WORK IN PROGRESS

a · i → K

Aikernel

Technical Guide

Version 1.3.1

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Table of Contents

For the Impatient.... 3
About this manual.... 4
Past, present, and future.... 4
Specification versus Implementation.... 4
Section 1.... 5
Chapter 1: Background and Introduction.... 6
Chapter 2: Architectural Overview.... 7
Chapter 3: User Module.... 9
The Interface.... 9
The Current UserModule implementation.... 11
Chapter 4: Session Module.... 13
The interface.... 13
The current SessionModule implementation:.... 13
Chapter 5: Application Module.... 14
The interface.... 14
The current Application implementation:.... 14
Chapter 6, I/O Module.... 17
The interface.... 17
The current I/O implementation.... 17
Chapter 7, Activator Module.... 19
The interfaces.... 19
The current Activator implementation.... 20
Chapter 8, Context Module.... 21
The interfaces.... 21
The current Context implementation.... 21
Chapter 9, Rolling your own Modules.... 23
Section 2.... 24
Chapter 10, Logic Overview.... 25
Chapter 11, Natural Language Processing.... 26
Chapter 12, More intelligence.... 27
Section 3.... 28
Chapter 13, How to develop a cell.... 29

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Author: Michael Rice
Contact: mrice@users.sourceforge.net
Created: 1-October-2003

For the Impatient

Don't have enough time to read through some long winded narcissistic technical document? Paste this link in your web browser to get started fast!

http://aikernel.sourceforge.net/documentation/quickstart.shtml

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

About this manual

This should be a pretty easy manual for any software developer or software enthusiast to use. It starts out with a quick introduction to the background of the Aikernel. It them proceed with a high level architecture overview. From there, the manual is split into to thee major sections.

Section 1: an introduction and an explanation of the major pieces of the infrastructural modules of the the Aikernel. Each module description is divided into two parts, a description of the interfaces -- the expected behavior of the module -- and the reference implementation developed right here at the Aikernel project.
Section 2: is a description of the logic and intelligent pieces of the Aikernel.
Section 3: describes the process of writing applications, or cells, so that you can write your own intelligent applications.

Past, present, and future

As the document discusses each piece, keep in mind that there is the "as is", the "will be", and the "could be" component to it. This is a living project that is written in the open so we never want to talk only about what it is, but what it will be too!

Specification versus Implementation

This document covers that core modules specifications and the reference implementation offered by the Aikernel team. As you read more about this product, this will make more sense to you. In a nutshell, the Aikernel is a modular product and you can override many pieces of it. This document describes this and they way that the server has been implemented already.

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Section 1

Overview and Core Modules

$a \cdot i \rightarrow K$

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Chapter 1: Background and Introduction

If you write software for living, or even do it just for fun, odds are that you are finding that your users are getting more and more sophisticated. They want more and more features from your products -- this is a challenge by itself. But humans being humans, we like to add a wrinkle. We want all these new features to be easy to use.

If you started out writing software for mainframes, or for the client/server paradigm, you remember designing windows, lots and lots of windows. What they now call a "rich" or "fat" client was the state of the art. The rich client was fun to develop because you typically modeled a database and then dragged and dropped widgets on the screen to design your application. If you were, say, a Visual Basic or Powerbuilder developer you would simply write code as handlers for the events that these graphical widgets emitted. This made it very easy to add more and more features, but as the number of check boxes, radio buttons, and text fields grew it indeed was a challenge to make it simple. (It also resulted in a lot of spaghetti code, but we are trying to look forward, not backward.)

Then came the web. The web is so compelling because it makes it very easy to reach a worldwide set of users, but it is painful to give them the same high level of features using only the web browser. The benefits typically far outweigh the disadvantages, however, and a new generation of tools from Microsoft and from open source developers is certainly making this task easier. The rise of the web also gave rise the desire to create new service oriented architectures like web services and refocused developers attention on building highly scalable server-based solutions.

As we are nearing our collective 10 year experiment with the web it is time to start leveraging our investment and to go to find new frontiers to take our users and our clients. The web has made incredible inroads to a mass market and given them the promise that computers can indeed make our lives better, not just faster and more efficient, but actually better. But our work is by no means complete. I think any savvy computer user knows that we have a lot more we can deliver.

