Use Cases for Rich Internet Applications

Jevon M. Wright, Jens Dietrich

School of Engineering and Advanced Technology,
Massey University, Palmerston North, New Zealand
j.m.wright@massey.ac.nz, j.b.dietrich@massey.ac.nz

Abstract. In order to define the functionality of Rich Internet Applications (RIAs), we first need to investigate the use cases presented by new technologies such as AJAX. In this report we detail a comprehensive set of use cases that covers most of the functionality available in RIAs, and document them in a standard use case format.

Current version: April 15, 2009

1 Background

As part of the research in defining modelling languages for RIAs, an important step was to identify the requirements presented by these types of web applications. This involved a study of existing RIAs to identify their common concepts, and the functionality within each application, enabled by new technologies such as AJAX [1]. These functionalities were summarised into a selection of use cases, which are presented below in a standard use case format.

This is similar to previous work in evaluating the expressiveness of existing modelling languages, in terms of hypermedia concepts [2], basic web application concepts [3], and comparing the implementation of a conference system [4]. This white paper differs in that we systematically define the use cases of Rich Internet Applications, which was then used to evaluate existing modelling languages [5].

More information on this modelling language for RIAs, including technical details of a benchmarking application satisfying these use cases, is available online at `http://openiaml.org`.

2 Actors

Without going into too much detail, the actors involved in these use cases are described below. A simple UML use case diagram showing the relationships between actors is provided in Figure 1.

- **Visitor**: an anonymous visitor to the application.
- **User**: a logged in/authorised Visitor.
- **Developer**: the developer of the site. Does not have to be a User.
- **Administrator**: a site administrator, who is also a Developer.

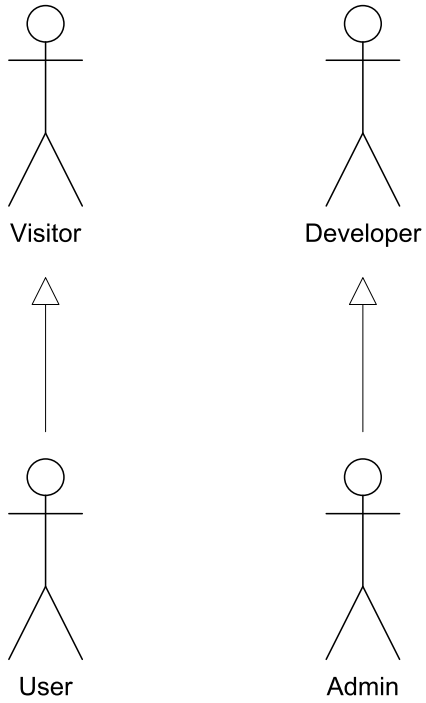

Fig. 1. Use Case Actors

- **Client**: also known as a Browser; the user interface running on the Visitor's machine, such as *Firefox* or *Internet Explorer*.
- **Server**: the application host, such as Apache *httpd*.
- **Remote Server**: another application host, separate from the Server.
- **Software**: separate application software running on the same machine as the Client, but not commonly used to browse the Internet.
- **Device**: a separate piece of hardware, such as a mobile phone.

3 Use Cases

We derived our use cases from a study of existing web applications (as mentioned in our benchmarking paper [6]); in particular, attention was focused on the following sites:

1. **Gmail**: Web-based e-mail by Google.
   `http://www.gmail.com`
2. **Calendar**: Google Calendar, a collaborative online calendar.
   `http://calendar.google.com`
3. **Reader**: Google Reader, an offline-enabled feed reader.
   `http://reader.google.com`

4. **Docs**: Google Docs, a collaborative office suite.
   `http://docs.google.com`
5. **Last.fm**: A social network-enabled music site.
   `http://www.last.fm`
6. **Pages**: Google Page Creator, an online web publishing suite.
   `http://pages.google.com`
7. **Facebook**: A social networking platform.
   `http://www.facebook.com`

These use cases are presented in a standard use case format. It appears that they can be loosely grouped into subject areas (e.g. *database-driven*, *events*, *user interface*, *security* et al.), and in our benchmarking paper where we fully define the requirements of RIAs, we do exactly this.

UC-01	**View Data**
Description	A website visitor can view a list of products on a website. This information is stored in a relational database.
Preconditions	A database exists; a list of products exists in the database
Actors	Visitor, Server
Normal Sequence	1. The Visitor requests a list of products in the database 2. The Server connects to the database and lists all available products on one page
Postconditions	
Exceptions	1. There are no products in the database; either an error is thrown or an empty list is displayed 2. The database cannot be accessed; an error message is displayed
Comments	Well supported in existing systems; a basic requirement of database-driven websites

UC-02	**Update Data**
Description	A user can update their user account details on a website. Their account details are persistently stored in a relational database.
Preconditions	A User has a user account on the website/database and is logged in (see *User Authorisation*)
Actors	User, Server
Normal Sequence	1. The User requests the Edit User Account page 2. The Server presents the Edit User Account page, which contains "name" and "password" fields 3. The User changes the name displayed in the "name" field 4. The User submits the page back to the server 5. The Server changes the "name" of the current User to the value provided by the User 6. The User is redirected to the home page
Postconditions	
Exceptions	1. The User does not have the permissions to edit their user account (e.g. the account is blocked); an error message is displayed 2. The database cannot be accessed; an error message is displayed
Comments	See *View Data*

UC-03	**Pagination**
Description	A website visitor can view a list of products on the website. If there are more than 10 products, they are displayed in a pages, with first, previous, next, last buttons.
Preconditions	Preconditions in *View Data*
Actors	Visitor, Server
Normal Sequence	1. The Visitor requests a list of products in the database 2. The Server connects to the database 3. There is more than 10 items; the Server displays the first 10 items, along with buttons to navigate forwards and backwards 4. The Visitor selects the navigation button to go forwards or backwards: (a) The Server changes the range of products currently displayed (b) The Server displays the new list to the Visitor, with the navigation buttons updated to show the new position
Postconditions	
Exceptions	1. Exceptions in *View Data* 2. The Visitor tries to navigate outside of the valid range; the result is restricted to the nearest bound page (i.e. first page or last page) 3. The list of products change whilst the user is browsing the pages; the Server ignores the change
Comments	1. Web applications may refresh the list and navigation buttons using AJAX technology [1] 2. If the list of products may change, the Server may wish to inform the user, store the initial results in memory, or display the updated list anyway

UC-04	**User Action Auditing**
Description	For auditing purposes, any edit or delete a user makes to a persistent data store or object will be logged to a logging table.
Preconditions	1. A persistent data store or object is marked as auditable 2. There is a logging data store to populate with auditing information 3. The User is authenticated
Actors	User, Server
Normal Sequence	1. The User performs an action 2. The Server requests to edit or delete a value in the data store or object 3. The Server makes a copy of the change in the database in the logging data store, along with the current User details and the time 4. The original data store or object is edited or deleted, and execution resumes
Postconditions	
Exceptions	
Comments	1. The "copy of the change" may be an entire copy, or only the delta change (*diff*) 2. Auditing the actions of anonymous users (Visitors) could also be supported 3. Systems tend to lack support for modelling cross cutting concerns like these; this could be implemented through point-cut like features on the design level with stereotypes/naming patterns, or an implementation solution like Java Servlet filters

UC-05	**Debug Mode**
Description	For debugging purposes, complex operations may have support for debug statements interspersed in the code.
Preconditions	1. Debugging statements are dispersed throughout the application 2. Administrators can turn on/off debug mode 3. Administrators can change the current level of debug messages recorded 4. A debugging log exists
Actors	Administrator, Server
Normal Sequence	1. The Administrator turns on the Debug mode 2. For every debugging/logging statement in the code executed by the Server, (a) If the debug statement is covered by the current debug level, (b) The statement is logged into the debug log by the Server along with the current time 3. This continues until the Administrator turns off the Debug mode
Postconditions	
Exceptions	1. The debugging table does not exist; it is created or an error message is sent to the Administrator and debugging mode is temporarily turned off 2. the debugging table cannot be written to; an error message is displayed to the Administrator and debugging mode is temporarily turned off
Comments	1. Debugging may slow down execution (a) How to toggle debug mode without restarting the application? 2. See *User Action Auditing* for an alternative debugging method

UC-06	**Server Transaction Support**
Description	A bank customer uses a web application to transfer money. The transaction should only succeed if every aspect of the transfer succeeded; otherwise the transaction should be rolled back.
Preconditions	The system is in a consistent state; the web application represents a banking application
Actors	User, Server
Normal Sequence	1. The User submits a transfer request 2. The Server starts the transaction 3. The Server deducts the amount from the source account 4. The Server adds the amount to the destination account 5. The operation is logged to a logging table (optional) 6. The Server checks to make sure all operations succeeded 7. The Server commits the transaction
Postconditions	The system is in a consistent state
Exceptions	1. Step 3, 4 or 5 fail; the transaction is rolled back 2. The operation does not reach to Step 7; the transaction is rolled back after a specified timeout time
Comments	Many web applications are front-ends for transactional systems

