



OpenPortal Whitepaper & Design Document

7-06-99

Brad Neuberg

Paolo de Dios

What Is OpenPortal?

OpenPortal is an open-source Java-server platform under the GNU Public License (GPL) and LGPL that makes a new generation of web sites possible. These new websites are *open*, *growable*, *transportable*, *changeable*, and *interoperable*. OpenPortals all share the following characteristics -

The 10 Characteristics of an OpenPortal

1. ***Everyone* is a user.**
2. ***Everything* is changeable and editable.**
3. **Users can add, create, and modify sites extensively into unknown new directions.**
4. **An OpenPortal is as open or closed as you like – and everywhere in between.**
5. **All the interesting stuff happens when openness is taken to the point of craziness.**
6. **Websites become discussions, and discussions become websites.**
7. **The website supports its own growth.**
8. **The walls between web sites are broken down, and OpenPortals can interoperate.**
9. **If they don't want to play, wrap 'em as components.**
10. **You may not even know you're on an OpenPortal (because you're not).**

These OpenPortal Ten Commandments are explained below.

1. *Everyone* is a user.

Traditionally, most websites divide the people who make, design, and use a website into separate categories:

A Programmer creates the programs that run on a server. These programs help to dynamically generate the site.

A Graphic Designer creates the html templates that help generate the site.

An Owner lays claim to the server and sets the direction of the site.

A Content Creator creates the actual stuff that is on the site.

An Editor decides which of the content created by the Content Creators is worth keeping.

An Administrator takes care of the site on a systems level.

An last, but not least, is the poor User who is left to consume the website made by his superiors.

Programmers, Graphic Designers, Owners, Content Creators, Editors, and Administrators all do their work directly on the server side using special tools that are separate from the website itself, and are usually located in the same physical location:

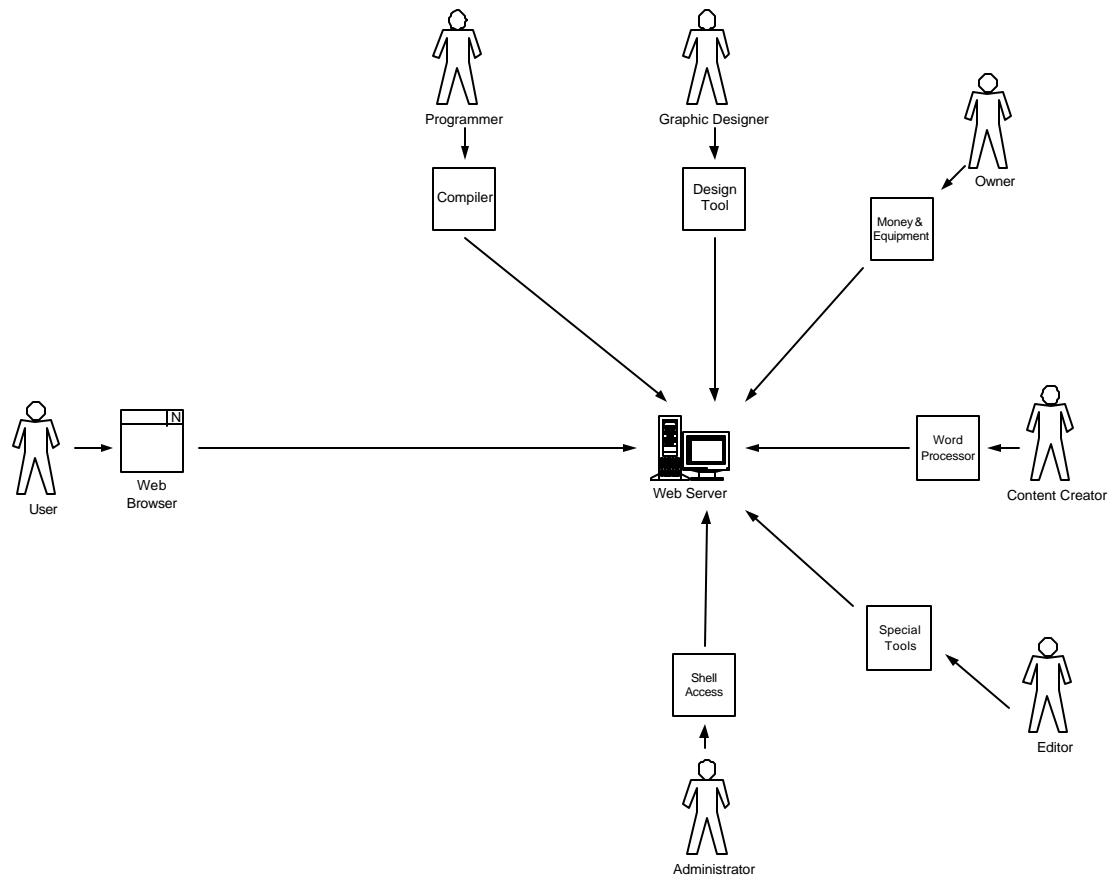


Exhibit A - How Folks Get their Jobs Done - People on the server-side use special tools to make and maintain the website, while users use a browser to access the website.

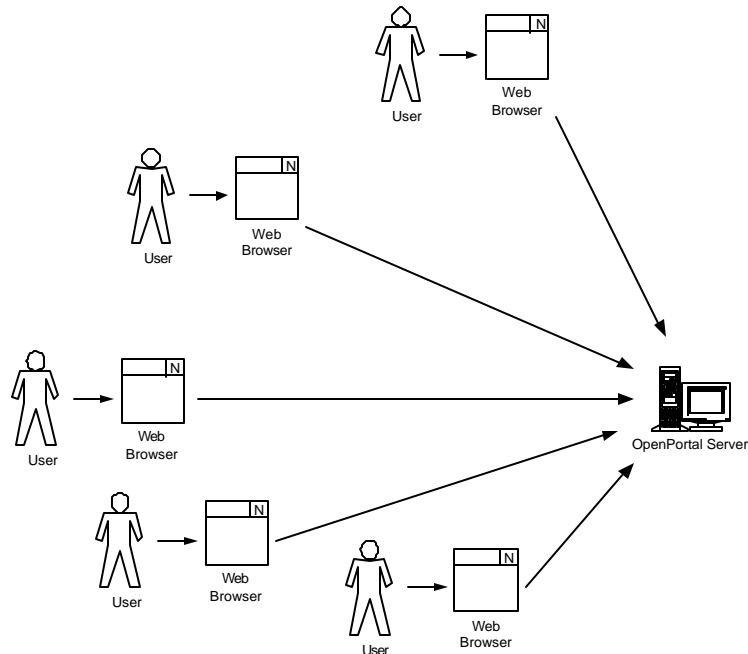
Users consume the website through their browsers – yummy. What happens if a user wishes to somehow become a Programmer, Graphic Designer, Owner, Content Creator, Editor, or Administrator? Sorry, too bad – stay in your place, behind the browser!

In the last two years portals have attempted to change this paradigm a bit with a feature known as *personalization*, which usually boils down to the following three features:

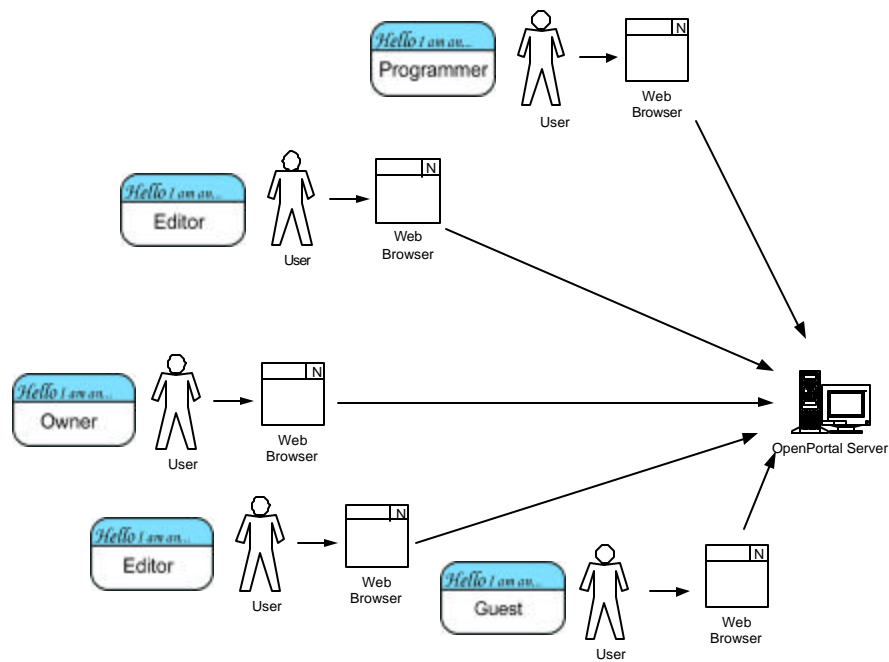
- change your background color!
- personalize things with this predefined cookie cutter!
- here's some generic content from our partners!

Portals are too afraid to truly allow personalization, to truly understand what this means, because it would involve a redefinition of what web sites are. Hence OpenPortal's appearance.

In OpenPortal *everybody* starts out as a User behind the browser – there is no one on the server side!



Then, in various ways, users gain *Nametags* that give them extra privileges to become the roles they wish and desire:



Any user, whether they are in Bangladesh or Bridgeport, can potentially gain any role. This is the power of making everyone a user.

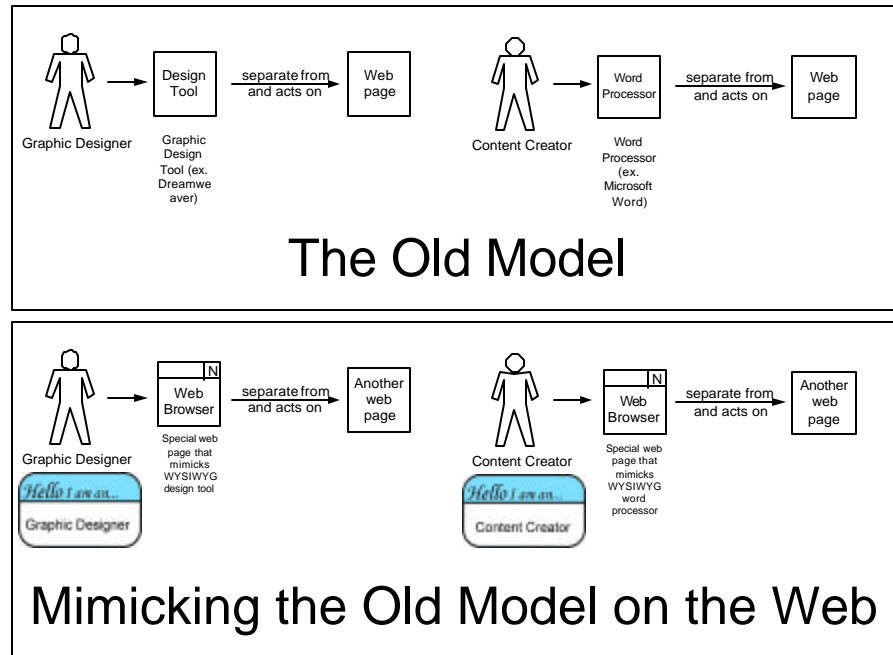


How do these OpenPortal users fulfill their job roles?

2. Everything is changeable and editable.

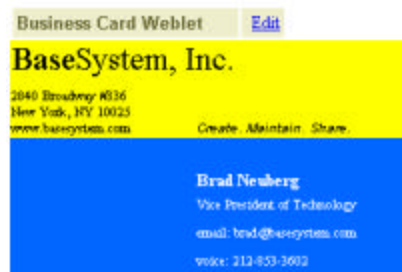
Once you begin to assume that anyone anywhere in the world can potentially lead an OpenPortal website through a browser, you must provide some sort of browser-based tools to help these people.

A first response in providing these tools is to build specialized html web pages, dynamic html web pages, or java applets that act on the web site, just like how in the old model graphic designers and programmers used design tools and compilers separate from a web server to modify a web site:

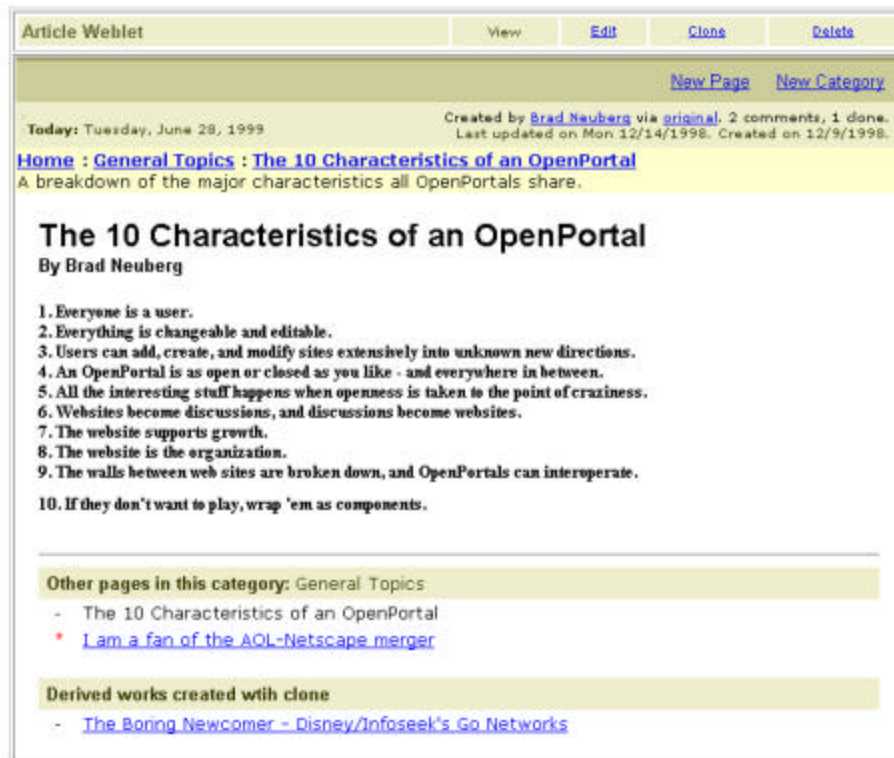


This approach is known as an *application-centric* approach. This is the PC approach – the politically correct Personal Computer.

OpenPortal is different. It takes a nod from technologies like OpenDoc that put the focus on **documents** and not on **applications**. In OpenPortal everything is a document. These documents are known as *weblets* because they live on the web. There are weblets that represent every type of document you can imagine: Business Card weblets, Article weblets, Comment weblets, Toolbar weblets, Poll weblets, and even weblets that represent the User and the Site itself:



Business Card Weblet



Article Weblet



Poll Weblet

Not only does a weblet encapsulate a document type, it also includes *all the tools necessary* to modify and change that weblet. You can imagine the weblet including a miniature editor customized just for that weblet. This is the heart of the document-centric approach – the focus is on documents, not applications. Applications are ancillary to documents and embedded within the documents themselves. For example, the diagram below shows how a Business Card weblet includes a tiny editor inside of it that can be invoked by clicking on an Edit hyperlink:



Note that even while OpenPortal provides a new way to change weblets and web pages through browsers, it does not necessarily foreclose the old way of doing things. We Embrace and Extend those as well ;) OpenPortal caches all web pages, weblets, and templates as flat text files on the server-side, directly mirroring the OpenPortal web-site into the filesystem, so that it can be administered by perl scripts, text editors, graphic design tools, etc. by those lucky-enough to be on the server-side.

3. Users can add, create, and modify sites extensively into unknown new directions.

Remember that OpenPortal is not just about editing specific parts of a website – it is about letting users build a website into entirely new directions unforeseen by the original web-site creator. To support this, webllets need to be more than just editable – they need to be addable, removable, and createable by users across all of an OpenPortal site. *Easy Command Language* (ECL) is the tool that makes this possible. ECL is a simple command-language that allows a user to directly issue commands to an OpenPortal server. ECL not only lets power-users manipulate webllets, but it also provides the tools necessary to build user-interfaces that can manipulate webllets for beginning users. In future versions of OpenPortal ECL will be hidden to everyone but the power-user by more sophisticated dynamic html and dynamic html graphical user-interfaces.

The concept behind ECL is that users tell an OpenPortal server what they would like to do in plain English:

- Edit this webllet
- Display my business card
- Add new article
- Login
- Display all members
- Delete this webllet

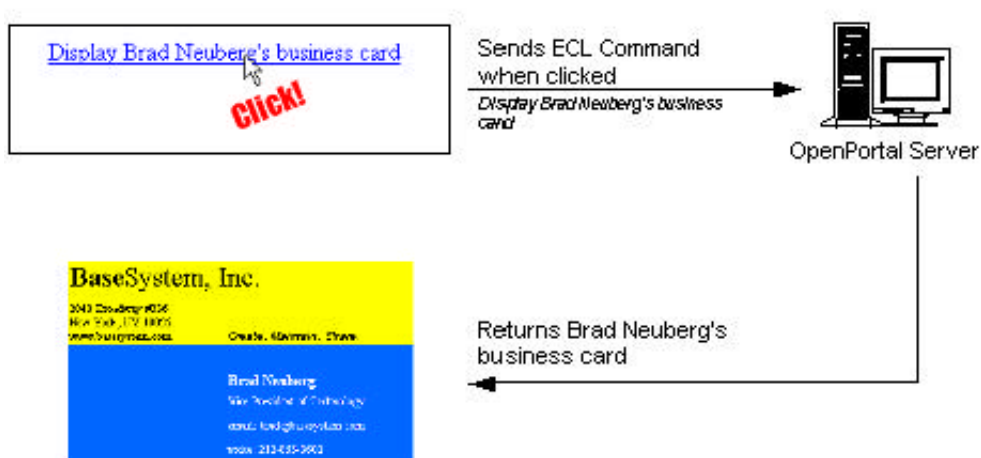
There are two ways in which ECL commands are issued by the user, either by *clicking on ECL hyperlinks* or by *issuing ECL commands in the edit form of a webllet*.

Issuing ECL Commands by Clicking on Hyperlinks

ECL commands are “hidden” behind OpenPortal hyperlinks:

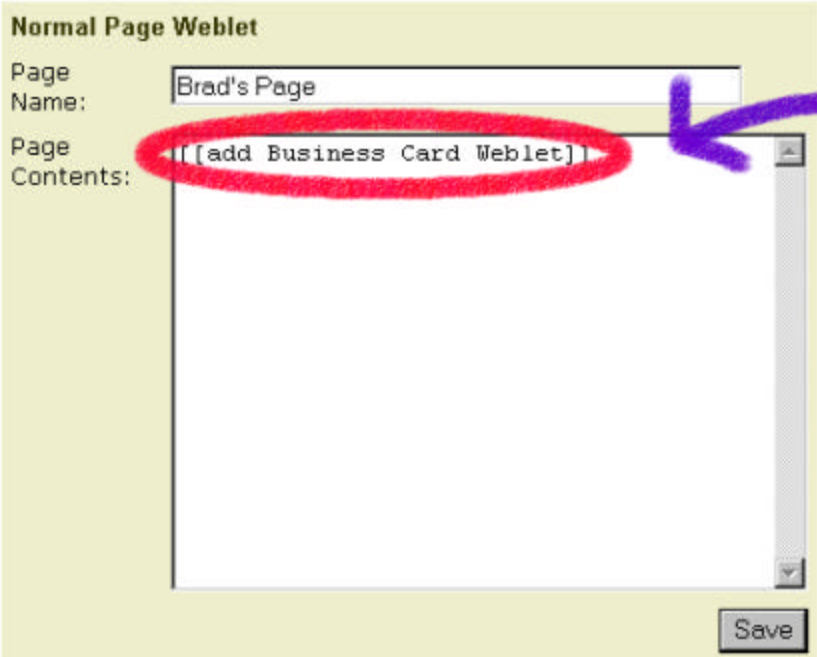
[Display Brad Neuberg's business card](#)

so that when a user clicks on the hyperlink the associated ECL command is sent to the OpenPortal server:



Issuing ECL Commands in the Edit Form of a Weblet

ECL commands can also be entered by the user into *edit* forms by surrounding the ECL command with double brackets:



The screenshot shows a web application interface for editing a weblet. The title is "Normal Page Weblet". There are two main input fields: "Page Name:" with the value "Brad's Page" and "Page Contents:" with the value "[[add Business Card Weblet]]". The "Page Contents:" field is circled in red. A purple arrow points from the text "ECL command surrounded by double brackets" to the red circle. Below the "Page Contents:" field is a large empty text area. At the bottom right is a "Save" button.

Normal Page Weblet

Page Name: Brad's Page

Page Contents: [[add Business Card Weblet]]

Save

ECL command surrounded by double brackets

[[add Business Card Weblet]]

or by surrounding the ECL command with arrows:

Normal Page Weblet

Page Name:

Page Contents:

Save

ECL command surrounded by arrows
-->add Business Card Weblet<--

Double-brackets instructs the server to run an ECL command *right when* the Save button is hit. In the example above:

Normal Page Weblet

Page Name:

Page Contents:

Save

when the user clicks the Save button, the *add Business Card Weblet* ECL command will be run by the OpenPortal server and the results of the command will be embedded in the web page, which in this case is a new business card weblet:

Normal Page Weblet

View

Edit

Clone

Delete

[New Page](#) [New Category](#) [New](#) Business Card [Weblet](#)

Sample Page

Today: Tuesday, June 28, 1999

Created by [Brad Neuberg](#), no clones.
Last updated on Mon 12/14/1998. Created on 12/9/1998.

Business Card Weblet

Edit

BaseSystem, Inc.

2840 Broadway #536
New York, NY 10025
[www.basesystem.com](#) *Create. Maintain. Share.*

Brad Neuberg

Vice President of Technology
email: [brad@basesystem.com](#)
voice: 212-853-3602

Derived works created with clone

Surrounding the ECL command with arrows instructs OpenPortal to automatically hyperlink this command when the web page is returned and to run the ECL command *when the user clicks on it*. In the example from above:

Normal Page Weblet

Page Name:

Page Contents:

-->add Business Card Weblet<--

when the user clicks on the Save button, OpenPortal does not execute the ECL command but automagically hyperlinks it instead:

Normal Page Weblet	View	Edit	Clone	Delete
New Page		New Category	New	Business Card <input type="button" value="v"/> Weblet
Sample Page				
Today: Tuesday, June 28, 1999		Created by Brad Neuberg , no clones. Last updated on Mon 12/14/1998. Created on 12/9/1998.		
add Business Card Weblet				
Derived works created with clone				

when the user clicks on the hyperlink *then* the ECL command is run:

Normal Page Weblet	View	Edit	Clone	Delete
--------------------	------	----------------------	-----------------------	------------------------

New Page New Category New	Business Card ▾	Weblet
---	-----------------	------------------------

Sample Page

Today: Tuesday, June 28, 1999

 Created by [Brad Neuberg](#), no clones.
 Last updated on Mon 12/14/1998. Created on 12/9/1998.

[add Business Card Weblet](#)

Business Card Weblet	Edit
----------------------	----------------------

BaseSystem, Inc.

2840 Broadway #636
 New York, NY 10025
www.basesystem.com
Create. Maintain. Share.

Brad Neuberg
 Vice President of Technology
 email: brad@basesystem.com
 voice: 212-853-3602

Derived works created with clone

With these two forms of ECL users can build an entire user interface:

Normal Page Weblet

Page Name:

Page Contents:

My Business Contacts:
-->Add Business Card Here<--

My To-Do Lists:
-->Add To-Do Weblet Here<--

When saved this page looks as follows:

Normal Page Weblet	View	Edit	Clone	Delete
New Page New Category New <input type="text" value="Business Card"/> Weblet				
Sample Page				
Today: Tuesday, June 28, 1999		Created by Brad Neuberg , no clones. Last updated on Mon 12/14/1998. Created on 12/9/1998.		
My Business Contacts: Add Business Card Here				
My To-Do Lists: Add To-Do Weblet Here				
Derived works created with clone				

When these links are clicked on *then* the ECL command is performed. For example, if the user clicks on the link *Add Business Card Weblet Here*, a new business card weblet is added:

Normal Page Weblet

View

Edit

Clone

Delete

Sample Page

Today: Tuesday, June 28, 1999

Created by [Brad Neuberg](#), no clones.
Last updated on Mon 12/14/1998. Created on 12/9/1998.

My Business Contacts:

[Add Business Card Here](#)

Business Card Weblet

Edit

BaseSystem, Inc.

2840 Broadway #B36
New York, NY 10025
www.basesystem.com

Create. Maintain. Share.

Brad Neuberg

Vice President of Technology
email: brad@basesystem.com
voice: 212-853-3602

My To-Do Lists:

[Add To-Do Weblet Here](#)

Derived works created with clone

It should be mentioned at this point that weblets don't play alone – they are usually grouped together in a *weblet container*. A weblet container is just a weblet that supports adding, removing, and creating new weblets inside of it. A good example of a weblet container is the Normal Page Weblet Container. This is a weblet that can hold other weblets and displays them on a single web page:

Normal Page Weblet

View

[Edit](#)

[Clone](#)

[Delete](#)

These are the weblets on this page:

Business Card Weblet

[Edit](#)

BaseSystem, Inc.

2840 Broadway #336
New York, NY 10025
www.basesystem.com

Create. Maintain. Share.

Brad Neuberg

Vice President of Technology

email: brad@basesystem.com

voice: 212-853-3602

Business Card Weblet

[Edit](#)

The OpenPortal Project

www.openportal.org

*Where websites become
discussions, and discussions
become websites.*

Paolo de Dios

System Architect

email: paolo@columbia.edu

voice: 212-555-5555

Poll Weblet

[Edit](#)

My favorite weblet is the



Business Card Weblet



Poll Weblet



Toolbar Weblet



Email Weblet



Groups Weblet

Vote

[[Results](#) | [Polls](#)]

Comments: **217** | Votes: **18274**

Each of the weblets in the Normal Page Weblet Container can still be edited:

These are the weblets on this page:

Business Card Weblet

Name:

Organization:

Address:

Slogan:

Role:

Email:

Phone- Number:

Save

Business Card Weblet

[Edit](#)

The OpenPortal Project

www.openportal.org

*Where websites become
discussions, and discussions
become websites.*

	Paolo de Dios
	System Architect
	email: paolo@columbia.edu
	voice: 212-555-5555

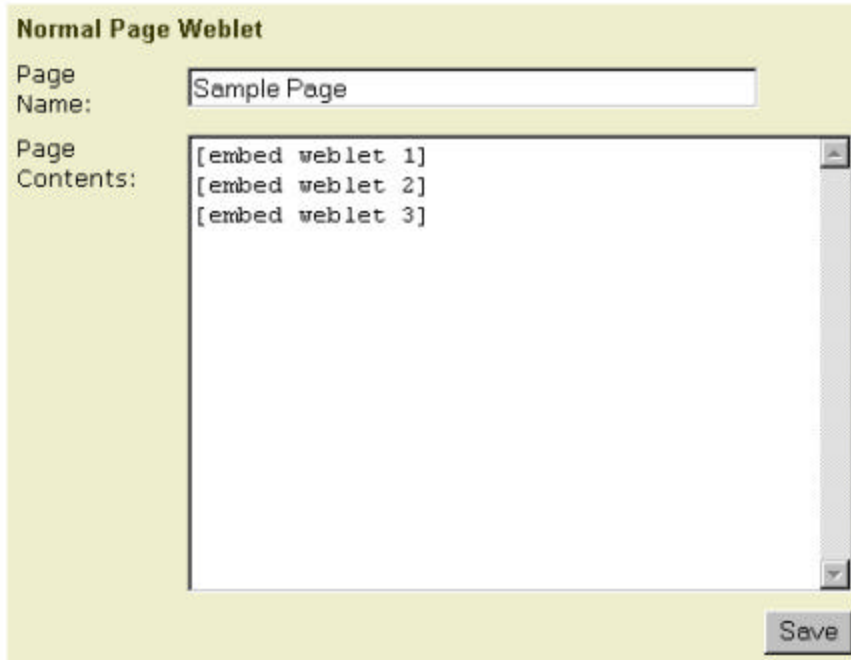
Poll Weblet

[Edit](#)

My favorite weblet is the

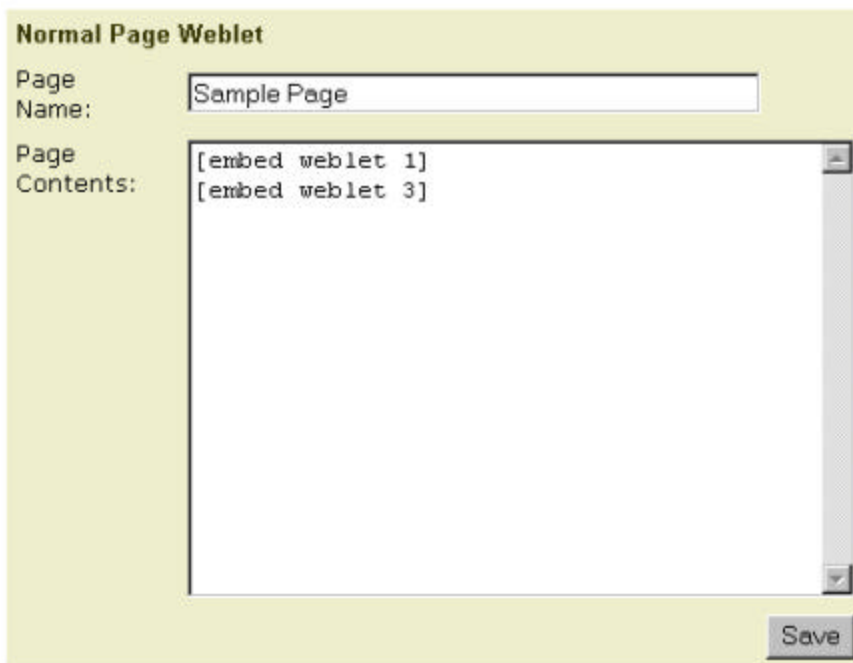
- ☐ Business Card Weblet
- ☐ Poll Weblet
- ☐ Toolbar Weblet
- ☐ Email Weblet
- ☐ Groups Weblet

What happens if you hit the *edit* Easy Command Language hyperlink for the Normal Page Weblet? You would get the following:



The screenshot shows a window titled "Normal Page Weblet". It has a "Page Name:" label followed by a text box containing "Sample Page". Below this, there is a "Page Contents:" label followed by a large text area. The text area contains three lines of code: `[embed weblet 1]`, `[embed weblet 2]`, and `[embed weblet 3]`. A vertical scrollbar is on the right side of the text area. At the bottom right of the window is a "Save" button.