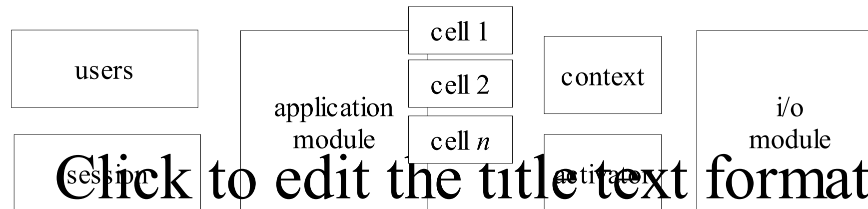
Now that we've got a worldwide infrastructure (ok, we still have some pretty big and depressing holes in that coverage) this project aims to take computing to the next level. Call it pervasive computing. Call in a natural user interface. Call it artificial intelligence.

The Aikernel is the "kernel" that will pop. Here's why: almost all server based products today are transactional in nature. But human life it not about transactions or even mouse clicks. It is about events. Events that happen all around us. The phone rings. The car runs out of gas. I say, "will I be able to change my meeting with Jim to 6 pm for happy hour?" And these events could happen within a few minutes of each other. If we want to build a new generation of products for our users, we need to tap into these events. Can you do it with a window on, say, a tablet PC? Sure. Will the user be required to spend an unreasonable amount of time recording these events on the tablet? You bet.

The Aikernel is a system that responds to events. In its current form it is best at listening to language events such as "I want the buy a new car". But it is flexible, it could be the platform to combine the events from life all around you into true computing power.

Chapter 2: Architectural Overview

The Aikernel is written to scale from the desktop to an internet-class server. As such, there are quite a few architectural pieces that having nothing to do with intelligence, and are there simply to create the framework for all this to happen. This overview should introduce you to both rather quickly: the infrastructure pieces and the logic pieces.



The Aikernel is now modular. Modular means that you can add and remove almost every piece of the server and still have it run intact. As you might have guessed, some modules are more important than others. The following is a quick summary of each of the core modules. They each have their own chapter devoted to them.

User. The user module is here to manage user accounts. This could just be one user in a desktop environment or thousands in an internet environment.

Session. This module keeps tabs on the sessions for users. Each user could have multiple open sessions, this is why there is a split between it and the user module.

Application. The application module loads and makes available applications to the users. Applications are also known as "cells".

I/O. This is the input/output module. It connects the Aikernel to the outside world. In the current reference implementation of the Aikernel this simple opens a telnet session with users and listens and emits textual data.

Activator. An activator is the smallest unit of stimulus available in the server environment. This module keeps track of

> *Room for Improvement*
>
> Watch for boxes like these for some hints about how we can work together to improve the Aikernel. This is for the open source participants that want to contribute to the project. It highlights some of the specific things that we should or could work on to improve future versions. It is by no means comprehensive but it should plant the seeds of change in your mind.
>
> - Change to a JNDI locator on these modules. This will allow the server to be distributed across multiple platforms or machines.
> - Change to JINI service. If we are thinking of JNDI, why not JINI? This will allow the services to locate themselves on an ad hoc basis.
> - Use MBeans. For better management of the server.
> - JBoss or other application server integration. Should it be done just at a beans level or all the way up at the server level, if available.
> - Service oriented architecture wrapper. Should it be able to send and receive information through web services?

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

all the different types of activators that could be fired in the server.

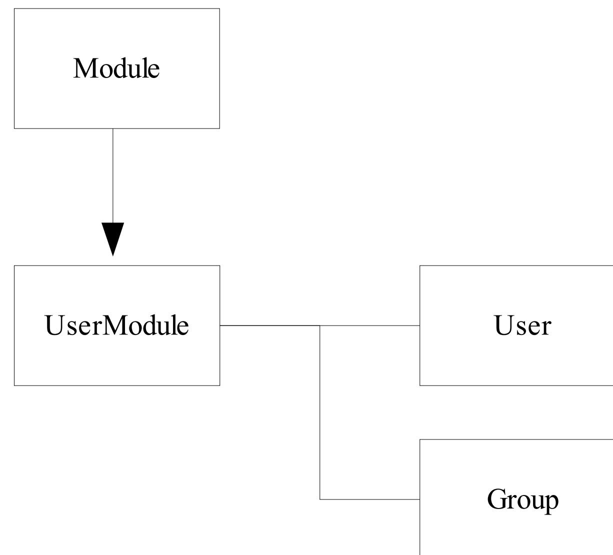
Context. A context is two things: a grouping of activators (remember, stimulus) and it represents the smallest functional level of your application. In the windowing world, the mouse clicking on the file menu in your application might be an activator. The fact that the user wants to open a file is a feature you have provided in your application. In the Aikernel world, we call this a context.

