HCI4 06

Elvin Lab

9 October 2006

This lab is intended to introduce you to a system you may wish to use in your HCI4 exercise: the Elvin messaging system.

Background

Elvin is a content-based messaging system. Producer processes can send messages with arbitrary content. Consumer processes can receive messages via subscriptions that specify the type of message content in which they're interested. For example, a set of "content providers" might send messages that include tuples, or elements, for name, date, time and location. Another process could "subscribe" to any messages with a location of "Boyd Orr". This offers a simple and effective way of enabling processes to communicate with minimum effort.

Messages (or notifications) consist of a set of elements (name, value pairs), e.g.,

```
Name="Phil", Age=39
```

A few simple data types are supported, including strings, integers and floating point numbers.

The Elvin system consists of two parts:

- a router that handles the actual transport of messages between producers and consumers and
- SDKs for several languages (e.g., C and Java), enabling producers and consumer programs to be written.

In what follows, a line with ➔ is something for you to do.

Finding Resources

Where is the Elvin router?

We will be using a router located on `aguijan`. It is referenced as follows:

```
elvin://aguijan.dcs.gla.ac.uk
```

Where is the Elvin Java SDK?

The Elvin Java SDK is available at:

```
\\anjouan\public\java-elvin-4.0.5
```

It includes jar files holding the required classes, plus source code for example applications. There are also a couple of useful batch files for running example producers and consumers.

➔ Copy the Elvin Java SDK directory to somewhere in your filespace.

Running Your First Producer

➔ `Go to java-elvin-4.0.5\examples`

`jep` is a simple Elvin event producer. It can be run via the batch file testProducer.bat. This file contains:

```
java -cp "java-elvin-demo-4.0.5.jar;..\java-elvin-
4.0.5.jar" org.elvin.test.je4.jep -e
elvin://aguijan.dcs.gla.ac.uk
```

➔Execute `testProducer.bat`. You should see a Windows console appear.

➔ Enter the following at the console, replacing <your login> with your unix account name and <your first name> with your first name:

```
<your login>: "<your first name>"
location: "boyd orr"
Age: 39
---
```

Note: you must end the message with the three hyphens. They are an "end of notification" marker.

However, you won't see anything yet until you have a consumer program to receive the message.

Running Your First Consumer

`jec` is a simple Elvin event consumer. It can be run via the batch file `testConsumer.bat`. This file contains:

```
java -cp "java-elvin-demo-4.0.5.jar;..\java-elvin-
4.0.5.jar " org.elvin.test.je4.jec -e
elvin://aguijan.dcs.gla.ac.uk -s "require(pdg)"
```

Notice the final argument, -s "require(pdg)", which will set this consumer to subscribe to any events that include an element identified with 'pdg'.

➔ Edit testConsumer.bat, replacing 'pdg' with your login name. Save it.

➔ Execute `testConsumer.bat`. Another console will appear, this time to hold output from Elvin. Arrange the two consoles so you can see each of them at the same time. Now each time you create a message with the producer that includes <your login> as an element name, the message will appear at the standard output of jec (the consumer's console).

➔ Try out a few messages in the producer console to see the effect.

Multicast

Messages can be consumed by multiple processes.

➔Find a friend or lab neighbour and arrange to subscribe to the events from their producer. You'll have to change the command line parameter of the consumer to an appropriate value.

Building an instant message system in 3 minutes

If John and Jane want to communicate, they would send messages using `jep` like this:

```
Phil: "Hello Jane"
---
Jane: "Hello Phil"
---
```

Each would create a consumer application using `jec` with this subscription:

```
require(Phil) || require(Jane)
```

This will result in a subscription to both events identified with 'Phil' and events identified with 'Jane'.

➔ `Find a friend or a lab neighbour and try it. Try adding more people.`

Subscription Expressions

So far we've only seen a single subscription expression, viz., "require(<item name>)". The table below shows some other potential subscription expressions.

Example	Description
require(Demo)	Notifications with an element named **Demo**
Email == "user@example.com"	Notifications with an element named **Email** which has a string value equal to "user@example.com"
wildcard(Email,	Notifications with an element named **Email** which

"*@example.com")	matches the wildcard expression.
regex(Subject, "[Tt]ransmeta")	Notifications with an element named **Subject** that matches the regular expression
Temperature >= 250	Notifications with an element named **Temperature** that is numeric and has a value greater than or equal to 250
Time < Record-Time	Notifications with an element named **Time** with a value less than the value of the **Record-Time** element. Both fields must be present and numeric for a match to occur.
contains(Subject, "laptop") && (Company == "Sony" || contains(Product,"Viao"))	You can compose complex expressions using the boolean operators

Writing an Elvin Application

Basic Java template applications are located in `hci4Examples. The relevant applications are:`

`hci4DemoProducer` and `hci4DemoConsumer`

Look at the source code. Each takes an Elvin router reference as its only argument. The producer creates a single notification and then terminates. The consumer subscribes to a "topic" and then writes its output to standard output. These two templates will be the basis of any Elvin application you might write.

➔ You can try them out via the two batch files, produce.bat and `consume.bat`. Have look at their contents.

➔ Execute consume.bat to start a receiver console. Now execute `produce.bat` a few times to create new messages.

Further Information

Full Javadoc is available in the Elvin Java SDK `doc` directory.

Additional information, including the full subscription language specification, can be found at:

`http://www.mantara.com/support/docs/`

UAR Command Line Options

ip
IP of proxy if using one (must have port as well if this is specified)
---
port
PORT of proxy if using one (must have IP as well if this is specified)
---
path
Path to output log files. Defaults to C:\local\grumps\ if not specified.
---
consentpath
Path to consent file. Consent file sets options for individual users.
The options which can be specified are listed in the "Other Options" section below.
If consentpath is specified and the file doesn't exist or the currently logged in user is not found in the consent file, then the UAR will exit.
A sample consent file format is given in .\SampleConsentFile.txt.
---
adaextensions
Allows advanced logging of user activity in AdaGide development environment for the ADA 95 language.
Activity logged is Compile, Build, Run, Comment and Uncomment.
---
maxfileopentime
The time in seconds that a logfile is kept open for before creating a new logfile.
---
nonobfusticatedkeylist
Path to a text file containing a list of Virtual Keycodes which should NOT be ignored if keypresses are not obfusticated.
Entries should be one per line and should be something like:
VK_ENTER
VK_BACK

Other Options:
"mc" - mouse clicks: can be "on" or "off"
"mm" - mouse moves: can be "on" or "off"
"mw" - mouse wheels: can be "on" or "off"
"wt" - mouse window: can be "on" or "off"
"at" - mouse application: can be "on" or "off"
"kc" - keyboard collection: can be "all", "hidden" or "off"
"wc" - window collection: can be "all", "hidden", or "off"
"process" - display only simple process name.
"fullprocess"; - display detailed process name.