

JAWAP: the Java Application Framework

Colin Grealish, Ericsson

Ericsson has been a major supporter of the Wireless Application Protocol formulation in recent years, and is eager for Internet services to be rendered in WML as well as HTML.

Ericsson expects most WAP applications to be developed in the same way as most HTML-rendered sites are today. Because of the benefits to Ericsson of extensive WAP use, it has developed JAWAP. JAWAP is targeted at the low-end WAP developer, but may be used on almost any platform.

JAWAP, the Java Application Framework, is a Java library that's freely available to developers inside and outside of Ericsson. The complete source code is downloadable from the Ericsson Developers' Zone at <http://www.ericsson.se>. The library facilitates the creation of WAP services, in the form of Java servlets, in a very quick and flexible manner. The very fact that Java, the JDK, and the JSDK are the development environment is in itself very useful to the amateur developer, since the simulation environment can be an average PC or laptop. This can contribute to a reduced cost of entry for emerging service development enterprises.

JAWAP Architecture

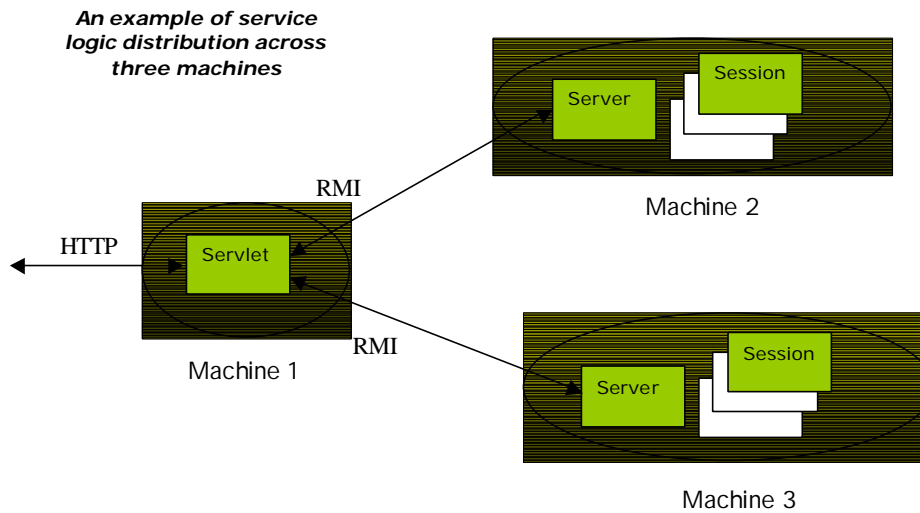
JAWAP provides the developer with a three-class model on which to build a service:

- The Servlet Class
- The Server Class
- The Session Class

The Servlet Class

The first class is the service itself, which is essentially an HTTP servlet extended to provide Remote Method Invocation (RMI) support. This allows the actual service logic to be implemented on a separate machine from the web server that hosts the service, thus providing the basis for load sharing. Again, this suits the 'budget' service developer, who may not be sure that scalability is required at the onset of service deployment.

This development approach allows easy migration to a load sharing solution, and is therefore scalable if the future need arises. As the scalability requirements become more apparent, it is possible to take advantage of the separation of service logic and servlet. It is also possible to have instances of the service logic on more than one server. To use this effectively, it is necessary to put some load distribution logic in the servlet. At the simplest level, this could be a "round robin" technique whereby sessions are activated in servers in circular order. A more complicated mechanism would be to determine which server has the highest spare capacity. This has the advantage of allowing servers to be taken out of service for maintenance without loss of service.



JAWAP has been designed such that if the servlet cannot reach a server over RMI, it generates a default error card. The application designer can override the information in the error card if necessary.

The Server Class

The next class of the design model is the server. This is always active during the deployment of the service, and communicates with the servlet class via RMI. Because it's always active, it is useful for things such as feed handling. An example could be a news headline service, where the headlines are fetched at ten-minute intervals and stored in the server to reduce delay and load. The server also keeps track of the active sessions of the particular service that are present at any one time.

The Session Class

The third part of the model is the session class. This provides support for sustained interactivity over HTTP to a particular session, and can significantly reduce the time and complexity of services for WAP development. Part of the session handling is a set of library methods for writing WML decks. Methods include such things as `beginDeck()`, `setCardBreak()`, and `setFont()`.

A particularly useful feature of the session class is its facility for button handling. Button handling in JAWAP has characteristics that are recognizable to applet designers, and the Java applet was obviously the inspiration. It allows links to be defined as buttons, and buttons to be defined as links. When using the button functionality as a button, or a link as a button, a "button listener" method is applied; this is all handled transparently to the developer, thus assisting smooth and intuitive service creation. For examples and screenshots of how this is implemented, refer to section 9.5 of the Developers Guide.