Each of the *embed* statements refers to each of the weblet's embedded in the weblet container. If you remove one of the embed statements:



This screenshot is similar to the previous one, showing the "Normal Page Weblet" configuration window. The "Page Name:" text box still contains "Sample Page". However, the "Page Contents:" text area now only contains two lines of code: `[embed weblet 1]` and `[embed weblet 3]`. The `[embed weblet 2]` line has been removed. The "Save" button remains at the bottom right.

and hit save, then the corresponding embedded weblet will be removed from the weblet container:

Normal Page Weblet

View

Edit

Clone

Delete

New Page

New Category

New

Weblet

Sample Page

Today: Tuesday, June 28, 1999

Created by [Brad Neuberg](#). no clones.
Last updated on Mon 12/14/1998. Created on 12/9/1998.

These are the weblets on this page:

Business Card Weblet

Edit

BaseSystem, Inc.

2840 Broadway #336
New York, NY 10025
www.basesystem.com

Create. Maintain. Share.

Brad Neuberg

Vice President of Technology

email: brad@basesystem.com

voice: 212-853-3602

Poll Weblet

Edit

My favorite weblet is the

☐

Business Card Weblet

☐

Poll Weblet

☐

Toolbar Weblet

☐

Email Weblet

☐

Groups Weblet

Vote

[[Results](#) | [Polls](#)]

Comments:217 | Votes:18274

The ECL *embed* command can also be used to embed weblets that have a name that are on different pages on the OpenPortal site. Some weblets can have a name, which is usually in the title bar of the weblet:

The screenshot shows a web portal interface. At the top, there's a header bar with the text "Normal Page Weblet" and buttons for "View", "Edit", "Clone", and "Delete". Below this, there's a navigation bar with links for "New Category", "New", and a dropdown menu showing "Business Card". A red circle highlights the text "Sample Page" in the main content area. Below this, there's a section titled "Brad Neuberg's Business Card" with an "Edit" button. A red circle highlights this title. Below the business card, there's a section titled "Derived works created with clone".

Sample Page
Weblet Name

**Brad Neuberg's
Business Card**
Weblet Name

This weblet name is usually set through the edit form. A weblet does not necessarily need to have a name; it will usually be given a default one if none is given. Using this weblet name one can embed weblets from all over the site on any OpenPortal page by using the *embed* command. As will be explained in section 4, "An OpenPortal is as open or closed as you like – and everywhere in between," all OpenPortal pages live in Sites which can have Areas. An OpenPortal can have multiple Sites, such as the "Linux Site", the "Windows 2000 Site", etc., and in each Site there can be multiple Areas, such as "/Linux Site/Main Area", "/Linux Site/News Area", "/Windows 2000 Site/Main Area", etc. When using the embed command, the full name of the weblet to be embedded must be given. For example, let's say that on the "Linux Site" there is an Area named "Repository Area". In this Repository Area could be a weblet named "Standard Toolbar":

Area Weblet

View

Edit

Clone

Delete

[New Page](#) [New Category](#) [New](#)

Business Card

[Weblet](#)

Linux Site : Repository Area

Today: Tuesday, June 28, 1999

Created by [Brad Neuberg](#), no clones.
Last updated on Mon 12/14/1998. Created on 12/9/1998.

Standard Toolbar - Toolbar Weblet

Edit

[Main Area](#)
[Today's News](#)
[Linux Commentary](#)

Derived works created with clone

From anywhere in the Linux Site this Standard Toolbar could be embedded onto any OpenPortal page by using the *embed* command:

Normal Page Weblet

Page Name:

Page Contents:

```
[embed "/Linux Site/Repository  
Area/Standard Toolbar"]  
  
Welcome to this page!
```

The *embed* command is considered the default ECL command, so in the text field above the word *embed* doesn't even have to be entered:

```
[ "/Linux Site/Repository Area/Standard Toolbar" ]
```

When the save button is hit the Standard Toolbar is embedded in the page:



The *embed* command is very useful for embedding items that are used throughout an OpenPortal site, such as the Standard Toolbar from the example above, or for referring to items when typing in a weblet, such as referring to your business card or a separate discussion that has occurred.

Automatic hyperlinks to named weblets can also be created using the ECL arrows --> and <---. One simply surrounds the weblet name with these arrows and OpenPortal will automatically create a hyperlink to this weblet:

Normal Page Weblet

Page Name:

Page Contents:

```
-->"/Linux Site/Repository Area/Standard  
Toolbar"<--
```

When the save button is hit the following is returned:

Normal Page Weblet	View	Edit	Clone	Delete
--------------------	------	----------------------	-----------------------	------------------------

[New Page](#)
[New Category](#)
[New](#)

Business Card ▾

[Weblet](#)

Sample Page

Today: Tuesday, June 28, 1999

Created by [Brad Neuberg](#), no clones.
 Last updated on Mon 12/14/1998. Created on 12/9/1998.

[/Linux Site/Repository Area/Standard Toolbar](#)

Derived works created with clone

If a weblet is referenced that does not exist, the tiny words [create this](#) are added and hyperlinked to the end of the unknown weblet:

Normal Page Weblet	View	Edit	Clone	Delete
New Page New Category New Business Card Weblet				
Sample Page				
Today: Tuesday, June 28, 1999		Created by Brad Neuberg , no clones. Last updated on Mon 12/14/1998. Created on 12/9/1998.		
Make sure to see Some Unknown Weblet create this				
Derived works created with clone				

When the [create this](#) hyperlink is clicked on the user is taken to a page that allows them to pick out what kind of weblet to make:

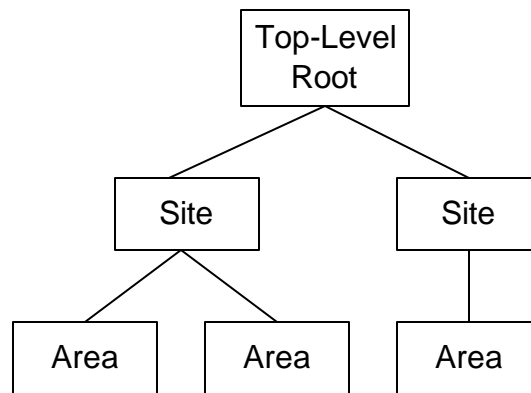
Once this weblet has been defined, the original page will display the link without a [create this](#) hyperlink:

Normal Page Weblet	View	Edit	Clone	Delete
New Page New Category New Business Card <input type="button" value="Weblet"/>				
Sample Page				
Today: Tuesday, June 28, 1999		Created by Brad Neuberg , no done's. Last updated on Mon 12/14/1998. Created on 12/9/1998.		
Make sure to see Some Unknown Weblet				
Derived works created with clone				

4. An OpenPortal is as open or closed as you like - & everywhere in between.

It is an extremely powerful notion to allow users to completely reconfigure and extend a site. With this ability comes the fear that any redefinition of power can cause. OpenPortal does not force you to have an open site and to make everything changeable – though it certainly encourages you to. Instead OpenPortal has an extensive permissioning system based on the English abilities of Easy Command Language, which allows people to run an OpenPortal as open or as closed as they wish.

OpenPortal allows users to create Sites. One OpenPortal server can have several Sites, all below a top-level root. Each Site can also have multiple Areas beneath it.



Each Site and Area can define general policies on what kind of Easy Command Language a user can execute when within them. Within each Site Users can have Roles, such as Editor, Owner, Member, and Guest. Within each Area and Site a user's Roles can be used to either restrict or enable ECL commands.

<u>Site</u>	<u>Area</u>	<u>Role/User</u>	<u>ECL Command</u>
Linux Site	Main Area	Owner	Can edit all

The table above shows how one can restrict or allow ECL commands based on Roles, Users, Areas, and Sites. In this example any user who has the Role of Owner in the Main Area of the Linux Site can edit everything. The next table shows more examples of restricting commands based on roles and users. In the first line the user Paolo de Dios is given permission to edit everything ("Can edit all") in the Discussion Area. In the second line a default security setting is set for everyone ("Default for Everyone") so that everyone cannot edit anything. Permissions are enforced in the order they are given, so that permissions higher in the table below are enforced and can over-ride lower permissions.

<u>Site</u>	<u>Area</u>	<u>Role/User</u>	<u>ECL Command</u>
Linux Site	Discussion Area	Paolo de Dios	Can delete all
Linux Site	Main Area	Default for Everyone	Cannot edit all

Sites and Areas can hold other weblet containers, but cannot hold nested Sites or Areas. They can also set policies on whether children Areas, weblet containers, and weblets can over-ride the security settings of their parents. For example, in the table below anyone who has the role of being Owner in every Area in the Linux Site can change children permissions of sub-Areas or weblet containers, while the Default for Everyone is set so that the everyone cannot set Area permissions but can set a weblet container's permissions in the Discussion Area.

<u>Site</u>	<u>Area</u>	<u>Role/User</u>	<u>ECL Command</u>
Linux Site	All	Owner	Can set area permissions
Linux Site	All	Default for Every one	Cannot set area permissions
Linux Site	Discussion Area	Default for Everyone	Can set weblet container permissions

Weblets and weblet containers can also have their own security policies attached to themselves.

A web-based user interface is used to set these policies for each Area, Site, weblet, or weblet container. They all have the same general form, an example is shown below for setting the properties of a Site named Linux Site:

The screenshot displays two web-based user interfaces for configuring site settings. The top section, titled "Linux Site Security", features a form with three dropdown menus: "In" (set to "this site"), "Everyone" (set to "Everyone"), and "can" (set to "can"). To the right, a list of permissions is shown, with "set security for area" selected. Below this is an "Add" button and a text field labeled "Other:" containing "Alternative ECL Command". A section titled "Security settings for this site in order of power:" contains a list of four items, with "In this site Editors can edit all weblets" highlighted. At the bottom of this section are "Remove", "Save", and "Change" buttons. The bottom section, titled "Linux Site User Roles", shows a dropdown menu for "By default, all users are" set to "Owners". Below this, a list of roles is shown, with "Brad Neuberg is an Owner" selected. At the bottom of this section are "Remove", "Save", "Add", and "Change" buttons. A final dropdown menu shows "Brad Neuberg" is an "Owner".

Linux Site Security

In:
 Other:

Security settings for this site in order of power:

Linux Site User Roles

By default, all users are:

Roles:

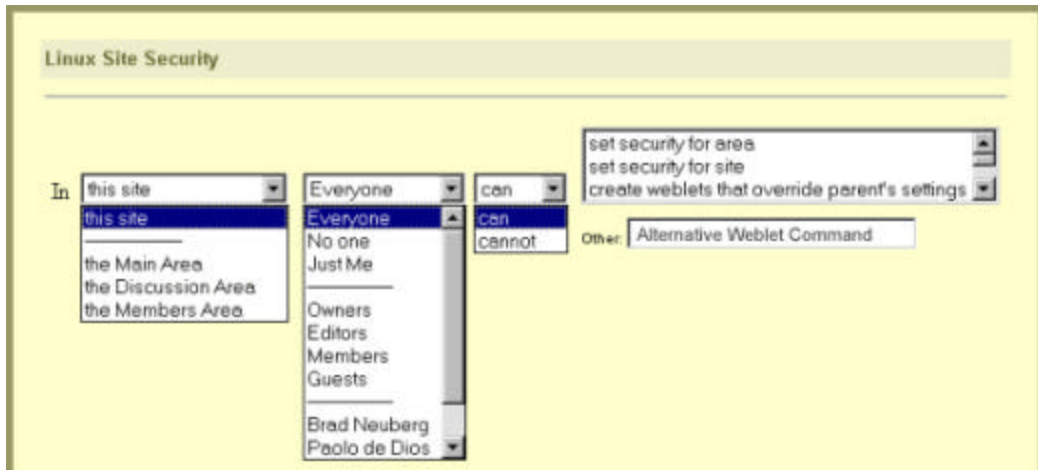
is an

The user interface has two sections; a top section in which security settings are set by creating the appropriate phrase from pull-down menus, and a bottom settings where all the security settings for the site are listed. There is also a bottom section for assigning users different Roles for Sites. For example, in the screenshot above the top section has the following security phrase spelled out from the pull-down menus:

In this site Everyone can set security for area

These new phrases can be added to the site by hitting the "Add" button, and the new phrase will be added after whatever phrase was highlighted in the lower section.

The pull downs for the top-down section are as follows:



In the upper right portion of the user interface is a scrolling list that has all possible ECL commands enumerated (i.e. "set security for area", "set security for site", etc.). The ECL commands which can have security set on them are as follows, with descriptions where appropriate.

- set security for area
- set security for site

These two commands give someone permission to set the security properties for an area or a site. If someone is allowed to set the security, a form similar to the ones above is returned.
- create weblots that override parent's settings

This gives a user permission to create a weblot that can override the weblot's parent security, possibly allowing more permissive or restrictive use of the weblot then the parent would provide. For example, using this setting would allow someone to create an editable Article weblot in an area where nothing can be edited.
- assign all roles
- assign Owner role
- assign Editor role
- assign Member role
- assign Guest role

These five commands gives a user the power to assign roles to other users in an area or a site. For example, a user could be given the power to assign the Member role to a new user.
- other (fill in command in box below)

This selection is used for typing in ECL commands that have not been enumerated. This is commonly used for setting ECL commands on individual weblots (i.e. edit "My Business Card").
- do everything with all weblots

This gives a user free reign over all weblots in an area, though this does not give them permission to change a site or area's security settings or to assign roles.
- edit all weblots
- view all weblots
- delete all weblots
- clone all weblots

These ECL commands give a user permission to run edit, view, delete, or clone commands on any weblot.
- set security for all weblots

This allows a user to change the security properties of a weblet; note that this does not include the ability to change the security permissions of the area or site.

- create all weblets
- create Business Card Weblet
- create Normal Page Weblet
- create Article Weblet
- create Toolbar Weblet

Every available weblet is enumerated and a 'create' option is put into the list. This allows one to restrict the creation of certain types of weblets to certain users.

- do everything with all Business Card Weblets
- edit all Business Card Weblets
- view all Business Card Weblets
- delete all Business Card Weblets
- clone all Business Card Weblets
- move all Business Card Weblets
- set security for all Business Card Weblets
- do everything with all Normal Page Weblets
- edit all Normal Page Weblets
- view all Normal Page Weblets
- delete all Normal Page Weblets
- clone all Normal Page Weblets
- move all Normal Page Weblets
- set security for all Normal Page Weblets

For each type of weblet all possible commands that can be run on this weblet is enumerated.

Above are two example enumerations for Business Card Weblets and Normal Page Weblets.

The bottom portion of the security form shows all the security settings for the site. Three buttons can be used to manipulate these: 'Remove', 'Save', 'Add', and 'Change'. Hitting Remove removes a highlighted security setting from both the client and server. Hitting Save saves a modified ECL command and all modifications. Hitting Add causes the ECL command that has been specified in the top-portion of the user interface to be inserted into the bottom portion. Hitting Change loads the selected ECL command into the top-portion.

There is also a bottom section for assigning users different Roles for Sites. The form to do this is located at the bottom of the Site form above. The default role for all users can be set with this form. Roles are listed in a list-box, and can be Removed, Saved, Added, and Changed by clicking on the appropriate buttons and selecting from the lower pull-downs (i.e. "Brad Neuberg" is an "Owner").

The form for setting an area's security policies looks similar to the site security form:

The screenshot shows the 'Discussion Area Security' form. At the top, there's a title bar. Below it, the form is divided into several sections. The first section has a label 'In the Discussion Area' followed by a dropdown menu set to 'Everyone', a 'can' dropdown, and a large text box containing three ECL commands: 'set security for area', 'create weblets that override parent's settings', and 'other (fill in command in box below)'. To the right of this is an 'Add' button. Below the 'Add' button is an 'Other:' label followed by a text box containing 'Alternative ECL Command'. The second section is titled 'Security settings for this area in order of power:' and contains a list box with two entries: 'In the Discussion Area Editors can clone all weblets' and 'In the Discussion Area Everyone can view all weblets'. At the bottom of this section are three buttons: 'Remove', 'Save', and 'Change'.

All that is different is that the Area is already restricted and the ECL command 'set security for site' is removed upper right box. Also, Areas cannot have their own assigned roles; roles are only assigned from Sites.

Weblet Containers also have their own security properties form:

The screenshot shows the 'Weblet Container Security' form. It has a similar layout to the Discussion Area Security form. The first section has a label 'In this container' followed by a dropdown menu set to 'Everyone', a 'can' dropdown, and a large text box containing two ECL commands: 'create weblets that override this container's settings' and 'other (fill in command in box below)'. To the right of this is an 'Add' button. Below the 'Add' button is an 'Other:' label followed by a text box containing 'Alternative ECL Command'. The second section is titled 'Security settings for this area in order of power:' and contains a list box with two entries: 'In this weblet container Editors can clone all weblets' and 'In this weblet container Everyone can view all weblets'. At the bottom of this section are three buttons: 'Remove', 'Save', and 'Change'.

The upper-right ECL command box includes all the same commands as the Site box, without the 'set security for area' and 'set security for site' commands.

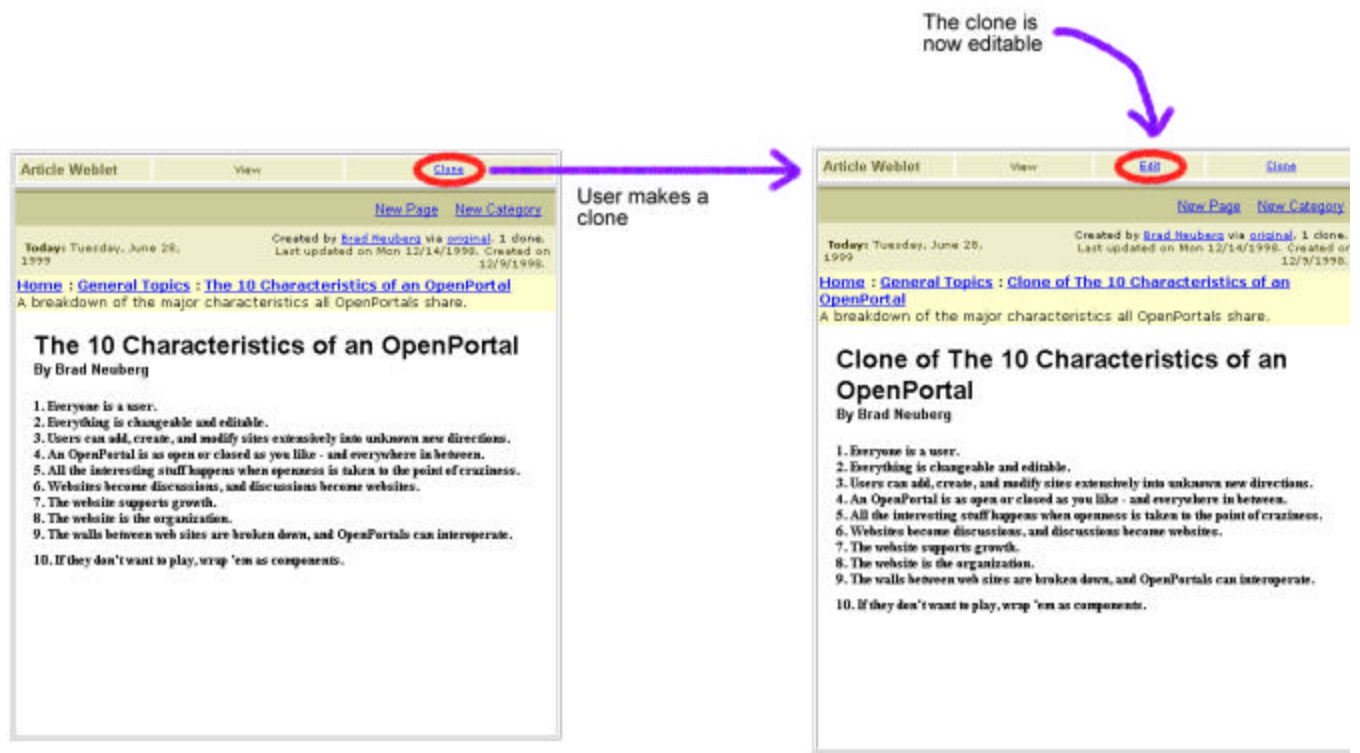
Weblets have the simplest security settings:

The screenshot shows a window titled "Business Card Weblet Security". Inside, there are three dropdown menus: "In the Discussion Area" set to "Everyone", "can" set to "can", and a third menu with options "do everything to this weblet", "edit this weblet", and "view this weblet". Below these is an "Add" button. A text label reads "Security settings for this weblet in order of power:". Underneath is a scrollable list box containing the text "In the Discussion Area Editors can clone this weblet". At the bottom are "Remove", "Save", and "Change" buttons.

The upper-right ECL command box has the following commands:

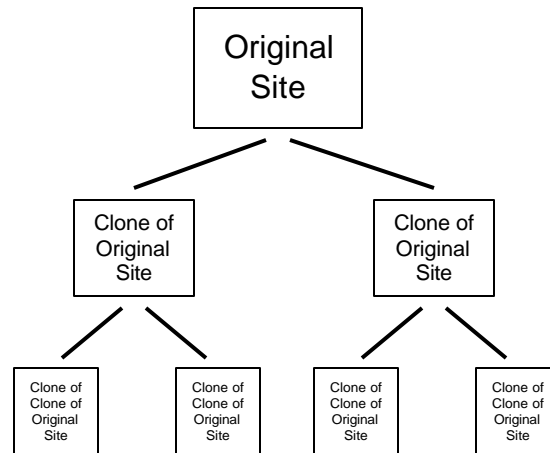
```
do everything to this weblet
edit this weblet
view this weblet
delete this weblet
clone this weblet
set security for this weblet
```

An interesting feature that balances being too open with being too closed is the *clone* feature that some weblets support. It is sometimes desirable to have some weblets be uneditable, such as a research paper that has been posted on an OpenPortal. However, it might also be useful to allow others to *clone* this paper and then let them make changes only on the clone, but not on the original:



5. All the interesting stuff happens when openness is taken to the point of craziness.

While OpenPortal can be used to build old, boring closed sites, all the fun starts to happen when you push and edge out into crazy openness. This is when the proverbial sh*t hits the fan. Wouldn't it be interesting to make a closed site using OpenPortal but allow anyone to *clone* that website and change it to make their own? Imagine the interesting results that would occur as the mutations of your website went from one clone, to two clones, to four clones, to sixteen, growing exponentially and changing each time?



Some of these clones would be worthless, but wouldn't one of them be a gem that takes your website in a brilliant direction unforeseen by you? Isn't this the open-source idea – applied to websites?

In future versions of OpenPortal not only will the website be changeable, but users themselves will be able to create entirely new weblets, extending OpenPortal in directions unforeseen to the OpenPortal team. OpenPortal is itself built with weblets: because of this, OpenPortal is a system that supports the evolution of its own process of evolution – the entire foundation of OpenPortal itself, not just its content, will be changeable through OpenPortal itself!

Some shouts from the audience:

“It will collapse if you make it that open...”

“But you can't allow such openness!”

“No one needs that level of openness anyway...”

“Every other company and person will destroy you if you're that open!”

Weren't these the original criticisms railed against the Internet, and isn't it damn more interesting that its closest competition, the closed monolithic telecom network? The same criticisms were shot at the World Wide Web, which is vastly more interesting than the closed Information Superhighway peddled by the cable companies.

OpenPortal is about building a massively open system that can handle and tolerate its own rapid evolution and change. While you can build a traditional closed system with OpenPortal, the more you experiment with crazy openness on your own OpenPortal the more interesting your site will become.

6. Websites become discussions and discussions become websites.

When users have the ability to create entirely new pieces of a website, the website itself simply becomes one giant bubbling conversation. But on OpenPortal chaos does not ensue because weblets provide just enough order to keep things structured. If editing an OpenPortal page were more like writing with a WYSIWYG (What-You-See-Is-What-You-Get) editor like Microsoft Word, then there would be absolutely no structure to keep things ordered. With a few phrases of ECL users can create entirely new site weblets, article weblets, comment weblets, etc. in response to other site weblets, article weblets, comment weblets, etc. What once used to be static pages can now branch out into entirely new sections created by users. The Main Page can be filled with Comment weblets, or any type of weblet.

"But don't web-based threaded discussion boards already provide a place for discussions?"

Yes, if you only care about having flat discussions that have no more structure than a hierarchy. OpenPortal allows you to *leap out* of a thread-discussion boards structure into entirely new directions. Free yourselves from the shackles of threaded-discussion boards! Imagine being able to respond to a comment *with an entirely new OpenPortal Site, made while your typing the response* **It's not about WYSIWYG** – it's about discussion, but at a higher-level than simple threaded discussion boards can handle.

7. The website supports its own growth.

An OpenPortal allows the creation of new Sites and Areas within it. From any edit form a user can enter the ECL command *create Site Weblet* or *create Area Weblet*:



The screenshot shows a web form titled "Some Weblet" with a light yellow background. It contains two input fields: "Weblet Name:" with the text "Some Name" and "Some Property:" with the text "[[create Site Weblet]]". A "Save" button is located at the bottom right of the form.

Unlike other weblets, the Site or Area weblet will not be embedded in the page you type the command in – this is why it can be typed from anywhere in an OpenPortal. When the user hits *save* a form will come back to create the new site:

Site Weblet [Set security settings](#)

Site Name:

Site Description:

Site Areas :

Page Contents :

The user can click the *Create Area* button to create a new Area in this site:

Area Weblet [Set security settings](#)

In Site:

Area Name:

Site Description:

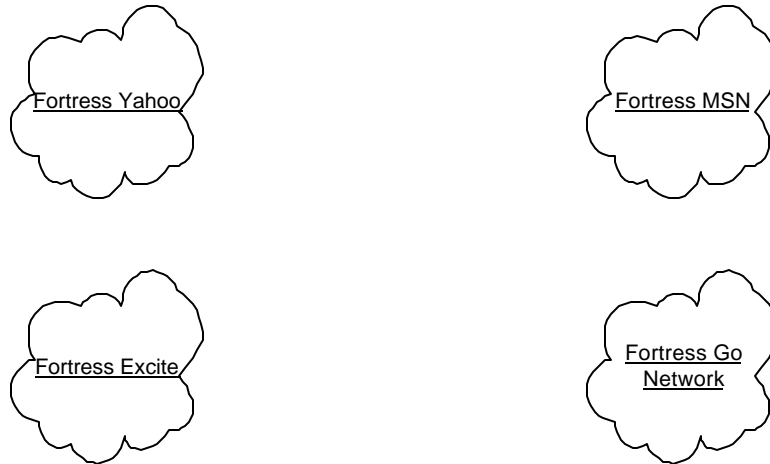
Page Contents :

The user can now begin populating this area with new weblets.

The user that creates a new Site instantly become the Owner of that Site and can set permissions and give out roles. The original creator is therefore free to make the Site as open or as closed as he wishes.

8. The walls between web sites are broken down, and OpenPortals can interoperate.

Let's examine the state of the web today. Major websites and portals sit like monolithic cathedrals on the web landscape:

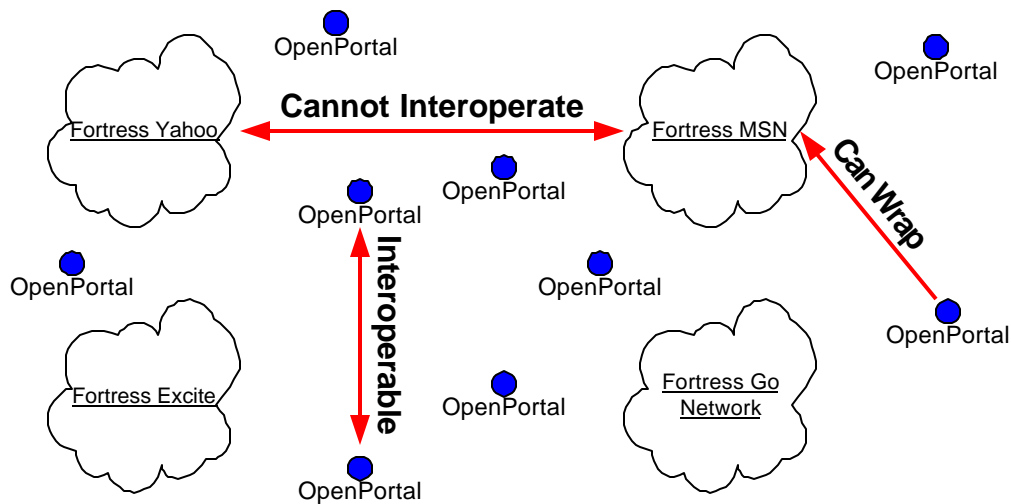


Each one of these cathedrals wants to be the cathedral, sucking in and controlling everyone else – none of them would ever dare to have their fortresses interoperate, except through corporate mergers!