Logic. This actually a number of different modules, but it is lumped together here into one. It will be covered in more detail in Section 2. The logic modules collectively respond to input from the external world and work on the input to identify activators. In most cases the result of this work will be to identify a context that can be sent to an application for processing. All of these modules implement the LogicModule interface, which, in turn, implements the standard Module interface.

Chapter 3: User Module

This is a good module to start with because it is a simple one. Its purpose is to provide a mechanism for many different users to be able to access your Aikernel server. We'll start the discussion by describing the application programming interface (API) defined, then we'll talk about the actual behavior of this implementation.

The Interface



The UserModule is a core module interface that extends the interface Module. Using the word extend here is in the most precise definition of the Java programming language. Basically the UserModule is an interface that has no implementation that, by definition, picks up all the behaviors common to all Modules (that's why we say it extends the Module interface). We are focusing here just on the methods provided by the UserModule. You can learn more about Module's in the "Writing your own Module" chapter.

The functions are pretty much what you'd expect. The module provides methods for creating users, for removing users, for saving changes to users, and for authenticating users. The API also defines a generic User interface, and provides transaction wrappers for certain transactions such as authentication. More on that later.

The first interface is defined in the aik.module package, the UserModule. The following methods are highlighted here:

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

login(...)	This uses a userId and a password and returns a LoginResult object to give the caller information about the authentication attempt.
create(...)	This requires that you pass it a valid system User object and will create the new User object based on the system id and the password passed. It returns the User object as a result.
delete(...)	Again, using a logged in system User object, the system will delete a user and return a UserTransactionResult.
changePassword (...)	This changes the a user's password, again using a system user account and returning a UserTransactionResult.
addCellToUser (...)	This should handle the process of determining whether the user can add a particular cell to its profile and handle the interaction with the Application Module to do this.
addUserToGroup (...)	This is similar to cell method above.
retrieveGroups (...)	This returns the Groups that are attached to available in the system. It requires a system user as a parameter.

The second interface is the User interface. It is held in the aik.system.user package. It is basically a handle to an actual user that gets passed around in the system for other module functions. If you were writing your own custom user module you will need to implement this interface and override the following functions:

getSession()	This returns the Session attached to the user.
getSystemId()	This returns the system identifier of the user.
getUserId()	Returns the more common user id
retrieveGroups()	Returns an array of the Groups assigned to this user.

The third interface is the Group interface. This is a common way of managing users and giving system administrators a bigger tool to help manage security. Again, when writing your own UserModule implementation you'll need to create a class that implements this interface. You'll need to follow through with the following methods:

getDisplayName()	Returns the display name of the Group, like “Power Users”
getSystemId()	Returns the system identifier of this Group

Finally, there are a few other interfaces that are defined to make this module more useful to other modules. The first is the LoginResult. This is a way for a another module to call a single login method

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

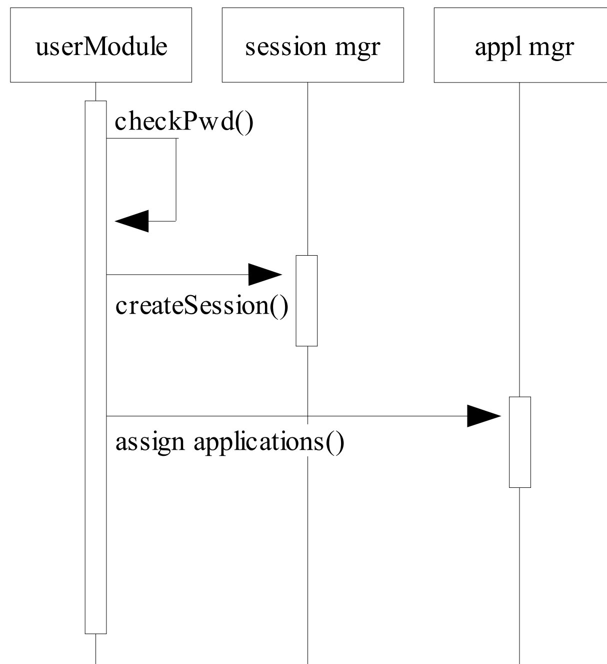
(see the UserModule above) and get not only the User object if the system authenticated properly, but also some information about why the authentication attempt failed.

