PHIN Applications Portal

By Walt Davis

Oregon Department of Human Services
Systems Architecture

Introduction

The Applications Portal and its underlying building blocks represent a comprehensive and reusable solution to web-based development using Java technologies.

The first couple of sections discuss Business Drivers and Design Goals influencing the portals construction.

Remaining sections serve as introductions to the understanding of several foundational points of view: Dynamic Content Generation, The Portal Interface, The Directory API, and a Repeatable Construction Process.

Business Objectives

While the business goals are often hard to identify; for the Applications Portal these may be represented by the following interpretations:

- Adopt industry and department standards that lead to common goals.
- Create solutions that make it easier for systems to be integrated.
- Modernization through web based applications.
- Reduce confusion for users who currently need to access web-based applications from multiple locations, i.e., central location from which to access applications.
- Improve efficiency through a simplified sign-on, thereby reducing the need for multiple passwords.
- Shared directories.
- Enterprise level solutions.
- Website branding that is meaningful to the user base.
- Look and feel consistent with State web applications.
- Provide administrative tools for integrated functions common to all applications.
- Authenticate users through the State's Identity Management and Access Control (TIM/TAM) where appropriate.

As part of the development processes these drivers where used to help create design goals.

Design Goals

Much of the design motivation comes from the WebMD/Healtheon content delivery platform for healthcare consumer portals (MSN, Lycos, Excite, and Readers Digest).

Key Design Goals (not in order of importance):

1. Low learning curve for all levels of programming experience.
2. Reduced maintenance and future development costs.
3. Java developer and HTML designer equivalence.
4. Minimize HTML application logic.
5. Equal support for application and web/portal development.
6. Strong security.

7. Share user profiles across applications and portal.
8. Central place from which to access web applications.
9. Simplified and single sign-on.
10. Customization of portal instances.
11. Comprehensive and well organized reusable building blocks.

Dynamic Content Generation

Facing many decisions around what frameworks to use and having gained valuable experience in porting the NBS (NEDSS Base System), the NEDSS (National Electronic Disease Surveillance System) team had been looking for alternative solutions to the more widely accepted and published industry standard frameworks such as EJB's, Struts, Caster, Velocity, Hibernate, and so forth.

These frameworks often contain huge code bases that are rigid in nature. Some are unable to meet our design goals. A good example of this is goal 3. Typically once a web designer has built the HTML and a Java developer has integrated it into the application using Struts; then the web designer can no longer maintain the HTML. This seriously impacts the web designer's ability to control the look & feel of a website and make modifications without the help of a Java programmer.

As a new member of the team and having successfully completed several full life cycle web based Java products I was able to offer alternatives. One of these being the construction of a web UI framework for generating dynamic content in a way similar to WebMD's platform.

To do this we needed to parse HTML and read an embedded language for inserting dynamic content. This is much different than the standard tag libraries found in other frameworks. Here you actually embed another language within the HTML that allows for looping and conditional logic yet limits application logic.

The risk of building the parsing portion ourselves was too high. However, with some research the team found HTML.Template.java. This was a lightweight HTML parser that did exactly what we needed. The templating language was simple and provided powerful insertion techniques. With this all that remained was to provide wrapper functionality for organization and modeling of an applications' user interface logic.

This would be the 4th generation of such a framework whereby each generation's development has been improved upon by separate teams of developers. Typically each developer having experience in Java, existing frameworks such as Struts, or many early home brewed JSP (Java Server Pages) frameworks.

Reusing this experience the NEDSS team was able to build a simplified framework for dynamic content delivery called WebEdifice. This framework unlike industry standard frameworks does a better job at meeting design goals 1 – 5, 10, and 11.

One draw back to using this framework is compared to industry supported frameworks there is little literature available to support its learning. On the other hand with the aid of someone familiar with the framework; at least one project has demonstrated it's possible to bring up a website with relatively little learning. Since this also meets all design goals WebEdifice appears to be a viable solution.

The Portal Interface

It is important to recognize that there are different types of portals.

What makes the Applications Portal unique is its design for web-based applications. It provides common and reusable features for managing web applications that would otherwise be client/server. This is much different than a portal such as iGoogle (a Widget Portal) or WebMD (a Content Management Portal); likewise, neither are Google and WebMD the same type of portal.