UC-07	**Local Data Storage**
Description	A shopping cart is implemented on a User's client, with the shopping cart data submitted to the Server only at checkout.
Preconditions	The User is browsing a list of products (see *View Data*); the Client can store data locally
Actors	User, Client, Server
Normal Sequence	1. For each product the User wants to purchase, (a) The User requests to add the product to their shopping cart (b) The Client stores the product addition to a local copy of the shopping cart. This data is stored throughout the entire session and not transmitted to the Server. (c) The User can continue to purchase more products 2. The User requests to checkout their shopping cart 3. The Client submits the stored shopping cart data to the Server 4. The Server takes the data stored from this shopping cart and charges it to their account
Postconditions	
Exceptions	1. The application loses network connectivity; the shopping cart will likely be lost. See also *Persistent Client Data*
Comments	1. For client-side storage, technologies such as Javascript memory, cookies, Java applets, URL rewriting and browser filesystem storage could be used. This should be transparent to all actors involved. 2. This could be modelled with stereotypes such as "server", "client", "cacheable" etc 3. Important if client has unreliable connection (e.g. ad hoc/wireless network) 4. Important if high level of system availability is required (e.g., companies outsourcing office applications to web based providers) 5. Technology wise, there are some development aiming at this feature, e.g. Dojo Toolkit [7], Google Gears [8] 6. One potential implementation could be covered in *Store Data in Local Database*

UC-08	**Server Data Access**
Description	The User can download an e-mail data object (like an e-mail draft) from the server to their client machine to work on
Preconditions	There is an existing e-mail data object to work on, that exists in a database
Actors	User, Server
Normal Sequence	1. The User opens an editable page for the e-mail 2. The Client downloads the e-mail object from the Server 3. The User edits the e-mail without recontacting the Server; this is achieved by editing the object in Client memory 4. The User is finished editing the e-mail, so submits it back to the Server 5. The Client submits the changed object data to the Server 6. The Server reconstructs the object based on the Client data and saves it back to the database
Postconditions	The e-mail object is changed and saved back to the database
Exceptions	1. The browser or active Internet connection goes offline while the user is editing the e-mail; the User should be informed.
Comments	1. This is different to a User working on data stored exclusively on their local machine, as in *Persistent Client Data* 2. Data accessed locally can be automatically saved through *Client Timer Support* 3. See also *Local Data Storage*

UC-09	**Persistent Client Data**
Description	An e-mail can be worked on, over many sessions, with the data staying persistent on the client's device
Preconditions	
Actors	User, Client, Server
Normal Sequence	1. The User starts writing an e-mail on their Client 2. The User closes the editing session 3. The Client saves a copy of the e-mail to some storage device 4. After some time, the User re-opens the e-mail editing session 5. The Client opens the saved copy and resumes editing 6. The User finishes their changes and submits the final copy to the Server for delivery 7. The local copy of the e-mail is removed
Postconditions	
Exceptions	1. After an extended period of time (e.g. a year) the data may be lost 2. Stored data may be deleted by the User themselves (e.g. clearing cookies)
Comments	1. Like in *Store Data in Local Database*, this should be transparent to the actors involved 2. Could be implemented through technologies mentioned in *Store Data in Local Database* 3. See *Local Data Storage* for client storage comments 4. E-mail delivery is covered in *E-mailing Users*

UC-10	**Temporary Server Data**
Description	A user is working on an image editing client; the image needs to be stored on the server, but only temporarily during the editing session
Preconditions	The User can upload an image to the Server
Actors	User, Server
Normal Sequence	1. The User uploads an image to work on 2. The Server saves this image to a temporary storage device 3. The User performs editing operations on the image (e.g. rotate, flip, add text, resize), which may take a long time and transit over many pages 4. The Server keeps the changes and original images in temporary storage 5. Once the User is finished, the final image is sent via e-mail 6. Once the session is over, or a new image is uploaded, the temporary data is deleted or marked for deletion
Postconditions	The image is not stored on the Server
Exceptions	1. The editing session is disconnected; the User may permanently lose the temporary image, or the session could be automatically restored (*Restore Server Session*)
Comments	1. The temporary server data could be stored either in files, or in a database 2. The temporary data should be able to persist over server reboots, though this may incur a performance penalty 3. E-mail delivery is covered in *E-mailing Users*

UC-11	**Uploading Files**
Description	To send a file via e-mail, the user can upload a file to the server, which is temporarily stored before being sent
Preconditions	The User can upload a file to the Server
Actors	User, Server
Normal Sequence	1. The User is creating an e-mail to send 2. The User attaches a file to the e-mail, by uploading a file to the Server 3. The Server downloads the file from the User and saves it in a temporary location 4. The Server informs the User the file has been attached and is ready to send 5. The User submits the completed e-mail to the Server 6. The Server attaches the file(s) to the e-mail and sends the e-mail
Postconditions	
Exceptions	1. The temporarily uploaded file is removed unexpectedly; the System displays an error message to the User, and has to re-upload their file before they can continue sending the e-mail 2. The uploaded file cannot be stored; the System displays an error message to the User, and sends an error message to the Administrator 3. The uploaded file is of an invalid or unexpected type; the System displays an error message to the User, and the temporary file is removed 4. The uploaded file is too big; the System displays an error message to the User, and the temporary upload data is discarded
Comments	1. The file must be able to persist over server reboots in this situation; this addresses fault tolerance and availability 2. It may be useful to add a progress dialog to the file upload; this could be accomplished by another service, Ajax + Javascript, etc 3. E-mail delivery is covered in *E-mailing Users*

UC-12	**Restore Server Session**
Description	While working on an image on the server, the User's session is disconnected; once the User returns, it is possible to continue the previous session
Preconditions	The User is authorised
Actors	User, Server
Normal Sequence	1. The User is working on an image manipulation program (see *Temporary Server Data*) 2. The session is disconnected prematurely, for example a power failure on the User's machine 3. The User returns to the website and re-authorises themselves 4. 5. The User requests to resume the existing session. 6. The User continues using the application with the old session data
Postconditions	The User is still authorised, and is using the same session data
Exceptions	1. The User requests to save the session data; the old session data is saved into permanent storage for access later, and a new session is started 2. The User requests the session data is destroyed; the old session is removed and a new session is started 3. The old session is destroyed before the session can be resumed; the session data will be lost 4. The old session is corrupted or loading the session would result in an inconsistent system state; the User is informed that they cannot reload it and must start a new one
Comments	1. Step 4 could be enhanced with a "preview existing sessions" step, allowing the User to specify which sessions to load or remove 2. This is similar to Microsoft Word's recovery process 3. There would probably have to be a time limit to how long sessions could be retained. Lost sessions could persist over server reboots, if required. 4. Some session data may not be suitable for recovery, for example, temporary authorisation with a bank transfer 5. (This is different to Firefox's Restore Session; Firefox does not actually restore the sessions, only the pages that were in use. This use case is server-based.) 6. This would be a way to load and save data from sessions active on multiple machines

UC-13	**User Authorisation**
Description	A Visitor can log in with certain credentials to become an authorised User
Preconditions	A Visitor has a session (see *Session Support*); there is a list of existing Users in the application
Actors	Visitor, User, Server
Normal Sequence	1. A Visitor visits a website, and goes to the "login" page 2. The Visitor enters in a username and password, and submits the form to the Server 3. The Server checks their username and password against the list of existing user accounts, and verifies that the account exists 4. The Server updates the Visitor's session to indicate that they are an authorised User 5. The Visitor becomes a User, and is redirected to a User-only page
Postconditions	
Exceptions	1. the entered credentials are invalid; the Visitor is informed and is prevented from continuing 2. The User account found is blocked from authorising; an error message is displayed to the Visitor 3. The Visitor loses its session while authorising; a new Session is created, or an error message is displayed to the Visitor, informing them they cannot currently login.
Comments	1. This is a basic requirement of database-driven websites 2. If the user cannot login, the Visitor can either register a new account, or retrieve their lost information (e.g. *Password Reset*) 3. This is usually not directly modelled in web applications but provided as service 4. However, the access rights controlling other resources must be defined 5. The registration of new User accounts is covered in *Account Registration* 6. User authentication can also occur through *Basic* or *Digest* HTTP authentication

UC-14	**Password Reset**
Description	A user cannot login with their given credentials; they know their username, but not their password. They choose to reset their password.
Preconditions	The user account has an associated e-mail address
Actors	Visitor, Server
Normal Sequence	1. The Visitor asks the Server to reset their password for a given username 2. The Server resets the User's password to a random password. 3. The Server e-mails this random password to the User's e-mail account. 4. The Visitor receives this e-mail, and uses this new password to login as a User, using *User Authorisation*. 5. The User is asked to change their password before they can continue with their session.
Postconditions	
Exceptions	1. The password e-mail cannot be sent; an error message is displayed to the User, and the Administrator is notified of the problem. 2. The User does not change their password in Step 5; this step will repeat before *User Authorisation* can complete.
Comments	1. The "random password" may be implemented through a secret link which does not affect the user account's password at all. 2. In Step 2, the Server can request for additional account information, e.g. asking a secret question stored with the account.