Here are all the JAWAP elements and methods that a programmer can use in an application. These methods are used to generate WML decks, and roughly speaking, each method generates a section of WML.

Elements	Associated Class	Related Methods
User Interface	Wapplication	setParagraphStyle()
		addAnyWML()
		addWMLFile()
		setFont()
Deck	-	beginDeck()
		beginParagraph()
		beginTextStyle()
		addNewLine()
		endTextStyle()
		endParagraph()
		setCardBreak()
		endDeck()
Button	Button	add()
select + option	ExitList	add()
	Font	
Img	Image	add()
Anchor	Link	link.SetFont()
		add()
select + option	List	list.add()
		add()
input	TextField	add()
	TextArea	TextArea.append()
		TextArea.setFont
		add()
timer	WAPTimer	add()
table	-	add()
		BeginTable()
		SetTableLocation()
		EndTable()

The session is kicked off with the abstract `initialize()` method being called by the server, which must be overridden. This method includes any parameters passed in the URL.

A Sample Application

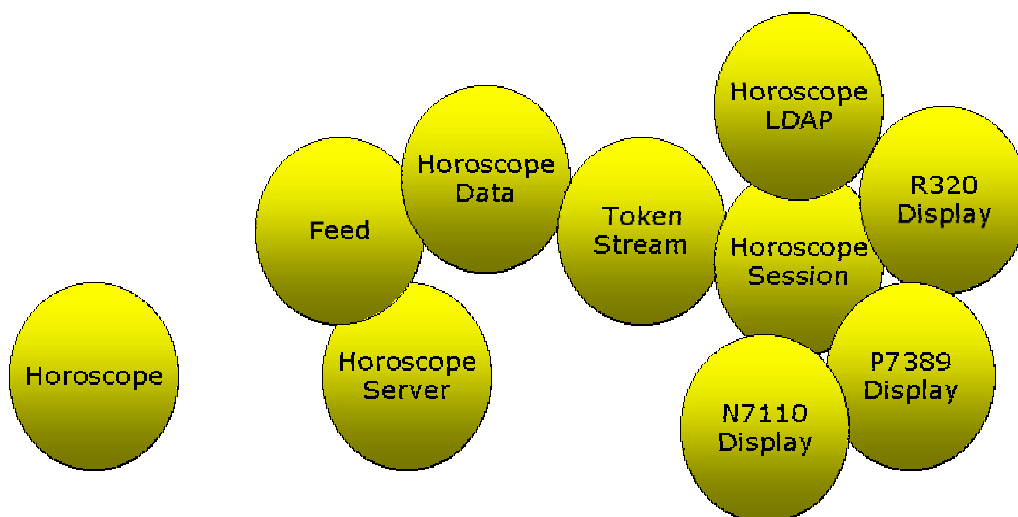
The first application we developed using JAWAP was a simple horoscope feature: *The Russell Grant Horoscope*. This was a service that provided a user with their horoscope. The following requirements were applied:

- The first time a user uses the service, a screen is presented requesting the user's star sign.
- The horoscope for that star sign is then displayed.
- The user's star sign is recorded in an LDAP directory for future use.
- The user has the option of changing their star sign.
- A help feature is provided if the user is not sure of their star sign.
- When the user returns to the service, the star sign recorded in the LDAP directory is referenced and the relevant horoscope displayed.
- The feed is provided in XML and fetched via HTTP. The feed is updated on a daily basis.
- The Ericsson R320, Nokia 7110 and Motorola Timeport P7389 are to be supported.

As specified, the feed was to be provided in XML. Here is a snippet of the Aries and Taurus code on the 16th February 2000.

```
<horoscope type="daily" at="20000216">
  <star sign ="Aries. Mar21-Apr20">
    <text>Your home environment is due for a shake up. Whether you're a student of
    Feng Shui or a master, you could do worse than to steal some decorative ideas from
    this old oriental way of living. The introduction of anything wooden or to do with
    communication is excellent.</text>
  </star sign>
  <star sign ="Taurus. Apr21-May21">
    <text>You'll be up with the movers and shakers if you open your mind to new
    ideas and techniques. Getting involved with anything from the World Wide Web to
    computers per se will lead to all kinds of interesting future opportunities. A
    travel choice should be made today.</text>
  </star sign>
</horoscope>
```

The JAWAP framework was used, and the following implementation classes were developed:



- Horoscope: Simply the identification of the service, locatable by a URL. No load distribution is provided for the first implementation.
- HoroscopeServer: This is active for the deployment of the service. On instantiation, the feed object is created and a thread of the feed class is started.