In simple terms the Applications portal is a common web interface through which to access a businesses web-based client/server applications and having the following key features:

1. Shared Directory.
2. User Profile Management.
3. Self Registration.
4. Forgotten Password Retrieval.
5. Forgotten ID Retrieval.
6. Application Launcher.
7. Application Manager.
8. Application Access Authorization.
9. External Credentials Manager.
10. Strong Security.

Because the Applications Portal uses a highly dynamic content delivery mechanism it is relatively simple to launch multiple instances. Each instance may have its own directory, look & feel, security requirements, and applications, or these things may be shared. To illustrate this flexibility consider the meaning of PHIN. Its purpose is to brand the portal as being the Oregon Public Health Information Network portal. If so desired the HTML designer could very easily have change this to FamillyNet.

Shared Directory

A shared directory contains the most common profile information of any user. This is defined by the Open Group RFC2251-RFC2256 and RFC1274. The Open Group in conjunction with many companies around the world has defined this as a standard for all to use.

By using the most common attributes for directory entries the directory can be used in the widest possible scope. This level of scope is considered the enterprise view. Specialized directories may exist but they should reference and extend an authoritative (or master) directory.

Within the applications themselves a profile may also be extended to include additional information. This information should be contained within the application that cares about it. The importance of doing this is not to pollute the directory with content that would otherwise not be global. Ignoring this rule typically leads to directory bloat that over time is increasingly difficult to work with. If the extended information is shared then a better design is to provide a service within the application that allows other applications access.

User Profile Management

The Applications Portal contains an administrative page for managing all user profiles. Procedures such as account suspension and reset are done from this page. Another page allows users to update their own personal information.

Self Registration

If someone accessing the portal does not yet have an account they may themselves create one. A temporary password is emailed ensuring a valid email address. Upon accessing the portal for the first time users must first change their temporary password. Typically you only have a few days to do this before account suspension.

Forgotten Password Retrieval

When a user forgets his/her password it can be retrieve without having to call support. The user will be prompted to enter personal information that only he/she knows and a temporary password will be emailed. Upon logon the user will use a temporary password and is immediately asked to change it.

Forgotten User ID Retrieval

Users typically do not forget their user ID but just in case they do it can be retrieved. The user is asked for personal information that only they would know and the ID is emailed back.

Application Launcher

The application launcher is the main or home page of a user. The home page lists out each application that the user has access to with links and descriptions. Another page allows the launcher to be configured so only to contain applications that are of interest.

Application Manager

Administrators have access to a page that lets them add/remove/disable applications. Applications can be simple links to other websites or they can be links to applications that have been fully integrated using shared features of the portal. When an application is fully integrated then it can take advantage of the Application Access Authorization feature of the portal.

Application Access Authorization

Some applications require approval in which case the approving person is emailed an application access request. Once approved the user is notified by email that access has been granted and it will be displayed with within the launcher.

External Credentials Manager

This is an un-built but scheduled feature of the portal that allows a user to maintain logon credentials for external websites. This feature allows external applications to be integrated with minimum effort and once completed will simulate single-sing-on to applications with different security mechanisms.

Strong Security

As we will see in the next section the Applications Portal has strong security through a powerful directory API.

The Directory API

The importance of a directory API (Application Program Interface) is to remove from the application developer the need to understand backend LDAP (Light Weight Directory Access Protocol) connectivity and from having to implement missing functionality.

This would be the job of a security API developer. Not only does developing such an API require its own effort, it makes no sense to reprogram this functionality every time; doing so would surely make it difficult if not impossible to provide any kind of standardization across applications.

By building the API with a re-implementable interface it makes dealing with changes in directory technologies more manageable by not having to modify the applications themselves, i.e., build another implementation of the API and re-deploy libraries.

At present there is one implantation of this API for LDAP backbends adhering to the Open Group InetOrgPerson definition. It has been tested with several LDAP directory vendors including OpenLDAP, eDirectory, and iPlanet.

Key features of this implementation of the directory API are:

1. LDAP compliant directory.
2. State machine driven security policies.
3. Utilizes an attribute re-mapping engine.
4. Role based access.
5. Stores the most common and widely accepted attribute data.
6. Implements a reusable API.
7. Meets or exceeds DHS security policies.
8. Single-sign-on.
9. Single credential set.
10. Quasi-two factor authentication (question/response).
11. LDAP vendor neutral.
12. Hashed password.