UC-15	**Session Support**
Description	Visitors to the website are tracked over the stateless HTTP protocol with an implementation of sessions.
Preconditions	
Actors	Visitor, User, Server
Normal Sequence	1. The Visitor visits the web application for the first time. 2. The Server creates a new session for the Visitor, and links this session to the current Visitor 3. After a period of inactivity, the session is lost.
Postconditions	
Exceptions	1. The Visitor cannot be tracked using existing session technologies; an error message is displayed to the Visitor and cannot continue using the web application. 2. The Visitor is authenticated as a User when the session is lost (Step 3); the Visitor will have to re-authenticate as a User (*User Authorisation*).
Comments	Sessions can be implemented through many technologies, such as cookies, IP tracking, or URL rewriting. The choice of technology should be transparent to the actors.

UC-16	**Account Registration**
Description	To use a shopping account properly, a Visitor needs to register their account details with the website to create a User account.
Preconditions	1. Visitors can create new User accounts 2. User accounts can be stored in a data store 3. The current Visitor is not authenticated as a User (*User Authorisation*)
Actors	Visitor, User, Server
Normal Sequence	1. The Visitor requests to create a new User account 2. The Server responds with an application form, which includes fields for e-mail address, username, and password. 3. The Visitor enters in their details and submits it to the Server. 4. The Server creates the new User account, and marks it to disabled. The User is e-mailed instructions on how to activate their account. 5. The Visitor receives the e-mail instructions, and follows the instructions to activate their account. 6. The Server marks the account as enabled, and redirects the Visitor to the login page (*User Authorisation*).
Postconditions	A new User account has been created
Exceptions	1. Invalid email address is submitted; the Visitor is informed of the error and must correct the application form. 2. Email cannot be delivered; the Visitor is either notified immediately, or the registration is ignored. 3. The given username (or email address) already exists; the Visitor is informed of the problem, and is given the option to choose a different username, or attempt to login with the username (in case they had forgotten they already had an account). 4. The user does not follow through on the account activation; the account is automatically deleted after a period of inactivity. 5. Erroneous account details are submitted, e.g. the wrong e-mail address; the delivered e-mail includes instructions on how to unsubscribe or cancel the account.
Comments	1. It is standard practice to encrypt the user password with a one-way hash with a hidden salt. 2. It is becoming acceptable for e-mail addresses to fulfill the role of usernames. 3. It is becoming less acceptable to force registration on users for trivial actions like downloading and submitting orders. 4. For some websites, forced account activation (steps 4-6) may not be necessary; this is generally only used to reduce spam or fake accounts.

UC-17	**Automatic User Authorisation**
Description	Instead of having to enter in their user details on every website visit, they are automatically logged in.
Preconditions	Preconditions in *User Authorisation*
Actors	User, Client, Server
Normal Sequence	1. As the User is authenticating (*User Authorisation*), they select an option to automatically log in ("remember me") 2. The Server stores re-authentication data on the Client 3. The User leaves the website; the session is closed, but the stored data remains 4. The User returns to any page on the website 5. The Server automatically detects that the User has re-authorisation data; this information is verified on the Server, and if successful, the user is automatically re-authorised
Postconditions	Postconditions in *User Authorisation*
Exceptions	1. The re-authorisation data is incorrect; the user is not logged in, the re-authorisation data is removed, and an error message is displayed to the Visitor. 2. The re-authorisation data cannot be stored on the Client; an error message is displayed to the Visitor.
Comments	1. Re-authorisation data is usually stored in cookies. 2. When the user is automatically logged in, a message is displayed once to the User, informing they have automatically been logged in. 3. A more modern method of storing this data is not to store the username and password on the server; but instead, store a "login key" on both the server and the client, which is used to verify the login. This also has the added benefit of surviving password changes, but has the problem of reduced security of access from compromised machines, unless existing login keys can be revoked. 4. Generally sites using automatic authorisation also re-request for the user's password before embarking on critical operations, such as changing user account details or deleting their account. 5. Some sites, such as banking sites, should never have automatic authorisation enabled.

UC-18	**Static Views (HTML)**
Description	A calendar web application wishes to publish an HTML-only version of a calendar
Preconditions	The calendar page can be represented purely in HTML
Actors	Visitor, Server
Normal Sequence	1. A Visitor requests the HTML-only version of a calendar page 2. The Server constructs the page by removing all scripting from the page 3. The Visitor receives the HTML-only page 4. For a User to add events to the calendar, (a) The User clicks the Add Event button, which loads the event entry page (b) The User fills in the form, with no help from Javascript or dynamic technologies (c) The User submits the form, and the Server saves the new event
Postconditions	
Exceptions	
Comments	1. Useful for users which do not have Javascript enabled, or old/slow/incompatible browsers 2. It may be probably possible to convert any rich features to HTML? 3. If necessary, the Server could emulate a user session on the server itself. An emulated session could assist in development of features that are not globally supported, e.g. ActiveX features emulated for Firefox users 4. May be related to *Multiple Browser Support* 5. See also *Backwards-Compatible Scripting*

UC-19	**Asynchronous Form Validation**
Description	Form validation can occur via asynchronous communication with the server; see *Client Form Validation* for a Client-only version
Preconditions	Preconditions in *Client Form Validation*
Actors	Visitor, Client, Server
Normal Sequence	1. The Visitor is filling out an entry field in a form (e.g. an e-mail address in a registration form) 2. Once the field is filled out, the Client submits the value to the Server for validation 3. If the Server finds the input is invalid (e.g. the address is malformed), it returns the result to the Client 4. The Client graphically highlights the field, and displays a message next to it informing the Visitor the input is invalid, why it is invalid, and how to fix it ("this e-mail address already exists in our database; would you like to login instead?") 5. The Visitor corrects the error (e.g. a valid email address) 6. The input is submitted again to the Server, and it is now correct; the field highlight and error message is removed
Postconditions	Postconditions in *Client Form Validation*
Exceptions	1. The corrected value (Step 6) is incorrect; the process goes back to Step 3 2. The Client cannot contact the Server for validation; a message may be displayed to the Visitor, but validation should also occur on the Server after submission
Comments	1. See *Client Form Validation* comments 2. Care must be taken to ensure that the connection is not saturated with validation requests (e.g. validating on every key stroke) 3. In some cases (e.g. unique usernames), the server result could be cached locally, so multiple inputs of the same data does not re-poll the server 4. Sometimes, this depends on the capability of the client, and it is desirable to decide where to validate a form at request time 5. Even though validation may occur on the client, it still **must** occur on the server before any data is retained

UC-20	**Client Form Validation**
Description	Fields can be validated on the Client via Javascript
Preconditions	There is a field that needs to have a valid value
Actors	Visitor, Client
Normal Sequence	1. The Visitor is filling out a destination e-mail address form (sending an e-mail) 2. Once the field is filled out, the e-mail address is parsed on the Client for validity (perhaps through regular expressions) 3. If the address is invalid: (a) The field is highlighted and an error message is displayed, and possibly a way to fix the problem ("this e-mail address is invalid; there is no @ character.") (b) The Visitor goes back to the invalid field and corrects the problem (c) The Client re-checks the field for validity, and it is valid (d) The highlight around the field is removed, and the error message is removed
Postconditions	The field has a valid value
Exceptions	1. The Client does not have Javascript enabled, or validation cannot occur; an error message is displayed to the User, and validation must occur on the Server (*Server Form Validation*)
Comments	1. See *Server Form Validation* comments 2. All Client-side operations and validation must be treated as insecure, and all validation must be re-executed on the Server 3. In some cases, it may be appropriate to disable sending the form until all attributes in the form are valid 4. The client and server form validations could be integrated together into one set of rules 5. The key difference between this use case and *Client Form Validation* is that this use case only occurs once the entire form is completed

UC-21	**Server Form Validation**
Description	When a form is submitted to the Server, the Server can assess the form for validity; if it is invalid, the form is redisplayed to the Visitor with the errors highlighted
Preconditions	There is a field in a form that needs to have a valid value
Actors	Visitor, Server
Normal Sequence	1. The Visitor is presented with a user registration form 2. The Visitor fills out the form with incorrect data and submits it to the server 3. The Server validates all the inputs of the form, and recognises that some of the inputs are invalid (e.g. malformed e-mail address, illegal username) 4. The Server redirects the Visitor back to the form entry page, but highlights all the incorrect fields, and displays a list of problems with the submission 5. The Visitor corrects all the values on the form, and resubmits the form to the Server 6. The Server re-validates all the inputs of the form, and recognises that they are all valid. The registration is saved to the database.
Postconditions	The field has a valid value
Exceptions	1. The Server cannot carry out the validation; an error message is displayed to the User and Administrator, and execution is not allowed to continue
Comments	Instead of simply saying “this input is invalid”, the Server could respond with a list of possibilities (“did you mean X?”)