The Current UserModule implementation

Support for the UserModule in the current version of the Aikernel is very rudimentary. It basically uses properties files to store users, their passwords, and their groups. This was done a) because it was very quick and effective and b) it supports a diverse set of environments without putting too much load on new Aikernel users. If we had, for example, required LDAP or a SQL database to support the calls, this might have dissuaded people from installing and running the application. We could also have supported multiple mechanisms but it seemed like a better idea to focus on the core purpose of the application and hope the when people start really using the Aikernel they can provide their own implementations of these numerous authentication systems.

The reference implementation is in the aik.impl.modules.user package.

At a basic level, there are some things that you'll need to do if you want to roll your own UserModule. These things aren't defined in the interfaces (i.e., can't be enforced by Java interface alone) but are required to keep the system consistent with a diverse suite of modules. Let's say that again, a little more simply: if there are a lot of modules available for download, and you deviate from this way of "doing business" then the other modules will likely fail and so will your server.



Most of f the behavior requirements stem from the initial login of the user. If you are implementing your own UserModule, you need to "provision" the user at the time they login with certain

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

information.

The first is the session. You'll need to reach out to the SessionModule and instantiate a new session that will be attached to this user.

The second is the application suite. You'll need to negotiate with the ApplicationModule to setup the suite of applications that they user will be using throughout this session. In the reference implementation, we store the applications (remember, these are also call cells) that each user wants to start their session with in a file called "profile.properties"

The following files are used in the reference implementation of the UserModule:

- users.properties - this stores the users as the "key" of the properties file and the password as the "value"
- groups.properties - this stores a group number in the value of the properties file and the users id as the key.
- profiles.properties - this lists the user id as the key and the list of system identifiers of the cells that the user wants on startup separated by a comma. Note the trailing comma after the last cell id. This is necessary because of the way we are using the StringTokenizer to split the value part of the properties file.

Room for Improvement

- ☒ Support LDAP authentication.
- ☒ Use the Java new standard JAAS architecture.
- ☒ The system doesn't provide a mechanism to check user's permissions to do certain actions. The only place this happens now is in the ApplicationModule, but that module is doing the permission checking. It would be better to centralize this here in the UserModule.
- ☒ In the future, perhaps the system shouldn't require that the UserModule implementors do the provisioning of the user. Perhaps they should just through certain events and allow the other modules to respond to this.

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Chapter 4: Session Module

This is a simple module like the UserModule, but it is a little hard to understand if you don't have a good understanding of the UserModule. If you haven't already ready that chapter, at least go back and skim the opening paragraph.

A session is a simple piece of code but it is important for the system to have so it can support multiple users accessing the server at the same time. First, we'll take a look at the interface API's of the SessionModule and then we'll look at how it is implemented in the current Aikernel reference implementation. The session interfaces are defined in the [what] package. The reference implementation is in the [what] package.

The interface

The first interface to consider is the SessionModule interface. It is a core module and it extends the Module interface. If you aren't sure what this means, there is a brief explanation of it in the User Module chapter. This module has a simple set of methods. They are:

[methods here]

The second interface is the actual Session itself. It is the handle to the session being evoked. It is what prevents one event from firing in the wrong session. It has the following methods:

[methods here]

The current SessionModule implementation:

Because of the simplicity of this module, there wasn't a whole lot of work to create a functional reference implementation. Basically when a session gets created it simply holds that session in a Hash table. It remembers the time that the session was last accessed. It does this by timestamping the system each time a method of substance gets called on the session object. The module implementation starts a thread with a queue scanner that looks at all the open sessions and tries to kills sessions that are over a certain age. The age is specified in the aik.conf file. For more about this file, see the chapter on developing your own custom modules.

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Chapter 5: Application Module

Now we are getting into more of what separates the Aikernel from other server based applications. Most applications that are server based need to maintain information about users and about sessions, but now we will talk about applications and cells. Kind of like your operating system, the Aikernel doesn't do much for a user without some applications to run. Applications in the Aikernel are much like a web application or an application on your desktop without a graphical interface. They mostly interact with the user through text channels by responding to events fired by the Aikernel. For more information about Cell or Application development, refer to section 3 of this Guide.

This is the module that takes the responsibility to load and keep track of applications that have been setup in your Aikernel installation. This module, like the others in previous chapters is based on the Module interface. The API is defined in the [what] package, while the actual ApplicationModule is contained in the aik.modules package.