There is a separate more technical document that discusses the security and implementation details of the API; therefore, we will not be discussing such details here.

Extending the directory definition for InetOrgPerson to include additional attributes is relatively easy; however, the more a directory schema deviates from the Open standard then the more advisable it is to implement a custom schema in a separate directory and link the two directories using an object UID.

Repeatable Construction Process

One portal application was abstracted into a foundational set of classes that could be used to instantly bring up a new application.

At least two applications have been produced using these classes; in one case developers with little Java experience where able to create a functional website in only a few days.

During the creation of these classes the flag ship application from which all others where to be modeled after was on an accelerated development path. This means that the abstracted classes have limited integration with The Portal Interface.

Summary

Although adoption has been slow other departments within DHS are using this solution to provide their needs. A good example of this is the DHS Eligibility project. They where able to rapidly build a HIPPA based transaction service. The core of this web service was functional within days. This then gave the team much needed time to focus on transaction processing.

As we have seen the Applications Portal represents an elegant solution specifically targeting the development of web applications that would otherwise be client/server.

A remaining aspect to the continued success of the Applications Portal is acceptance and adoption of its solution by those maintaining it and customer base using it.

Given the right circumstances and under the right leadership other departments within DHS can experience the same level of success as seen in the PHIN Applications Portal; however, make no mistake in thinking that here lays a golden hammer to all problems because there will always be an exception.

Success does not necessarily dictate adoption of The Portal Interface. Perhaps just The Directory API or the Dynamic Content Generator is useful, after all they do provide solutions to common problems.

Appendix A: Questions & Answers

Q. **When everyone wants to add their own attributes to the directory that have little use beyond the concerning application how do you prevent directory bloat?**
A1. The more common solution is to use a master/slave architecture and create a master directory that only stores common attributes to all then slave directories are created that can be; objects would be linked back to the master using a UID.
A2. A more organized approach is to have a single master directory and let each application extend the object definitions. If other applications need this information then it may be access through a service. It's still a master/slave architecture but you have reduced the complexity to maintaining one directory and it promotes a better division of responsibilities among applications in a distributed environment.

Q. **How easy would it be to create a new implementation of the directory API?**
A. That entirely depends on the complexity of the backend directory technology and its deviation from open LDAP standards.

Q. **Does portal integration obviate the need for a web application to have its own authentication mechanism? ... its own login page?**
A. An application may have its own mechanism, use The Directory API, or both. Some apps have a configuration setting that lets them redirect back to the portal logon so to achieve a seamless look and feel. On the other hand if the portal is down its nice to still be able to use your applications.

Q. **If an application is integrated with the portal, can it still be accessed independently of the portal? Is this advisable?**
A. Yes. This of course is a design choice. By redirecting back to the portal you save development time by not having to implement another logon page. On the other hand if you do not want this point of failure then you should add your own logon page. Either way when using the API the mechanism is the same.

Q. **Is there extra work involved to be able to access it independently of the portal in addition to within the portal?**
A. There are foundational classes and HTML templates available for use so while there is additional work, it should be minimal.

Q. **Is it possible for an application to use a completely different authentication mechanism but still be integrated?**
A. From a hyper link perspective yes; otherwise, there are plans to build a credentials manager that would use header injection when redirecting to external applications.

Q. **If we were to upgrade the directory API to full two-factor authentication, would portal-integrated applications benefit at no additional cost?**
A. No. The current quasi-two factor form is a question/answer response and the amount of work for the upgrade would depend on the two-factor form.

Q. **Do you have any case studies of existing applications that have switched from a different proprietary or one-off security module to using the directory API, and the associated costs?**
A. Nothing substantial but like anything else the benefits should be weighed against the costs. If costs are greater than the benefits and with limited resources time is probably better spent elsewhere.

Q. **Do you have any case studies of existing applications that have been integrated post-production into the portal, and the associated costs?**
A. There has been some testing with external application on completely different authentication mechanisms whereby they where integrated with header injection. The work effort was minuscule and in some cases required no changes to the external application.