Enter OpenPortal. In a future OpenPortal release weblots will become mobile weblots, able to move between OpenPortals. OpenPortals will be able to form networks with each other. This entire OpenPortal network will be open, just like the Internet and the World Wide Web. Using mobile weblots OpenPortals will be able to support the following between them:

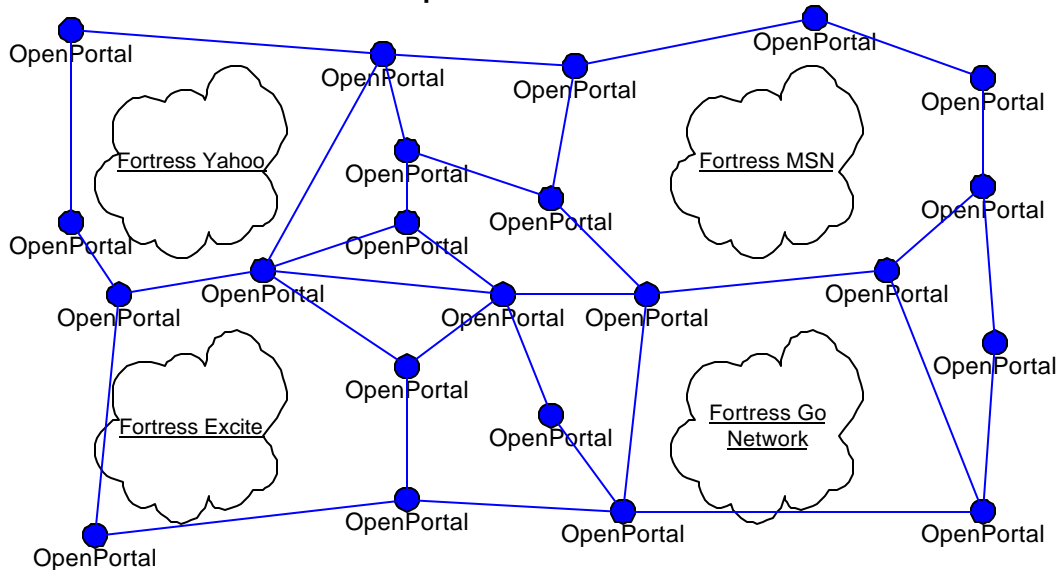
- Weblots from one site can be automatically embedded and linked to using Easy Command Language
- Every user will get a clipboard – using dynamic html they can drag any weblot onto this clipboard from one OpenPortal site and paste it onto another. In the background the two OpenPortals are exchanging the mobile weblot.
- Subscribing to a weblot amounts to simply copying and pasting a weblot from a remote OpenPortal. A link is retained to the old remote weblot, so that whenever the old weblot changes the new "pasted" weblot changes as well.
- Compound documents of weblots can be created, with some of the weblots actually being from other OpenPortal sites and being updated whenever the original changes
- A universal log-in network can be created across OpenPortals for higher-level user-services
- Users can "carry" their web-sites around with them from OpenPortal to OpenPortal, as if it were in their back pockets.
- Many other exciting features

It will be exciting when OpenPortals begin to interoperate in future versions. Each of these OpenPortals will start as tiny rain drops on the internet, insignificant when compared to the huge puddles that are the major portals and major web sites; however, these OpenPortals will actually work together:



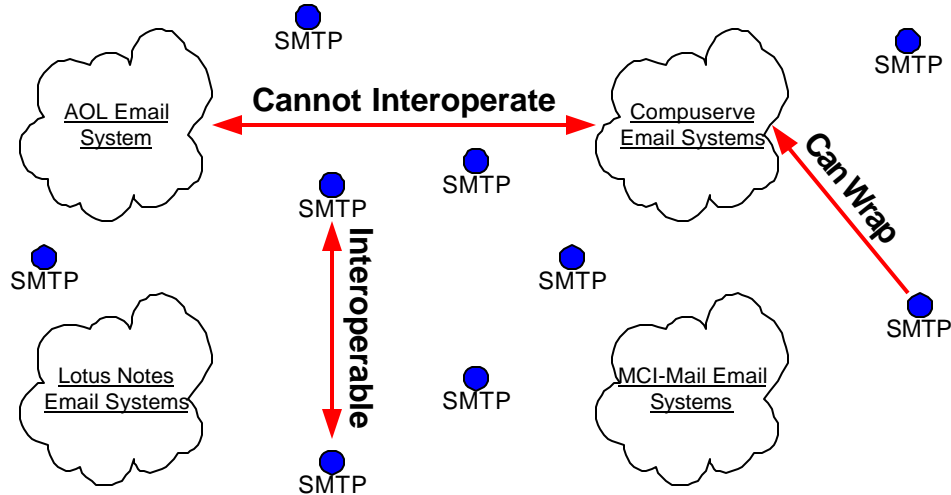
But when a thousand rain-drops begin to merge together, forming larger and larger puddles, the portals and large sites will have to listen: web-sites are not **cathedrals**, they are a **bazaar**. In the beginning we were nothing but a few rain-drops, but when a few rain-drops coalesce they suck in all the puddles:

The OpenPortal Network



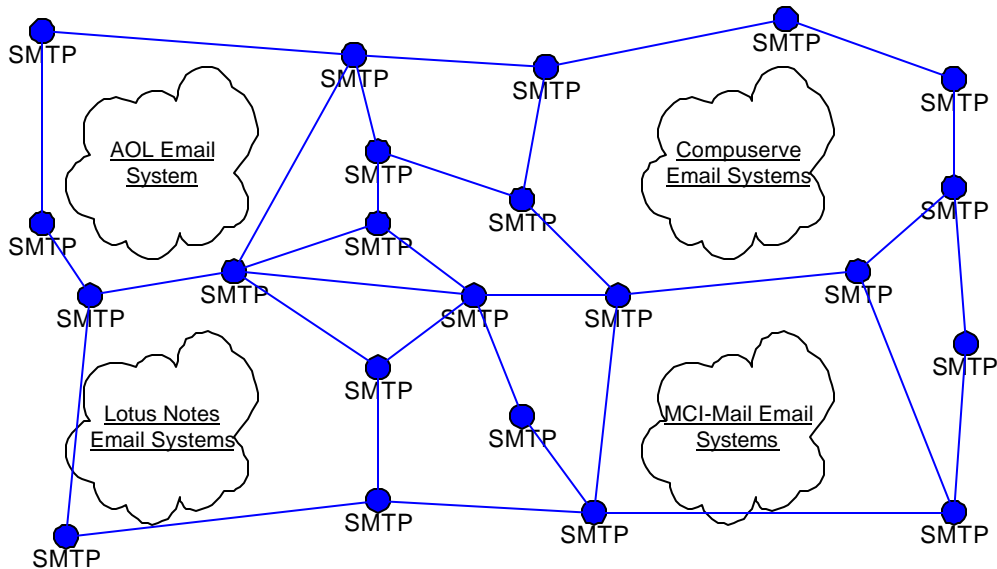
This phenomenon has occurred in the past for many of the major internet technologies: internet email, the World Wide Web, etc. The diagram below illustrates how in 1990 most of the major online services and software packages that supported email, such as Compuserve, AOL, Lotus Notes email systems, and others, could barely support interoperability of email systems between competing services, if they even tried. Around this time SMTP internet email servers began to proliferate; they were tiny and located mostly in a university setting:

Email Systems About ~1990



Before anyone knew what happened these tiny servers had surrounded most of the major online services, conglomerating themselves into one giant network that actually supported interoperability between themselves *and* their competition:

The SMTP Network



These diagrams illustrate what OpenPortal *really* is: a light-weight component-level standard agreed upon by websites, called weblets, which are controlled by the users through an OpenPortal.

It is only through the openness of the bottom-up process that we can build an interoperable web controlled and built by its users. OpenPortal is the mp3 of web-site technology. Hopefully if you can -->change the technology<--, you can -->change the rules<--.

9. If they don't want to play, wrap 'em as components.

We can't expect the big boys to play nice when OpenPortal comes along, so OpenPortal includes a controversial technology: the ability to wrap portions of other, non-OpenPortal websites as weblets. For example, the weblet below wraps a portion of the Excite Communities website as a weblet:

Excite Communities Weblet		Edit
Start new group Invite new members Add Excite Communities Event Login to Excite Communities	Communities Name:	Some Groups Name
	Communities Address:	somegroup@excite.com
	Communities Description:	This is a test

Here is another web portal service wrapped as a weblet. In this case it wraps a portion of a website known as eGroups as a weblet, to reuse its group functionality within an OpenPortal without anyone even knowing it actually makes calls back to the eGroup website:

Group Weblet		Edit
Start new group Invite new members Add Group Event Add Group Poll Login to your Group	Group Name:	Some Groups Name
	Group Address:	somegroup@eGroups.com
	Group Description:	This is a test

Both of these weblets can now be mixed and combined with other weblets, all on the same OpenPortal page. Some of the other weblets could themselves actually be wrappers around other websites. OpenPortal can then use these weblets to force the websites to interoperate, or to provide a unified portal to the user based on many other websites functionality.

10. You may not even know you're on an OpenPortal (because you're not).

Some people can choose to throw most of OpenPortal away and just use the Weblet Framework (see the document "Creating a Weblet") A *weblet* is a reusable piece of web-functionality that uses Easy Command Language and html as its front-end and two technologies known as *weblet descriptors* and *Weblet Server Pages (WSP)* on the middle-tier. Weblets can wrap potentially any back-end technology, whether it is JavaBeans, relational databases, CGI-BIN scripts, or even other websites. Technically, a weblet is nothing more than a bundle of properties, ECL commands, and template scripts. These are all declared in a file known as a *weblet descriptor*. Here is the weblet descriptor for a Business Card Weblet:

```
<? xml version="1.0" ?>
<weblet>
  <!--The Business Cards properties: name, organization, slogan, role,
                                email, and phone number -->
  <property name="name" default-value="Your Name"/>
  <property name="organization" default-value="Your Organization"/>
  <property name="slogan" default-value="Your Slogan"/>
  <property name="role" default-value="Your Role"/>
  <property name="email" default-value="Your Email Address"/>
  <property name="phone_number" default-value="Your Phone-Number"/>
  <!--The Business Cards commands: display, edit, and save -->
  <command full-command="display this" embed="BusinessCard.wsp"/>
  <command full-command="edit this" embed="BusinessCard.wsp"/>
  <command full-command="save this" embed="BusinessCard.wsp"/>
</weblet>
```

This weblet descriptor is just a flat-text file that sits in the filesystem. It establishes the properties and commands for a business card weblet. Notice the three `<command>` tags. These establish the *edit*, *display*, and *save* commands for the weblet. You can choose to throw away all of these commands and create your own new commands, *completely dropping the edit command if you want*. For example, you could create a "send card owner email" command and a "add card owner to contacts list" command by adding the following to lines to the weblet descriptor file:

```
  <command full-command="send card owner
email"embed="BusinessCard.wsp"/>
  <command full-command="add card owner to contacts list"
embed="BusinessCard.wsp"/>
```

Now, if the user enters the ECL commands "send card owner email" anywhere in the edit form of this weblet or clicks on a hyperlink titled send card owner email, then this command will be found in the weblet descriptor file and run.

Every command in a weblet is associated with a Weblet Server Pages(WSP) file that is executed when the command is executed by the user:

```
<command full-command="edit this" embed="BusinessCard.wsp"/>
```

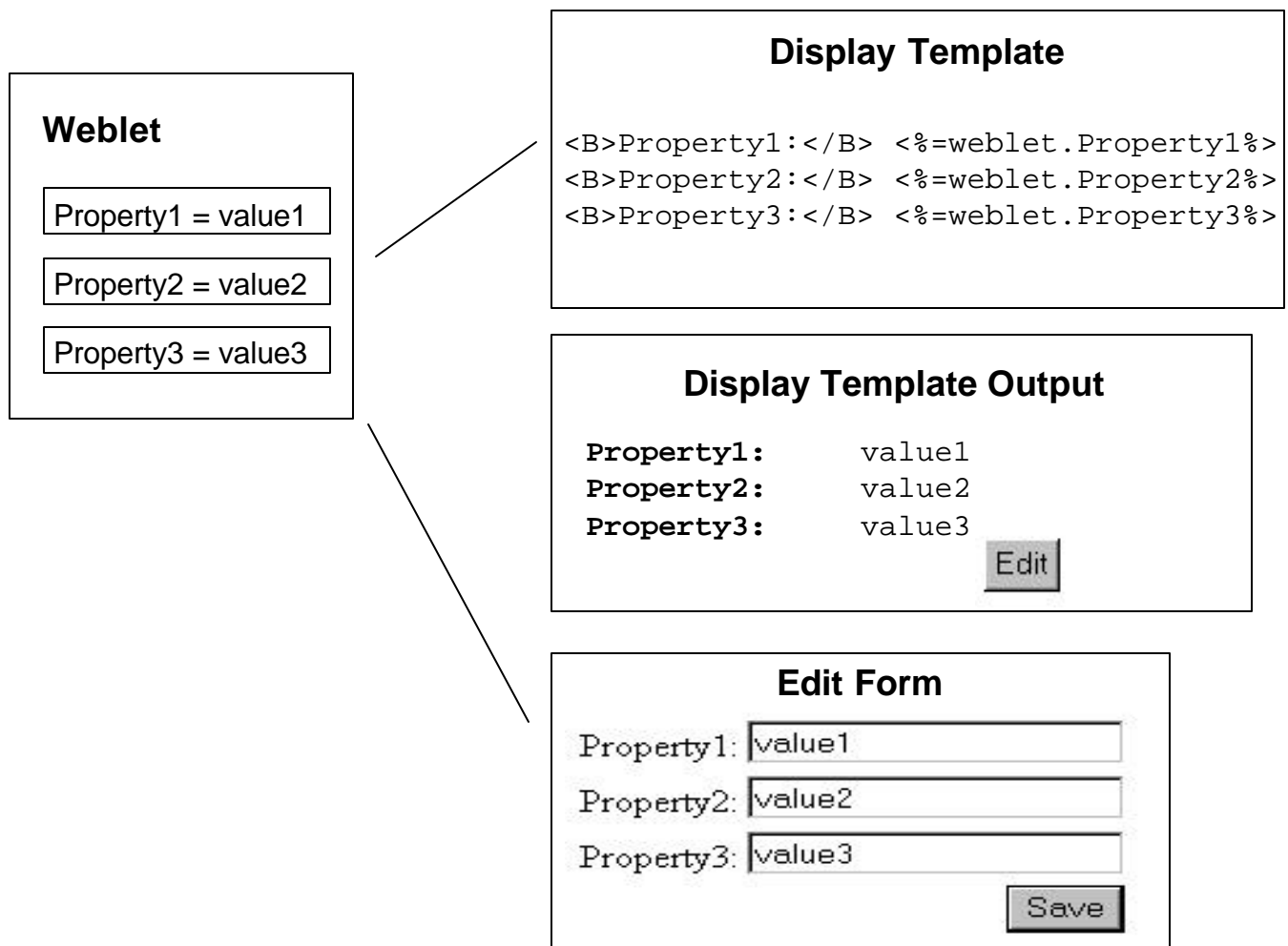
This WSP file is just like Java Server Pages (JSP) or Active Server Pages (ASP) (it's actually just a subset of the two), and contains a mix of html and java, javascript, or webl that is executed on the server side. In the future *anything* could be called from the *embed* parameter, whether it's a perl script, COM control, PHP script, server-side include, or Frontier script. In this way you can expose the functionality of sophisticated server-side perl scripts as simple human executable ECL commands.

Creating a Weblet

The Basic Weblets

Before we begin to create weblets, we must examine what kinds of weblets are possible. Most weblets can be divided into two types: property weblets and service weblets. Property weblets are very simple; they are just a list of properties that are strings. For example, a business card weblet could consist of five basic properties: name, organization, email, phone-number, and address. The diagram below is a simple property weblet with three properties: Property 1, Property 2, and Property 3. A display template can then display these properties in a web browser. The `<%=weblet.Property1%>` expression is a Weblet Server Pages (WSP) phrase that displays the value of Property1. Property weblets don't just include a template to display themselves in html; they also include an edit form that can be generated on demand to change a weblet's properties.

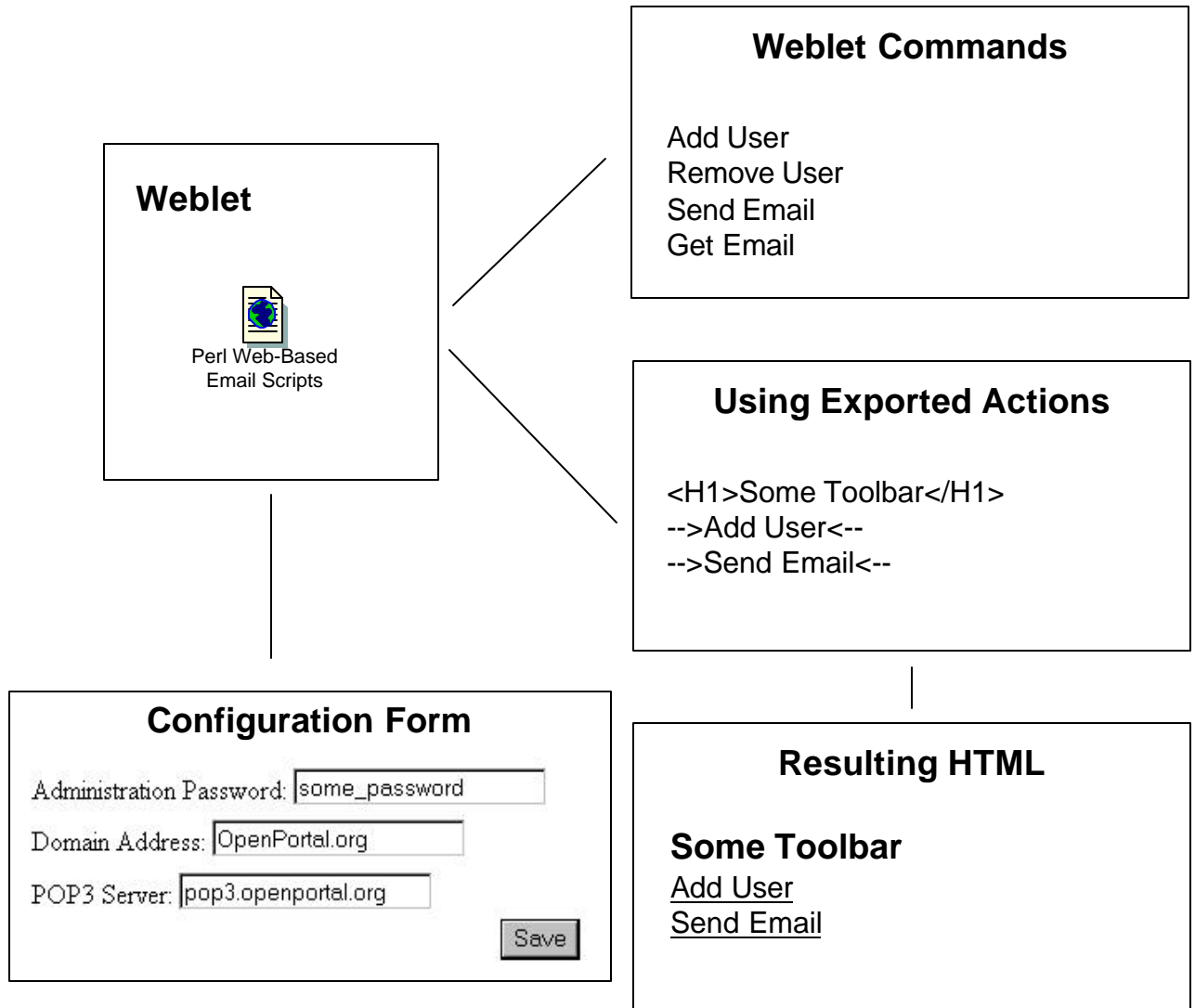
Property Weblet



The second kind of weblet is a service weblet. This kind of weblet wraps an underlying service and then exposes these services as Weblet Commands. This can be used to wrap other web-technologies, like perl scripts or Java Server Pages (JSP) files, or to offer easy to use services through OpenPortal. In the

example below a web-based email system that was written in perl is wrapped by a weblet that exports email commands, such as Add User. These commands could then be used from any weblet by entering →Add User←, which would cause the Weblet Command parser to automatically hyperlink Add User and initiate that action when clicked on. While a property weblet provides a form to change its properties, service weblets usually have configure forms to configure the services. An example configuration form is shown below that allows a user configure the email weblet through a web browser.

Service Weblet



Service weblets and property weblets are not mutually exclusive; most weblets will probably be a mix of the two. For example, a weblet could provide some properties that are in-place editable as well as export some services

Different Skills, Different Needs

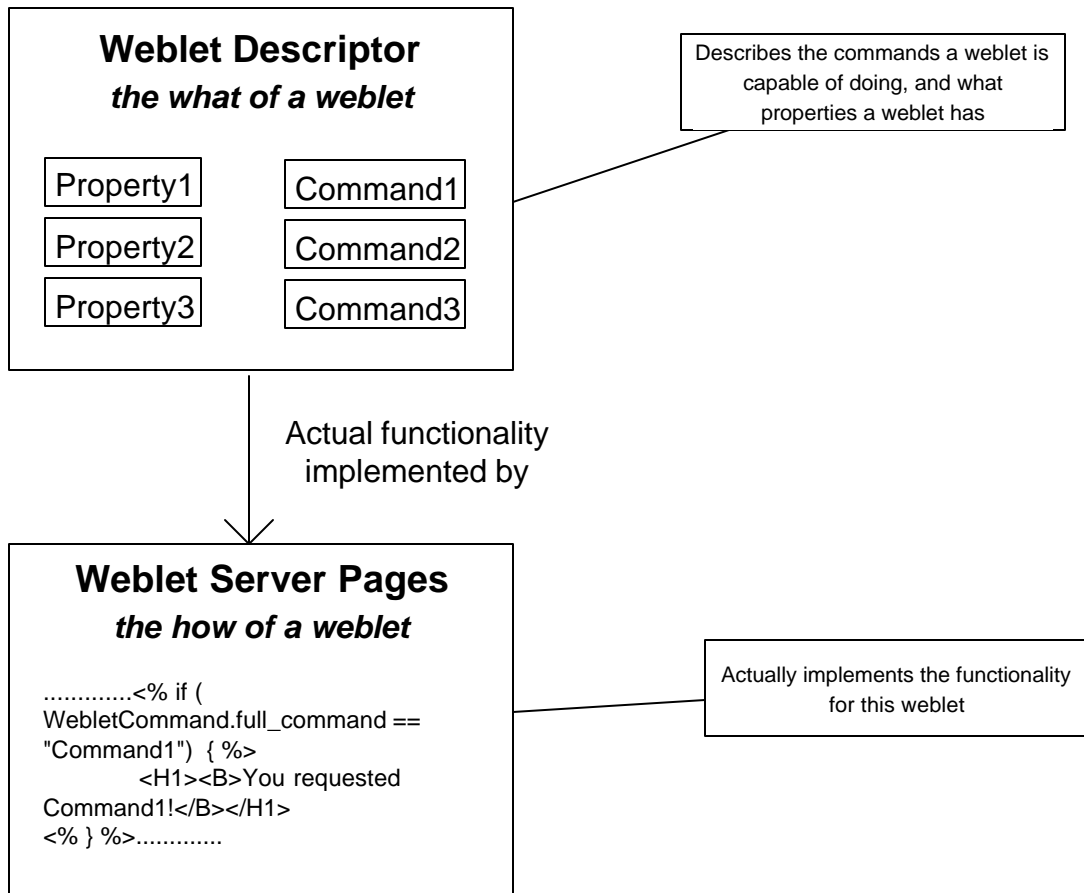
There are many different kinds of users with different needs. The Weblet Base divides these users into two types: those on the server-side who have direct shell access to an OpenPortal server, and those on the client-side who are operating through a browser.

Creating a Weblet on the Server-Side

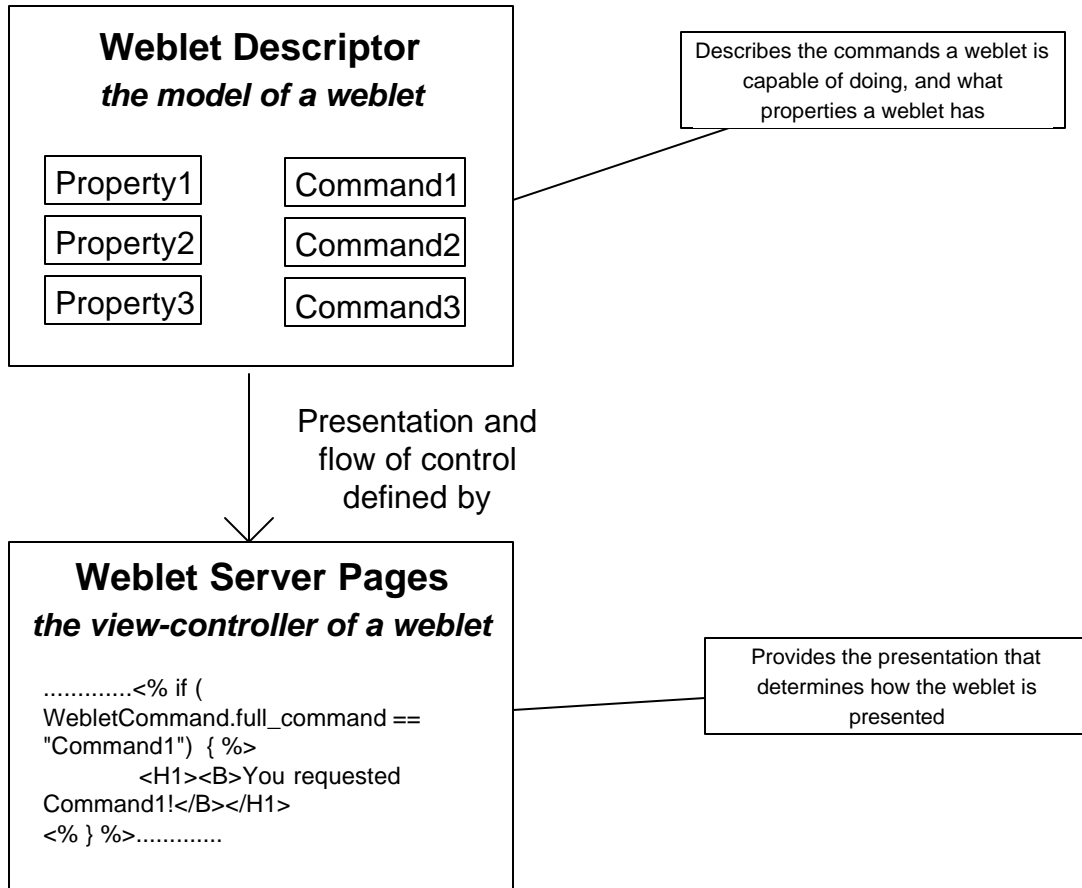
Many computer programming languages separate *what a program does* from *how the program does it*. This is called separating the *what* from the *how*. For example, one can specify that a program sorts items (the *what*) from how the sorting algorithm actually works (the *how*). The C language does this by separating variable and function declarations into a *header* file and the actual C code for the header file into a *source* file. Java has the ability to separate the *what* into something called an *interface*, while the actual *how* is taken care of by a Java class that implements the interface. There are many good reasons to separate what a program does from how it actually performs it. One is that it makes programs easier to maintain and change, since one can easily change how a program actually works "under-the-hood" without changing what it actually does.

A similar concept is the *Model-View-Controller* pattern. In this design pattern a *model* describes what something does while a *view-and-controller* describes how the model is visually presented and controlled by the user. The model contains no presentation logic; it simply simulates some object (i.e. it is a model). The view and controller modules are usually combined together into one, since it is natural to specify how something looks along with how a user manipulates the view. As an example of the model-view-controller pattern, one could have a model that simulates a business card, providing methods to get and set this business card's properties. A separate view and controller module could display this business card visually on a monitor and allow a user to manipulate the business card using a mouse. Java Server Pages (JSP) follows this pattern; in JSP a *JavaBean* acts as a model, while a JSP page manipulates this *JavaBean* model to create an HTML presentation and to respond to user requests through the browser. The model-view-controller pattern allows one to change how a system looks and is controlled without affecting the model. In the business card example, one could provide all types of new presentations and responses to user input for this business card model without having to change the model.

Creating weblets integrates both principles. Following the principle of separating the *what* of a weblet from its *how*, a weblet consists of two pieces: one piece promises the *what* of the weblet, exposing the properties and commands that the weblet supports, while the other piece provides the actual *how* that gives the weblet its functionality:



Following the second principle, these two pieces can be seen as the *model* and *view-controller* for the weblet:



These two weblet pieces are contained in two separate files, a *weblet descriptor* file and a *Weblet Server Pages* (WSP) file. A weblet descriptor file has the file extension *.weblet*, while a WSP file has the extension *.wsp*. A weblet descriptor file declares the commands and properties that a weblet supports, and also specifies which WSP file to execute for which commands. A WSP file contains scripting code and HTML presentation code.