UC-22	**Multiple Browser Support**
Description	The application can be rendered identically to multiple browsers, or take advantage of browser-specific features
Preconditions	
Actors	Client, Server
Normal Sequence	1. The Client requests a page 2. The Server identifies the browser from the user request, in this case Internet Explorer 6: (a) The Server constructs a page response that is standard to all target browsers (b) The Server applies fixes to solve the differences in rendering techniques, for example Internet Explorer 6's incorrect CSS *float* model (c) The Server sends this result to the Client
Postconditions	
Exceptions	1. No user agent can be identified; it sends the standard page response (Step 2a) 2. The user agent is incorrectly identified by the Server; there should be a way for the Visitor to change the target browser, but it should be as transparent as possible
Comments	1. The current user agent can be identified using the standard HTTP headers, or inline Javascript/CSS code. 2. This could be cleanly implemented with a conditional processing instructions [9] 3. This could also lend itself to multiple output formats: (a) XML + XSL (client-side) (b) XHTML (1.0, 2.0) (c) HTML (3.0, 4.01, 5.0) (d) etc. 4. This technique can be useful to handle different mobile phone clients (*Mobile Phone Support*)

UC-23	**Mobile Phone Support**
Description	Website visitors using a mobile phone can automatically be redirected to a design specifically created for mobile users
Preconditions	The Visitor is visiting a normal web application with a mobile phone; a web application designed for mobile phones exist
Actors	Visitor, Server
Normal Sequence	1. The Visitor requests a page on the website 2. The Server realises the user is on a mobile phone 3. The Server redirects them to the mobile phone version of the same web application
Postconditions	The Visitor is browsing the web application designed for mobile phones
Exceptions	
Comments	1. Identification of mobile phones can be achieved in the same way as *Multiple Browser Support* 2. Mobile pages tend to be smaller with less content than PC equivalents 3. Mobile pages also tend to have less graphics and limited Javascript support 4. The Visitor should be able to disable this automatic redirection if desired 5. The use case *Multiple Browser Support* can help rendering the same mobile site to multiple implementations of mobile platforms

UC-24	**Remote Data Source**
Description	A calendar application has been asked to import another calendar (in iCal format) from an external web server.
Preconditions	The application can import iCal calendar sources; calendar events can be stored in a local database
Actors	User, Server, Remote Server
Normal Sequence	1. The User has requested to import an external iCal file 2. The Server contacts the Remote Server and requests the iCal file 3. The Server downloads and parses the iCal file 4. For each event identified, (a) It checks to see if the event already exists in the database (b) It adds the event to the database
Postconditions	The remote calendar has been loaded into the local database
Exceptions	1. The Server cannot access the Remote Server; an error message is displayed to the User 2. The iCal file cannot be parsed; an error message is displayed to the User 3. A single iCal event cannot be parsed; the event is ignored, and a message is displayed to the User 4. An event already exists in the database; it can either be ignored, or updated with the new event data
Comments	1. This does not cover removing events that no longer exist in a remote source 2. This process occurs only once; for accessing a remote data source regularly, see *Active Remote Data Source*

UC-25	**Active Remote Data Source**
Description	A calendar application has been asked to continuously import iCal events from an external web server, actively keeping the local calendar updated at all times
Preconditions	1. Preconditions in *Remote Data Source* 2. The Server has a registered remote data source for a given calendar 3. The Server has calendar data which has not been updated for more than 24 hours
Actors	Server, Remote Server
Normal Sequence	1. The Server locates the remote data source 2. The Server uses *Remote Data Source* to update the data 3. The Server marks the current time as the last calendar update
Postconditions	Postconditions in *Remote Data Source*
Exceptions	1. Exceptions in *Remote Data Source*
Comments	1. See *Remote Data Source* 2. In some cases it can be useful to force a data refresh outside of the regular interval 3. For data which rarely changes, the Server could automatically reduce its access interval 4. Multiple requests to the same data source could be cached locally 5. This process could be aggregated into one automated process/server

UC-26	**Data Feeds**
Description	The application can provide external data feeds about new blog entries to external clients/servers via RSS.
Preconditions	Blog entries can be fed through an RSS feed [10]
Actors	Server, External Server
Normal Sequence	1. An External Server requests an RSS feed of the most recent blog entries 2. The Server compiles an RSS feed of the most recent blog entries and returns it to the External Server
Postconditions	
Exceptions	1. The RSS feed is requested too many times by one client; the client is automatically denied access for a few hours, and the administrator is notified, to prevent denial of service attacks
Comments	1. This could be generalised into a specific type of *Web Service* request 2. Some data feeds could use emulated user permissions, see below 3. Modelling may support RSS as general mechanism, i.e. to have modelling elements that can extract RSS summaries from resources, and to provide an abstraction from concrete RSS dialects (RRS1.0, RSS2.0, ATOM). This needs to be customisable, w.r.t. display, security, etc.

UC-27	**Web Service**
Description	The application can provide external functionality support to its services via web service communication protocols such as SOAP [11], such as converting currency.
Preconditions	A web service API is published for the local web application; the web application has exchange rates between US$ and Yen.
Actors	Server, External Server
Normal Sequence	1. An External Server requests a web service function call via SOAP, to convert US $50 to Yen. 2. The Server retrieves its exchange rate for US$ and Yen, and converts this into a number 3. The Server responds to the External Server with the result using the SOAP framework
Postconditions	
Exceptions	1. The External Server does not have the permissions to call this web service; the request is denied
Comments	1. If this web service also calls a web service itself, we must make sure to prevent infinite loops of web service calls 2. Alternative web service communication protocols includes JSON [12] and the outdated XML-RPC protocol

UC-28	**Back/Forwards Button Control**
Description	The application can control where the back and forward buttons go, even on applications which do not change the browser history by default
Preconditions	The User is using an e-mail application which is entirely rendered on the client
Actors	User, Client
Normal Sequence	1. The User is in the e-mail inbox page 2. The User clicks on an e-mail, which replaces the view with the e-mail contents 3. The Client adds the e-mail inbox page to the browser history 4. The User clicks on the reply link, which adds a reply box to the bottom of the e-mail, and redirects focus to the bottom of the email 5. The Client adds the e-mail page to the browser history 6. The User enters in a reply 7. The User clicks the inbox link 8. The User redirected to the e-mail inbox page
Postconditions	
Exceptions	1. Step 4, the user clicks Back; the User is taken to the e-mail inbox page, and the e-mail page is the next page 2. Step 6, the user clicks Back; the User is taken to the e-mail page, and the reply box is shown in the next page 3. Step 8, the user clicks Back; the User is taken to the e-mail page, with the last reply still in the content field. The e-mail inbox is shown in the next page, but the user is asked if they want to lose their changes if they go forwards.
Comments	1. If the User ended up sending the e-mail (after Step 6), going back to the reply box would have presented an empty reply 2. An application should always have back button support; disabling this button causes great user stress 3. Sometimes going back may require either the old view to be displayed (writing a reply), or the view refreshed (e-mail inbox), depending on the situation

UC-29	**Opening New Windows**
Description	The User can open links in new windows, even if the application is solely rendered via the Client
Preconditions	
Actors	User, Client
Normal Sequence	1. The User is in the e-mail inbox page 2. The User clicks on an e-mail, but asks for it to be opened in a new tab/window 3. The e-mail is displayed in the new tab/window by the Client, with no previous browser history
Postconditions	
Exceptions	1. The browser cannot open a page in a new window or tab; the page is opened in the current window instead, and the browser history is affected accordingly (see *Back/Forwards Button Control*)
Comments	1. Some links should not be allowed to open in new windows (e.g. actions), but these should not be rendered as text links, but as buttons 2. Also refer to *Pop-up Window Support*

UC-30	**Client-Side Application**
Description	The entire application user interface can be rendered by Javascript and XML by the Client, instead of HTML from the Server
Preconditions	
Actors	User, Client, Server
Normal Sequence	1. The User requests the e-mail website 2. The Client downloads the Javascript to render the page 3. The Client executes the Javascript, which downloads an XML file of inbox e-mails 4. The Client draws the user interface, using the information from the XML 5. The User can interact with this user interface like any normal website, except most operations are run on the Client, not on the Server
Postconditions	
Exceptions	1. The Javascript cannot be executed, or the user interface cannot be rendered in this browser; the User is redirected to a non-Javascript page 2. The XML cannot be loaded; the User is redirected to a non-Javascript page 3. The loading process takes too long; the User is redirected to a non-Javascript page
Comments	1. This is not an example of a browser displaying XML+XSL; it is the use of Javascript and XML data islands to construct an HTML DOM in memory at run time on the client 2. Requires *Back/Forwards Button Control* 3. This naturally tends towards storing some data on the Client side, and operating on it there; see *Local Data Storage*

UC-31	**Communication with Software**
Description	A web application can send messages to external software applications
Preconditions	An e-mail site has a chat service; an external software application is also connected to this chat service
Actors	User, Server, Software
Normal Sequence	1. A User is logged into a chat program, with the desktop Software 2. On the Server, the User receives a new e-mail message 3. The Server sends a notification to the desktop Software about the new e-mail, which renders as a pop-up box 4. The User clicks on the notification 5. The User is sent to a special URL using their Client, which opens in a new browser window 6. The Server displays the e-mail
Postconditions	The User is viewing the received e-mail
Exceptions	1. The Software cannot open a new browser window; an error message is displayed to the User 2. The authentication between the User and the Software expires before the Server can re-authenticate the User; the process is stopped at Step 6, and the User is asked to re-authenticate themselves
Comments	1. Usually when the notification is clicked, the User is sent to a special URL, which is constructed to automatically authenticate the user (*User Authorisation*) if they are not already authenticated 2. Very common in chat programs integrated with e-mail clients, e.g. Google Talk and GMail, or MSN Messenger and Hotmail 3. The notification (Step 3) could either be passive (polling) or active (an open connection is sent notifications) 4. Passive polling can be achieved using a *Web Service* by the Software