- Feed: The content is fetched via HTTP at instantiation. The feed is updated daily, being refreshed at midnight.
- HoroscopeData: The current set of horoscopes is stored in this class and passed to the horoscope session when a user initiates a service session.
- TokenStream: This class is used to parse the HoroscopeData to select a particular star sign.
- HoroscopeSession: The WAP session is held in this class.
- HoroscopeLDAP: The logic to fetch the user's profile is implemented in this class. For the demonstration, a dummy class is used; otherwise a directory server would be needed.
- R320Display: The implementation for the Ericsson R320.
- N7110Display: The implementation for the Nokia 7110.
- P7389Display: The implementation for the Motorola Timeport P7389.

A display class exists for each WAP browser supported. For this application, the Ericsson R320, Nokia 7110 and Motorola P7389 are supported.



The Ericsson R320, illustrated above, has the smallest screen of the three devices. But if word wrap is used, the content is compatible with all the devices.

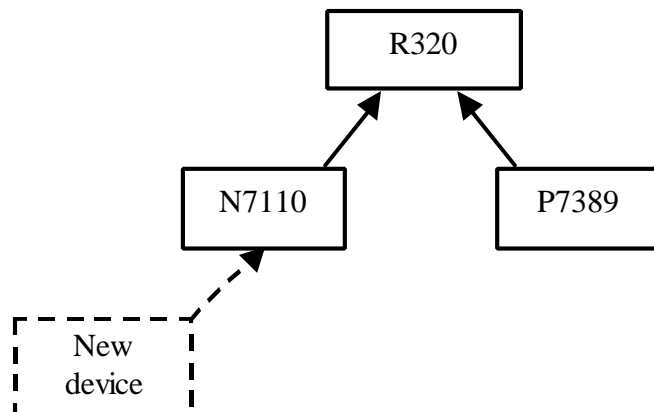


For the Nokia 7110 there is no horizontal navigation. This does not pose a compatibility problem, as the application does not have any side-by-side links. The deck size for the Nokia is different from the R320, and so this value has to be checked.



The Motorola Timeport P7389 uses the Phone.com browser, which is shown above, and has some differing characteristics from the other two browsers. The most significant of these is that the deck title is not automatically placed at the top of the screen.

Because the display is implemented in a Java class, inheritance can be used optimally. The following class model is applicable:



The Ericsson R320 was developed first, and selected as the basis for all the other devices. The classes N7110 and P7389, for the Nokia and Motorola, were extended versions of the R320, with only small changes. This was a very neat solution, as there was substantial commonality between the devices. (The button set, for example, was common to all.) The diagram above shows a possible extension for a future Nokia release where there is commonality with the 7110.

Ericsson

Ericsson does not expect to generate income directly from selling WAP services (apart from some niche applications that require specialist expertise), but if WAP does become widely available, Ericsson stands to gain real revenue from:

- Terminals, such as the MC218 organizer, R320 WAP phone and R380 EPOC-enabled phone
- Infrastructure, such as base stations, access routers, portals and even telephone exchanges due to increased telephone traffic

- Consultancy, training, and solutions support for new operators and developers of WAP services

An example of the type of niche application that Ericsson is developing is the BT/Telfort Euro 2000 site. This site was to be available during the championship, and the number of users across Europe was very difficult to estimate. The service also had to be "right first time", as there was not an opportunity to re-engineer the service. Because of various factors, one of them being that loading was so critical, JAWAP was not used for this site.

*The Euro 2000 site is currently still active. See the URL:
<http://193.78.100.17/scripts/SA/PHSA/wml/common/main.wml>*

Recent forecasts are that Ericsson will gain only 1% of wireless sites, and that some of these will be more strategic than profitable. We at Ericsson are very satisfied with JAWAP, and we're happy to assist in the development and deployment of services with a low cost of entry. The following are downloadable, free, and form the basis for any startup service development:

- Java 2 SDK, JAVA Development Kit: <http://java.sun.com/j2se/1.3>
- JSDK, JAVA Servlet Development Kit: <http://java.sun.com/products/servlet>
- JAWAP, Java Application Framework: <http://www.ericsson.com/developerszone>
- Ericsson WAP IDE, Ericsson R320 Emulator:
<http://www.ericsson.com/developerszone>
- Nokia 7110 SDK, Nokia 7110 Emulator: <http://www.forum-nokia.com>
- Phone.com SDK, Motorola P7389 Emulator: <http://www.phone.com>