Sequence of Actions for Weblet Descriptor File and Weblet Server Pages File

When a user issues a weblet command, such as "edit", a specific weblet descriptor is found for the type of weblet that the command is performed on. In the example below the weblet type is a *Business Card.weblet*. Inside this weblet descriptor is a list of all the commands that the weblet supports. In this case there is only one command, "edit." Every weblet command is associated with a *Weblet Server Pages* (WSP) file. In the example below the "edit" command is associated with the WSP file *Business Card.wsp*. Since the user requested the "edit" command, the associated WSP file *Business Card.wsp* is executed and the HTML results are returned.

Business Card Weblet

Edit

BaseSystem, Inc.

2840 Broadway #336
New York, NY 10025
www.basesystem.com

Create. Maintain. Share.

Brad Neuberger

Vice President of Technology

email: brad@basesystem.com

voice: 212-853-3602

User clicks on hyperlink

Command tag for "edit" is found

Weblet Descriptor

Business Card.weblet

```
<weblet>  
  <command action="edit" embed="BusinessCard.wsp">  
</weblet>
```

WSP script associated with "edit" command tag is executed

Weblet Server Pages

Business Card.wsp

```
<H1>HTML that defines the business card's edit form  
goes here</H1>  
<% // a scriptlet block to check for edit conditions %>
```

HTML is output

Business Card Weblet

Name:

Brad Neuberger

Organization:

BaseSystem, Inc.

Address:

2840 Broadway #336
New York, NY 10025
www.basesystem.com

Slogan:

Create. Maintain. Share.

Role:

Vice President of Technology

Email:

brad@basesystem.com

Phone-Number:

212-853-3602

Save

Weblet Descriptor Files

A weblet descriptor file is written in XML. It begins with a standard XML directive:

```
<? xml version="1.0" ?>
```

Next comes the declaration of a weblet:

```
<? xml version="1.0" ?>
<weblet>
```

If the weblet is a weblet container than this tag would be

```
<? xml version="1.0" ?>
<webletcontainer>
```

If a weblet is a service weblet, then the attribute *service-weblet* should be set to true:

```
<weblet service-weblet="true">
```

By default this is set to false. There can only be one copy of this weblet if it is a service-weblet (i.e. it is a singleton). If someone enters a weblet command without any target:

```
-->Login User<--
```

Then this command will automatically be sent to the one weblet that has this command. This allows one to export weblet commands that are available anywhere in the site as services. In the Login User example above, a Login weblet could be created that is a service weblet which exports this command.

After the weblet tag comes the declaration of a property:

```
<? xml version="1.0" ?>
<weblet>
  <property name="property1" default-value="defaultValue1"/>
```

This declares that the weblet has a property named *property1* and that the default-value given to this property when a new weblet of this type is created is *defaultValue1*. The default-value argument is optional, and if not given then default-value for a property is just the empty string "". The name of a property must be between a-z, A-Z, 0-9, or the special characters underscore _.

A weblet can have several properties:

```
<? xml version="1.0" ?>
<weblet>
  <property name="property1" default-value="defaultValue1"/>
  <property name="property2" default-value="defaultValue2"/>
  <property name="property3" default-value="defaultValue3"/>
```

All weblets share certain *standard-properties*, such as an owner or the date the weblet was created. These standard-properties do not need to be declared in the weblet descriptor, but it is sometimes useful to override one of their default values:. All standard properties are defined with the `<standard-property>` tag.

```
<? xml version="1.0" ?>
<weblet>
```

```

    <property name="property1" default-value="defaultValue1"/>
    <property name="property2" default-value="defaultValue2"/>
    <property name="property3" default-value="defaultValue3"/>
    <standard-property name="default_input" default-
value="<input type=text size=42 maxlength=80>"/>

```

This tag over-rides the standard-property *default_input*, which has to do with the inplace-editing feature of weblets (this is described below).

Command declarations come after property declarations, describing the commands that the weblet supports:

```

<? xml version="1.0" ?>
<weblet>
    <property name="property1" default-value="defaultValue1"/>
    <property name="property2" default-value="defaultValue2"/>
    <property name="property3" default-value="defaultValue3"/>
    <standard-property name="default_input" default-
value="<input type=text size=42 maxlength=80>"/>
    <command action="performSomeAction" perform-on="onSomeObject"
embed="someFile.wsp"/>

```

This tag declares that the weblet has a command *performSomeAction onSomeObject*, and when this command is activated (say by a [hyperlink](#)), then the Weblet Server Pages file named *someFile.wsp* should be executed and its HTML results embedded.

A weblet can have several commands:

```

<? xml version="1.0" ?>
<weblet>
    <property name="property1" default-value="defaultValue1"/>
    <property name="property2" default-value="defaultValue2"/>
    <property name="property3" default-value="defaultValue3"/>
    <standard-property name="default_input" default-value="<input
type=text size=42 maxlength=80>"/>
    <command action="performSomeAction" perform-on="onSomeObject"
embed="someFile.wsp"/>
    <command action="anotherAction" perform-on="onSomeObject"
embed="someFile.wsp"/>

```

The action attribute usually identifies some verb, such as *edit*, *set*, and *display*, while the perform-on attribute is usually a direct object of the verb, such as *this*, *properties*, *business card*. Multiple actions and perform-on's can be specified by using a comma:

```

<weblet>
    <property name="property1" default-value="defaultValue1"/>
    <property name="property2" default-value="defaultValue2"/>
    <property name="property3" default-value="defaultValue3"/>
    <standard-property name="default_input" default-value="<input
type=text size=42 maxlength=80>"/>
    <command action="performSomeAction" perform-on="onSomeObject"
embed="someFile.wsp"/>
    <command action="anotherAction" perform-on="onSomeObject"
embed="someFile.wsp"/>

```

```
<command action="add,create" perform-on="this"
embed="anotherFile.wsp"/>
```

This new command tag states that whenever the weblet commands "add this" or "create this" is requested by the user, the WSP file named anotherFile.wsp is executed and its results are embedded. Using commas to specify several actions or perform-on's is useful for specifying a weblet command that may have several different ways of being expressed. For example, the weblet commands "add this" and "create this" are basically equivalent, and commas allow this to be expressed as one command tag.

Instead of providing the parts of speech (i.e. the action and the perform-on) for a command, the full command can be provided through the full-command attribute:

```
<? xml version="1.0" ?>
<weblet>
  <property name="property1" default-value="defaultValue1"/>
  <property name="property2" default-value="defaultValue2"/>
  <property name="property3" default-value="defaultValue3"/>
  <standard-property name="default_input" default-value="<input
type=text size=42 maxlength=80"/>"/>
  <command action="performSomeAction" perform-on="onSomeObject"
embed="someFile.wsp"/>
  <command action="anotherAction" perform-on="onSomeObject"
embed="someFile.wsp"/>
  <command full-command="some long command" embed="someFile.wsp"/>
```

Every command tag must provide the embed attribute. This provides a script file that is executed when the command is requested by the user. This script file is passed a reference to the weblet itself, the weblet command that was requested by the user, and the request and response objects that are a part of the servlet API; this is covered in more detail in the section on Weblet Server Pages. The WSP filename that is given in the embed attribute is within the Java Naming Directory Interface (JNDI) namespace; if the filename has no directory slashes at the beginning of it, it is searched for in the weblet's local directory.

Other scripting languages and technologies can be called other than Weblet Server Pages. It should be possible to make JSP and ASP scripts callable from the embed attribute, though this is not planned for the current release. Currently the only technology other than WSP that can be called are java methods in the weblet itself. A method on the weblet class itself can be called by using the "this" operator and the method name:

```
<command full-command="some long command"
embed="this.someMethod()"/>
```

This will call the method someMethod() on the weblet itself when the command specified is encountered. This is useful for calling default methods in the weblet base-class for certain default actions. For example, when a weblet-container receives the "add" command it should call a predefined method in the WebletContainer base class named addWeblet(

```
<weblet-container>
  <command action="add" perform-on="this" embed="this.addWeblet()"/>
```

Any method that is called from the embed tag must be able to take the weblet itself as a reference, the weblet command, and the request and response objects as arguments.

Command tags are not exclusively executed; if the weblet command requested by the user is declared in several command tags, then each of the scripts listed by these tags will be run one after another and their output will be concatenated together. For example, if a weblet has the following two command tags:

```
<command action="save" perform-on="this" embed="this.saveWeblet()"/>
<command action="save,display" perform-on="this" embed="someFile.wsp"/>
```

and the weblet command "save this" has been requested by the user, then the first command tag will execute first by calling this.saveWeblet(), followed by the second command tag which will execute someFile.wsp and concatenate its output onto the first output.

Many times the perform-on attribute will be set to "this." Since the perform-on attribute is meant to be the direct-object of the action, it is useful to have a short-hand way of deducting whether a requested weblet command actually refers to the weblet defined in the weblet descriptor itself. It is impossible and inflexible to hard-code the actual name of the weblet into the perform-on attribute. For example, if there is a business card named "Brads Business Card", and a user requests the weblet command 'edit "Brads Business Card"', this could be hard-coded as:

```
<command action="edit" perform-on="\Brads Business Card\"">
```

However, it would be useful if the system automatically checked to see whether the weblet descriptor that is being called matches the perform-on attribute; if so, it converts the value of perform-on *in the original weblet command* into the word "this." In the example above, if the weblet that is being called is named "Brads Business Card" and the target of the "edit" command is "Brads Business Card", then the value of the perform-on attribute in the actual weblet command variable, WebletCommand, is changed to "this". This means that the above command tag can be converted to:

```
<command action="edit" perform-on="this">
```

which is much more flexible and generalized and less dependent on the actual name of the weblet.

A set of Dynamic HTML properties are also defined for the <weblet> tag and the <property> tag. The attribute *draggable* can be added to either a <weblet>, <weblet-container>, or <property> tag:

```
<? xml version="1.0" ?>
<weblet draggable="true">
  <property name="property1" default-value="defaultValue1"
draggable="true"/>
  <property name="property2" default-value="defaultValue2"/>
  <property name="property3" default-value="defaultValue3"/>
  <standard-property name="default_input" default-value="<input
type=text size=42 maxlength=80>"/>
  <command action="performSomeAction" perform-on="onSomeObject"
embed="someFile.wsp"/>
  <command action="anotherAction" perform-on="onSomeObject"
embed="someFile.wsp"/>
  <command full-command="some long command" embed="someFile.wsp"/>
</weblet>
```

The draggable attribute states that either the weblet, weblet-container, or property is draggable. The WSP script that is called can check this property, and can either honor it or not when attempting to create the javascript and Dynamic HTML that is part of making weblets drag and droppable. If the draggable attribute is left off it defaults to false.

Another Dynamic HTML attribute is the *inplace-editable* attribute. This attribute is similar to *draggable* in that it can only be placed either on a <property> tag, and denotes that the property can be edited simply by clicking on the element:

```
<? xml version="1.0" ?>
```



```

<weblet draggable="true">
  <property name="property1" default-value="defaultValue1"
draggable="true"/>
  <property name="property2" default-value="defaultValue2" inplace-
editable="true"/>inplace-
editable="false"/>

```

If the *inplace-editable* attribute is left off it defaults to true. Like the *draggable* attribute, underlying WSP scripts can check whether a given property has the *inplace-editable* attribute set to true and decide whether to honor this flag. It is merely a suggestion to the underlying presentation WSP.

When a property that is *inplace-editable* is clicked on, it is replaced by some kind of form input. The default is that the property is replaced with a standard `<input type=text>` tag, though this default can be over-ridden by setting the standard-property *default_input* to something else:

```

<? xml version="1.0" ?>
<weblet draggable="true">
  <property name="property1" default-value="defaultValue1"
draggable="true"/>
  <property name="property2" default-value="defaultValue2" inplace-
editable="true"/>inplace-
editable="false"/><standard-property name="default_input" default-
value="<input type=text size=42 maxlength=80>"/>
  <command action="performSomeAction" perform-on="onSomeObject"
embed="someFile.wsp"/>
  <command action="anotherAction" perform-on="onSomeObject"
embed="someFile.wsp"/>
  <command full-command="some long command" embed="someFile.wsp"/>
</weblet>

```

For any property which has had *inplace-editable* be set to true and which does not define its own custom input, the *default_input* is used for this property when it is clicked on. For example, in the above weblet descriptor block when *property2* is clicked on in a browser it is replaced with the *default_input*, which is `<input type=text size=42 maxlength=80>`. Individual *inplace-editable* properties can also provide their own *inplace-input* attribute for what kind of input they are replaced with when clicked on:

```

<property name="property2" default-value="defaultValue2" inplace-
editable="true" inplace-input="<textarea>"/>

```

This will replace *property2* with a `<textarea>` when it is clicked on rather than the *default_input* of `<input type=text size=42 maxlength=80>`. Note that for both the *default_input* standard-property and the *inplace-input* attribute only a form input type may be given. This form input must not define the *name* or *value* attributes of the input, and cannot include more than one tag. The inplace-editing engine automatically fills these values in according to certain characteristics.

```
<property name="property2" default-value="defaultValue2" inplace-
editable="true" inplace-input="<textarea>something</textarea>" />
```

INCORRECT

```
<property name="property2" default-value="defaultValue2" inplace-
editable="true" inplace-input="<textarea wrap=virtual>" />
```

CORRECT

```
<property name="property2" default-value="defaultValue2" inplace-
editable="true" inplace-input="<input type=text name=property2>" />
```

INCORRECT

```
<property name="property2" default-value="defaultValue2" inplace-
editable="true" inplace-input="<input type=text value=something>" />
```

INCORRECT

A final example weblet descriptor is provided for a business card weblet:

BusinessCard.weblet:

```
<? xml version="1.0" ?>
<weblet draggable="true">
  <property name="name" default-value="Your Name" />
  <property name="organization" default-value="Your Organization" />
  <property name="slogan" default-value="Your Slogan" />
  <property name="role" default-value="Your Role" />
  <property name="email" default-value="Your Email Address" />
  <property name="phone_number" default-value="Your Phone-Number" />
  <command action="save,display" perform-on="this"
embed="BusinessCard.wsp" />
  <command action="edit" perform-on="this" embed="BusinessCard.wsp" />
  <command action="set" perform-on="security, security settings,
settings, security properties" embed="StandardSecurityForm.wsp" />
</weblet>
```

Weblet Server Pages

Weblets are actually scripted separate from the weblet descriptor in a Weblet Server Pages (WSP) file, which has the extension wsp. A WSP file consists of scripting control code and HTML presentation code. WSP is based on Active Server Pages (ASP) and Java Server Pages (JSP), and is a light-weight subset of both standards.

All WSP files begin with a language directive that states what language the script in the WSP file is written in:

```
<%@ language="javascript" %>
```

This directive is inspired by the JSP specification. The following languages are currently supported in WSP:

- Javascript/ECMAScript (language="javascript|ecmascript")
- WebL (language="webl ")
- Java

The preferred and default language for WSP pages is Javascript. All examples in this section use Javascript.

The rest of the WSP file consists of *scriptlets* and HTML. A scriptlet is a block of code between the tags `<%` and `%>` in the language defined in the language directive:

Example.wsp:

```
<%@ language="javascript" %>
<H1>Hello world!</H1>
<%
    var someVariable = "blah";
    function someFunction() {
        write(someVariable);
    }
%>
<CENTER>Some more HTML</CENTER>
```

If a weblet descriptor existed that had the following:

```
<weblet>
    <command action="display" embed="Example.wsp">
</weblet>
```

Then when a user requested the weblet command display, Example.wsp would be executed and the results would be returned as a string. The scriptlet block would execute and its results would be embedded in Example.wsp's output:

```
<H1>Hello world!</H1>
blah
<CENTER>Some more HTML</CENTER>
```

While WSP files can be this simple, in general a WSP file is used to perform and present a user's weblet command request. Several objects are exposed to the WSP scripting language in the scripting language's native format to help process the user's weblet command request:

- Standard JSP objects
 - request
 - response
 - servlet
 - session
 - input
 - output
 - parameters
- Special WSP objects
 - weblet
 - WebletCommand
 - WebletManager
 - naming

The standard JSP objects can be referenced exactly as one would in java. The following would be legal references:

- `servlet.getServletContext().getRealPath(myFilename)`
- `request.getRemoteUser()`
- `response.setHeader("Content-encoding", "binary")`

The *servlet* object corresponds to the Java servlet's *this* object. Since the same servlet is shared by all WSP pages, *servlet* is actually a global object.

The *request* and *response* objects are the same as their corresponding counterparts in the *service* method argument list. These objects are refreshed with each WSP page invocation.

session is actually a shorthand for *request.getSession(true)*. By its nature, this object is static for the duration of the client's connection.

input is actually a shorthand for *request.getInputStream()*, while *output* is shorthand for the string result that is returned by the execution of the WSP file.

The *parameters* object contains the collection of parameters passed to the WSP page through the request's *query string*. Single-valued parameters are stored as scalars, multi-valued parameters are stored as arrays.

An individual parameter (for instance, *filename*) can be referenced as *parameters.filename* or as *parameters["filename"]*.

Referencing non-existent parameter properties should not cause any errors. If a non-existent parameter is referenced then the offending javascript statement is simply ignored.

WSP provides special objects to make it easy for WSP scripts to manipulate the weblet descriptors that called them. The first is the *WebletCommand* object. This object exposes all of the details concerning the weblet command that the user requested. It has the following properties:

WebletCommand.action – the action requested by the user

WebletCommand.perform_on – the object on which the action was requested

WebletCommand.full_command – the full command (action + perform_on)

WebletCommand.weblet_name – the name of the weblet on which the command is executed on

WebletCommand.current_user – a reference to a User object for the user that executed the command

WebletCommand.current_container – a reference to the parent weblet container of the weblet that the command was executed on

WebletCommand.current_site – a reference to the Site object that the target weblet is in

WebletCommand.current_area – a reference to the Area object that the target weblet is in

WebletCommand.full_name – the full path-name of the target weblet

The *weblet* object provides access to the weblet that called the WSP file and information about the weblet. Using Javascript one can access everything within the *weblet* object using the Document Object Model (DOM). The *weblet* object exposes all of the values of a weblet's properties that were defined in the weblet descriptor:

```
<% write(weblet.property1);  
      weblet.property2 = "hello world";  
%>
```

Further, any of the attributes that were defined for these properties in the weblet descriptor, such as *draggable* or *inplace-editable*, are accessible as well, using the Document Object Model:

```
<%  
weblet.property1.draggable  
weblet.property2.inplace-editable  
%>
```

All of the properties can be referenced as follows, using Javascript:

```
<%  
weblet.all.tags("property");  
%>
```

This returns an array of all the properties.

A weblet's standard-properties is referenced as follows:

```
weblet.standard_properties.property_name
```

where *property_name* is some standard property, such as the owner:

```
<%  
weblet.standard_properties.owner = "Brad Neuberg";  
%>
```

A different weblet command than the one currently executing in the WSP file can be invoked on the weblet as follows:

```
<%  
    weblet.embed("Full Command", WebletCommand, request, response);  
%>
```

where *Full-Command* is some weblet command like "display business card". If the command does not exist no error is thrown and the embed() method simply returns.

WebletManager exposes methods for weblets to gain a context to the naming service, as well as a method to embed other weblets.

The *naming* object exposes the root of the naming service so that operations may be performed in the directory services.

It is recommended that WSP files not output <html>, <head>, or <body> tags, since several WSP files could be chunked together by a weblet container and the existence of multiple <html> or <head> tags in each of these chunked weblets could confuse browsers.

Just as in Java Server Pages, the shortcut tag <%= someVariable.property %> exists to print out a variable's current value. For example, to print out the value of the weblet property *organization* in some weblet, one would do as follows:

```
<H1>Hello I am a member of <%= weblet.organization %></H1>
```

which if weblet.organization was set to "OpenPortal" would print out:

```
<H1>Hello I am a member of OpenPortal</H1>
```

Dynamic HTML Tags

OpenPortal provides some convenience tags that can be used in WSP scripts to help create Dynamic HTML interfaces. All of these tags are based on XML, and have the namespace *weblet* attached to them. These tags help build the following functionality into weblet user interfaces:

- Inplace-editing

In the future these tags and the DHTML subsystem will be extended to allow the creation of extremely powerful user interfaces for weblets in a way that is transparent to both the developer and cross-platform.

Inplace-Editing Tag

To add in-place editing to a property in a WSP file, place the tag `<weblet:INPLACE-EDIT>` around the html which displays the property. For example, if one has a weblet that has the property *organization* in it and wishes to make it inplace-editable, one would surround it with the inplace-edit tag:

```
I am a member of <weblet:INPLACE-EDIT property-  
name="organization"><%=weblet.organization%></weblet: INPLACE-EDIT>
```

If the value of *organization* was "OpenPortal", then this would print out:

I am a member of OpenPortal

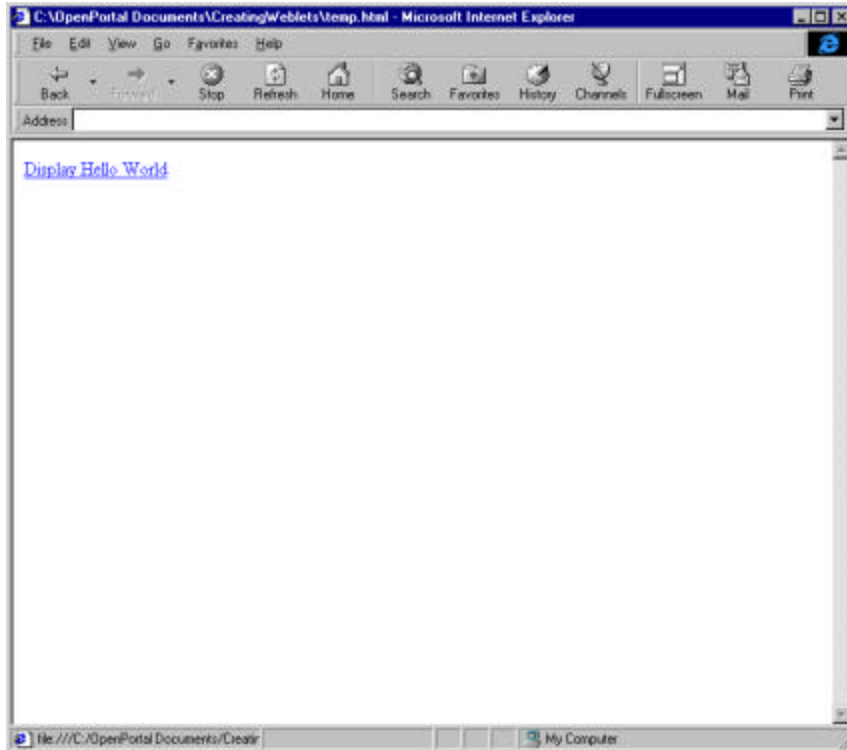
and when a user clicked on the word 'OpenPortal' the word would instantly turn into a small edit field for the value of `weblet.organization` to be changed.

OpenPortal automatically inserts the correct javascript and information in a WSP file when sent to the client if it contains `<weblet:INPLACE-EDIT>` tags.

Examples of Creating Weblets

The Simplest Weblet – Hello World

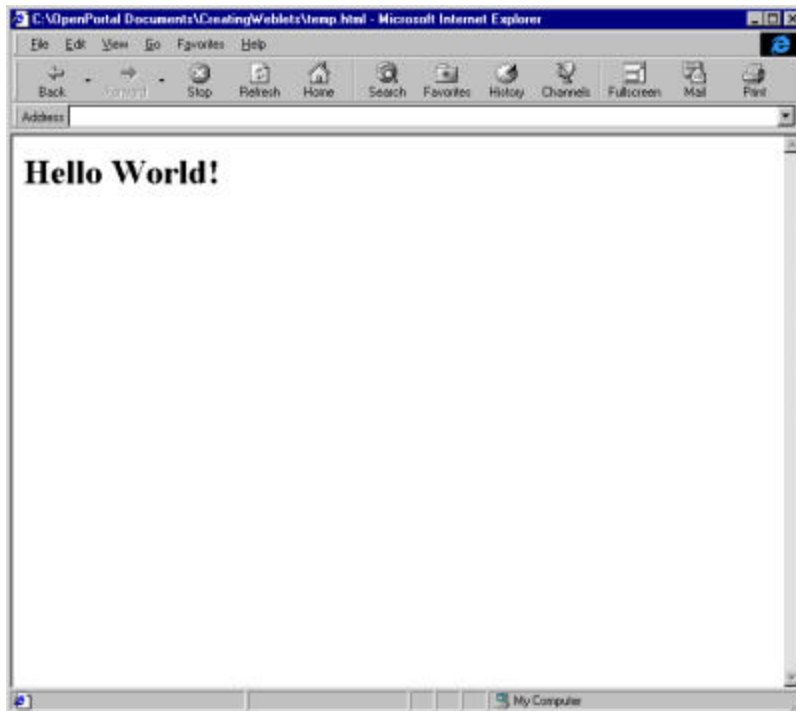
Let's start with the absolute simplest weblet, one that prints "Hello World" when it receives the weblet command "display hello world". When the user clicks on the web page below:



whose HTML looks like this (remember that weblet commands are case-insensitive:

```
-->Display Hello World<--
```

The following is displayed:



To make this weblet, a weblet descriptor *.weblet* file and a Weblet Server Pages *.wsp* file need to be made.

HelloWorld.weblet:

```
<? xml version="1.0" ?>
<weblet service-weblet="true">
  <command full-command="display hello world" embed="HelloWorld.wsp"/>
</weblet>
```

HelloWorld.wsp:

```
<%@ language="javascript">

<H1>Hello World!</H1>
```

The *service-weblet* attribute in *HelloWorld.weblet* establishes that this weblet is a service-weblet, and that there will only be one copy of this weblet on the site. If anyone types in the command `-->display hello world<--` then this weblet will be called. The *<command>* tag establishes that the "display hello world" command should invoke and run the WSP file *HelloWorld.wsp*.

Notice that *HelloWorld.wsp* does not need to check what command called it. *HelloWorld.jsp* could have been written as

```
<%@ language="javascript">

<% if (WebletCommand.full_command == "display hello world") { %>
<H1>Hello World!</H1>
<% } %>
```

This accomplishes the same thing as the previous version and is not necessary unless desired. If we wanted to make a more complex version of Hello World that displayed Goodbye World when the commands

"display Goodbye World" or "Goodbye World" are called, and displays "Hello World" when the commands "display Hello World" or "Hello World" are called by the user, then we could do it as follows:

HelloWorld2.weblet:

```
<? xml version="1.0" ?>
<weblet service-weblet="true">
  <command full-command="display hello world, hello world"
embed="HelloWorld.wsp"/>
  <command full-command="display goodbye world, goodbye world"
embed="GoodbyeWorld.wsp"/>
</weblet>
```

When "display hello world" or "hello world" are called, *HelloWorld.wsp* is embedded:

HelloWorld.wsp:

```
<%@ language="javascript" %>
<H1>Hello World!</H1>
```

When "display goodbye world" or "goodbye world" are called, *GoodbyeWorld.wsp* is embedded:

GoodbyeWorld.wsp:

```
<%@ language="javascript" %>
<H1>Goodbye World!</H1>
```

Dividing the commands into two separate files is an easy, quick, and reusable way of making the Weblet Server Pages, though they could be put into one file as follows:

HelloWorld2.weblet:

```
<? xml version="1.0" ?>
<weblet service-weblet="true">
  <command full-command="display hello world, hello world"
embed="World.wsp"/>
  <command full-command="display goodbye world, goodbye world"
embed="World.wsp"/>
</weblet>
```

World.wsp:

```
<%@ language="javascript" %>
<% if (WebletCommand.full_command == "display hello world" ||
WebletCommand.full_command == "hello world") { %>
<H1>Hello World!</H1>
<% }
      else if (WebletCommand.full_command ==
"display goodbye world" ||
WebletCommand.full_command == "goodbye world") { %>
<H1>Goodbye World!</H1>
<% } %>
```

We could add a *property* to this weblet that underlying Weblet Server Pages files could use to format themselves as follows:

HelloWorld3.weblet:

```
<? xml version="1.0" ?>
<weblet service-weblet="true">
  <property name="message" default-value="some message goes here"/>
  <command full-command="display hello world, hello world"
embed="HelloWorld.wsp"/>
  <command full-command="display goodbye world, goodbye world"
embed="GoodbyeWorld.wsp"/>
</weblet>
```

HelloWorld.wsp:

```
<%@ language="javascript" %>
<H1>Hello World! By the way, here's your <%=weblet.message %></H1>
```

GoodbyeWorld.wsp:

```
<%@ language="javascript" %>
<H1>Goodbye World! By the way, here's your <%=weblet.message %> </H1>
```

We could have an automatic edit form generated for this weblet by adding an edit command to it:

HelloWorld4.weblet:

```
<? xml version="1.0" ?>
<weblet service-weblet="true">
  <property name="message" default-value="some message goes here"/>
  <command full-command="display hello world, hello world"
embed="HelloWorld.wsp"/>
  <command full-command="display goodbye world, goodbye world"
embed="GoodbyeWorld.wsp"/>
  <command action="edit" perform-on="this" embed="this.edit()"/>
</weblet>
```

An edit form will automatically be generated for a user to customize the property *message*. Alternatively, we could make *weblet.message* be inplace-editable so that if the user clicks right where *weblet.message* is printed out it will turn into a tiny edit field where the value can be changed:

HelloWorld5.weblet:

```
<? xml version="1.0" ?>
<weblet service-weblet="true">
  <property name="message" default-value="some message goes here"
inplace-editable="true"/>
```

HelloWorld.wsp:

```
<%@ language="javascript" %>
```

```
<H1>Hello World! By the way, here's your <weblet:INPLACE-EDIT
property-name="message"><%=weblet.message %></weblet:INPLACE-
EDIT></H1>
```

GoodbyeWorld.wsp:

```
<%@ language="javascript" %>
<H1>Goodbye World! By the way, here's your <weblet:INPLACE-EDIT
property-name="message"><%=weblet.message %></weblet:INPLACE-
EDIT></H1>
```

A Property Weblet - Business Card

Our next example weblet is a business card weblet. This is an example of a property weblet.