UC-32	**Mobile Phone Communication**
Description	A user can get notifications via mobile phone (SMS)
Preconditions	1. The web application receives e-mails 2. The User has a registered mobile phone number, and the number is contactable from the Server
Actors	User, Server, Device (mobile phone)
Normal Sequence	1. On the Server, the User receives a new e-mail 2. The Server sends a notification message to the User via a mobile network 3. The mobile network transports the notification to the User's mobile phone 4. The Device (mobile phone) receives the notification and displays the message to the User
Postconditions	
Exceptions	1. The mobile network is down, or cannot be contacted; the User does not receive the notification until the network is back up, or the notification is lost 2. The mobile number no longer exists, or does not support messaging; the Server removes the association with the User and informs the User the next time they log in 3. The mobile phone is offline; the User will receive the message when they next turn on their phone
Comments	1. On some networks, it may cost to send messages; the application may need to maintain some sort of user account balance 2. The notification message sent could include a URL, similar to *Communication with Software*, which the User can open and browse with their mobile phone

UC-33	**E-mailing Users**
Description	The User can be sent e-mails
Preconditions	The web application supports internal messages; the User has requested to be e-mailed about new messages sent internally
Actors	User, Server
Normal Sequence	1. In the web application, the User is sent a new internal message 2. The Server composes an e-mail to be sent to the User, including the internal message 3. The Server sends the e-mail 4. The User receives the e-mail 5. The User reads the internal message included
Postconditions	
Exceptions	1. The e-mail bounces immediately (invalid address); the notification service is disabled, and the User is notified the next time they log in 2. The e-mail bounces after a while (cannot contact server); the message is discarded or added to a queue to retry later 3. The User never receives or downloads the e-mail; nothing happens
Comments	1. This is a basic requirement of database-driven websites 2. One important aspect of sending e-mail messages is managing bouncebacks, and the lifecycle of e-mail accounts; this needs to be considered in any serious web application 3. Also related to *Mobile Phone Communication*, emails are another example of an event delivery mechanism

UC-34	**E-mail Unsubscription**
Description	Any Visitor can unsubscribe their e-mail address from the web application, to prevent any further e-mails
Preconditions	The web application can send e-mails (*E-mailing Users*)
Actors	Visitor, Server
Normal Sequence	1. The Visitor requests to be unsubscribed from all future e-mails to a particular address 2. The e-mail address is sent a confirmation e-mail, along with instructions on how to reverse the process 3. The Visitor clicks the link in the confirmation e-mail to disable all future e-mails 4. The Server adds this e-mail address to a "do not contact" list 5. Any future time an e-mail is being sent out: (a) The destination address is checked against the list of people to not contact (b) If the address exists in the invalid list, an error message is displayed, and the message is discarded
Postconditions	The Visitor is never e-mailed again
Exceptions	1. The e-mail address has never been contacted before; the e-mail address is still added to the "do not contact" list 2. There are some e-mails in the e-mail queue for the user; these e-mails are deleted once confirmation has been given by the user, and if possible, error messages will be displayed to the Administrator
Comments	1. The ability to permanently unsubscribe from web services is important to gain user trust, and also satisfy privacy regulations 2. Any further e-mails to this user must be sent manually, and cannot be sent by the system 3. The Visitor may, in the future, remove themselves from the "do not contact" list, by following the instructions in the e-mail in Step 2, or contacting the administrators 4. Steps 2 and 3 might not be possible, depending on privacy requirements

UC-35	**Persistent Errors**
Description	If an error occurs during an operation (such as sending an e-mail to an invalid address) when the source of the operation cannot be contacted, the error message is stored until later
Preconditions	The system can send e-mails (*E-mailing Users*); error messages can be stored by the Server for particular Users
Actors	User, Server
Normal Sequence	1. The User tries to send an e-mail to an address that doesn't exist 2. The e-mail is put into a local queue on the Server while it is trying to be sent 3. The User leaves the web application 4. The e-mail bounces back to the Server as undeliverable; the Server stores an error message for the User 5. The User returns to the web application 6. The error message is displayed from the Server along with the original message
Postconditions	The error message is removed from the Server once displayed
Exceptions	1. The User never returns to the web application; nothing happens
Comments	The error message could also be delivered via e-mail to the User

UC-36	**User Content Security**
Description	Content can be shared between users, with specific permissions
Preconditions	1. There exist three Users: User 1, User 2, User 3 2. A User 1 has created and published a blog entry 3. User 2 and User 3 cannot usually access the blog entry
Actors	User 1, User 2, User 3, Server
Normal Sequence	1. User 1 assigns reading rights to User 2 2. User 2 tries to view the blog entry 3. The Server checks the entry permissions and allows User 2 to view the entry 4. User 3 tries to view the blog entry 5. The Server checks the entry permissions, but displays an error message to User 3 and does not let the User continue
Postconditions	1. User 2 can view the entry 2. User 3 cannot view the entry
Exceptions	1. User 2 views the blog entry, but then the rights are revoked; User 2 can continue to view the entry until the page is closed
Comments	Assigned permissions should be accessible and modifiable by User 1

UC-37	**Private User Content Security**
Description	A User can share private feeds to another Visitor, which emulates the first User's account to achieve the desired rights
Preconditions	Preconditions in *User Content Security*
Actors	User, Visitor, Server
Normal Sequence	1. The User shares a private URL with the Visitor to view an RSS feed of all their blog entries 2. The Visitor visits this private URL without being authenticated 3. The Server temporarily assigns the Visitor the rights of the User, to allow them to view the request 4. The Server checks the entry permissions and allows the Visitor to view the entry
Postconditions	Postconditions in *User Content Security*
Exceptions	1. Exceptions in *User Content Security* 2. The User revokes or regenerates the private URL; the original URL will no longer provide access
Comments	1. Often this is achieved by adding a special authentication key to a request 2. If a third party discovers this private URL, they can gain unauthorised access to this private data 3. Another common example is sharing a private calendar with a friend, which creates a temporary connection with very specific rights 4. Used in applications to share private data with other users, or to integrate with external feed readers without requiring separate authentication 5. This differs from *User Content Security* as that it does not assign permissions to any particular registered User

UC-38	**User Collaboration**
Description	Different users can work on the same calendar
Preconditions	1. An editable calendar is shared between two users, User 1 and User 2 2. Calendar data can be stored locally (*Server Data Access*)
Actors	User 1, User 2, Server
Normal Sequence	1. User 1 goes to the shared calendar 2. User 2 goes to the shared calendar 3. User 1 adds a new event to the calendar 4. The new event is submitted to the Server 5. The Server notifies User 2 of the new event 6. User 2's Client adds the new event to its model 7. User 2 edits the new event, changing the date 8. User 2 submits the change to the Server 9. The Server notifies User 1 of the new event 10. User 1's Client updates the existing event in its model with the changes
Postconditions	The system state is consistent
Exceptions	1. User 1 and User 2 make a change at the same time on different events; both users are notified of each others change 2. User 1 and User 2 make a change at the same time on the same event; the event is changed to the last update, and both users are notified of the change
Comments	1. Steps 5 and 9 do not have to be instant; they could only occur every 60 seconds, for example; but the core concept of this is that it is interactive and realtime 2. Other examples of this include Writely, Spreadsheets

UC-39	**Interactive Map**
Description	A visitor is given an interactive graphical map of the world, which they can move around using their mouse
Preconditions	A map can be rendered as an image; this image is broken up into many smaller, separate pieces
Actors	Visitor, Client, Server
Normal Sequence	1. The Visitor is displayed an image of part of a map 2. The Visitor uses the mouse to drag the map to the right 3. The Client moves the map image according to the mouse movement 4. If the newly-revealed parts of the larger map image have not been loaded yet: (a) The Client requests the Server for the map parts (b) The Server returns the image parts (c) The Client displays the new images by replacing older images in the map
Postconditions	
Exceptions	1. The Client cannot contact the Server; an error message is displayed, or a "missing image" is displayed 2. The Server does not have map imagery for the missing co-ordinates; a "no image exists; zoom out" image is returned instead
Comments	1. This allows for Clients to browse a large image by only downloading the parts relevant to the current query 2. There must be some way to jump to specific co-ordinates in the larger map, without having to browse the map first 3. The obvious example is Google Maps

UC-40	**Drag and Drop**
Description	The user can use drag and drop to intuitively move e-mails from one folder to another
Preconditions	
Actors	User, Client, Server
Normal Sequence	1. The User is displayed a list of e-mails in their inbox 2. The User uses the mouse to drag and drop an e-mail from the inbox to the trash icon 3. While being dragged, a highlight of the e-mail is displayed under the mouse cursor 4. When dropped, the Client: (a) Contacts the Server to move the selected e-mail to the trash (b) Removes the selected e-mail from the inbox, and updates the trash messages count 5. The Server moves the selected e-mail to the trash
Postconditions	
Exceptions	1. The escape key is pressed while dragging; the operation is cancelled 2. The e-mail is dragged to an invalid area on the screen; nothing happens 3. The e-mail is dragged outside the Client window; nothing happens
Comments	