Q. **Does the design of the portal itself, along with its integration requirements; encourage good web interface design practices? ... web application design practices?**
A. Yes. Special care was taken to develop well organized and meaningfully components using object oriented practices. And special attention was giving to its design ensuring that web designer retains ownership over Java developer.

Q. **What are the portal integration requirements, web interface wise?**
A. There are no per say requirements but having a similar look & feel probably makes for a less confusing user experience. In practice it will not always be beneficial to integrate look & feel but instead pick the low hanging fruit such has single credential set and sign on.

Q. **Do all business domains have to share the same directory and portal user interface?**
A. No. In fact if you have well defined domains each having its own set of applications then the needs of those domains are often different to a point that makes it more manageable to have multiple instantiations.

Q. **Regarding the embedded HTML language could you also not have achieved the same thing with JSP?**
A. Yes you could; however, web designers do not make good java programmers and JSP supports placing too much application logic in the UI. Placing application logic in the UI is generally thought of as a bad programming practice.

Q. **How much does it costs to license HTML.Template.java?**
A. Nothing, HTML.Template.java is OpenSource.

Q. **Is it possible to do single-sign-on across applications running on different web servers?**
A. For security reasons no. Crossing web servers creates an SSL certificate issue during session creation. And the Directory API uses session information to insure a secure authentication. It is however possible to simulate it through header injection.

Q: **Does the Directory API utilize its own session timeout separate from the SSL connection?**
A. Yes. This creates a more secure website and is part of the single-sign-on mechanism.

Q: **For applications that require two factor authentications, how do they integrate with the Portal?**
A: From a practical point of view such applications must perform the authentication on their own, in which case the Portal just serves as a central jumping point from which applications are accessed.

Q. **How much wood would a woodchuck chuck if a woodchuck would chuck wood?**
A. If a woodchuck could chuck wood then a woodchuck would chuck a lot of wood.

Appendix B: HTML.Template.java

The following has been directly copied from http://html-tmpl-java.sourceforge.net/. It is not a work of this documents author and is being provided as a convenience to the reader.

HTML.Template has five main tags - <tmpl_var>, <tmpl_if>, <tmpl_unless>, <tmpl_loop> and <tmpl_include>.

The if, unless and loop tags are block tags and need to be closed with corresponding </tmpl_if>, </tmpl_unless> and </tmpl_loop> tags.

In addition, the <tmpl_if> and <tmpl_unless> tag may have an optional <tmpl_else> part.

The <tmpl_include> tag is evaluated when the template is created, all other tags are evaluated when the template's output is requested.

The <tmpl_var> tag is the simplest tag, and may contain a single scalar value. <tmpl_var> can occur anywhere in the template.

Block tags may be nested, ie, an if tag may contain other tags, including other block tags.

The <tmpl_loop> tag is a bit more complicated than the other tags. It allows you to create a section of text that will be displayed repeatedly for every item in the loop control variable. Inside the <tmpl_loop>, you place <tmpl_var>s.

In the java, you use a Vector that contains a list of Hashtables, each representing one record/group of <tmpl_var>s to be displayed in the loop.

The loop iterates over the vector, replacing variables in the text block with their parameter values from each Hashtable. Loop tags create a new namespace. All code within a loop block is evaluated for every iteration of the loop.