This weblet will have the following properties:

- name
- organization
- address
- slogan
- role
- email
- phone-number

and the following commands:

- display
- edit
- save

This business card weblet will look as follows when given the *display* command:



Each of these properties will be in-place editable, so that when someone clicks on the organization name – "BaseSystem, Inc.", an in-place DHTML edit form will instantly be embedded where the value can be changed:

Business Card Weblet [Edit](#)

2840 Broadway #536
New York, NY 10025
www.basesystem.com [Create](#) [Maintain](#) [Share](#)

Brad Neuberg
Vice President of Technology
email: brad@basesystem.com
voice: 212-853-3602

and if changed:

Business Card Weblet [Edit](#)

2840 Broadway #536
New York, NY 10025
www.basesystem.com [Create](#) [Maintain](#) [Share](#)

Brad Neuberg
Vice President of Technology
email: brad@basesystem.com
voice: 212-853-3602

will instantly reflect this change:

Business Card Weblet [Edit](#)

OpenPortal

2840 Broadway #536
New York, NY 10025
www.basesystem.com [Create](#) [Maintain](#) [Share](#)

Brad Neuberg
Vice President of Technology
email: brad@basesystem.com
voice: 212-853-3602

When this weblet is given the *edit* command (by clicking on the [Edit](#) hyperlink above), it will return the following form which can be used by those without Dynamic HTML browsers:

Business Card Weblet

Name:

Organization:

Address:

Slogan:

Role:

Email:

Phone-Number:

The properties and commands for the business card weblet are defined in the following weblet descriptor file:

Business Card.weblet:

```
<weblet draggable="true">
  <property name="organization" default-value="Organization"/>
  <property name="address" default-
value="Address1<br>Address2<br>Address3<br>"
    inplace-input="<textarea rows=3 cols=30>"/>
  <property name="slogan" default-value="Slogan"/>
  <property name="name" default-value="Name"/>
  <property name="role" default-value="Role"/>
  <property name="email" default-value="Email"/>
  <property name="phone_number" default-value="Phone-Number"/>
  <standard-property name="default_input" default-value="<input
type=text size=42 maxlength=80"/>
  <command action="save" perform-on="this"
embed="BusinessCardSave.wsp">
  <command action="display,view,save" perform-on="this"
embed="BusinessCardDisplay.wsp">
  <command action="edit" perform-on="this"
embed="BusinessCardEdit.wsp">
</weblet>
```

Notice that the HTML that defines the three commands for this weblet are in three separate files. Also notice that the *save* command is given twice:

These three files are:

BusinessCardSave.WSP:

```
<%@ language="javascript" %>
<!-- Save the weblet -->
<%
  if (parameters.organization != null)
    weblet.organization = parameters.organization;
  if (parameters.address != null)
    weblet.address = parameters.address;
```

```

    if (parameters.slogan != null)
        weblet.slogan = parameters.slogan;
    if (parameters.parameters != null)
        weblet.name = parameters.name;
    if (parameters.role != null)
        weblet.role = parameters.role;
    if (parameters.email != null)
        weblet.email = parameters.email;
    if (parameters.phone_number != null)
        weblet.phone_number = parameters.phone_number;
    naming.rebind(weblet.standard_property.name);
%>

```

Notice how a reference to the calling weblet is obtained in the WSP file by using the predefined standard object *weblet*:

```
weblet.organization = parameters.organization;
```

Also notice how once the weblet's new values have been stored in the weblet it is saved back into the directory service using the exposed *naming* object:

```
naming.rebind(weblet.standard_property.name);
```

BusinessCardDisplay.wsp:

```

<%@ language="javascript" %>
<!-- Display the weblet -->
    <table border="0" cellspacing="2" cellpadding="3" width="206">
        <tr bgcolor="#EEEECC">
            <td><font face="Arial, Helvetica, sans-serif" size="-1"
color="#666633"><b>Business
            Card Weblet</b></font></td>
            <td align="right" width="26"><font size="-1">-->Edit<-
</font></td>
        </tr>
    </table>

    <table border="0" cellspacing="0" width="300">
        <tr bgcolor="#FFFF00">
            <td width="57%" height="34" valign="top"><font size="5"
face="Times New Roman, Times, serif"><weblet:INPLACE-EDIT property-
name="organization"><%=weblet.organization%></weblet: INPLACE-
EDIT></font> </td>
            <td width="43%" height="34">&nbsp;</td>
        </tr>
        <tr bgcolor="#FFFF00">
            <td width="57%"><font size="1"><weblet:INPLACE-EDIT property-
name="address"><%=weblet.address%></weblet:INPLACE-EDIT></font>
            </td>
            <td valign="bottom" width="43%"><i><font size="1" face="Arial,
Helvetica, sans-serif"><weblet:INPLACE-EDIT property-
name="slogan"><%=weblet.slogan%></weblet:INPLACE-EDIT></font></i></td>
        </tr>
        <tr bgcolor="#0066FF">
            <td width="57%"></td>
            <td width="43%"></td>
        </tr>
    </table>

```

```

        <tr bgcolor="#0066FF">
            <td width="57%"></td>
            <td width="43%"><font color="#FFFFFF"><b><font
size="2"><weblet:INPLACE-EDIT property-
name="name"><%=weblet.name%></weblet:INPLACE-
EDIT></font></b></font></td>
        </tr>
        <tr bgcolor="#0066FF">
            <td width="57%" height="24"></td>
            <td width="43%" height="24" valign="top"><font color="#FFFFFF"
size="1"><weblet:INPLACE-EDIT property-
name="role"><%=weblet.role%></weblet:INPLACE-EDIT></font></td>
        </tr>
        <tr bgcolor="#0066FF">
            <td width="57%"></td>
            <td width="43%" valign="top"><font color="#FFFFFF"
size="1">email: <weblet:INPLACE-EDIT property-
name="email"><%=weblet.email%></weblet:INPLACE-EDIT></font></td>
        </tr>
        <tr bgcolor="#0066FF">
            <td width="57%"></td>
            <td width="43%" valign="top"><font color="#FFFFFF"
size="1">voice: <weblet:INPLACE-EDIT property-name="phone-number">
<%=weblet.phone_number%></weblet:INPLACE-EDIT></font></td>
        </tr>
    </table>

```

BusinessCardEdit.wsp:

```

<!-- Edit the weblet -->
    <form method="post" action="?<%=WebletCommand.toURLString()%>">
        <table border="0" cellspacing="2" cellpadding="3">
            <tr bgcolor="#EEEECC">
                <td colspan="3"><font face="Arial, Helvetica, sans-serif"
size="-1" color="#666633"><b><font color="#333300">Business Card
Weblet</font></b></font></td>
            </tr>
            <tr>
                <td><font face="Verdana, Arial, Helvetica, sans-serif" size="-
1">Name:</font></td>
                <td>&nbsp;</td>
                <td><input type="text" name="name" value="<%=weblet.name%>"
size="42" maxlength="80"><td>
            </tr>
            <tr>
                <td><font face="Verdana, Arial, Helvetica, sans-serif" size="-
1">Organization:</font></td>
                <td>&nbsp;</td>
                <td><input type="text" name="organization"
value="<%=weblet.organization%>" size="42" maxlength="80"><td>
            </tr>
            <tr>
                <td><font face="Verdana, Arial, Helvetica, sans-serif" size="-
1">Address:</font></td>
                <td>&nbsp;</td>
                <td><input type="text" name="address"
value="<%=weblet.address%>" size="42" maxlength="80"><td>

```

```

        </tr>
        <tr>
            <td><font face="Verdana, Arial, Helvetica, sans-serif" size="-
1">Slogan:</font></td>
            <td>&nbsp;</td>
            <td><input type="text" name="slogan"
value="<%=weblet.slogan%>" size="42" maxlength="80"><td>
        </tr>
        <tr>
            <td><font face="Verdana, Arial, Helvetica, sans-serif" size="-
1">Role:</font></td>
            <td>&nbsp;</td>
            <td><input type="text" name="role" value="<%=weblet.role%>"
size="42" maxlength="80"><td>
        </tr>
        <tr>
            <td><font face="Verdana, Arial, Helvetica, sans-serif" size="-
1">Email:</font></td>
            <td>&nbsp;</td>
            <td><input type="text" name="email" value="<%=weblet.email%>"
size="42" maxlength="80"><td>
        </tr>
        <tr>
            <td><font face="Verdana, Arial, Helvetica, sans-serif" size="-
1">Phone-Number:</font></td>
            <td>&nbsp;</td>
            <td><input type="text" name="phone_number"
value="<%=weblet.phone_number%>" size="42" maxlength="80"><td>
        </tr>
        <tr>
            <td>&nbsp;</td>
            <td>&nbsp;</td>
            <td align="right"><input type="submit" value="Save"></td>
        </tr>
    </table>
</form>

```

Wrapping An Existing Portal as a Weblet – eGroups

In this example a Group weblet is created. It is created by taking a portion of an existing portal, named eGroups, and wrapping eGroups so that its functionality can be reused by OpenPortal. eGroups is a portal that facilitates the creation of web-based groups. The language WebL is used to manipulate the eGroups site.

The Group weblet has the following user interface when it receives the *display* command:

Group Weblet		Edit
Start new group Invite new members Add Group Event Add Group Poll Login to your Group	Group Name:	Some Groups Name
	Group Address:	somegroup@eGroups.com
	Group Description:	This is a test

Every Group weblet has the following properties:

- group name
- group address
- group description
- group toolbar

The group toolbar property is interesting. It is inplace-editable, so that a user can click right on it to change it's menu contents, and then hit save to have it instantly updated!

Group Weblet		Edit
<div> <div> -->Start new group<-- -->Invite new members<-- -->Add Group Event<-- -->Add Group Poll<-- -->Login to your Group<-- </div> <div> ▲ ▼ </div> </div> <div>Save</div>	Group Name: Some Groups Name Group Address: somegroup@eGroups.com Group Description: This is a test	

Every Group weblet has the following commands:

- display
- edit
- save
- start new group
- invite new members
- add group event
- add group poll
- login to your group

Here is the weblet descriptor that describes these properties and commands:

Group.weblet:

```
<? xml version="1.0" ?>
<weblet draggable="true">
  <property name="group_name"/>
  <property name="group_address"/>
  <property name="group_description"/>
  <property name="group_toolbar" inplace-editable="true"
    default-value="
```

```

-->Start new group<--
-->Invite new members<--
-->Add Group Event<--
-->Add Group Poll<--
-->Login to Groups<--"
        input="<textarea rows=5 cols=30>" />
        <command action="start" perform-on="new group" embed="Group.wsp" />
        <command action="invite" perform-on="new members"
embed="Group.wsp" />
        <command action="add" perform-on="groups event" embed="Group.wsp" />
        <command action="add" perform-on="groups poll" embed="Group.wsp" />
        <command action="login to" perform-on="your group"
embed="Group.wsp" />
        <command action="display" perform-on="this" embed="Group.wsp" />
        <command action="edit" perform-on="this" embed="this.edit()" />
        <command action="save" perform-on="this" embed="this.save()" />
</weblet>

```

All of the commands funnel into Group.wsp, which does the actual checking of which command was requested:

Group.wsp:

```

<%@ language="web1" %>

<!-- Define all functions -->
<% var startGroup = new fun()
    var page = GetURL("http://www.egroups.com/listman", [.
method="display_startnewlist" .]);

    // manipulate HTML here
end;

var inviteMembers = new fun()
    var page = GetURL("http://www.egroups.com/GroupMembersPage",
[.
method="performAction",listName=weblet.group_name,selectedView="all",ne
wMemberName="",Button_InviteNewMember="Invite+new+member".]);

    // manipulate HTML here
end;

var addEvent = new fun()
    var page = GetURL("http://www.egroups.com/cal", [. md="a",
listname=weblet.group_name .]);

    // manipulate HTML here
end;

var addPoll = new fun()
    var page = GetURL("http://www.egroups.com/vote", [. md="a",
listname=weblet.group_name .]);

    // manipulate HTML here
end;

```

```

var login = new fun( )
    var page = GetURL("http://www.egroups.com");

    // manipulate HTML here
end;
%>

<!-- Handle all commands -->
<!-- Start new group -->
<% if (WebletCommand.full_command = "start new group")
    startGroup();
end;
%>

<!-- Invite new members -->
<% if (WebletCommand.full_command = "invite new members")
    inviteMembers();
end;
%>

<!-- Add Group Event -->
<% if (WebletCommand.full_command = "add group event")
    addEvent();
end;
%>

<!-- Add Group Poll -->
<% if (WebletCommand.full_command = "add group poll")
    addPoll();
end;
%>

<!-- Login to your Group -->
<% if (WebletCommand.full_command = "login to your group")
    login();
end;
%>

<!-- Display this-->
<% if (WebletCommand.full_command = "display this") %>
    <table border="0" cellspacing="2" cellpadding="3" width="206">
        <tr bgcolor="#EEEECC">
            <td><font face="Arial, Helvetica, sans-serif" size="-1"
color="#666633"><b>Group
Weblet</b></font></td>
            <td align="right" width="26"><font size="-1">->Edit<-
</font></td>
        </tr>
    </table>
    <!-- simplified HTML -->
    .
    .
    .
    <td><%=weblet.group_toolbar%></td>

```

```

.
.
.
<td>Group Name:</td>
<td><%=weblet.group_name%> </td>
.
.
.
<td>Group Address:</td>
<td><%=weblet.group_address%> </td>
.
.
.
<td>Group Description:</td>
<td><%=weblet.group_description%> </td>
<% end; %>

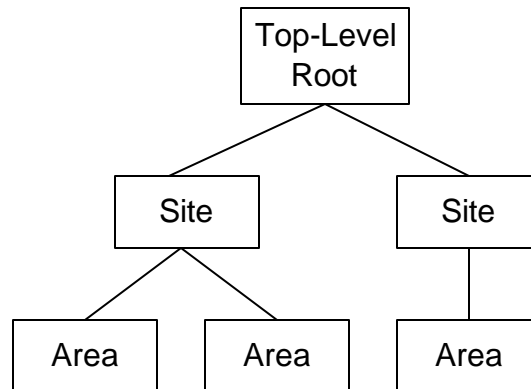
```

Note that not all of the WebL script to manipulate the eGroups HTML is in the functions above.

Introduction to Security in the Weblet Foundation

Vision: Support OpenPortals that are completely open or completely closed, and everywhere in between – and let the users decide which they want through an easy interface.

OpenPortal allows users to create Sites. One OpenPortal server can have several Sites, all below a top-level root. Each Site can also have multiple Areas beneath it.



Each Site and Area can define general policies on what kind of Easy Command Language a user can execute when within them. Within each Site Users can have Roles, such as Editor, Owner, Member, and Guest. Within each Area and Site a user's Roles can be used to either restrict or enable ECL commands.

<u>Site</u>	<u>Area</u>	<u>Role/User</u>	<u>ECL Command</u>
Linux Site	Main Area	Owner	Can edit all

The table above shows how one can restrict or allow ECL commands based on Roles, Users, Areas, and Sites. In this example any user who has the Role of Owner in the Main Area of the Linux Site can edit everything. The next table shows more examples of restricting commands based on roles and users. In the first line the user Paolo de Dios is given permission to edit everything ("Can edit all") in the Discussion Area. In the second line a default security setting is set for everyone ("Default for Everyone") so that everyone cannot edit anything. Permissions are enforced in the order they are given, so that permissions higher in the table below are enforced and can over-ride lower permissions.

<u>Site</u>	<u>Area</u>	<u>Role/User</u>	<u>ECL Command</u>
Linux Site	Discussion Area	Paolo de Dios	Can delete all
Linux Site	Main Area	Default for Everyone	Cannot edit all

Sites and Areas can hold other weblet containers, but cannot hold nested Sites or Areas. They can also set policies on whether children Areas, weblet containers, and weblets can over-ride the security settings of their parents. For example, in the table below anyone who has the role of being Owner in every Area in the Linux Site can change children permissions of sub-Areas or weblet containers, while the Default for Everyone is set so that the everyone cannot set Area permissions but can set a weblet container's permissions in the Discussion Area.

<u>Site</u>	<u>Area</u>	<u>Role/User</u>	<u>ECL Command</u>
Linux Site	All	Owner	Can set area permissions
Linux Site	All	Default for Every one	Cannot set area permissions
Linux Site	Discussion Area	Default for Everyone	Can set weblet container permissions

Weblets and weblet containers can also have their own security policies attached to themselves.

A web-based user interface is used to set these policies for each Area, Site, weblet, or weblet container. They all have the same general form, an example is shown below for setting the properties of a Site named Linux Site:

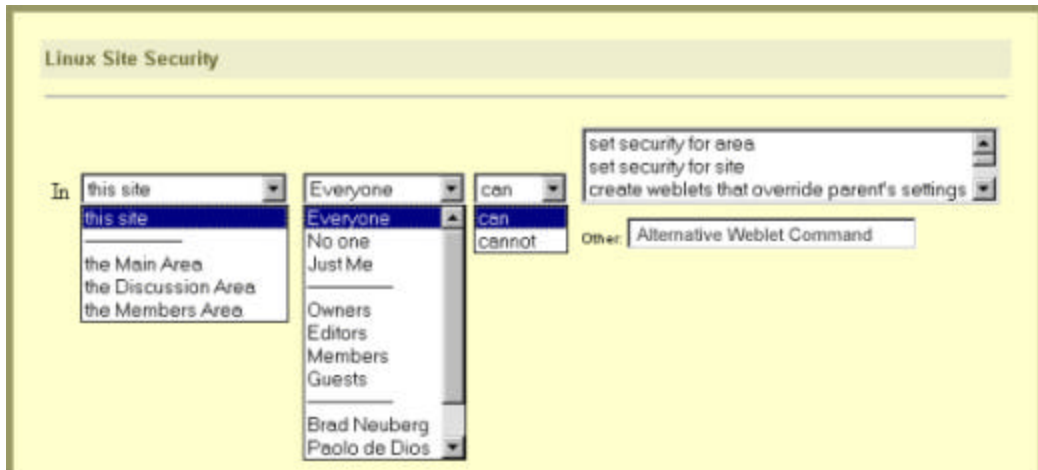
The screenshot displays two web-based configuration panels. The top panel, titled "Linux Site Security", features a form for defining security policies. It includes a sequence of dropdown menus: "In" (set to "this site"), "by" (set to "Everyone"), and "can" (set to "can"). A text box to the right contains a list of actions: "set security for area", "set security for site", "create weblets that override parent's settings", and "Alternative ECL Command". Below these fields is an "Add" button. A section titled "Security settings for this site in order of power:" contains a list of four security phrases, with the third one, "In this site Editors can edit all weblets", highlighted in blue. Below this list are "Remove", "Save", and "Change" buttons. The bottom panel, titled "Linux Site User Roles", shows a dropdown menu for "By default, all users are" set to "Owners". Below this, a section labeled "Roles:" contains a list of five user roles: "Brad Neuberg is an Owner", "Christopher Tse is an Editor", "Julio Hershberg is an Editor", "Jane Poland is a Member", and "Bryan Pitts is a Member". At the bottom of this panel, there is a "Remove" button, a "Save" button, an "Add" button, a "Change" button, and a summary line showing "Brad Neuberg" is an "Owner".

The user interface has two sections; a top section in which security settings are set by creating the appropriate phrase from pull-down menus, and a bottom settings where all the security settings for the site are listed. There is also a bottom section for assigning users different Roles for Sites. For example, in the screenshot above the top section has the following security phrase spelled out from the pull-down menus:

In this site Everyone can set security for area

These new phrases can be added to the site by hitting the "Add" button, and the new phrase will be added after whatever phrase was highlighted in the lower section.

The pull downs for the top-down section are as follows:



In the upper right portion of the user interface is a scrolling list that has all possible ECL commands enumerated (i.e. "set security for area", "set security for site", etc.). The ECL commands which can have security set on them are as follows, with descriptions where appropriate.

- set security for area
- set security for site

These two commands give someone permission to set the security properties for an area or a site. If someone is allowed to set the security, a form similar to the ones above is returned.
- create weblots that override parent's settings

This gives a user permission to create a weblet that can override the weblet's parent security, possibly allowing more permissive or restrictive use of the weblet then the parent would provide. For example, using this setting would allow someone to create an editable Article weblet in an area where nothing can be edited.
- assign all roles
- assign Owner role
- assign Editor role
- assign Member role
- assign Guest role

These five commands gives a user the power to assign roles to other users in an area or a site. For example, a user could be given the power to assign the Member role to a new user.
- other (fill in command in box below)

This selection is used for typing in ECL commands that have not been enumerated. This is commonly used for setting ECL commands on individual weblots (i.e. edit "My Business Card").
- do everything with all weblots

This gives a user free reign over all weblots in an area, though this does not give them permission to change a site or area's security settings or to assign roles.
- edit all weblots
- view all weblots
- delete all weblots
- clone all weblots

These ECL commands give a user permission to run edit, view, delete, or clone commands on any weblet.
- set security for all weblots

This allows a user to change the security properties of a weblet; note that this does not include the ability to change the security permissions of the area or site.

- create all weblets
- create Business Card Weblet
- create Normal Page Weblet
- create Article Weblet
- create Toolbar Weblet

Every available weblet is enumerated and a 'create' option is put into the list. This allows one to restrict the creation of certain types of weblets to certain users.

- do everything with all Business Card Weblets
- edit all Business Card Weblets
- view all Business Card Weblets
- delete all Business Card Weblets
- clone all Business Card Weblets
- move all Business Card Weblets
- set security for all Business Card Weblets
- do everything with all Normal Page Weblets
- edit all Normal Page Weblets
- view all Normal Page Weblets
- delete all Normal Page Weblets
- clone all Normal Page Weblets
- move all Normal Page Weblets
- set security for all Normal Page Weblets

For each type of weblet all possible commands that can be run on this weblet is enumerated.

Above are two example enumerations for Business Card Weblets and Normal Page Weblets.

The bottom portion of the security form shows all the security settings for the site. Three buttons can be used to manipulate these: 'Remove', 'Save', 'Add', and 'Change'. Hitting Remove removes a highlighted security setting from both the client and server. Hitting Save saves a modified ECL command and all modifications. Hitting Add causes the ECL command that has been specified in the top-portion of the user interface to be inserted into the bottom portion. Hitting Change loads the selected ECL command into the top-portion.

There is also a bottom section for assigning users different Roles for Sites. The form to do this is located at the bottom of the Site form above. The default role for all users can be set with this form. Roles are listed in a list-box, and can be Removed, Saved, Added, and Changed by clicking on the appropriate buttons and selecting from the lower pull-downs (i.e. "Brad Neuberg" is an "Owner").

The form for setting an area's security policies looks similar to the site security form:

The screenshot shows the 'Discussion Area Security' form. At the top, there's a title bar. Below it, the form is divided into sections. The first section has a label 'In the Discussion Area' followed by a dropdown menu set to 'Everyone' and a 'can' dropdown. To the right is a large text box containing three options: 'set security for area', 'create weblets that override parent's settings', and 'other (fill in command in box below)'. Below this is an 'Add' button. To the right of the 'Add' button is a text field labeled 'Other:' with the text 'Alternative ECL Command'. The second section is titled 'Security settings for this area in order of power:' and contains a list box with two entries: 'In the Discussion Area Editors can clone all weblets' and 'In the Discussion Area Everyone can view all weblets'. Below the list box are three buttons: 'Remove', 'Save', and 'Change'.

All that is different is that the Area is already restricted and the ECL command 'set security for site' is removed upper right box. Also, Areas cannot have their own assigned roles; roles are only assigned from Sites.

Weblet Containers also have their own security properties form:

The screenshot shows the 'Weblet Container Security' form. It has a similar layout to the Discussion Area Security form. The first section has a label 'In this container' followed by a dropdown menu set to 'Everyone' and a 'can' dropdown. To the right is a large text box containing two options: 'create weblets that override this container's settings' and 'other (fill in command in box below)'. Below this is an 'Add' button. To the right of the 'Add' button is a text field labeled 'Other:' with the text 'Alternative ECL Command'. The second section is titled 'Security settings for this area in order of power:' and contains a list box with two entries: 'In this weblet container Editors can clone all weblets' and 'In this weblet container Everyone can view all weblets'. Below the list box are three buttons: 'Remove', 'Save', and 'Change'.

The upper-right ECL command box includes all the same commands as the Site box, without the 'set security for area' and 'set security for site' commands.

Weblets have the simplest security settings:

Business Card Weblet Security

In the Discussion Area

Security settings for this weblet in order of power:

The upper-right ECL command box has the following commands:

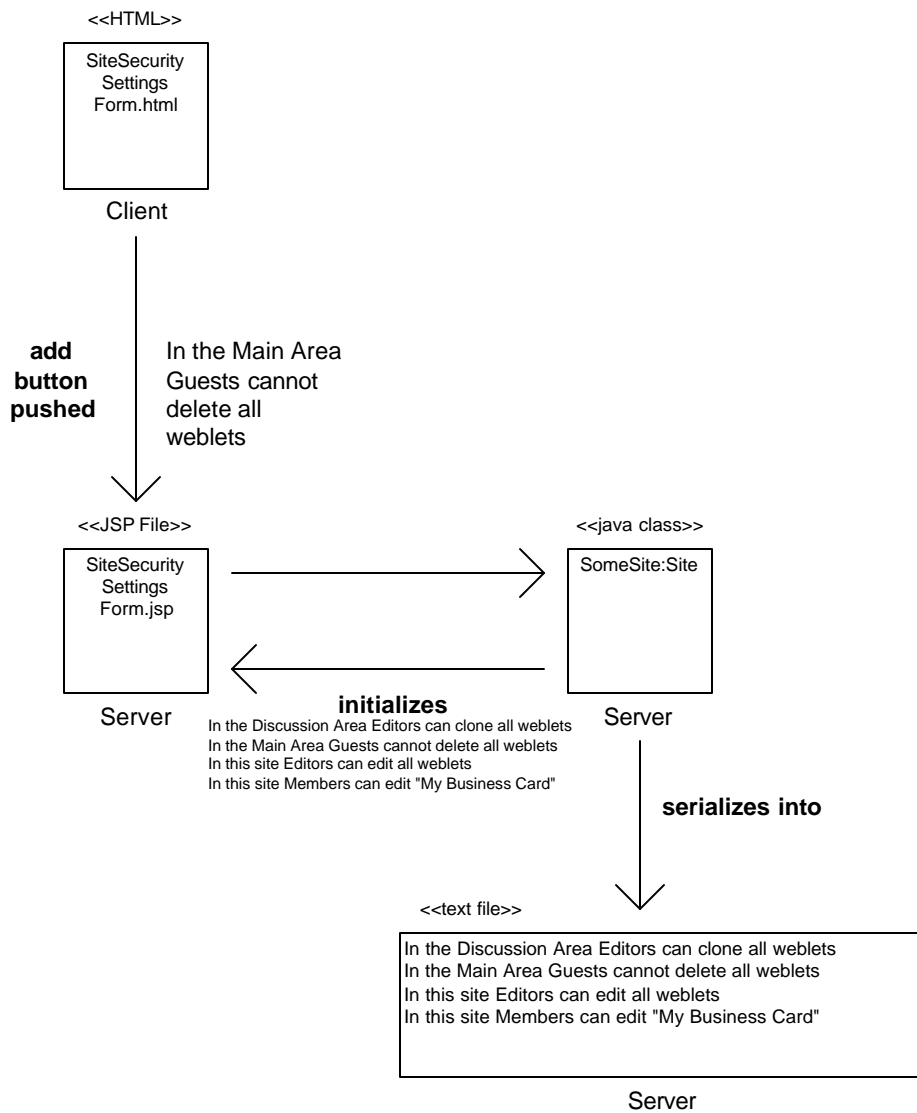
```
do everything to this weblet
edit this weblet
view this weblet
delete this weblet
clone this weblet
set security for this weblet
```

Easy Command Language Security Format

Security settings are converted and stored as ECL commandss. This makes it possible to simplify and encapsulate interacting portions of the Weblet Foundation. This is used in the following sub-systems:

- communication between client and server when the user is setting security properties using the site security form, the area security form, the weblet security form, and the weblet container security form
- serializing a weblet's security settings to a flat text file in a local file system through the JNDI
- interaction on the server side between the JSP files for the site, area, weblet, and weblet container security forms and the actual java weblet objects

These interactions are shown in the diagram below:



This section details how security permissions are transformed into security ECL commands.

