Object Identifier in addition to primitive types (e.g., numbers, strings). It could also include the derived representation types like Wright Specifications.

There was some suggestion that it *should* be a broadcast.

Change Events

When an object is altered by the FAM (via a method a call from some tool), a *change* is generated so that any views maintained by tools (particularly the UI) can be updated. A *view* is any representation maintained by a tool whose value is dependent on objects in the design database. The change event is broadcast to all tools in the system (including the tool which make the method call).

NOTE ONLY FAM-AWARE TOOLS RECEIVE CHANGE EVENTS SINCE THESE ARE DIRECTLY RELATED TO OBJECTS IN THE DATABASE. THERE IS NO EXPLICIT MECHANISM PROVIDED BY AESOP FOR TOOLS WHICH OPERATE ON EXTERNAL REPRESENTATIONS WHICH COULD CHANGE, OR EXTERNAL REPRESENTATIONS WHICH DEPEND ON OBJECTS IN THE DESIGN DATABASE.

Protocols

The event would include the *identifier of the object* on which the method was invoked, the *method name* and the *arguments* in the call. In the case of a tool with multiple views, it may also be useful to have a way of knowing which view generated the change so that the tool needn't update a view which has already been altered with a new value (I'm not sure I have a specific example, but this may be). The information about the call can be used as a "hint" so that views can be updated incrementally. We can't guarantee in order change event delivery but for most purposes we can get away this optimistic assumption[3]. The user interface can provide an explicit "update" if views become confused.

General Broadcast Events

There are almost certainly events other than change events which should be broadcast to all (or, all interested) tools in the system. These would at the very least include a "system going down" event. Any tool can announce an event of general interest and receive those events. In terms of implementation, of course, we don't need any new mechanism to deal with these, but we may want to think of Change events and General Broadcast events differently.

Fuzzy: What's the structure of these events?

Tool-to-Tool Communication

Is there anything other than the shelf that needs mechanisms for carrying on an on-going asynchronous conversation? Architecturally speaking, I'm not sure there's anything to prevent us from adding a notion that lets one tool identify another in the system.

---

[3] We can encourage the end-user to perform appropriate religious rituals to decrease the probability of something really egregious happening.

Shared State

This is not in the architectural diagram but of course both the database and external file systems are shared state connectors (or components depending upon whether you happen to be a big-ender or a little-ender). I'm not sure there's much to say here – it's just worth noting them explicitly since this state is certainly implied by the above description, in that references to the database and filesystem are flowing through all these connectors.

Sample Tools

This is an explanation of how specific tools make use of the mechanisms described above. This includes a more detailed discussion of the mechanisms (some of which we may want to move into the main text).

A Wright Checker

(this is Zhenyu's text)

I haven't had a chance to sychronize this description with the rest of the text. It should be smoothly integrated with the rest of this document eventually. This is more or less what Zheny

First there should be two methods in the FAM that are responsible for Invoke/Response. Call them Check() and GetAnswer(). GetAnswer is a callback which is called every time the response message is sent back.

When the UI wants to check one role and one port, it will call the Check() method in the FAM with some data (PortID, RoleID and TransactionID). First Check() will search the result cache in DB to see if there is already clean result. If so, it will return imediately (with a result or with "waiting for answer" then send a result message back to UI?). If not, Check() will do some data retrieval from the Database depending on what data format the UI want to send to checker, then Check() will compose a message in following format:

Tool_Class(WRIGHT_CHKER),Tool_Op(PR_CHECK),Val_Port_Type, Val_Port, Val_Role_Type, Val_Role, Extending data

Type may be:

0: ACME description
1: File Reference
2: WRIGHT spec
...(others?)

Then Check() will return to caller with "waiting for an answer". Then a Transaction List should be updated and the port and role waiting for checking answer should be LOCKED[4]. (If the user insists on breaking the lock, perhaps we should send a cancel message to checker, I am not sure)

The tool server gets this message. Then it will start the WRIGHT checker (if no one is active) or choose one checker(perhaps by load balancing if multiple copies exist). It then routes the message to the checker.

The Checker (wrapper) receives the message. If the checker is busy, the message is put in an input queue. If not, it examines Tool_Op and Val_Port_Type, Val_Role_Type and after some data format transformation calls FDR. After get the result, it will compose a message in following format:

---

[4] *If the FAM is going to return out of its transaction, this would mean having another locking mechanism. Do we need this?*

Check_Result, Extending data.

The event server then sends this message back to the FAM of caller UI. It is possible this Extending data will consists of the UI Instance ID so the event server can direct the message to correct UI, if you insist on not broadcasting the result,

The GetAnswer() will be called, it will release the lock on port and role, update the DB cache, then compose a message back to UI).

What the UI want to do upon receiving the result depends on itself.

A Background Design Checker

This will be a description similar to the one above.

It would be a good idea to enumerate several more tools, too.