UC-41	**Client Timer Support**
Description	The Client can automatically save a draft of the e-mail every ten minutes
Preconditions	The Client is working on a local e-mail (*Server Data Access*); the Client can access the Server
Actors	User, Client, Server
Normal Sequence	1. The User is working on an e-mail on the Client 2. Every ten minutes: (a) The Client submits a copy of the e-mail to the Server (b) The Server saves the e-mail as a draft copy (c) The Client displays "Auto-saved at (time)"
Postconditions	The e-mail object is automatically saved every ten minutes
Exceptions	1. The e-mail has not changed in the last ten minutes; the Server is not contacted 2. The Server cannot be contacted for any reason; the e-mail draft save is ignored, and the auto-saved message changes to "Could not auto-save" 3. The e-mail no longer exists on the Server; an error message is displayed to the User, and the User is asked if they would like to save the current draft
Comments	Usually implemented with a setTimeout() command through Javascript

UC-42	**Server Timer Support**
Description	The Server can automatically e-mail users new products every four hours
Preconditions	1. New products are added to a database on the Server 2. A User has requested that they are e-mailed every four hours of new product listings 3. This process has not occurred for at least four hours
Actors	User, Server
Normal Sequence	1. The Server gets a list of all the products added in the last four hours 2. The Server composes an e-mail listing all of these new products 3. The Server sends the e-mail to the User (*E-mailing Users*) 4. The current time is saved as the time this process was last executed
Postconditions	The User has received an e-mail of new products
Exceptions	1. The e-mail cannot be sent; see exceptions in *E-mailing Users* 2. There are no new products added in the last four hours; Steps 2 and 3 are skipped 3. The process does not complete; an error message is displayed to the Administrator
Comments	1. Other examples include new listings on auction sites, or digesting the replies from private messages 2. This is different from *Scheduled Events* as this use case is executed every 4 hours; *Scheduled Events* occurs at the same specified time every day

UC-43	**Page Caching**
Description	A frequently-requested home page can be cached to reduce unnecessary processing requirements on the Server
Preconditions	The home page has some sort of caching setting (five minutes)
Actors	Visitor, Server
Normal Sequence	1. A Visitor requests the home page 2. The Server checks the cache to see if a cached version exists, and its age 3. If the page is too old, the page is recomputed in this request and stored in the cache 4. The cached result is supplied to the Visitor
Postconditions	
Exceptions	1. No cached version of the current page exists; it is recompile from scratch (as in Step 3) 2. The current Visitor is an Administrator; the cache is ignored and the page is reconstructed every time
Comments	1. This can also be extended to handle page components; e.g. one section of the page is user account info, another is the home page 2. Cached pages can integrate with other technologies such as *memcached* or *Squid*, or other Content Delivery Networks (CDNs) 3. Page caching should not occur when the application is in a *Debug Mode*

UC-44	**Offline Application Support**
Description	A Client-based application can be taken offline and continued to be worked on, but no changes can be saved until it is taken back online
Preconditions	Preconditions in *Server Data Access*
Actors	User, Client, Server
Normal Sequence	1. The User is writing an e-mail 2. Access to the web application is taken offline, e.g. a network failure 3. The Client displays a message that the application is offline, but the current e-mail can still be edited 4. The User continues to work on the e-mail, without being able to save it or send it 5. The application is taken back online; the connection is restored with the Server 6. The ability to save and send the e-mail is restored
Postconditions	Postconditions in *Server Data Access*
Exceptions	1. The User abandon the e-mail while the application is offline; a warning message is shown and the inevitable data loss is confirmed 2. The User tries to send the e-mail whilst offline; an error message is displayed to the User informing they cannot send the e-mail until online again
Comments	1. This use case covers the functionality of an offline application, not the storage of data locally 2. A limited amount of information could be stored through the technologies mentioned in *Persistent Client Data* 3. In some cases, e-mails could still be sent by the client, except they would be queued up until network connectivity is restored

UC-45	**Loading Time Support**
Description	While an application is loading, the Server should inform the the User of the progress through application loading. If the application will take too long to load, the Server should request the User switches to a different version.
Preconditions	The Client can judge the loading process
Actors	Visitor, Client, Server
Normal Sequence	1. A Visitor tries to download a large application script 2. The Client starts the download timer 3. After 60 seconds, the Client suggests that the Internet connection is too slow 4. A message is displayed informing the User about the situation, allowing them to continue waiting (the default), or providing them with a link to a simpler site
Postconditions	The web application is loaded, or the User has been redirected to a simpler site
Exceptions	1. The application never loads, or takes more than five minutes; the loading process is abandoned and the User is taken to the simpler site by default
Comments	1. The *Static Views (HTML)* use case allows for static HTML pages that could be simpler than the default web application 2. See also *Multiple Browser Support*

UC-46	**Flash MP3 Support**
Description	An e-mail with an MP3 attachment can be streamed and played in the webmail application using a custom Flash component
Preconditions	The User has an e-mail with an MP3 attachment; the Client can render Flash components
Actors	User, Client, Server
Normal Sequence	1. The User opens an e-mail with an MP3 attached 2. The Server returns the e-mail page, along with a Flash Object 3. The Flash component is loaded by the Client and displays a play button 4. Once the play button is pressed, the MP3 is streamed from the Server by the Flash component, and played back to the User
Postconditions	
Exceptions	1. The User does not have Flash installed; the Server or Client detects the lack of Flash and instead displays a link to download the MP3 itself 2. The Flash Object fails to load; a link to download the MP3 file is displayed instead 3. The Server fails to stream the MP3; the Flash component is replaced with a link to download the MP3 file, or try contacting the Server again
Comments	1. If Flash is not installed in the Client, the Client can ask the User if they would like to install Flash 2. Other options for MP3 support include Java applets and the Quicktime plugin 3. The upcoming HTML5 standard [13] includes a native component for playing media files

UC-47	**Flash Communication Support**
Description	A web page wants to display a slider control for number of e-mails to display per page, using a custom Flash component
Preconditions	
Actors	User, Client, Server
Normal Sequence	1. The User requests the list of e-mails 2. The Server returns the list of e-mails, along with a slider Flash Object 3. The Client loads the Flash component along with the current setting as a parameter (50 e-mails per page) 4. The User drags the slider up to 75 5. The Flash component sends a message to the Client via Javascript of the new e-mails value 6. The Client requests the Server for the 25 previously undisplayed e-mails 7. The Server responds with the 25 e-mails 8. The Client adds the 25 e-mails to the bottom of the e-mail list
Postconditions	
Exceptions	1. The Client cannot load the hidden e-mails; the display remains unchanged, and the slider is reset 2. The Client cannot load a Flash component; a normal slider, or text entry field, is displayed instead
Comments	1. The Client should also be able to send messages back to the Flash component, possibly through Javascript 2. If Flash is not installed in the Client, the Client can ask the User if they would like to install Flash 3. Flash component are often used when fluidity and richness of user interface components is a major concern 4. This is a generalised use case of *Communication with Plugins*

UC-48	**Internationalisation Support**
Description	A web application can have support for multiple locales
Preconditions	A web application is translated into at least two locales
Actors	Visitor, Server
Normal Sequence	1. The Visitor visits the web application 2. The Server decides the most likely locale for the User, either from request location or HTTP request header 3. The Server checks to see which supported locale is the closest match 4. The Server presents the page with the translated components
Postconditions	
Exceptions	1. The automatically selected locale is incorrect; the Visitor can select a different locale
Comments	1. Different locales can be provided by different Servers or represented with different URLs 2. The chosen locale should be stored, probably with a cookie, for future requests

UC-49	**Logout Control**
Description	If a User logs out from a website, it should not be possible to repeat an activity as that User
Preconditions	The User is authenticated
Actors	User, Server
Normal Sequence	1. The User composes a message, and sends it through the web application 2. The User logs out and becomes a Visitor 3. The Visitor presses the back button on their browser, and tries to resubmit the message 4. The Server realises the Visitor has logged out, and instead displays an error message, asking the Visitor to re-authenticate
Postconditions	The Visitor cannot do any User actions
Exceptions	
Comments	1. If the Client could recognise the Visitor has logged out, the Client could display an error message instead of allowing the Visitor to browse back in the navigation history 2. This is an important security-related use case

UC-50	**Single Sign-In Solutions**
Description	An application can use a single centralised sign-on service (SSO) for authentication support
Preconditions	A single sign-on provider exists; the User has an account with this service
Actors	Visitor, User, Server
Normal Sequence	1. The Visitor authenticates with a web application 2. The application redirects the User to the SSO service 3. The User authenticates with their SSO details 4. The SSO page informs the Server of the login success 5. The Visitor is redirected to the Server, and is authenticated as a User
Postconditions	The Visitor is authenticated as a User
Exceptions	1. The User does not have an account with the SSO service; the SSO service can let the User create a new account 2. The given SSO ID has been blocked on the local web application; an error message is displayed to the Visitor, and authentication is blocked 3. The User cannot authenticate with the SSO service; the SSO service should help with re-establishing authorisation (e.g. lost password)
Comments	1. A common SSO service is Google Accounts 2. A common decentralised SSO service that provides identity, but not trust, authentication is OpenID [14]