The interface

[insert UML diagram here]

As we've done in previous chapters, the first interface we'll take a close look at is the ApplicationModule. Its job is to serve up information about the applications loaded in the system, and what applications have been "suited" to which users. It also extends the Module interface. Review the following functions in detail:

[insert methods here]

The next interface we'll take a close look at is the Application itself. This is just a handle to an application that could be bound to a particular User and Session. It provide some useful information about itself and provides hooks for the bindings. Review the methods here:

[insert methods here]

There are a few more interfaces to think about in the application module, but this is the last major one we'll consider here. It is the ApplicationSessionSuite and it acts as the grouping of User's currently instanced cells and their Session. Not all applications are always available to all users in the Aikernel server. The user should be able to add or remove applications from their suite when they want to and have enough permission to do so. These are the methods that it defines:

[insert methods here]

The current Application implementation:

There are a lot of ways to get this job done. In this project the simplest and cleanest approach was

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

selected as the first implementation of these interfaces. As you consider rolling your own implementation of the application API, you might can take into account your own issues which may increase or decrease the difficulty level of the system. In this implementation, there were two things we needed to do very well:

support application discovery and binding
keep tabs on application user sessions

For the first task, the project models the way that web archives (WAR's) are defined in the J2EE servlet/JSP specifications to some degree. When a cell is developed, all a cell developer needs to do is to bundle their application as a JAR file, rename the extension to .cell and put it in the deploy directory. In the reference implementation, you will need to restart the Aikernel server to pick up the new cell. This makes the cell available as an application in the palette and users are now free to pick it up and bind it to their sessions.

The second task was a little easier. This implementation of the application module simple defines a class that implements the ApplicationSessionSuite interface (referenced above) and stores each instance of the class at a Session level, and stores it in a standard Java hashtable.

Expected behaviors:
Like the User module documented in Chapter 3, there are certain behaviors that the application module needs to take on if it expects to cooperate with any module that might be installed in the Aikernel server. This behavior is related only to application loading.

The module implementation should support the .cell format and it should automatically discover a class in the .cell file that implements the Application interface as we have done in this implementation.
The module implementation should publish a deployment location for cells even if it is remove.
The module implementation should provide a new instance of each cell class to each unique session. This rather than reusing one instance and making it reentrant.
The module implementation should bind each of the cell instances to the Event sub system. For specifics on this, refer to the source code documentation.
Room for improvement

The class loading on the current implementation uses the default class loaders from Java. This seems to cause problems in today's application servers (such as JBoss) and web containers (Catalina/Jetty).
The module should support hot swapping of the cells in the deployment directory. This might go hand in hand with owning our own class loaders.
Should we support JNDI locating of applications, allowing behind the scenes distribution of applications. With this we could put a simple .xml descriptor of the remote application in the \deploy directory.
Perhaps a more advanced cell package could created that includes EJB's, web servlets/JSP, and more with some thing like a Cell ARchive (CAR) format.
The past implementations of the Aikernel supported a much richer set of configuration options and xml-based activator/context publishing. Perhaps some of this should be resurrected. There might be benefit in letter Aikernel server administrators configuration applications at the server level.

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Chapter 6, I/O Module

The input/output module of the Aikernel is where you can get a lot of bang for your buck as a module developer, or even a simple user of the Aikernel. As you might expect, the IOModule defined in the aik.modules package provides the eyes, ears, and mouth for the Aikernel. You can dramatically impact the way that the Aikernel works just by changing the way it connects to its world. This chapter discusses the how the API was defined and how the reference implementation works.

The interface

There is really only one main interface to consider with this module. It is the IOModule and it has the following functions:

You'll notice that there is one method to request data directly, but it should not be the primary mechanism for loading input data. Input data can happen anywhere in the Java Virtual Machine (VM) simply by creating a new InputEvent and calling the register() method on it. The retrieveInput() method defined by the IOModule is meant primarily as a way for the cells or other modules to directly retrieve data from the user or other input stream.

The current I/O implementation

[uml picture here]
The current implementation is directly squarely at developer and high tech users. It requires a telnet connection on a non-standard telnet port. The port that it opens a SocketServer on is configurable by modifying the aik.conf file. When the launch() method gets called (for more information on this see the custom module developer chapter) it creates a new class on a Thread that opens a SocketServer on the port you specified.