Security ECL commands are pretty much straight translations from the security form:

becomes the security ECL command:

In this site Everyone can set security for area

This becomes the command-URL (if the user is Brad Neuberger):

http://www.openportal.org?command=In+this+site+Everyone+can+set+security+for+area&username=Brad+Neuberger&site=Linux+Site&weblet_name=/Linux+Site&weblet_type=Site&openbasis_version=2.0

The context-free grammar of security ECL commands are as follows:

SecurityWebletCommand

```
SecurityWebletCommand ::= In Location Identity Permission
                        WebletCommand
```

Location

```
Location ::= this site | All-areas | this weblet container
```

All-areas

```
All-areas ::= area1 | area2 | area3 ... arean
```

Discussion: If an area name does not begin with the word "the" or "The", then an optional "the" can be appended before this name (i.e. "the Discussion Area"). The security forms do this when they are set. If the area name already has "the" or "The", then the security forms do not add the "the" or "The." For manually entered security ECL commands the "the/The" can be dropped (i.e. "In Discussion Area....").

Permission

```
Permission ::= can | cannot
```

Identity

```
Identity ::= Everyone | everyone | No one | no one | Just me | just
           me | Owners | owners | Editors | editors | Members | members
           | Guests | guests
```

WebletCommand

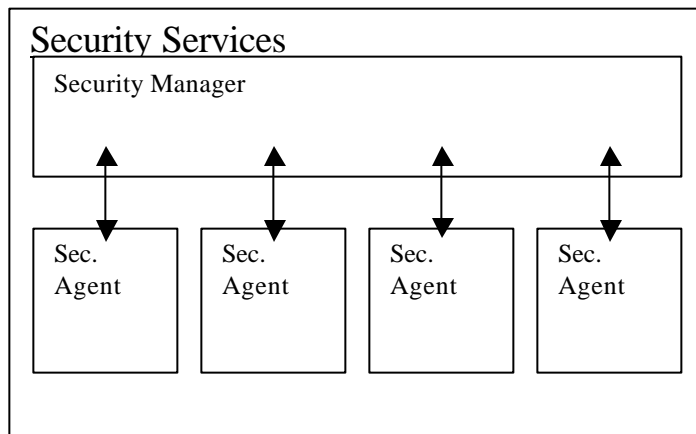
`WebletCommand ::= A valid ECL command`

Discussion: To avoid recursion SecurityWebletCommands cannot be used here, except for "set security for area" and "set security for site" which are actually not security ECL commands but are normal ECL commands.

Weblet Foundation Security: Architecture and Design Overview

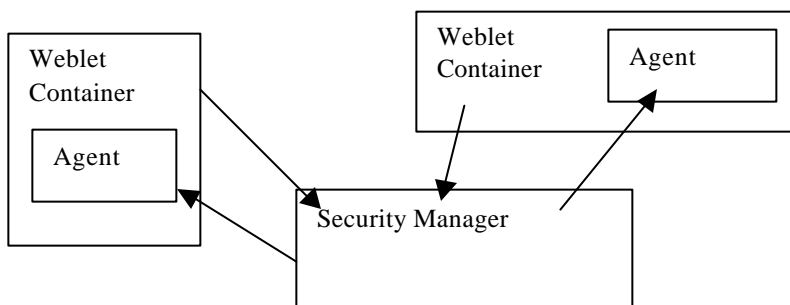
Iteration 1

In order to maintain a secure web environment, security features must be implemented to authenticate users and weblets in order to verify that all weblet and ECL command operations are from trusted users. In order to manage security auditing/validation etc, the Weblet Foundation will implement a Security Service as addition to its service layer architecture. Security Services consists of two main components, the Security Manager and the Security Agent.

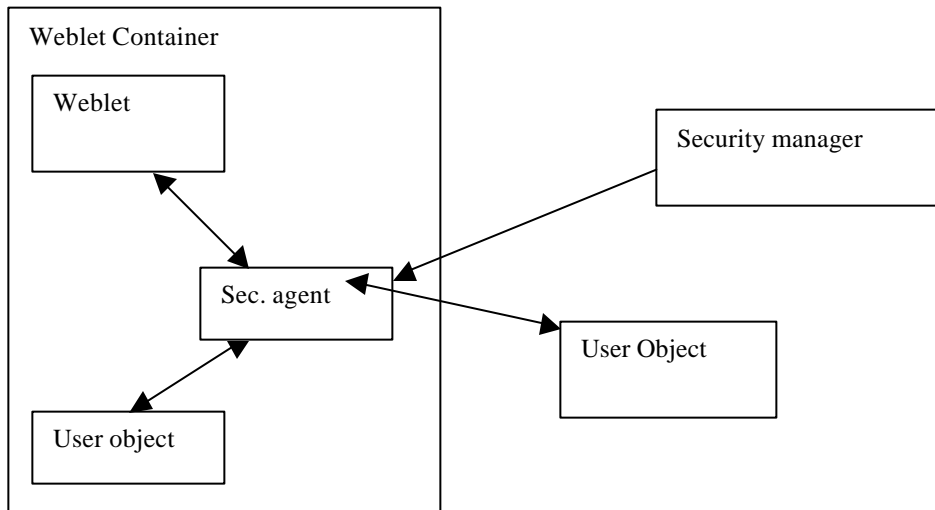


Essentially, the Security Manager is responsible for creating policies, by mapping object sources (weblets, users, other sites and areas) to sets of permissions for its Security domain, and then dispatching security agents to user or weblet objects to enforce these policies. Thus, there will be no central choke point for security enforcement that can be responsible for DoS (Denial of Service) attacks or spoofs, but still providing a centralized location for the administration of system policies. Security agents are an enhancement of a ticket based authentication system. Instead of simply encapsulating a single policy between two objects, security agents encapsulate all operations to validate communications between a group of objects given a collection of policies or permissions. It keeps weblet and user objects lightweight by delegating additional security logic to another component that can be easily maintained and changed.

In order to ensure that all weblets will be under the auspices of the security manager, a weblet container must register itself with the security manager. The security manager would then assign a security agent to the weblet container.



These agents will be responsible for enforcing security policies for weblets belonging to the same set.



The security manager will keep track of and manage these agents. It will manage security agent lifetimes and operations for logging and auditing. This architecture can even be extended to support session tracking

Security Manager:

- Keeps track of object sources
- Manages security agent lifetimes and activities
- Instantiates/assigns agents to specific domains

Security Agent:

- Queries weblets/weblet containers/sites in its immediate protection domain
- Maintains a permissions collection
- Maintains a policies collection
- Enforces security policies
- Has the same life cycle as its container/site

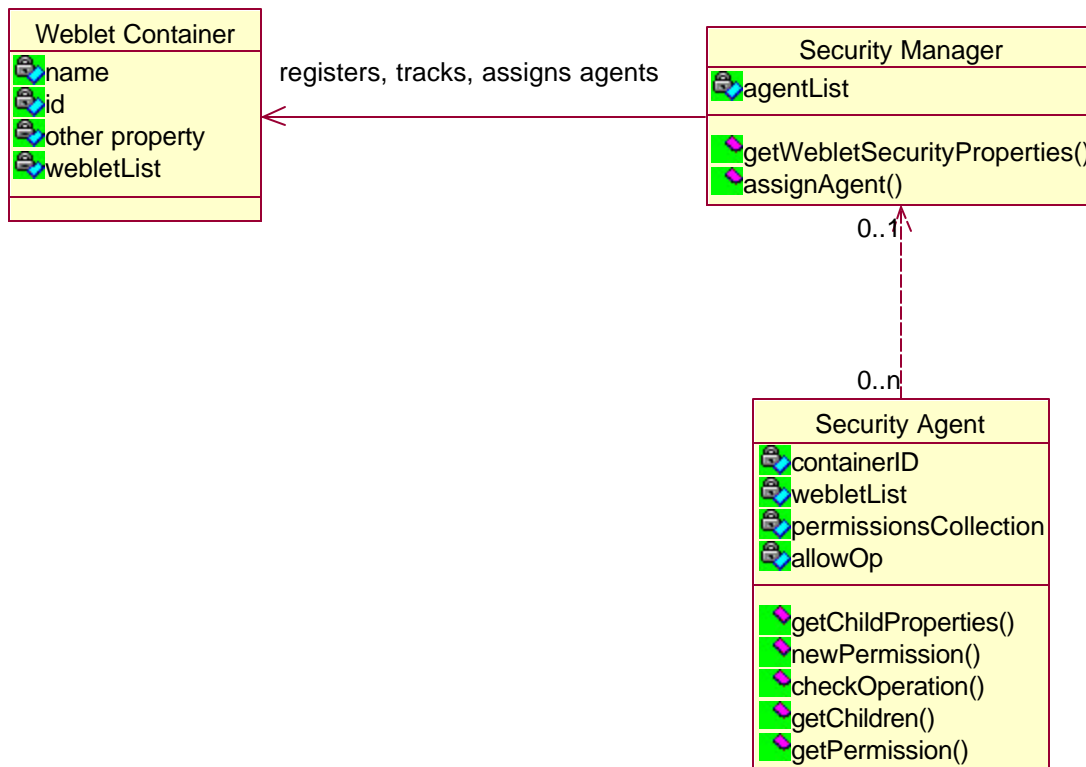
High Risk Areas/Open Issues:

- Relatively complex implementation
- Performance/scalability issues. Security should not take a relatively large portion of processing time.
- Client side authentication issues, e.g. sending passwords over the Internet.
 - Options
 - SSL 3.0
 - Client certificates
 - Base64 encryption

Weblet Foundation: Security Services Architecture Iteration 1

The security service architecture is modeled after a security manager and agent design pattern, whereby a security agent enforces all security policies applicable to the immediate objects in the site or area level.

In order to make itself known to the security system, the weblet container must register itself with the security manager. The security manager queries the weblet container/site for its security properties and then instantiates and assigns a security agent to that weblet/site container



The security agent is responsible for maintaining all the weblets in its protection domain. It queries all weblets added into the container for security properties and instantiates and maintains new permission objects. These permission objects in turn establish policies associated with this permission. In this way, a security check can be established by first seeking an operation in the permission list and then searching through the policy collections for an object-operation-object mapping. For example, if a user, Paolo, attempts to edit an article weblet, the security agent will search through the available operations in the permission list. Upon finding an edit operation, it will search the policy lists for a "Paolo" to "article weblet" mapping. A similar traversal is done to change permissions and policy mappings using the get/set methods. Perhaps the only operation that is not delegated to a security agent is the destruction of a weblet container. The security agent in charge of the parent container/directory context handles that operation.

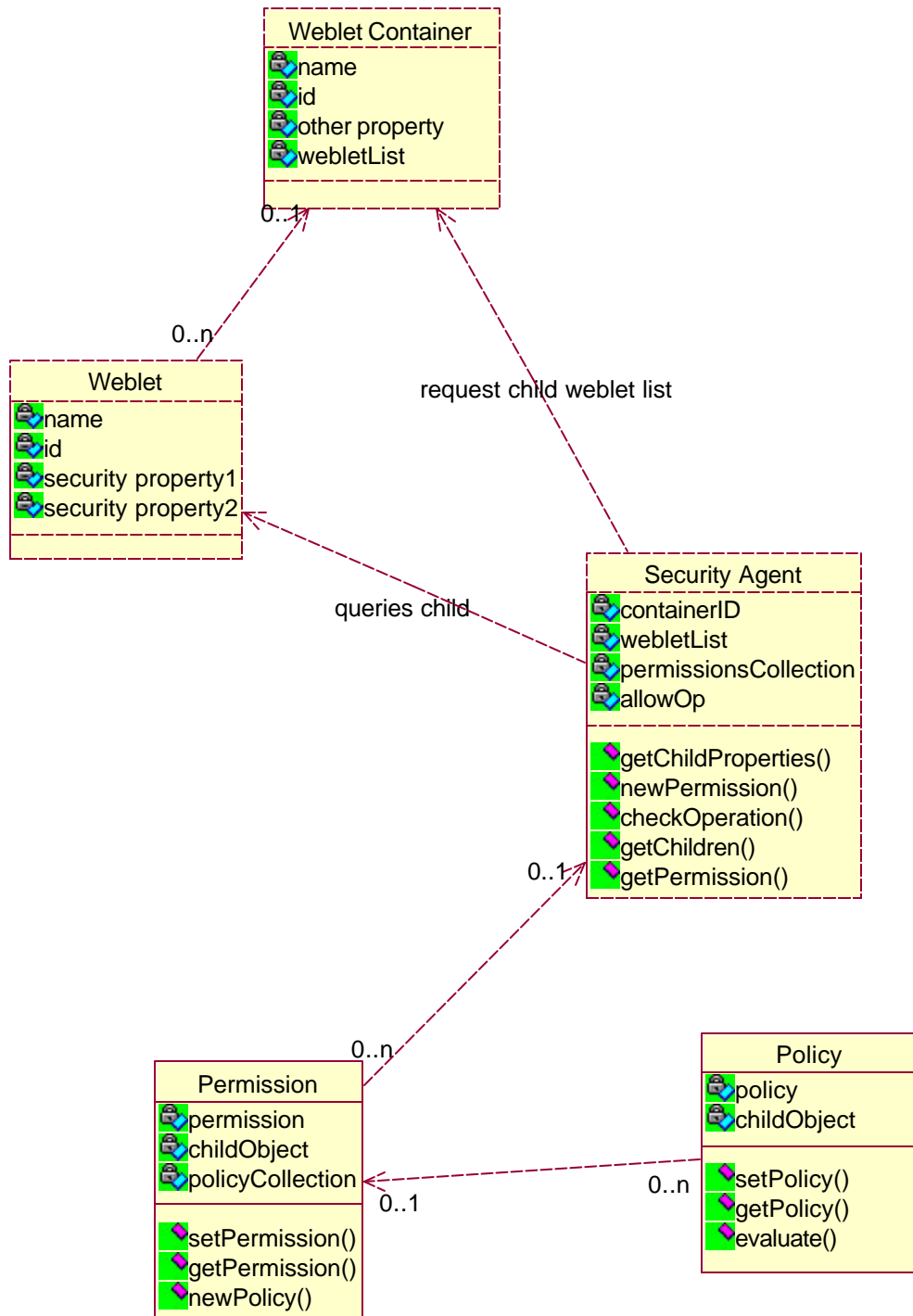


Table of Contents

1.Introduction	
1.1 Purpose	
2.Architectural Analysis	
2.1 Component Model	
2.2 Behavior Model	
2.3 Enterprise Model	
2.3.1 Logical Component Classifications	
2.3.2 Logical Behavior Classifications	
3.System design	
3.1 Design Views	
3.1.1 Logical Component View	
3.1.2 System Layered View	
3.1.3 System Deployment View	
3.2 Object Model	
3.2.1 Design Component Specifications	
3.2.2 Object specifications	
3.3 Operations Model	
3.3.1 Detailed Behaviors	
3.3.2 Operations Specifications	
3.4 Class Model	
4.Common Definition Language	
5.Appendix	

Introduction

The purpose of the System and Software Architecture Definition is to describe the structural, relational and behavioral mappings of the different components of OpenPortal. It attempts to map the requirements and responsibilities of OpenPortal into its software level abstractions and structure. Implementation specific details, such as algorithms and method specifications will be left to Javadoc source code documentation. This document aims at concurrency with major releases of the system.

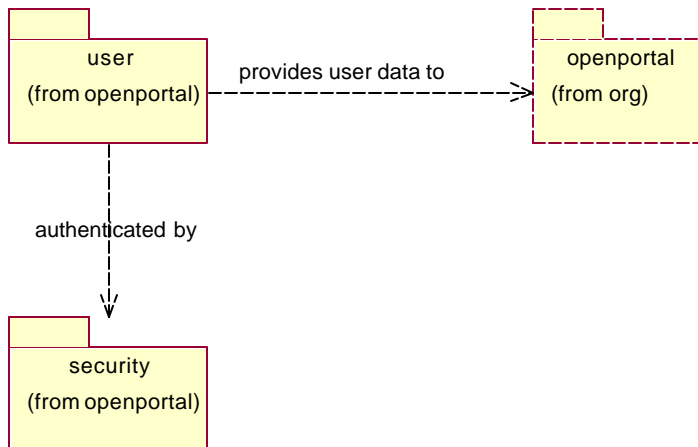
Architectural Analysis

Component Model

The Component Model is derived from OpenPortal's system responsibilities. OpenPortal consists of four custom components that utilizes a number of design components.

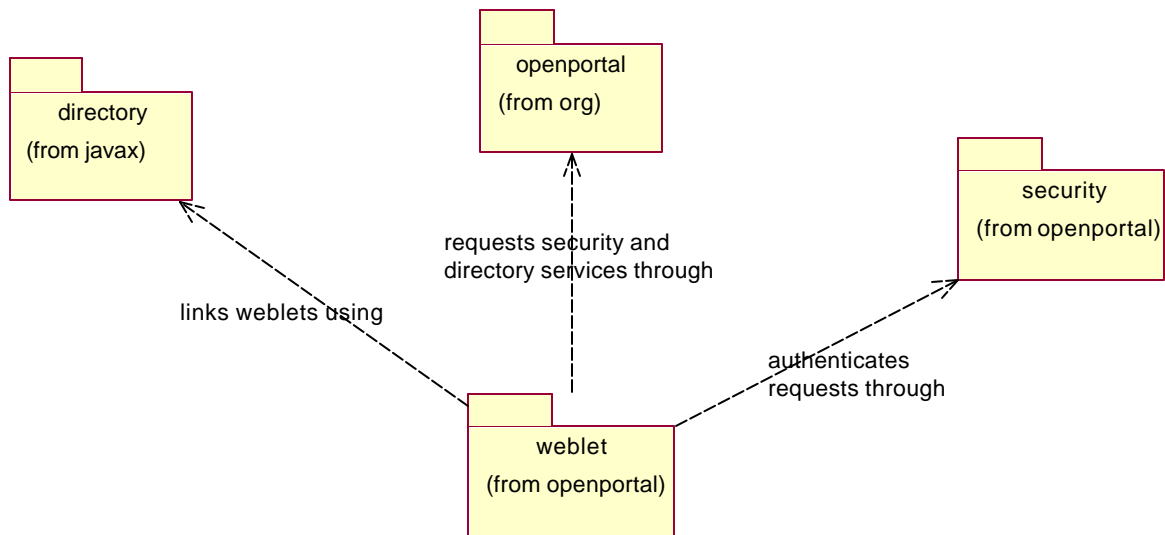
Component		COM-01
Defining Quality		Responsible for creating and maintaining user profiles
Name		User (org.openportal.user)
Attributes		Contains persistent user data collection
Behaviors		Represent user weblet service requests from OpenPortal
Relationships		weblet, and security
Roles		ProfileGenerator, user manager
Constraints		
	Dependencies	openportal
	Cardinality	1

COM-01 Relationships



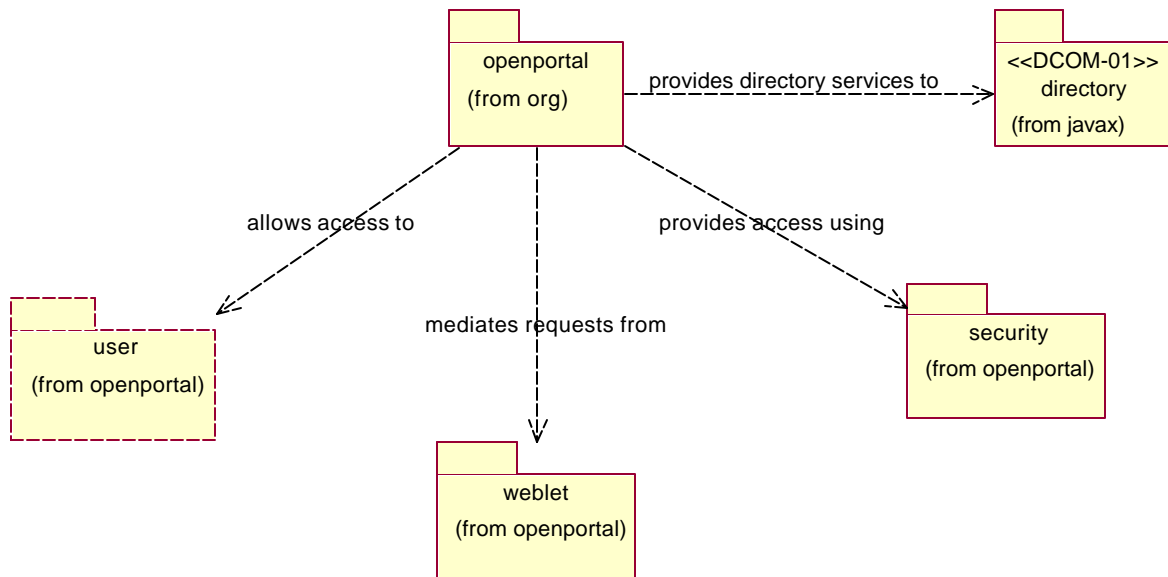
Component		COM-02
Defining Quality		Creates weblets and processes WSP and Weblet Command requests
Name		Weblets(org.openportal.weblet)
Attributes		
Behaviors		Create weblet, process WSP and weblet commands
Relationships		OpenPortal, directory, security
Roles		Weblet factory, weblet compiler
Constraints		
	Dependencies	GNU JSP, JNDI, security
	Cardinality	1

COM-02 Relationships



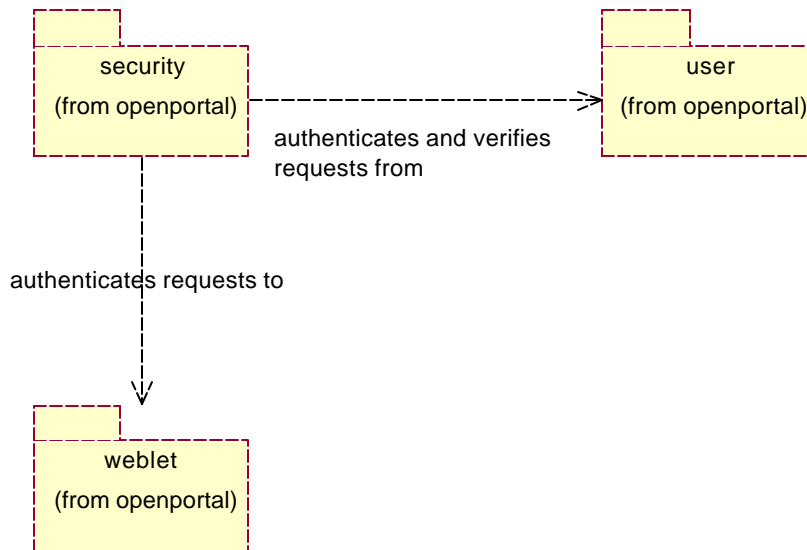
Component		COM-03
Defining Quality		Mediator of requests from other OpenPortal components
Name		OpenPortal (org.openportal)
Attributes		
Behaviors		Handle user requests, mediate service interaction
Relationships		User, weblet, security, directory
Roles		Request handler, OpenPortal service manager
Constraints		
	Dependencies	JNDI
	Cardinality	1

COM-03 Relationships



Component		COM-04
Defining Quality		Maintainer and enforcer of security policies
Name		Security (org.openportal.security)
Attributes		
Behaviors		Authenticate users, verify user requests, verify weblet command directives
Relationships		Weblets, user
Roles		Security agent, security manager
Constraints		
	Dependencies	user
	Cardinality	1

COM-04 Relationships

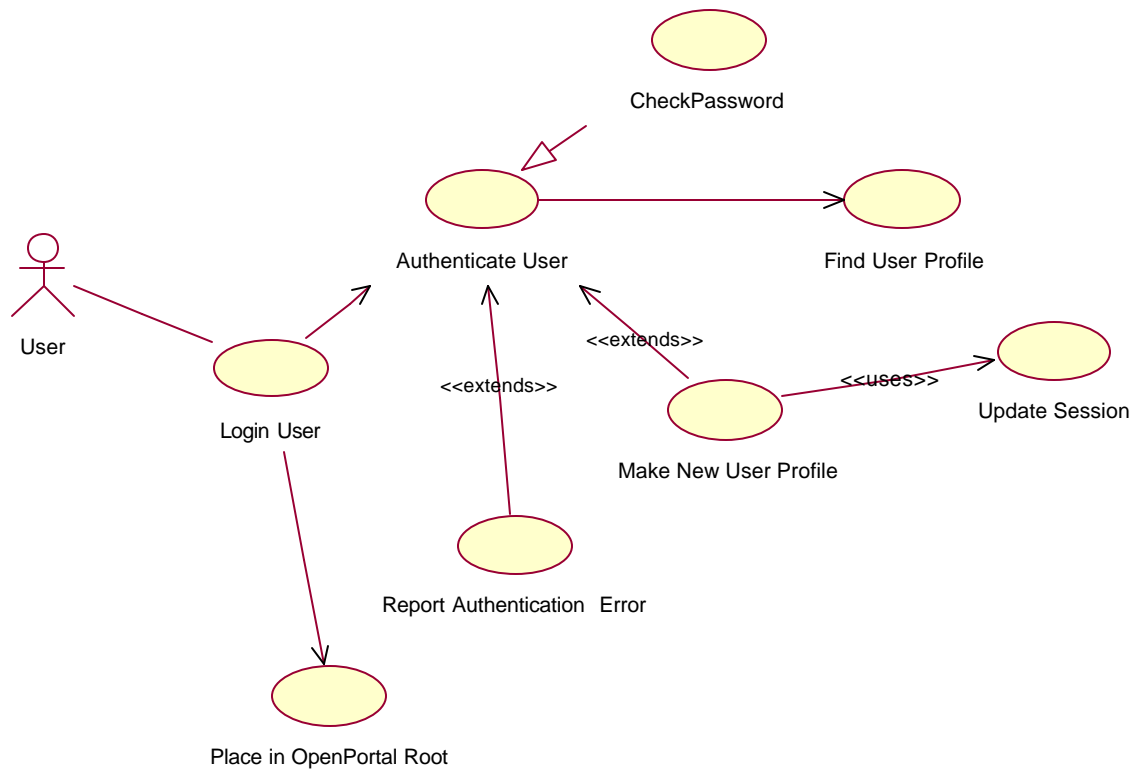


Behavior Model

The behavior model describes the different behaviors invoked by external actors or components that the system must handle. It is derived from the system responsibilities.

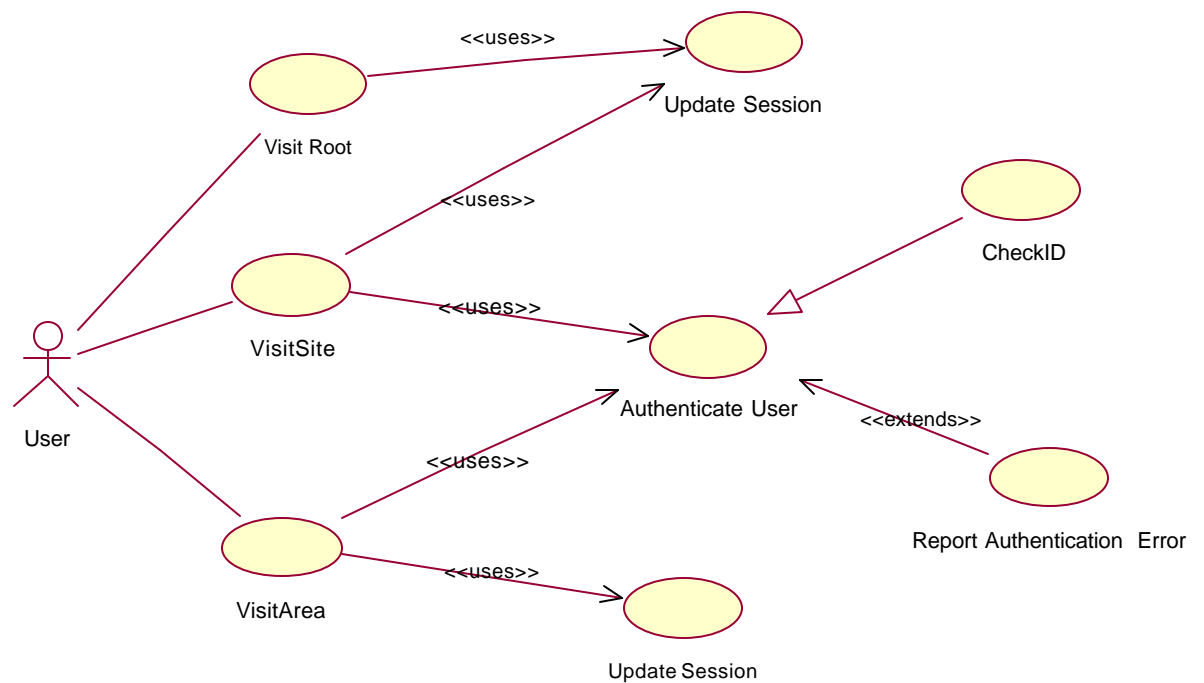
Identifier		BH-01
Name		Login to OpenPortal
Trigger		User submits a login and password
Description		Logs user into their home area, otherwise into the public area
Pre-Condition		User has a login profile
Post-Condition		User gets placed in their home area page
Input		ID, password
Output		User home page
Refers to		
Exception		No user profile, anonymous login

The user is not authenticated upon first entry into the OpenPortal website. Upon entry into the main page, the user is designated as guest and is allowed to view the public services available by the main page. To navigate other sites and areas listed in the openportal root, authentication may be required.