UC-51	**User Redirection**
Description	A visitor can search a database of sites with a search string, and be redirected to a given site
Preconditions	The Visitor can search a database of sites
Actors	Visitor, Server
Normal Sequence	1. The Visitor searches for the search string "puppies" 2. The Server returns a list of all the results for "puppies" in its database 3. The Visitor clicks the first search result, which leads to a click-through URL on the Server 4. The Server notes that the Visitor clicked the first search result for "puppies" 5. The Server sends the Visitor a redirection response to the actual search result URL, `puppies.com` 6. `The Visitor is redirected to the destination URL, puppies.com`
Postconditions	The Visitor is now on the third party website `puppies.com`
Exceptions	
Comments	1. The ability to redirect the user is a basic requirement of database-driven websites 2. Redirects can either be internal (to the current web application) or external (to third party web applications) 3. User redirection can be chained multiple times; care must be taken to prevent infinite redirections

UC-52	**Keyboard Shortcuts**
Description	A web application can implement keyboard shortcuts to help power users
Preconditions	The User has a browser which is capable of intercepting key strokes
Actors	User, Client
Normal Sequence	1. The User logs into their e-mail inbox 2. The User presses the “T” key to move to the Trash folder 3. The Client picks up the “T” key being pressed and “clicks” the Trash link 4. The User is redirected to the Trash folder
Postconditions	The User is viewing the Trash folder
Exceptions	1. The Client cannot intercept the key stroke (e.g. no Javascript support); nothing happens
Comments	1. Keyboard shortcuts are useful for power users 2. Keyboard shortcuts should not be too intrusive, and must not occur when trying to enter in actual data

UC-53	**Undo/Redo Support**
Description	Whilst working on an online document, the user can undo and redo actions
Preconditions	The User is editing a document locally (see *Server Data Access*), with "Bold", "Undo" and "Redo" buttons
Actors	User, Client, Server
Normal Sequence	1. The User sets a selection of text to be bolded, and presses the "Bold" button 2. The Client informs the Server, and the Server bolds the text on the document 3. The Client renders the bolded text 4. The User presses the "Undo" button 5. The Client requests an Undo from the Server 6. The Server unbolds the text on the document, and informs the Client of the change 7. The Client renders the un-bolded text 8. The User presses the "Redo" button 9. The Client requests a Redo from the Server 10. The Server bolds the text on the document, and informs the Client of the change 11. The Client renders the bolded text
Postconditions	The User is editing the same document
Exceptions	1. The previous action cannot be undone; the Undo button is disabled 2. An undone action cannot be redone; the Redo button is disabled
Comments	1. Could be implemented through an *Action* design pattern [15] 2. Combining *Undo/Redo Support* with *User Collaboration* may introduce some complex synchronisation problems

UC-54	**Browser-Based Chat**
Description	Users can open up a browser-based chat window with other users, connected through the server
Preconditions	1. There are two Users in the same chat room, opened with two Clients 2. Each Client has a connection to the Server 3. User 2 is already connected to the Server
Actors	User 1, User 2, Client 1, Client 2, Server
Normal Sequence	1. User 1 connects to the chat room 2. Client 1 opens a connection with the Server 3. The Server sends the Client 1 a list of previous chat messages, which are displayed to User 1 4. The Server sends a message to Client 2, informing them that User 1 has connected 5. User 2 sends a chat message 6. Client 2 sends the chat message to the Server 7. The Server sends the message to all connected Clients, displaying them to their Users
Postconditions	
Exceptions	1. The connection to the Server cannot be opened; an error message is displayed to the User, and the chat window is closed 2. The connection to the Server is lost; the Client attempts to reconnect the connection; once connected, the Server resends the Client a list of any activity whilst disconnected 3. The connection to the Server is lost, and cannot be reconnected; an error message is displayed to the User, and the chat window is closed
Comments	1. This is one implementation of a browser-based chat session, with the messages sent over the open connection and interpreted once received 2. A different implementation (more common) uses active polls on open connections to keep synchronised with server activity 3. This could be implemented without client-side Javascripts (regular page refreshes)

UC-55	**Pop-up Window Support**
Description	An application can open pop-up windows to assist users in adding images to e-mails
Preconditions	A User is composing an e-mail (see *Server Data Access*)
Actors	User, Client, Server
Normal Sequence	1. The User clicks a button to add an image to the e-mail 2. The Client opens a pop-up window displaying a list of available images 3. The User clicks an image to add to the e-mail 4. The Client closes the pop-up window and adds the image to the e-mail composition
Postconditions	The image is added to the e-mail display
Exceptions	1. The User does not select an Image, and closes the window; the Client does not add any image to the e-mail 2. The User ignores the pop-up window, and only interacts with it once the e-mail application has been closed; any further action on the window either does nothing, or displays an error message and closes the window
Comments	This can be implemented with any number of technologies, including a normal browser popup; a modal popup; a Java application; a Flash component; or a Javascript message dialog

UC-56	**Incompatible Client Warning**
Description	If the Visitor visits the website with a browser/client that cannot fully support all the features the site absolutely requires, the Visitor is given a warning
Preconditions	The web application has some elements that cannot be rendered in the Client
Actors	Visitor, Client, Server
Normal Sequence	1. The Visitor visits the website with their incompatible Client 2. The Server recognises the Client, compares it to the list of features needed for the website to operate correctly 3. The Server identifies the Client is incompatible or misses some features (a) Proceed anyway, at their own risk (b) Access an alternative site (e.g. *Static Views (HTML)*
Postconditions	
Exceptions	1. The Client cannot be identified; the Server assumes the Client is fully functional
Comments	1. Some website features may not be "required" but only optional, in which case the client detection should only occur when trying to access the feature 2. Runtime browser testing could be performed with scripting to work out if the Client is *actually* incompatible

UC-57	**Dynamic Objects**
Description	Depending on the Client accessing the web application, a map component in a web application can be rendered using scripting technology, or only static HTML
Preconditions	The map component cannot be rendered with scripting technology in their Client
Actors	Visitor, Server
Normal Sequence	1. The Visitor visits the website with their incompatible Client 2. The Server recognises the Client is incompatible or misses some features 3. The Server instead renders the map component with a static HTML rendering
Postconditions	The Visitor can continue to use the site in the same manner as the scripted rendering
Exceptions	1. The Client cannot be identified; the Server assumes the Client is fully functional, and uses the scripted technology rendering 2. The Client is mis-identified; the Visitor can force the web application to switch to a different rendering
Comments	This differs from *Multiple Browser Support* in that it is concerned with the rendering of individual elements, not complete applications

UC-58	**Store Data in Local Database**
Description	An application can use an offline technology, such as Google Gears [8], to store data locally in a local database which can be synchronised when network connectivity is restored
Preconditions	1. The User has Google Gears installed 2. Preconditions in *Communication with Plugins* 3. The User initially has network connectivity 4. The current web application supports offline operation
Actors	User, Client, Server
Normal Sequence	1. The User visits the web application 2. The Client creates a local database to store changes 3. The User loses network connectivity; the application continues to function as expected 4. The User interacts with the offline application 5. The Client saves changes to the local database 6. The User restores network connectivity: (a) The Client recognises the User has gone back online (b) The Client contacts the Server and publishes its changes (c) The Client deletes the information in the local database 7. The User can continue to use the application as expected
Postconditions	The system is in a consistent state
Exceptions	1. The User does not have Google Gears installed; a message is displayed to the User informing them that work will otherwise be lost when network connectivity is lost 2. The Client recognises its database is out of sync with the Server database; the Client informs the User and gives them synchronisation options
Comments	1. Data does not have to be submitted back to the server; it can be stored on the client indefinitely 2. It is a reasonable assumption that data cannot be expected to be stored forever on the client 3. The major issue with storing data locally is synchronisation, especially between multiple independent clients 4. The major use case of offline technology appears to be in interacting with mostly-read-only data sources 5. Individual resources (images, videos) can be stored through *Store Resources Locally*

UC-59	**Store Resources Locally**
Description	An application can use an offline technology, such as Google Gears [8], to store web application resources (images, CSS, Javascript etc) on the local machine, to allow the application to go offline
Preconditions	1. The User has Google Gears installed 2. Preconditions in *Communication with Plugins* 3. The User initially has network connectivity 4. The current web application supports offline operation
Actors	User, Client, Server
Normal Sequence	1. The User visits the web application 2. The Client creates a list of resources necessary for offline operation, including a remote image 3. The Client downloads these resources, including the remote image, and stores them locally 4. The User loses network connectivity 5. The User interacts with the Client, which needs to display the remote image (a) The Client retrieves the image from the local machine (b) The Client returns the image to the browser, which is displayed to the User identically to online operation
Postconditions	
Exceptions	1. Exceptions in *Store Data in Local Database* 2. An image is requested which is not in the local cache; either an empty image is displayed, or an error message is shown
Comments	1. Compared to *Store Data in Local Database*, the resources loaded in this method cannot be modified 2. The offline technology must handle modified resources whenever the resource cache is synchronised