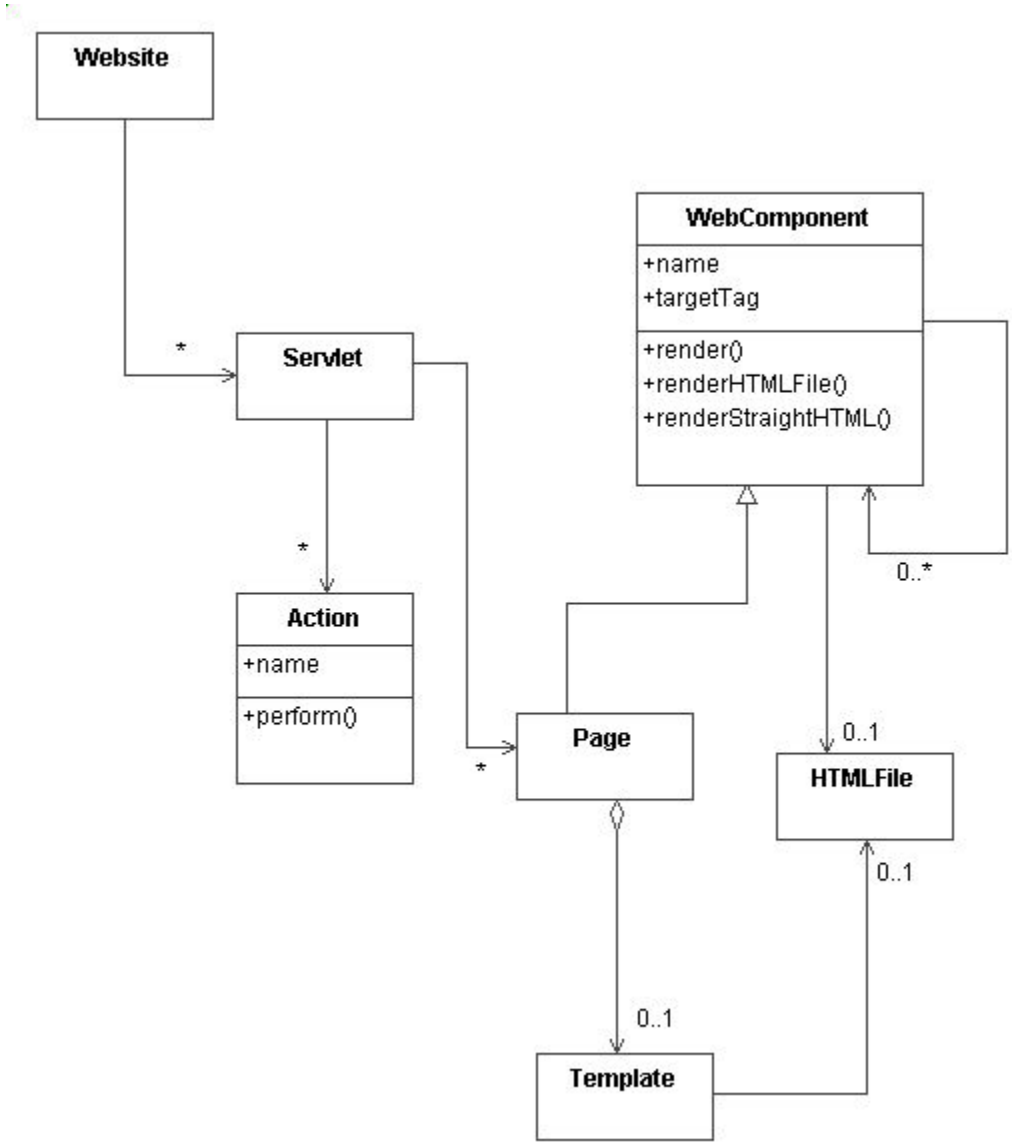
Appendix C: Directory Schema

```
#########################################################
# Is an extension of RFC2251-RFC2256 and RFC1274 schema.  #
#########################################################

# Attribute Definitions
attributetype ( 1.3.6.1.4.1.21259.1.1.2.1 NAME 'kAcctStatus'
        DESC 'account status'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{128} SINGLE-VALUE)

attributetype ( 1.3.6.1.4.1.21259.1.1.2.2 NAME ('kMI' 'kMiddleInitial')
        DESC 'middle initial'
        EQUALITY caseIgnoreIA5Match
        SUBSTR caseIgnoreIA5SubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26{1} SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.21259.1.1.2.3 NAME 'kAppRole'
        DESC 'role based access <appName:roleName>'
        EQUALITY caseExactMatch
        SUBSTR caseExactSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{256})

attributetype ( 1.3.6.1.4.1.21259.1.1.2.4 NAME 'kPagerNumber'
        DESC 'work pager number'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{128} SINGLE-VALUE)

attributetype ( 1.3.6.1.4.1.21259.1.1.2.5 NAME 'kHomePostOfficeBox'
        DESC 'home post office box'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{128} SINGLE-VALUE)

attributetype ( 1.3.6.1.4.1.21259.1.1.2.6 NAME 'kHomeStreetAddress'
        DESC 'physical address of residence'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{128} SINGLE-VALUE)

attributetype ( 1.3.6.1.4.1.21259.1.1.2.7 NAME 'kHomeCity'
        DESC 'city of residence'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{128} SINGLE-VALUE)

attributetype ( 1.3.6.1.4.1.21259.1.1.2.8 NAME 'kHomeCounty'
        DESC 'county of residence'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{128} SINGLE-VALUE)
```

```
attributetype ( 1.3.6.1.4.1.21259.1.1.2.9 NAME 'kHomeState'
        DESC 'state of residence'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{128} SINGLE-VALUE)

attributetype ( 1.3.6.1.4.1.21259.1.1.2.10 NAME 'kHomePostalCode'
        DESC 'home postal code'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{128} SINGLE-VALUE)

attributetype ( 1.3.6.1.4.1.21259.1.1.2.11 NAME 'kWorkCounty'
        DESC 'county of work'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{128} SINGLE-VALUE)

attributetype ( 1.3.6.1.4.1.21259.1.1.2.12 NAME 'kOldPassword'
        DESC 'old passwords'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.40{128})

attributetype ( 1.3.6.1.4.1.21259.1.1.2.13 NAME 'kUserPassword'
        DESC 'reserved for encrypted password use: SHA, MD5, ...'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.40{128} SINGLE-VALUE)

attributetype ( 1.3.6.1.4.1.21259.1.1.2.14 NAME 'kPasswordUseHistory'
        DESC 'number of old passwords that are tracked'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.21259.1.1.2.15 NAME 'kPasswordExpirationDays'
        DESC 'days before password expires'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.21259.1.1.2.16 NAME 'kDateOfLastPasswordChange'