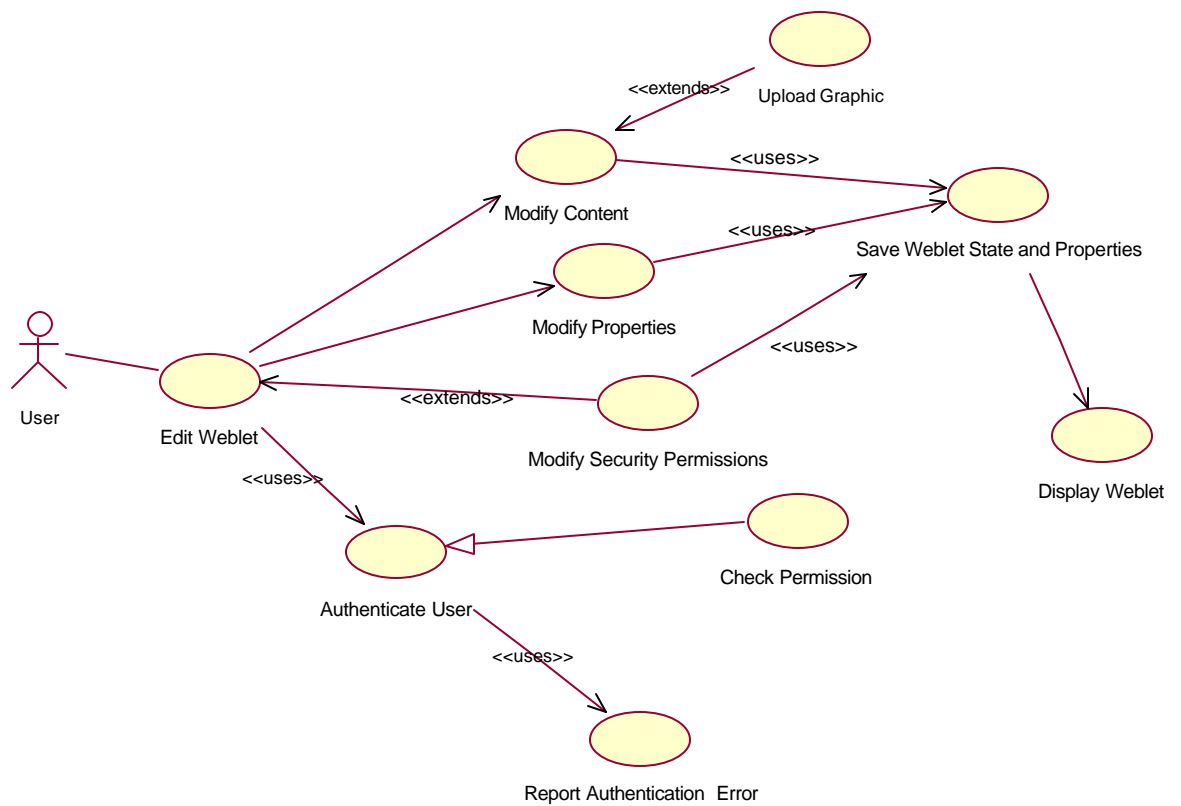


Identifier		BH-02
Name		User navigates OpenPortal site and area locations
Trigger		Hyperlink
Description		The user clicks on a hyperlink to follow an internal reference
Pre-Condition		Target page exists, user has appropriate viewing rights
Post-Condition		User is allowed to GET the requested page
Input		CommandURL
Output		HTML page
Refers to		
Exception		Page not found, user not allowed

The user is allowed to navigate the different *sites* and *areas* in OpenPortal provided that they have the permissions. Users are also assigned roles that have inherent permissions that will vary the behavioral use case of the system. Depending on a user's role or class, they may or may not have to be authenticated.

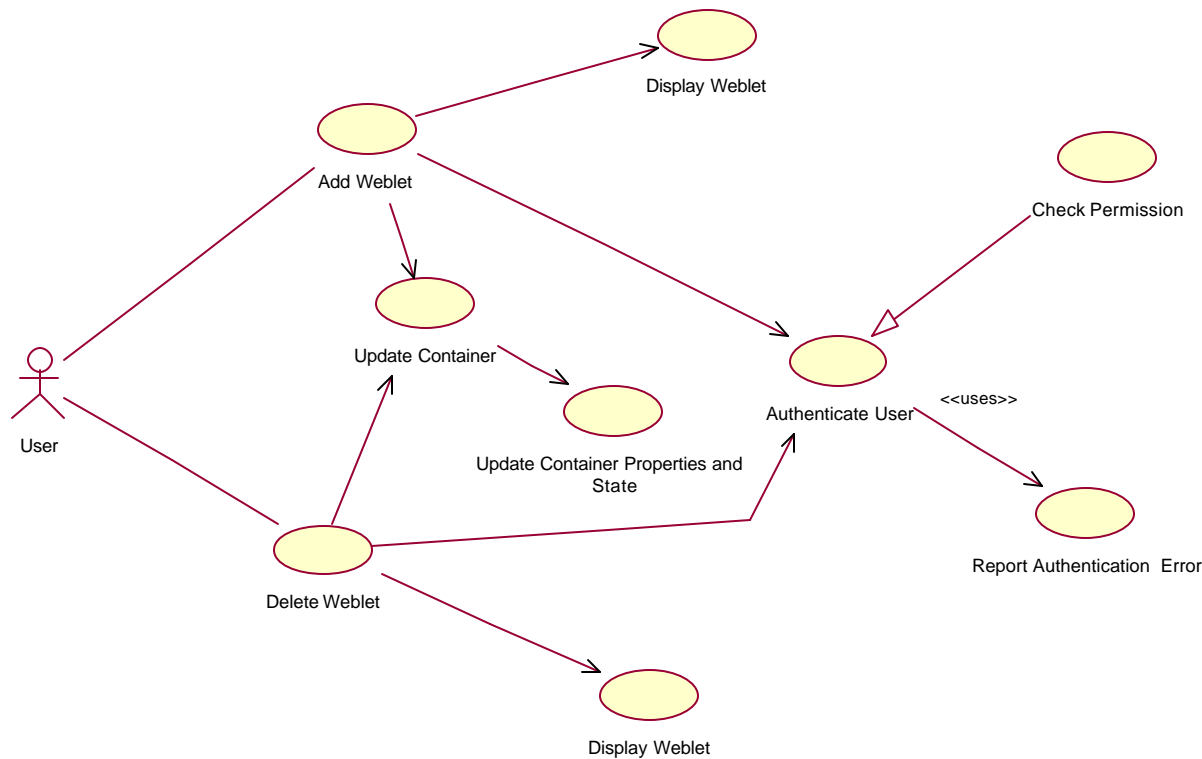


Identifier		BH-03
Name		Edit Weblet
Trigger		User Clicks on Edit
Description		Allows user to edit content, along with a weblet's properties, including security, positional and GUI elements.
Pre-Condition		User has the proper permissions
Input		Hyperlinked ECL CommandURL
Output		Edit form(type of edit form dependent on permission)
Refers to		
Exception		Permission denied



Identifier		BH-04
------------	--	-------

Name		Add or Delete a weblet
Trigger		Add or delete weblet command directive
Description		Adds or deletes a weblet from a container
Pre-Condition		User has the appropriate permissions
Input		[add or delete <weblet name>] commandURL or hyperlinked ECL commandURL
Output		Updated container page
Refers to		
Exception		Permission denied, weblet not found



System Design

Design View

Logical Component View
System Layered View
System Deployment View

Object Model

Design Component Specifications

These Components are derived from COTS products that must be configured or modified to work with OpenPortal.

Identifier		DCOM-01
Defining Quality		Naming and directory subsystem for weblet persistence
Name		JNDI (Java Naming and Directory Interface)
Attributes		
Assigned Behaviors		Lookup, search, bind, unbind
Relationship		OpenPortal Hub, Weblet Manager, Security Manager
Possible Roles		Directory manager, weblet caretaker
Constraints		Should provide access to the filesystem
Implementation		Java library

Identifier		DCOM-02
Defining Quality		Processes WSP directives and scriptlets
Name		JspServlet (PolyJSP engine)
Attributes		
Assigned Behaviors		Process WSP scriptlets, return HTML markup block
Relationship		WSP Engine
Possible Roles		WSP Compiler
Constraints		Should only process scriptlets and not JSP tags.
Implementation		Java based Interpreter package with a servlet interface

Identifier		DCOM-03
Defining Quality		Helps parse weblet command
Name		GNU RegExp – GNU Regular Expression Library
Attributes		
Assigned Behaviors		Parse command
Relationship		WebletCommand Parser
Possible Roles		Command tokenizer
Constraints		Should only return valid weblet command tokens
Implementation		Java based library

Identifier		DCOM-04
Defining Quality		Parses and interprets JavaScript
Name		ECMAScript

Attributes		
Assigned Behaviors		
Relationship		WSPEngine
Possible Roles		Interpreter
Constraints		
Implementation		Java based library and interpreter

Identifier		DCOM-05
Defining Quality		Parses and interprets WebL
Name		WebL3.0
Attributes		
Assigned Behaviors		Interpret
Relationship		WSPEngine
Possible Roles		Interpreter
Constraints		
Implementation		Java based library and interpreter

Identifier		DCOM-06
Defining Quality		Parse and process script files into XML structures
Name		XML4J
Attributes		
Assigned Behaviors		Parse weblet descriptor
Relationship		WSPEngine, Weblet Descriptor parser
Possible Roles		Xmlparser
Constraints		
Implementation		Java based library

OpenPortal Component Specifications

1. Main OpenPortal Component Framework.

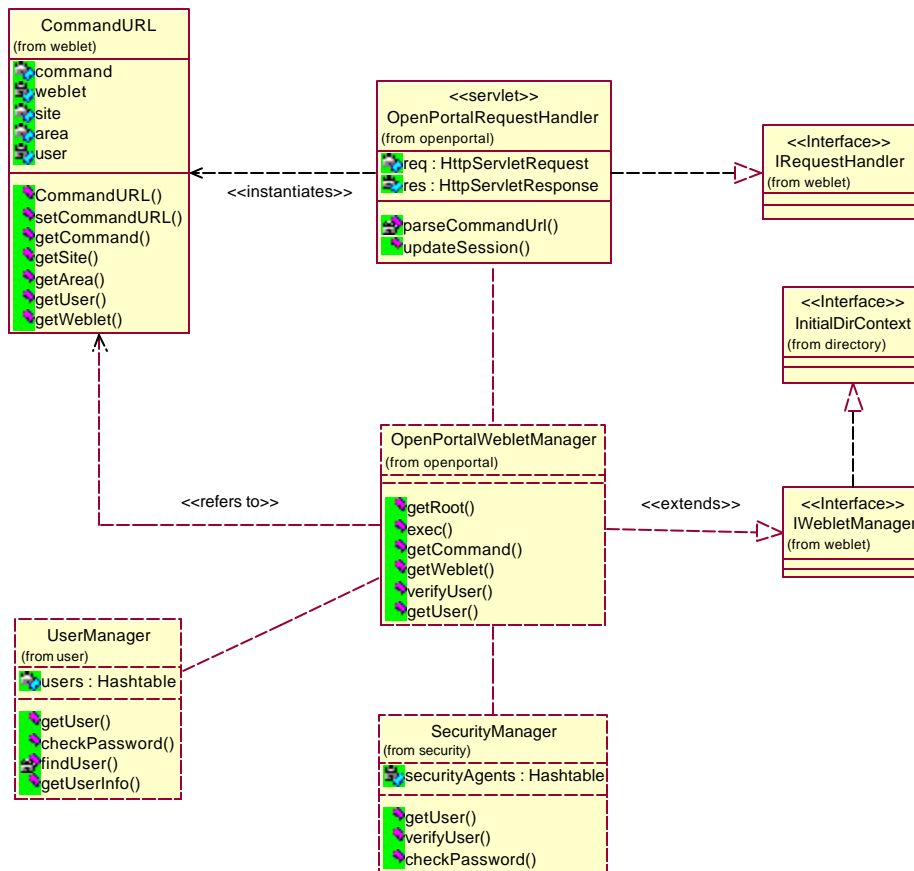
The Main OpenPortal framework establishes the relationships between the various sub-components that constitute OpenPortal. The OpenPortal framework is an extension of the weblet weblet framework. It extends the generic AbstractWebletManager to add security and user management capabilities. It is also the main component responsible for setting up an initial directory context that interfaces with JNDI (DCOM-01). All user HTTP requests are handled by the RequestHandler servlet and are processed as commandURL's. These commandURL's are used by the OpenPortalWebletManager to determine which weblet should be executed. The WebletManager also acts as a hub that allows weblets to communicate with the security and user managers.

Identifier		OBJ-01
Defining Quality		Handles user requests and outputs weblet responses
Name		RequestHandler
Object Interactions		Process user requests into CommandURLs Passes requests to the OpenPortalWebletManager
States		Handling request, handling response
Constraints		
Component membership		COM-01 OpenPortal
Refers to		
Implementation		Java Servlet

Identifier		OBJ-02
Defining Quality		Encapsulates user request and its context

Name		CommandURL
Object Interactions		OBJ-01 RequestHandler
States		
Constraints		
Component membership		OpenPortal
Refers to		
Implementation		Java class object

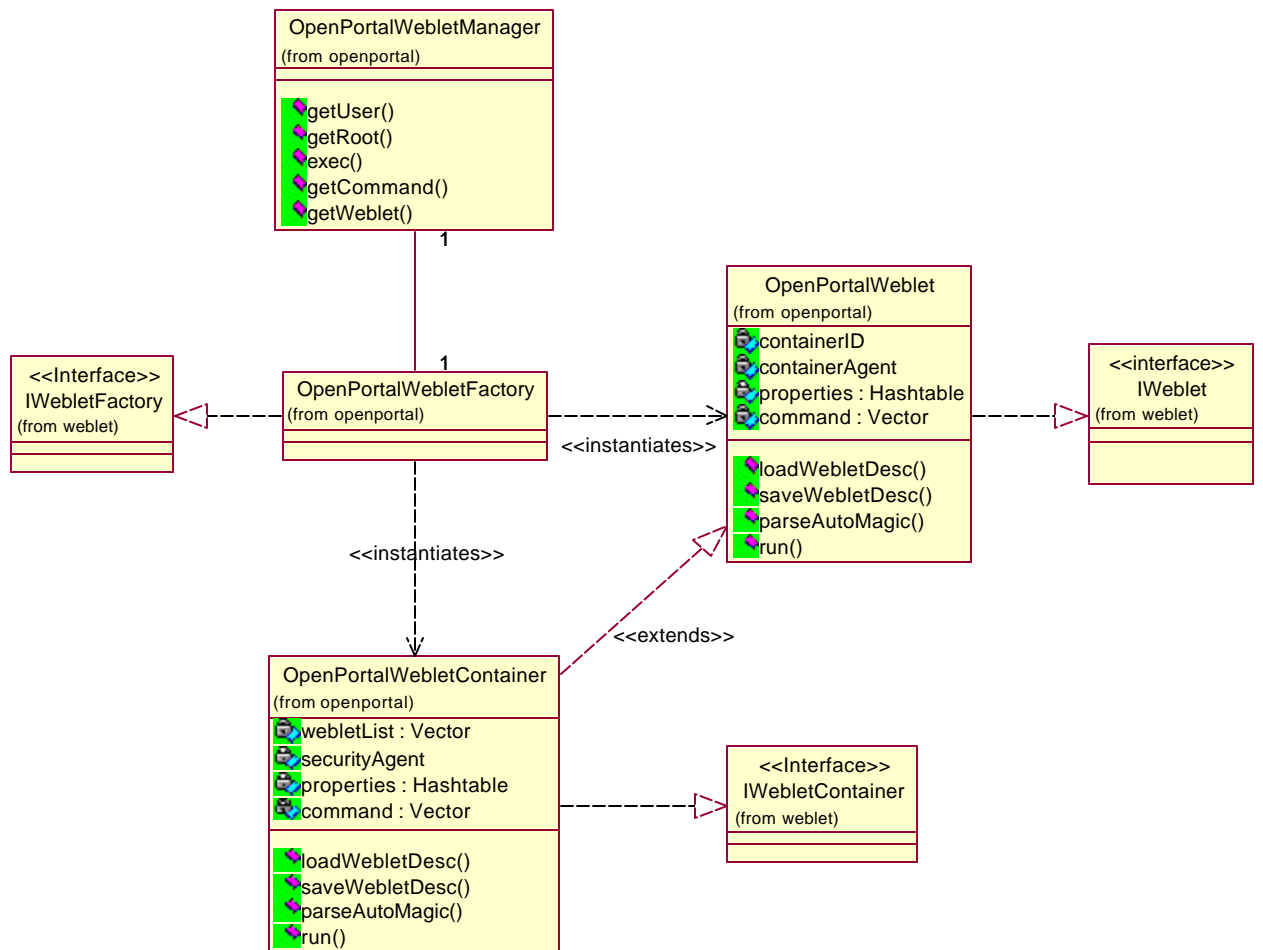
Identifier		OBJ-03
Defining Quality		Manages user requests to weblets and mediates weblet requests to the user and security managers
Name		OpenPortalWebletManager
Object Interactions		Request Handler, WSP Engine, ECL Engine, User Manager, Security Manager
States		
Constraints		
Component membership		Openportal, weblet
Refers to		
Implementation		Java class object



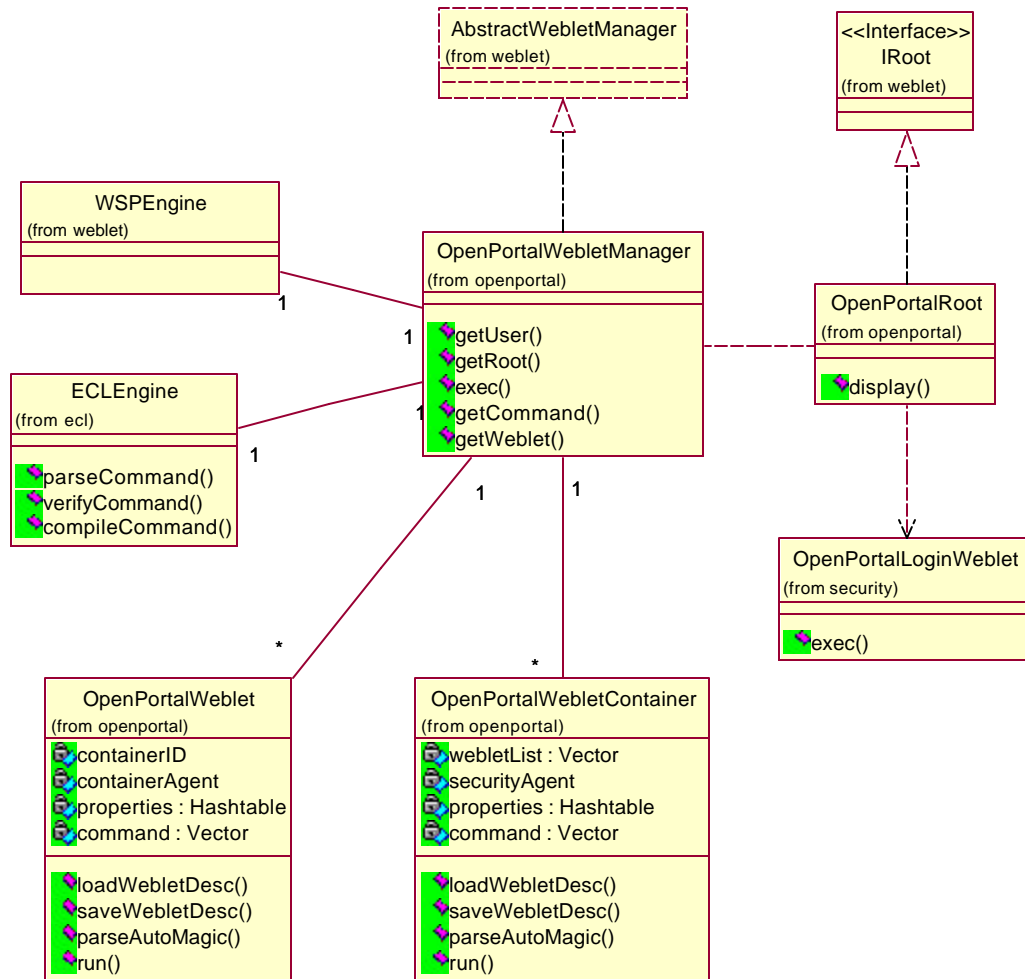
2. Weblet Framework

The weblet framework is one of the major components of the OpenPortal system. It is responsible for creating and managing weblets. It defines an interface for creating weblets and weblet containers through the weblet factory interface. The weblet manager is responsible for mediating requests to the weblet factory.

Identifier		OBJ-04
Defining Quality		Creates concrete weblet types
Name		OpenPortal weblet factory
Object Interactions		OpenPortal weblet manager
States		
Constraints		
Component membership		Weblets
Refers to		
Implementation		Java class object



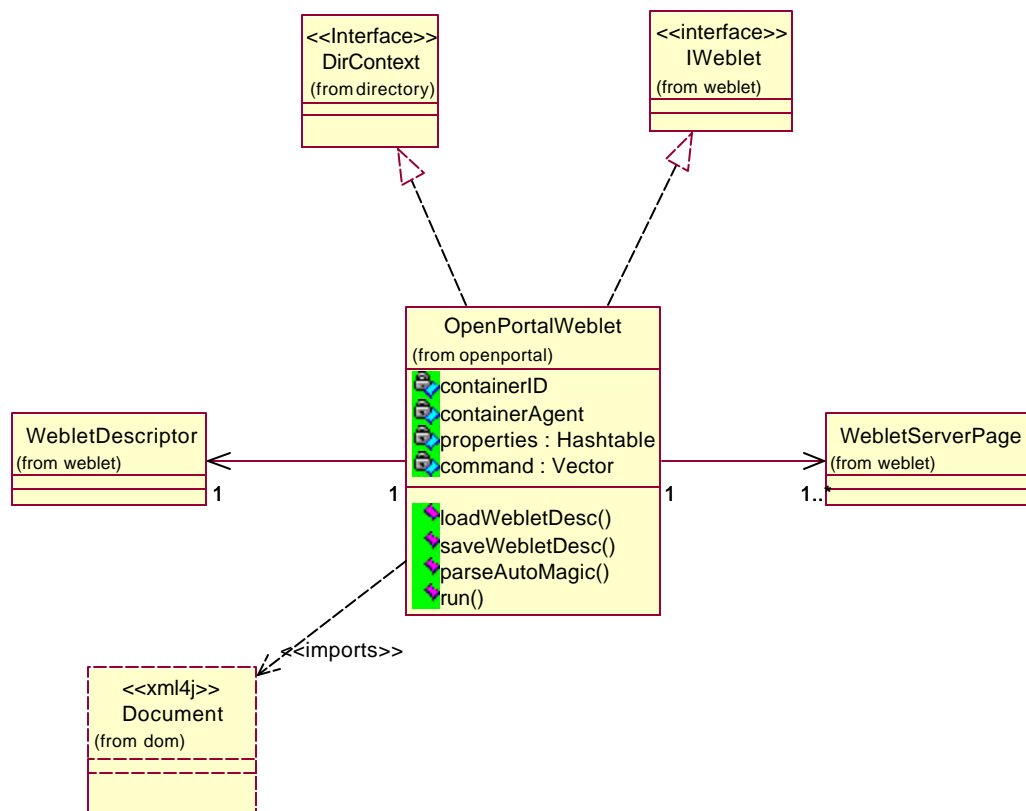
The weblet manager is also responsible for interfacing with the ECL and WSP engines. It also maintains a top level initial directory context with the OpenPortal Root object.



Identifier		OBJ-05
Defining Quality		Encapsulates a specific webpage or site functionality
Name		OpenPortalWeblet
Object Interactions		WebletServerPage, weblet descriptor, weblet factory, weblet manager
States		
Constraints		
Component membership		Weblet
Refers to		
Implementation		Java class object, WSP and weblet descriptor files

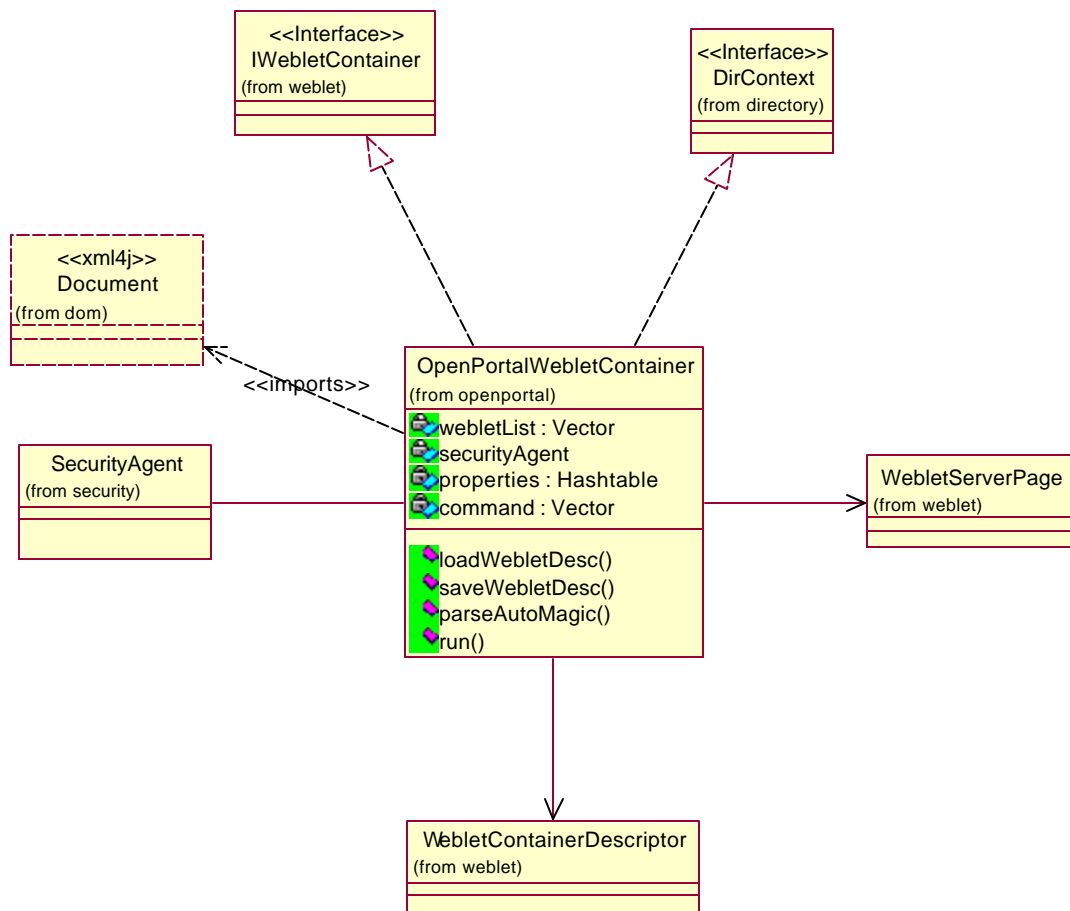
The basic OpenPortal weblet class implemets a standard weblet interface. It is required to implement the DirContext interface so that it can be stored in the filesystem using the JNDI. Weblets consist of the

weblet base class and the descriptor used to set the properties of the weblet. The weblet may consists of many WSP script files that embed content into weblet using the WSP engine.