UC-60	**Multiple Client Threads**
Description	A rendered web application can execute in multiple threads, possibly asynchronously, on the Client
Preconditions	The Client can run multiple threads
Actors	User, Client
Normal Sequence	1. A Client starts an operation 2. The Client splits the operation execution into a second thread 3. The User continues to interact with the Client in one thread, while the other thread works in the background 4. At some point, the second thread will terminate
Postconditions	The second thread has terminated
Exceptions	1. The Client cannot run threads; either the threads can be run virtually, or an error message is displayed to the User 2. The second thread terminates unexpectedly; either the thread is restarted, or an error message is displayed to the User
Comments	1. Currently no major browser supports multiple Javascript threads, but may be implemented in the future 2. A thread could be native or virtual (run with ticks in a single thread) 3. The implementation of the *WorkerPool* in Google Gears [8] achieves some level of multiple Client threading

UC-61	**Multiple Server Threads**
Description	A web application can execute in multiple threads, possibly asynchronously, on the Server
Preconditions	The Server can run multiple threads
Actors	Server
Normal Sequence	1. The Server starts an operation 2. The Server splits the operation execution into a second thread 3. At some point, the second thread will terminate
Postconditions	The second thread has terminated
Exceptions	1. The Server cannot run threads; either the threads can be run virtually, or an error message is displayed to the Administrator 2. The second thread terminates unexpectedly; either the thread is restarted, or an error message is displayed to the Administrator
Comments	Server-side multiple threading is quite common, however concurrency and scalability issues often restrict its use

UC-62	**Communication with Plugins**
Description	A client application can communicate with browser plugins, for example, interaction with Google Gears
Preconditions	A Client has the Google Gears plugin installed; the Client can directly communicate with installed plugins
Actors	User, Client, Server
Normal Sequence	1. The User visits a web application 2. The Client searches for an installed copy of Google Gears 3. The Client finds that Google Gears has been installed, and creates a new local instance of the Google Gears factory 4. The Google Gears factory can then be used to create additional objects, such as the *DataStore* and *Local Resource Pool*
Postconditions	
Exceptions	1. Step 3: The user does not have Google Gears installed; either an error message can be displayed, or the exception can be ignored
Comments	1. Plugin technologies include ActiveX (for Internet Explorer) and NPAPI (for Firefox) 2. Other use cases for communicating with plugins include identifying installed versions or executing functionality 3. *Flash Communication Support* is a particular instance of using plugins to interact with a rendered page

UC-63	**Scheduled Events**
Description	A Server can have a daily scheduled event (e.g. at 5am every day) to send new blog entries to every User on the system.
Preconditions	There is a list of Users on the system; the scheduled time has arrived (i.e. 5am)
Actors	Server
Normal Sequence	1. The Server iterates through recent blog entries and constructs a message of new entries 2. The Server iterates through all Users in the system 3. The Server sends an e-mail (*E-mailing Users*) to each User with this message 4. The Server proceeds onto the next user
Postconditions	All users have been processed.
Exceptions	1. The scheduled event is skipped; the event is executed as soon as possible 2. The scheduled event is cancelled or unexpectedly terminated; an error message is displayed to the Administrator, and the process may be restarted from where it left off
Comments	See *Server Timer Support* for a process which occurs regularly, instead of at a specific scheduled time.

UC-64	**Custom API Publishing**
Description	A blogging website can publish a *custom* API (e.g. the LiveJournal API) to allow external software/websites to post blog content
Preconditions	The Server has a published API endpoint, along with an expected API message format
Actors	Server, User or Remote Server
Normal Sequence	1. The User or Remote Server constructs a *PostMessage* message 2. The User submits the *PostMessage* message to the Server's published API endpoint 3. The Server receives and parses this *PostMessage* message 4. The Server creates a new blog entry, based on the input from the posted message 5. The Server saves the blog entry, and returns an "OK" message
Postconditions	
Exceptions	1. The message cannot be parsed; an error message is returned via the web service 2. The blog entry cannot be saved; an error message is returned via the web service
Comments	1. Compared to *Web Service*, this use case covers the publication of a *custom* API, not necessarily using a published protocol 2. This is often achieved through custom JSON [12], XML or REST APIs

UC-65	**Runtime Interface Updates**
Description	The Client can update the user interface of a calendar site based on User input, without reloading or redirecting the Client
Preconditions	The Client can run Javascript scripts; the User is viewing a monthly calendar page
Actors	User, Client
Normal Sequence	1. The User selects “two weeks” as the calendar view 2. The Client replaces the monthly view of the calendar with a fortnightly view, without reloading or redirecting the Client
Postconditions	The User is viewing a fortnightly calendar page
Exceptions	1. The Client needs to load data asynchronously from the Server, but cannot contact it; the Client instead reloads the current page, or an error message is displayed to the User
Comments	This use case includes loading data asynchronously using AJAX technologies

UC-66	**Out-of-Order Events**
Description	The Client can ignore events which come out of order, for example when requesting a list of search results that start with a particular prefix
Preconditions	The Client can send/receive asynchronous messages
Actors	User, Client, Server
Normal Sequence	1. The User enters in "a" 2. The Client sends an asynchronous request to the Server (*message 1*) for results starting with "a" 3. The Server receives *message 1*, and sends back a list of results (*response 1*) 4. The User enters in "ab" 5. The Client sends an asynchronous request to the Server (*message 2*) for results starting with "ab" 6. The Server receives *message 2*, and sends back a list of results (*response 2*) 7. The Client receives *response 2*, and displays a list of results starting with "ab" 8. The Client receives *response 1*, but recognises that message 2 was sent later; the Client ignores this response
Postconditions	A list of results starting with "ab" are displayed
Exceptions	1. *Response 2* does not arrive after a specified timeout period; the result from *response 1* is displayed, and *message 2* is resent
Comments	1. In some cases, it may be desirable to display both responses, even if they are out of order 2. In extreme cases, where the order of messages is truly important, an additional layer of network control should be implemented (similar to TCP)

UC-67	**Backwards-Compatible Scripting**
Description	Non-essential functionality in scripted web applications do not affect the operation of web applications on incompatible clients; for example, a button should light up when it is clicked
Preconditions	An incompatible client is accessing a web application with a button, that is using incompatible scripting
Actors	User, Server
Normal Sequence	1. The User clicks on a button 2. The Client tries to "light up" the button 3. The Client-side animation fails, but the failure is caught by the Client 4. The execution of the button clicking continues as normal
Postconditions	The button is clicked
Exceptions	
Comments	1. If scripted functionality is actually required, then the Client should be redirected to a more suitable application rendering, as in *Incompatible Client Warning* or *Multiple Browser Support* 2. The less desirable alternative is for the incompatible scripting to throw an error message, that prevents the button from functioning at all

UC-68	**Spellchecking**
Description	A form can be spellchecked, either at submit or at runtime.
Preconditions	A form can be edited by the User.
Actors	User, Server
Normal Sequence	1. The User enters in some incorrect text onto the form 2. The User submits the form to the server 3. The Server checks the submitted data for spellchecking, to ensure the text is entered in correctly 4. The submitted data is saved
Postconditions	The submitted data is fully spellchecked
Exceptions	1. Step 3, the submitted data is badly spelt; the Server presents the User with a list of possible corrections, and goes back to Step 1
Comments	1. Some Clients support spellchecking natively 2. If *Internationalisation Support* is supported, spellchecking must also take into account the different locales

References

1. Garrett, J.J.: Ajax: A New Approach to Web Applications. Technical report (2005)
2. Preciado, J.C., Linaje, M., Sanchez, F., Comai, S.: Necessity of Methodologies to Model Rich Internet Applications. In: WSE '05: Proceedings of the Seventh IEEE International Symposium on Web Site Evolution, Washington, DC, USA, IEEE Computer Society (2005) 7–13
3. Fraternali, P.: Tools and approaches for developing data-intensive Web applications: a survey. ACM Comput. Surv. **31**(3) (1999) 227–263
4. Schwabe, D.: A Conference Review System. In: First International Workshop on Web-Oriented Software Technology. (2001)
5. Wright, J., Dietrich, J.: Survey of Existing Languages to Model Interactive Web Applications. In: Proceedings of the Fifth Asia-Pacific Conference on Conceptual Modelling (APCCM 2008), Wollongong, NSW, Australia (2008)
6. Wright, J., Dietrich, J.: Requirements for Rich Internet Application Design Methodologies. In: Proceedings of the Ninth International Conference on Web Information Systems Engineering (WISE 2008), Auckland, New Zealand (2008)
7. Dojo Foundation: The Dojo Toolkit (2007)
8. Google Inc.: Google Gears (2007)
9. Peter-Paul Koch: Conditional comments (2009)
10. RSS Advisory Board: RSS 2.0 Specification. Technical report, RSS Advisory Board (2007)
11. W3C Group: Simple Object Access Protocol (SOAP) Version 1.2. Technical report, W3C Recommendation 27 April 2007 (2007)
12. D. Crockford: RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON). Technical report, The Internet Society (2006)
13. W3C Group: HTML 5: A vocabulary and associated APIs for HTML and XHTML. Technical report, W3C Working Draft 26 February 2008 (2008)
14. Recordon, D., Reed, D.: OpenID 2.0: a platform for user-centric identity management. In: DIM '06: Proceedings of the second ACM workshop on Digital identity management, New York, NY, USA, ACM (2006) 11–16
15. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional (1995)