        DESC 'date of Last Password Change'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26{128} SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.21259.1.1.2.17 NAME 'kSecret'
        DESC 'quasi two-factor authentication <question:response>'
        EQUALITY caseExactMatch
        SUBSTR caseExactSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{256})

attributetype ( 1.3.6.1.4.1.21259.1.1.2.18 NAME 'kPasswordExpires'
        DESC 'if true then password will expire'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.21259.1.1.2.19 NAME 'kMaxAuthFailures'
        DESC 'max number of auth failures before account is suspended'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.21259.1.1.2.20 NAME 'kFailedAuthCount'
        DESC 'number of authentication retries'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 SINGLE-VALUE )
```

```
attributetype ( 1.3.6.1.4.1.21259.1.1.2.21 NAME 'kMinPasswordLength'
        DESC 'minimum password length'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.21259.1.1.2.22 NAME 'kMinLogonIdLength'
        DESC 'minimum logon ID length'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.21259.1.1.2.23 NAME 'kAuthTokenTimeout'
        DESC 'minutes before authentication tokens are expired'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.21259.1.1.2.24 NAME 'kAuthToken'
        DESC '<timeStamp:logonID:encryptedPassword:hostIP:sessionID>'
        EQUALITY caseExactMatch
        SUBSTR caseExactSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{256})

attributetype ( 1.3.6.1.4.1.21259.1.1.2.26 NAME 'kPasswordResetGracePeriod'
        DESC 'number of days user has to change a reset password'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.21259.1.1.2.27 NAME 'kSignature'
        DESC 'MD5 hash of demographic data'
        EQUALITY caseExactMatch
        SUBSTR caseExactSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{256} SINGLE-VALUE)

attributetype ( 1.3.6.1.4.1.21259.1.1.2.28 NAME 'kRFU'
        DESC 'Reserved for Future Use'
        EQUALITY caseExactMatch
        SUBSTR caseExactSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15)

# Class Definitions
objectclass ( 1.3.6.1.4.1.21259.1.1.2.29 NAME 'kUserProfile' SUP inetOrgPerson
        DESC 'App User context Profile'
        MAY  ( kMI $ kAppRole $ kPagerNumber $ mobile $ homeTelephoneNumber $
                kHomeStreetAddress $ kHomeCity $ kHomeCounty $ kHomeState $
             kHomePostalCode $ kHomePostOfficeBox $ homePostalAddress $
             kWorkCounty $ kSecret $ kFailedAuthCount $ kOldPassword $
             kAuthToken $ kUserPassword $ kAcctStatus $ kSignature $
             kDateOfLastPasswordChange $ kPasswordExpires $ kRFU))

objectclass ( 1.3.6.1.4.1.21259.1.1.2.30 NAME 'kOrgUnit' SUP organizationalUnit
        DESC 'App User Context Organizational Unit'
        MAY (kPasswordExpirationDays $ kMaxAuthFailures $ kMInPasswordLength $
            kMInLogonIdLength $ kPasswordUseHistory $ kAuthTokenTimeout $
            kAppRole $ title $ kPasswordResetGracePeriod $
            kRFU))
```