When a new socket request comes in, it passes this connection to the new ClientSession object on a new Thread. That object retrieves data from the user's telnet client and transforms that information in a) events for the Aikernel or b) commands that use is trying to issue to the telnet session itself. The following commands are currently defined:

[commands here]

These commands are handled solely in the ClientSession class and never become events for the Aikernel to process. If the input comes in as anything other than an event (such as the user saying, "my car is red") then that information will be packaged into an InputEvent and registered into the Aikernel. It then gets processed by the next two modules.

Room for improvement:
Rather than a new Thread per user, it might be a good idea to do Thread sharing on this.

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Chapter 7, Activator Module

If you've been reading chapter by chapter, then you've probably forgotten what the Activator was for by now. The Activator is a key concept to understand about the Aikernel. An activator is the smallest significant stimulus that can be handled by the Aikernel. What do we mean when we say "stimulus"? It is a fluid concept that can mean different things to different cell developers (in other words, you as a cell developer get to define what an activator is to your cell).

To understand best, it might make sense to step back for a second and consider other examples from real life. In the United States, when you dial a telephone, you dial seven digits (unless you live in one of the big cities where you are probably dialing 10 digits). When you are finished dialing the phone, the telephone on the other end of the line rings. Each individual number is an event. If we were designing an Aikernel based central switch, would each digit be an "Activator"? Probably not, because each number is not enough to act on. But a whole number might be an "activator".

A more concrete example of an Activator, and one that is very common in the Aikernel comes from natural language. When the user says "the car is red", is the whole sentence the Activator? In this case, probably not because it is too much for your cell to act on. In this case you might want to define an activator for the word "car", one for the word "is", and one for the word "red". Incidentally, these activators get lumped into a common feature called a Context, but we'll talk more about that in the next chapter.

But we probably still don't know what to do with the word "is" unless we can think of it as an action. It helps if we know that the word "car" is a fact, and it helps if we know that the word "red" is a modifier. Activators are information that your user triggers either directly or indirectly and you want to act on with your cell. They the food that the Aikernel feeds your cell when input matches.

No we have stepped with both feet into the world of natural language processing. Yes, the Activator concept is a way of dramatically simplifying the task of extracting meaning from a noisy input channel (this time, human speech has a lot of noise built into it that makes meaning ambiguous to machines). No, it isn't good enough just by itself. There are a lot of support services that the system needs to provide to make this work in the real word. This includes thesaurus lookups: a car is the same as an auto, automobile, and maybe "ride" in some slang. Car could also be "cars" or some other morphologically different instances of the word. Any we might not want define activators for every adjective that could describe out car, and so we'll need some part of speech taggers too. These topics will get covered in more detail in Section 2, the Logic services.

The interfaces

There are really just two primary interfaces that we need to consider when we look at the Activator module. The ActivatorModule itself is defined in the aik.module package while the remaining interfaces are defined in the ailk.activator package. Let's start by looking at the ActivatorModule. Like the others before it, this module extends the Module interface. Let's take a look at the methods:

[functions here]

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

The next biggie to consider is the actual Activator interface. It is the handle to the implementation of the Activator being used. Note that the Activator construction happens through the ActivatorModule. If cell developers try to roll their own Activators, this will fail. Here are the methods:

[functions here]

The current Activator implementation

[insert uml here]

There is surprisingly little work to be done when creating an implementation of the ActivatorModule. The basic task is to create an implementation of an Activator and to create the factory implementation that can generate Activators when requested (in other words, the ActivatorModule).

Room for improvement:
Lots, but where is it?

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Chapter 8, Context Module

Contexts are almost as important as Activators but don't exist without them. If you don't understand Activators, please go back and review chapter 7 before proceeding. In its simplest, a context is just a collection of activators. It basically tells the Aikernel and its logic modules that when an input source (such as a user typing input) has strung together some activators that it can connect together.

Another way to think about Contexts is to think of them as functions of your application. Recall the Activators from the example in chapter 7 that we designed for dialing a phone. In this case, a simple phone number might be a function that says "I want to call another number", you might have different contexts for retrieving voice mail, calling for operator assistance, each which would have different dialing sequences as activators. In the other example, using natural language, we might want to support "I want to buy and new car and I want a red one" as a context that support buying cars.

The Context module support these abilities. Like the other modules it is defined by an API and by an implementation.