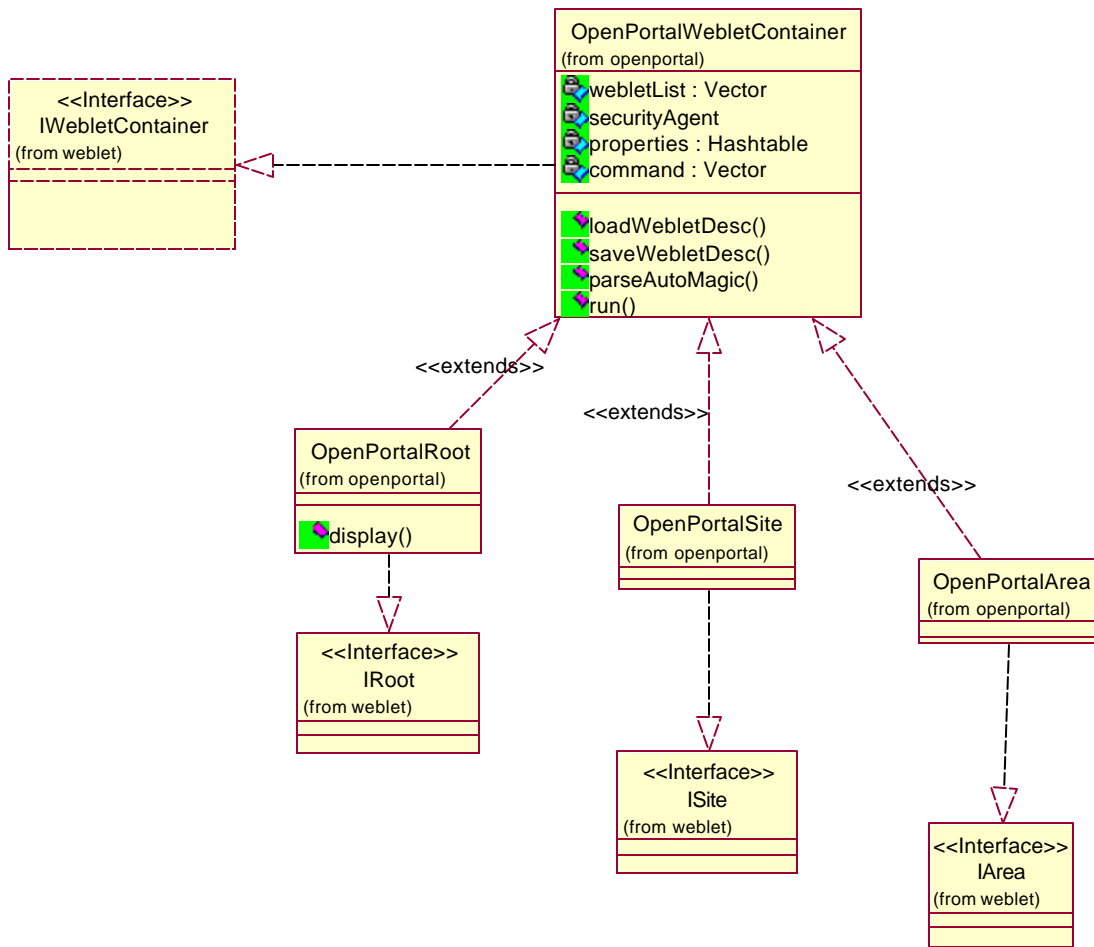


Identifier		OBJ-06
Defining Quality		Contains other weblets
Name		OpenPortalWebletContainer
Object Interactions		OpenPortalWebletManager, Weblet
States		
Constraints		
Component membership		Weblets
Refers to		
Implementation		Java class object, WSP and weblet descriptor files

Weblet containers implement the basic weblet container interface. They are required to implement the DirContext interface so that they can be stored in the filesystem using the JNDI. The weblet container class extends the weblet base class, adding capabilities for referencing and embedding weblets. Weblet Containers utilize weblet descriptors to initialize its properties and to determine which weblets it contains. Embedded WSP files are processed using the WSP engine. The weblets referenced by the container are processed in the order they are specified in the weblet container descriptor.

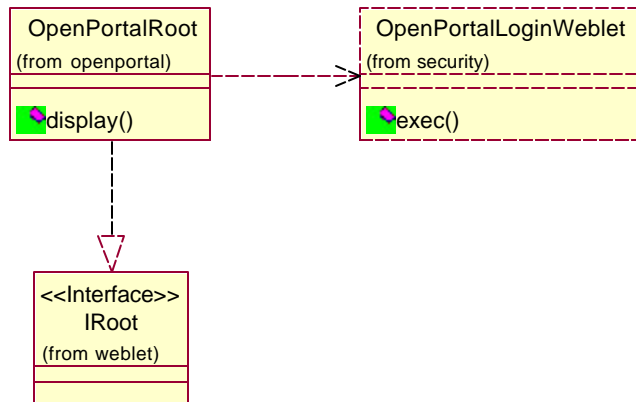


OpenPortal utilizes a hierarchical containment structure. These containers include the site root, a site, and an area. These different containers extend the base weblet container class and implements their respective container interfaces. Each container type places additional constraints in the nesting of weblets. The RootContainer can reference any weblet container. A SiteContainer can only reference AreaContainers and the weblet class. AreaContainers can only reference the weblet class.



Part of the standard weblet services of OpenPortal is the LoginWeblet contained at the root level of OpenPortal. In addition to logging users into OpenPortal, it is responsible for interfacing with the user manager to create new user profiles.

Identifier		OBJ-07
Defining Quality		Logs users in, interfaces with user manager to create new user profiles.
Name		LoginWeblet
Object Interactions		OpenPortalRoot
States		
Constraints		
Component membership		OpenPortal
Refers to		
Implementation		Java class object, weblet



3. ECL subsystem

Most command directives, enclosed with the “[“ and “]” or the “→” and “←” delimiters, are specified in an edit form or in the case of a security settings form, compiled from a series of pull down menus. Essentially, these commands pass through as either a singular block(s) of directives or part of a larger document context.

ECL commands are parsed through the parser, and turned into expression objects so that they can be more easily mapped into their corresponding command methods. After parsing, these expression objects are passed to the ECLProcessor for mapping to native objects. The command processor will execute the action associated with the command. It will find the corresponding HTML that realizes the command and pass it back through the chain of responsibility and then to the user.

In order to ensure the security of a specific command directive, the command must go through the security agent of the current containment context before it can be executed. The ECL engine will be able to verify the command by asking the weblet manager to run the command URL through the security agent. If a command directive is verified to be safe it will be allowed to execute in the command processor.

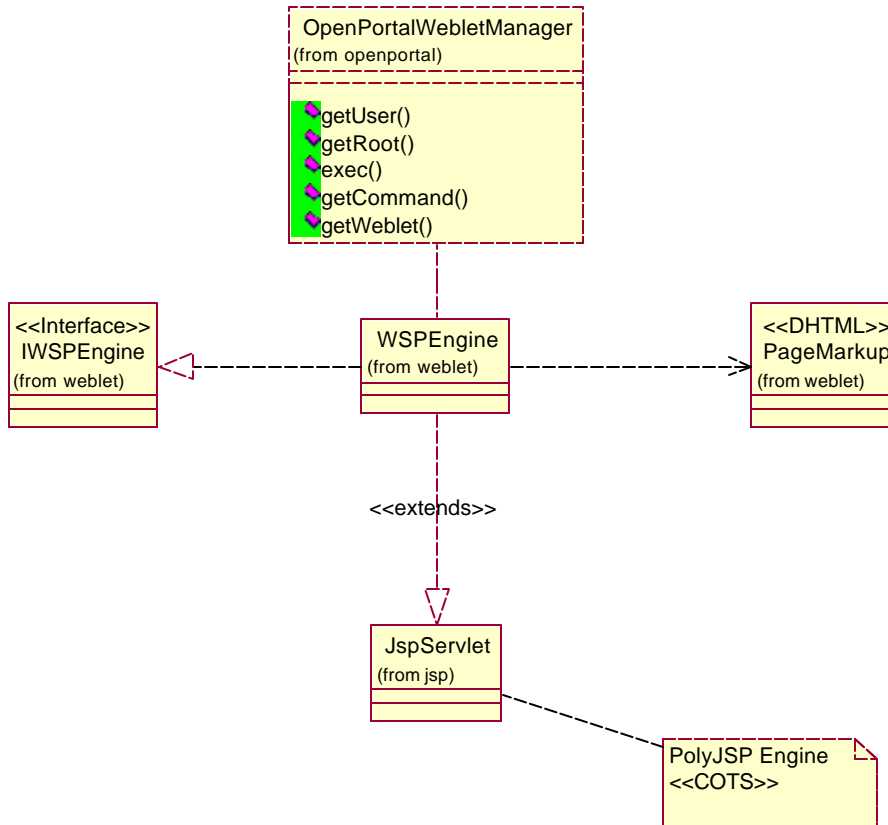
Identifier		OBJ-08
Defining Quality		Provides the interface to the parser and processor
Name		ECL Engine
Object Interactions		Weblet Manager
States		
Constraints		Command mappings are specified in the weblet descriptor only
Component membership		Weblet
Refers to		
Implementation		Java class object

Identifier		OBJ-09
Defining Quality		Parses ecl commands
Name		ECL Parser
Object Interactions		ECL Engine, ECL Expression
States		
Constraints		Can only parse commands based on the grammar defined by the regular expression
Component membership		Weblet
Refers to		
Implementation		Java class object

4. WSP subsystem

The WSP engine is an interface to PolyJSP engine (DCOM-02). It is responsible for passing “.wsp” files for processing and receiving the results for the weblet manager. It conforms to the WSPEngine interface for providing WSP services to the weblet manager.

Identifier		OBJ-12
Defining Quality		Interfaces with PolyJsp engine for wsp file processing
Name		WSPEngine
Object Interactions		DCOM-02 PolyJSP OpenPortalWebletManager
States		
Constraints		
Component membership		Weblet
Refers to		
Implementation		Java class object



5. Security Services

The security service architecture is modeled after the master and slave design pattern, whereby a security agent enforces all security policies applicable to the immediate objects in the site or area level.

In order to make itself known to the security system, the weblet container must register itself with the security manager. The security manager queries the weblet container/site for its security properties and then instantiates and assigns a security agent to that weblet/site container

The security agent is responsible for maintaining all the weblets in its protection domain. It queries all weblets added into the container for security properties and instantiates and maintains new permission objects. These permission objects in turn establish policies associated with this permission. In this way, a security check can be established by first seeking an operation in the permission list and then searching for a matching user-to-operation mapping. A policy file is also used to determine which permissions are mapped to a particular weblet.

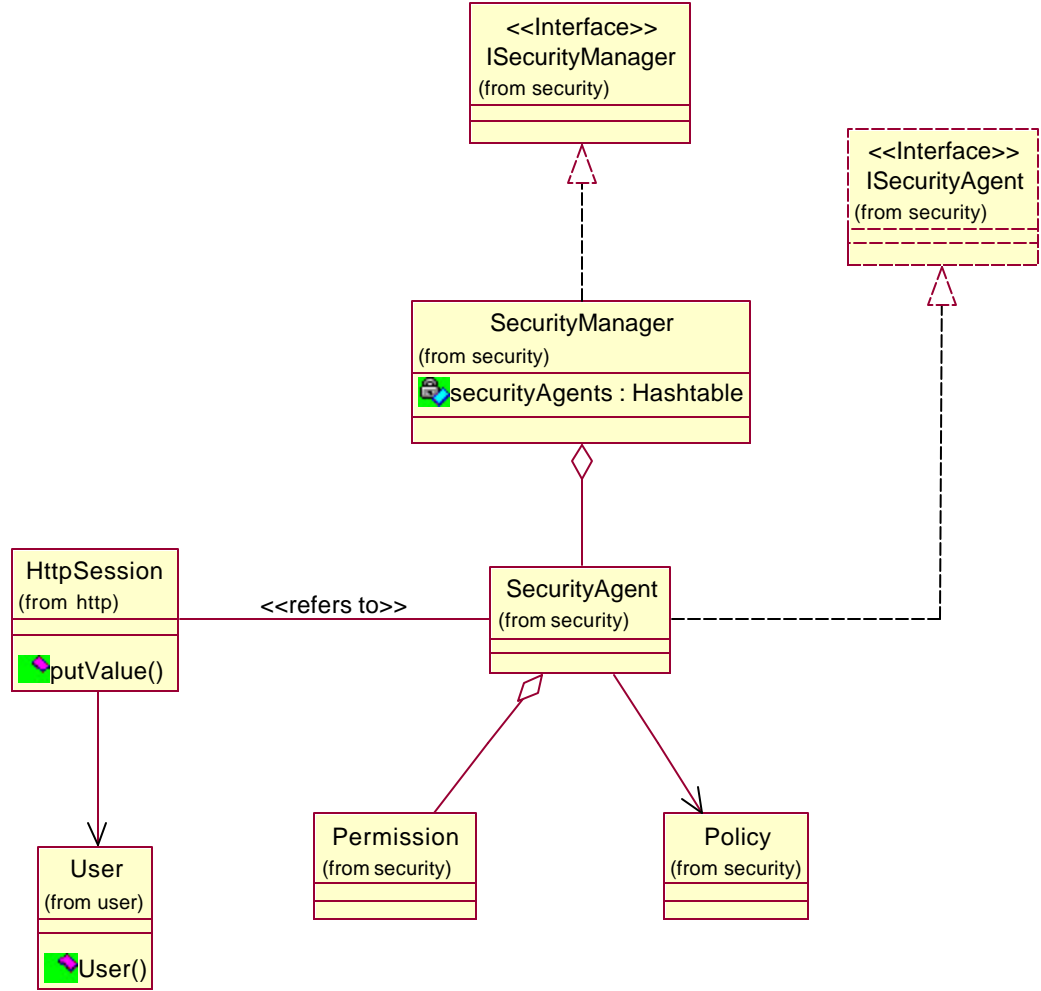
For example, if a user, Paolo, attempts to edit an article weblet, the security agent will search through the available operations in the permission list. Upon finding an edit operation, it will search the policy lists for "Paolo". A similar traversal is done to change permissions and policy mappings using the get/set methods. Perhaps the only operation that is not delegated to a security agent is the destruction of a weblet container. The security agent in charge of the parent container/directory context handles that operation.

Identifier		OBJ-13
Defining Quality		Manages security agents
Name		Security Manager
Object Interactions		OpenPortalWebletManager
States		
Constraints		
Component membership		Security
Refers to		
Implementation		Java class object

Identifier		OBJ-14
Defining Quality		Enforces security policies for a weblet container
Name		Security Agent
Object Interactions		Security Manager, OpenPortalWebletContainer
States		
Constraints		Can only enforce the policies that it is defined in the policy file.
Component membership		Security
Refers to		
Implementation		Java class object

Identifier		OBJ-15
Defining Quality		Maintains a set of permissions for a given command
Name		Permission
Object Interactions		SecurityAgent
States		
Constraints		
Component membership		Security
Refers to		
Implementation		Java class object

Identifier		OBJ-16
Defining Quality		Maintains a permission to code source(weblet and weblet container) mapping
Name		Policy
Object Interactions		Security Agent
States		
Constraints		
Component membership		Security
Refers to		
Implementation		text



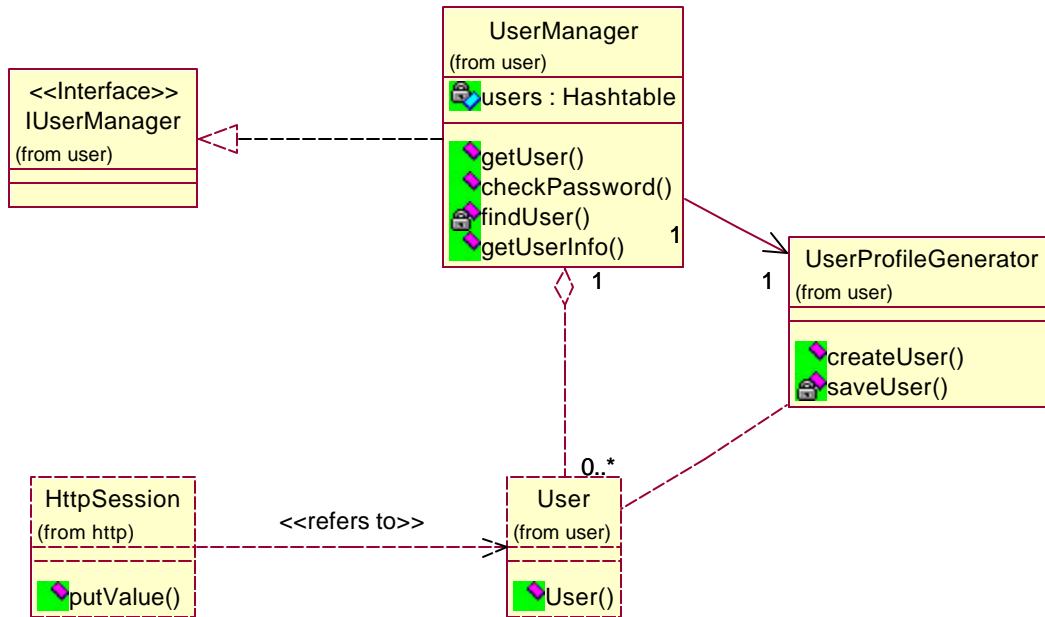
6. User Manager

The User Manager is responsible for creating and maintaining a collection of users that have access to OpenPortal. It implements the user manager interface. New user profiles are created using the UserProfileGenerator class.

Identifier		OBJ-17
Defining Quality		Manages the collection of user objects
Name		User Manager
Object Interactions		OpenPortalWebletManager
States		
Constraints		Can only be accessed through the weblet manager
Component membership		User
Refers to		
Implementation		Java class object

Identifier		OBJ-18
Defining Quality		Generates user profiles
Name		UserProfileGenerator
Object Interactions		UserManager, User
States		
Constraints		Only create profiles based on userID and passwords Only accessible by the User Manager
Component membership		User
Refers to		
Implementation		Java class object

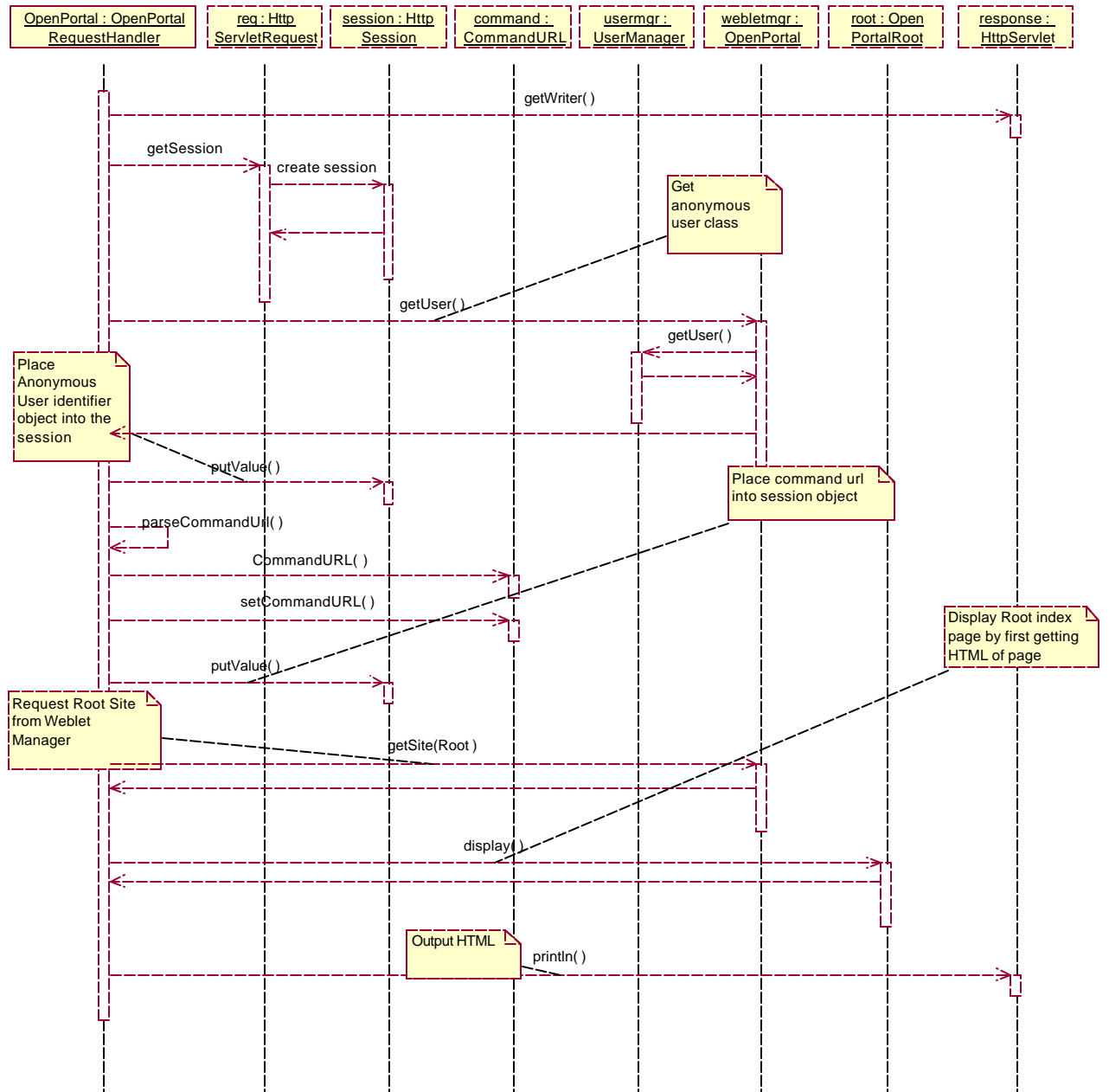
Identifier		OBJ-19
Defining Quality		Represents user information
Name		User
Object Interactions		User Manager, UserProfileGenerator
States		
Constraints		Only stores basic information about a user. Id and password only
Component membership		User
Refers to		
Implementation		Java class object



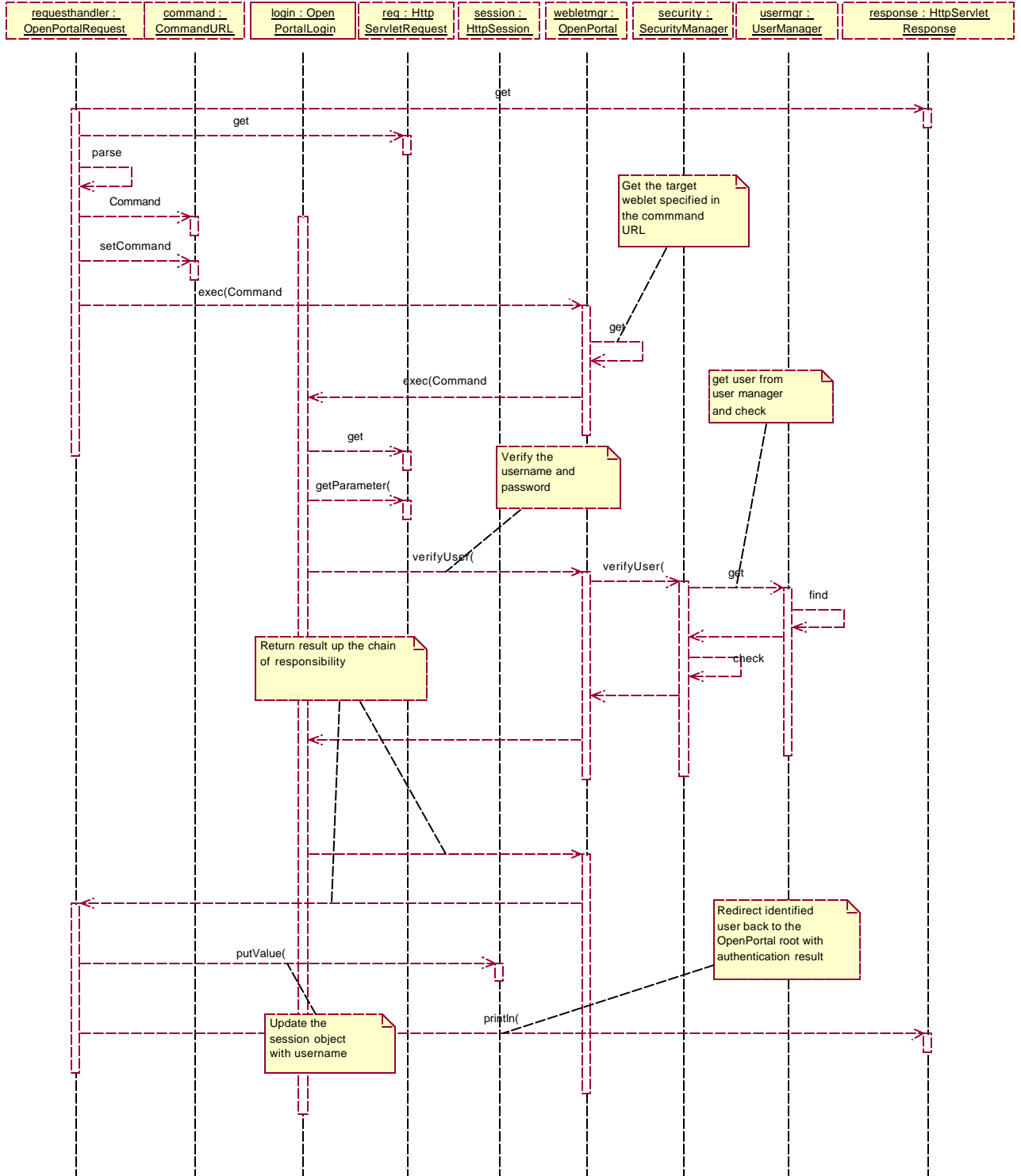
Operations Model

Operations Specifications

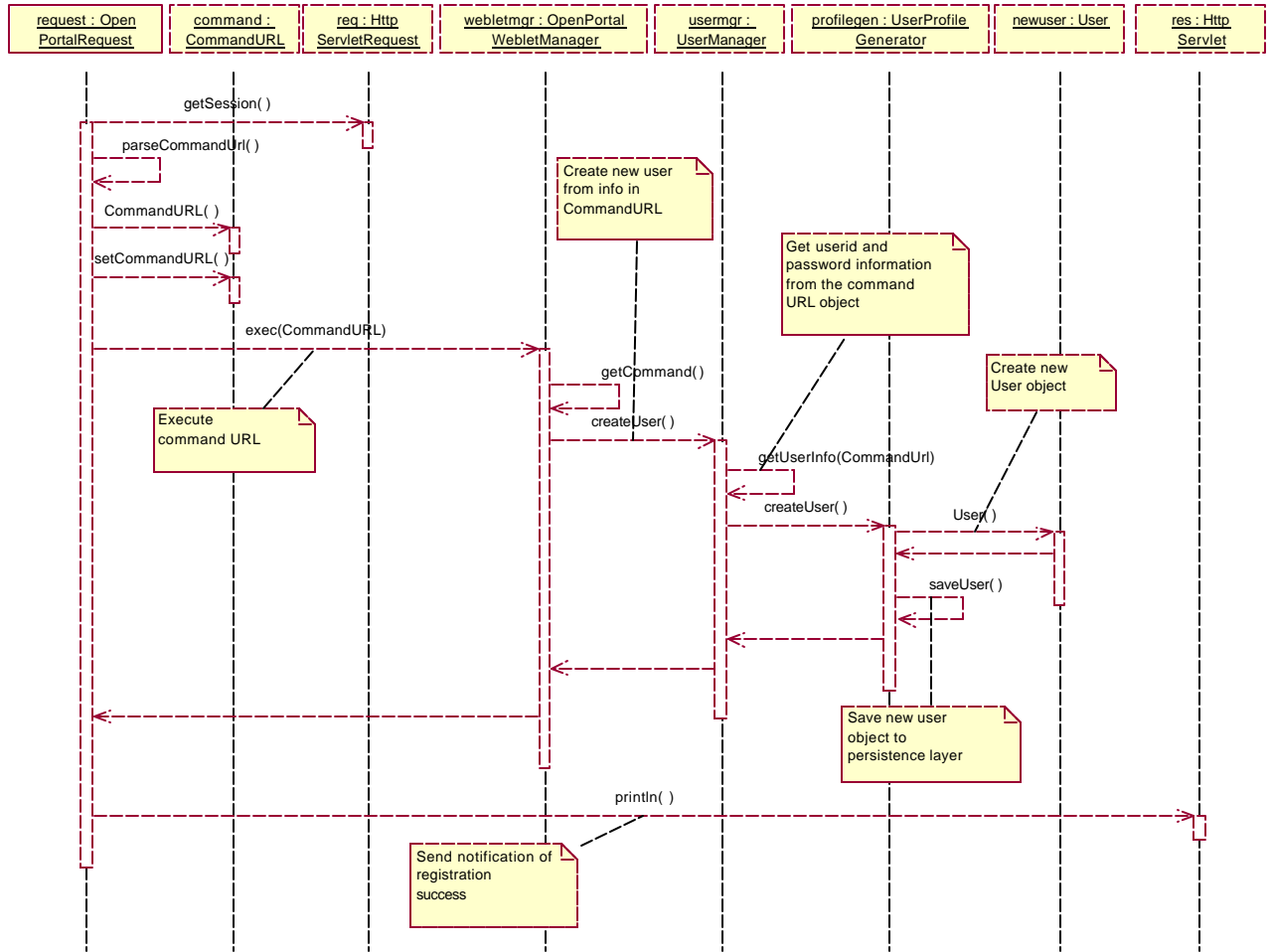
1. Initial Access to OpenPortal



2. User Login (Weblet Access)



3. New User (Creating a new user profile)



Data Model

Weblet Descriptor Data Type Definition

```
<!DOCTYPE WebletDescriptor [  
  <!-- Weblet -->  
  <!ELEMENT weblet (property*, standard-property*, command*)>  
  <!ATTLIST weblet draggable (true | false) false>  
  <!-- Weblet Container -->  
  <!ELEMENT webletcontainer (property*, standard-property*, command*)>  
  <!ATTLIST webletcontainer draggable (true | false) false>  
  <!-- Property -->  
  <!ELEMENT property EMPTY>  
  <!ATTLIST property name CDATA #REQUIRED>  
  <!ATTLIST property default-value CDATA #REQUIRED>  
  <!ATTLIST property inplace-editable (true | false) true>  
  <!ATTLIST property draggable (true | false) false>  
  <!ATTLIST property inplace-input CDATA "<input type=text size=20">>  
  <!-- Standard-Property -->  
  <!ELEMENT standard-property EMPTY>  
  <!ATTLIST standard-property name CDATA #REQUIRED>  
  <!ATTLIST standard-property default-value CDATA #REQUIRED>  
  <!ATTLIST standard-property inplace-editable (true | false) true>  
  <!ATTLIST standard-property draggable (true | false) false>  
  <!ATTLIST standard-property inplace-input CDATA "<input type=text size=20">>  
  <!-- Command -->  
  <!ELEMENT command EMPTY>  
  <!ATTLIST command action CDATA #IMPLIED>  
  <!ATTLIST command perform-on CDATA #IMPLIED>  
  <!ATTLIST command full-command CDATA #IMPLIED>  
  <!ATTLIST command embed CDATA #REQUIRED>  

```

Appendix