Appendix D: WebEdifice



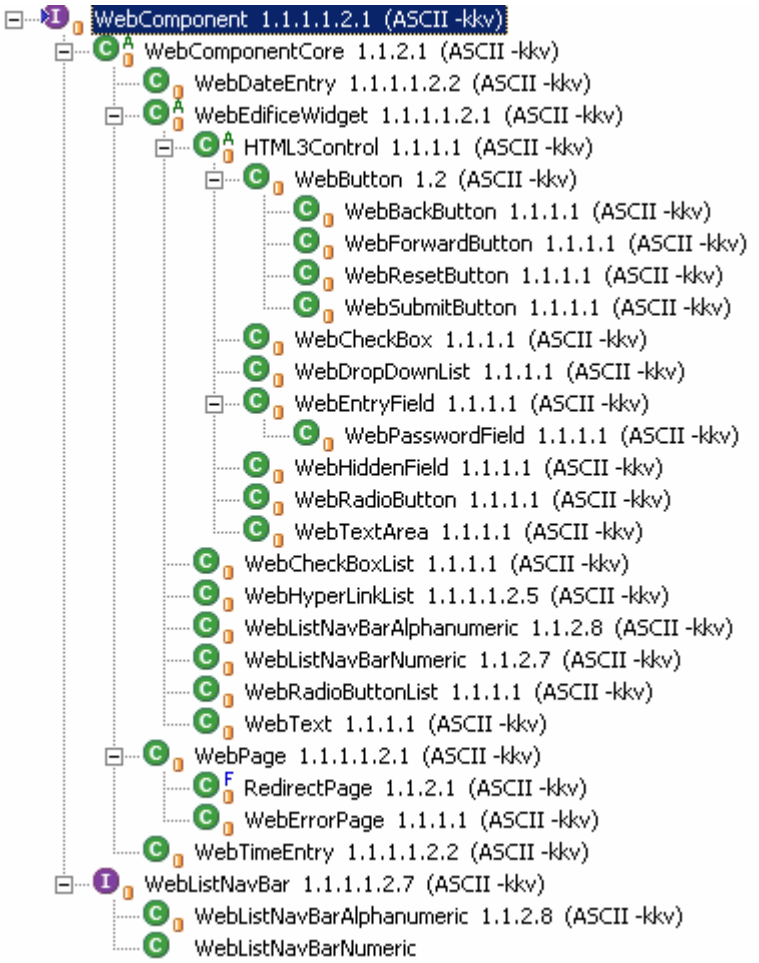
WebEdifice is a framework for building web interfaces to Java applications. The NEDSS team developed this framework during construction of eSentinel.

Its core elements are:

- **Servlets** - represent user roles that capture the look/feel/workflow in a roles based or use case driven system.
- **Pages** - takes data from the domain model layer of a three-tier application architecture then presents a web interface view of that data.

- **Actions** - represent things that a user will do that cause something to get done; take data from a web form and update data in the domain model, invoke a method within the domain model that generates a report, and so forth.
- **Components** - represent user interface objects that render HTML. Pages are the basic building blocks in constructing the user interface while widgets (dropdown boxes, entry fields, radio buttons, ...) are contained within a page and layouts change a widgets appearance.
- **Templates** – represent a reusable HTML file that defines a common look and feel for all pages using it.

The reusable components library:



Appendix E: The Directory API

Not a complete list of all API signatures!