The interfaces

The ContextModule interface is defined in the aik.modules package and the remaining interfaces are defined in the aik.context module. This is very similar to the Activator interfaces where the two important interfaces to consider are the ContextModule and Context interface. The ContextModule implements the Module interface, and it also defines the following functions:

[methods here]

The Context is the actual handle or reference to a Context that has been defined. The context has the following key members:

[methods here]

The current Context implementation

Even thought it is a simple API, this is one of the most complex implementations in our entire Aikernel reference implementation. It was done like this for two reasons: a) to showcase the power of the modular application design and b) to play some fun games with the Aikernel in general.

The main objective of the Context system is to build and server contexts to modules that want to publish them. Usually these are just cells. This is the easy part. We have a simple realization of the ContextModule interface and we have duly implemented the methods of the Context interface. This is

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

where it gets complicated. We decided to implement something call "context following" and take a generic approach to persisting this.

Let's look at both pieces of this separately, following the context and persisting it. Context following is one way to improve the Aikernel's decision making ability. It give the logic processing more information about the statistical likelihood of one context following another. Say for example that the user is using the AiRSS Service cell (check the aikernel.sourceforge.net project site for more information about this cell). Typically, the user will use this cell to first browse what RSS feeds are available on the server, and then list the items on a particular. So, imagine this dialog:

user> I want to browse the feeds
aik> Ok, I have two feeds, one from Java and one from Slashdot
user> first one, please
aik> Ok, the following stories are available . . .

This is far more complicate than it looks to the user. In the first turn, the user activates the "Browse Channels" context on the AiRSS Cell. Then the user gives only a tiny bit of information on the next turn to "List News Items on Channel X". The reference implementation allows the Aikernel to learn the connection between the two contexts. In order to do these, we added some additional methods to the basic Context implementation (well, actually, these became part of the standard API). We then needed to create a dependency between the Activator Module implementation (you'll see more of this in Section 2) and this context implementation. Since they are not part of the methods defined in the ContextModule or Context methods, we needed a way to store this information.

So we created an additional Module. This module is used only by the Activator and Context implementations provided by the reference implementation. However, it makes it possible for module developers and people who want to find a way to make this context information "stick" to override the piece of the kernel that actually stores the relationships among contexts into their own database. Please refer to the source code and source code documentation for more details about this.

Room for improvement
You betcha. Where?

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Chapter 9, Rolling your own Modules

As we showed in Chapter 8 there are instances where you will want to make new modules available to your cells. This is easy to do in these simple steps.

Create a class that implements aik.module.Module
Make sure that class is on the classpath of the Aikernel
Add a <module entry to the aik.conf file using the examples provided in the file
Restart the server

Now your module will be available to other modules and to cells. To access the module, you can call the ModuleManager class, which has static methods that give you access to your new module.

Room for improvement:
The system should automatically add files from the lib directory into its class path
Again, it might be a good idea to support JINI and JNDI module accessing and configuring with a deployable XML file.

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Section 2

The Logic Interfaces and Implementation

$a \cdot i \rightarrow \mathrm{K}$

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Chapter 10, Logic Overview

It was mentioned in the overview that the logic classes are basically modules that can act on input data and generate events to stimulate cells to do something. This is mostly true but you might not see the distinction when you are looking at the code itself. Logic classes should generally implement the LogicModule, but in this version of the Aikernel the LogicModule interface simply extends the Module interface and doesn't provide any other services. Eventually it will.

This is how the logic classes generally work today
[diagram today: the event manger, the single arbitrator and how it calls out to the other modules

This is how it would be better:
[diagram chained transformation and analysis of payload data]

Chapter 11, Natural Language Processing

This Aikernel started out as strictly an intelligent chat system, so it is only natural that this is where its true strength lies today. Please review the following diagram to understand all the big pieces and parts:

[pieces and parts]

Describe each piece and call out the code to reference.

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Chapter 12, More intelligence

Recall that the purpose is not to tie in only to the user's language but all the events that make up a day to day life. This will require a new generation of sensors, IOModules, and logic. Algorithms like this:

Genetic algorithms for batch offline learning from past events

Neural networks to improve learning and training

Agent technology to improve what it can learn, and what it can act on

Knowledge bases to improve data driven events.

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Section 3

Cell Developers Guide

$a \cdot i \rightarrow K$

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?

Chapter 13, How to develop a cell

THIS DOCUMENT IS A WORK IN PROGRESS! WANT TO HELP?