User Queries	getUser(logonID) Return user profile for user with the specified logon ID. getUsersOfApp(appName) Return those users who have been granted access to the specified application. getNonusersOfApp(appName) Return all users who have not been granted access to the specified application.
Authentication	isUserAuthenticated(logonID, sessionIP, sessionID) Answer true if a user has been authenticated for the specified HTTP session IP address and ID. authenticateUser(logonID, password, sessionIP, sessionID) Authenticate that the supplied password matches the users password. If it does and there are no issues with the account then a user profile is returned. Upon success an authentication token is created whereby the user is not required to re-authenticate unless of course the token has timed out.
Authorization	isUserAuthorized(logonID, appName, roleName) Answer true if a user is authorized to access an application for a the specified role. grantAccess(logonID, appName, roleName) Grant the specified user access to an application and role. revokeAccess(logonID, appName) Revoke a users access from the specified application.
Role Queries	getRolesForApp(appName) Return all roles for the the specified application. getRolesForUser(logonID, appName) Return those roles that a user has been granted access to for the specified application.

Appendix F: Integration Options

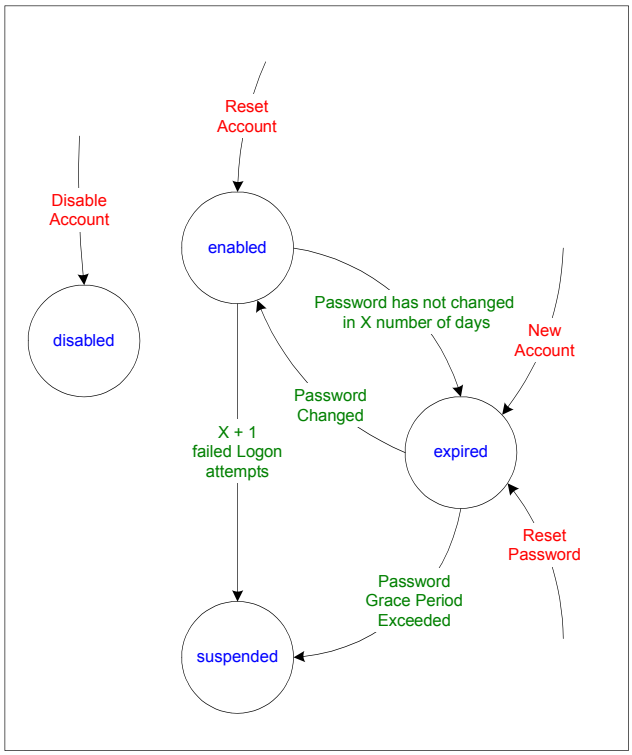
There are essentially three ways in which to integrate applications with the "Applications Portal": 1) The Directory API, 2) header injection, and 3) directory synchronization using IDI (IBM Directory Integrator).

Using The Directory API provides the highest degree of integration. Applications integrated with this option are sure to have the latest information and there is no need to setup separate processes for synchronization of data.

Header injection is perhaps the easiest way to integrate but requires the user to maintain a list of credentials. When a user is redirected to an external system these credentials will be injected through an http put. So the server needs to have a handler that looks for this injection and will automatically sign the user in. This allows the appearance of single-sign-on.

For option 3 requires knowledge of where the directory information is stored and how to access it. This information is used to setup jobs that synchronize directory data. Header injection is still required for the appearance of single-sign-on however the user would not have to maintain a credentials list.

Appendix H: 'Account Status' State Machine



Actions

New Account - * creates new user account with password equal to logon ID.

Disable Account - disables the account.

Reset Account - * when an account has been disabled or suspended it must be reset before returning to normal operations.

Reset Password - * changes user password to their logon ID.

** Upon first logon user is required to change password; if user does not do so within the allowed grace period then account is suspended.*

Disable - user is no longer able to access applications and absolutely no activity on the account is allowed.

Enabled - the account is in good working order and user may freely access applications.

Expired - the users password has expired and it must be change before applications are accessible.

Suspended - occurs when: 1) there has been a number of failed authentication attempts and 2) the user has not changed, within the allowed grace period, their password after initial account creation, account reset, or password reset.

Constants

Days Before Password Expiration - specifies the number of days before user is required to change their password.

Password Grace Period - number of days user has to change their password; applies to newly created accounts, reset accounts, and when the password is reset.

Max Failed Auth Attempts - specifies the max number of failed authentication attempts that once exceeded the account becomes suspended.

Variables

Number of Failed Auth Attempts - number of consecutive failed auth attempts, is reset to zero upon successful logon.

Date of Last Password Change - is changed when user changes his/her password and upon account reset but not password reset.

Account Status - tracks the state of a users account.

Appendix I: Access Control Interaction Diagram