Adaptive Web Site (AWS) Framework
High-level Design Document
AWWG

December 4, 2003

Contents

1 Purpose 2

2 Background 2
  2.1 Adaptive Web Sites ................................................................. 2
  2.2 AWS Frameworks ....................................................................... 2
  2.3 The AWS Project ....................................................................... 2

3 AWS framework goals 3
  3.1 General Goals ........................................................................... 3
  3.2 Adaptation Goals ....................................................................... 3
  3.3 Framework Roles ....................................................................... 4

4 Framework Development 4
  4.1 Design Stages ........................................................................... 4
  4.2 Modular Decomposition ............................................................. 6
  4.3 Module Interdependence .............................................................. 8
  4.4 Operational Environment .......................................................... 10

A Framework Scenarios 14
  A.1 Builder .................................................................................. 14
  A.2 Author ................................................................................... 14
  A.3 Adapter .................................................................................. 15
  A.4 Presentation Generator ............................................................ 16
  A.5 User ....................................................................................... 16
    A.5.1 Content ........................................................................... 16
    A.5.2 Presentation ................................................................. 17
    A.5.3 Navigation ................................................................. 17
1 Purpose

This document captures the high-level requirements and design considerations of the AWS framework we intend to build. Within this, it describes the high and mid-level goals, requirements and environment of this engineering project. It illustrates these goals using scenarios which informally describe different perspectives of the system.

This document assumes some familiarity with adaptive website concepts, although it does describe our vision of an adaptive web site targeted by our framework. For background material, the reader is referred to “Toward An Adaptive Web: The State of the Art and Science” by [Kilfoil et al, 2003].

This document is organized as follows: section 2 presents a brief overview of the AWS project, including its most general goals; section 3 describes the specific goals of the framework and the roles which we have identified as being involved in the creation of an adaptive web site; section 4 describes the architecture and the detail of the AWS framework and demonstrates the process of developing an adaptive web site using the framework. Finally, appendix A describes portions of an adaptive web site from the perspectives of a number of roles or from within the system itself.

2 Background

2.1 Adaptive Web Sites

An adaptive web site (AWS) transforms a page request into a final page response by considering information about the page requested, about the user, about the way the site has been used, about the environment of the site and about the environment of the user. Transformations may include modifications to content (such as the inclusion or exclusion of images, paragraphs or links) or may include modifications to the presentation of the content (such as swapping different presentation format for equivalent content, transforming a content atom into another format, or removing a piece of content).

2.2 AWS Frameworks

An adaptive web site framework describes the interplay between the content of a web site, the usage of that content, the users who consume the information, and the packaging and navigation structure of that content. From this, we can see that it has several inputs coming from those interacting components. It should have a way to make useful information out of this data, so it might do usage mining, structure mining, content mining, and user profile acquiring as well as clustering to find groups within each or across multiple streams of data.

Usage can refer to several different things. It can refer to the low-level browsing of single pages (used in, for example, session reconstruction from page views); it can refer to transactions, which are collections of page views to accomplish a purpose; it may refer to finding sequences of concepts that a person browses (which is similar to transactions, but is even more abstract); it may refer to finding sets (or sequences) of product purchases; it may refer to finding time-related patterns of activity or time-independent patterns; it may refer to finding patterns within a single user’s activity, or to finding patterns common to large groups of people.

2.3 The AWS Project

The AWS project aims to design an adaptive web site framework, and design, code, test, implement and maintain a functional web site built around that framework for at least one content domain. This prototype is intended to demonstrate key features of an adaptive web site. The web site will have the appearance of real data (if not actually having real data). The framework will be flexible enough to cover more than one content domain. The web site will be closed-corpus, in the sense that content is all local to the site and can be fully known to the system (as opposed to a general filter or search engine which must grab snapshots of the universe of pages over which it functions); it should have a method for adding new pages to the system, and tools which can assist the people in different system roles, including system administrator, application designer, web developer, and content provider.
3 AWS framework goals

3.1 General Goals

There are several general goals which describe performance measures about both the framework and adaptive websites derived from the framework.

- **Versatility.** The framework should be applicable to multiple application domains.
- **Flexibility.** While the framework should define multiple adaptations, triggers for adaptation and transformations, it should be flexible enough that these can be removed or customized according to the application design.
- **Extendability.** Where feasible, the framework definitions should allow extension, so that new definitions can be developed from other fundamental definitions. For example, a new type of miner can be developed from an existing definition of a miner. It may even be possible to define new fundamental objects within the system, although this is likely infeasible.
- **Scalability.** Some consideration must be given to the amount of overhead a complex framework will add to the resources required to satisfy user requests, as well as the amount of additional work offline processes require.

3.2 Adaptation Goals

The primary job of an adaptive web is to adapt. This adaptation can occur in many ways and use a number of sources of information. The AWS framework should consider at least the following adaptation goals as a minimum:

- **Navigational adaptation.** This includes providing local and global guidance, link annotation, link ordering and collaborative recommendation.
- **Content adaptation.** This includes dynamic page generation, content selection (combining content building blocks not explicitly marked as part of the same page), content filtering (removing content blocks marked as part of the same page) and content ordering.
- **Presentation adaptation.** This includes swapping equivalent blocks which have different forms (such as a block of text for an image, or a download link for a video clip), modifying the colours or font of a page to match a user preference, and generating a different output format (such as generating WML for one user and HTML for another).
- **Personal adaptation.** The framework should allow for an individual user to have adaptation related to their particular needs.
- **Arbitrary groupings.** The concept of a group should be well-supported within the framework.
- **Group adaptation.** The framework should allow for adaptation to be performed based on group membership. This also allows for more anonymous user interaction.
- **Multiple output formats.** The framework should not impose an output format within the data where possible, but allow for multiple output formats (such as HTML, XML, XHTML or WML) to be used.
- **Allow for miners to play a role.** A miner is an automated process that looks for patterns within the data. Typical miners within the context of adaptive web sites include those looking for common groupings of users based on usage behaviour or demonstrated interests. By defining a basic miner, we can provide an API for new miners to be created and connected to the rest of the system easily.
3.3 Framework Roles

Our framework considers several roles:

- **Framework Designer.** This is essentially our role in this project. This role is responsible for laying out the most basic components of the system and the ways that these components can be used, modified, extended or created.

- **Application Designer.** This role builds the specific application from the framework. This includes establishing the ground rules and guidelines which the author will follow, deciding what information needs to be tracked and how it will be gathered, including the user profile and meta-data.

- **Site Designer.** Once a particular application has been decided upon, it is up to this role to define the particular instance of that application.

- **Graphic/Layout Designer.** This role is responsible for designing the look-and-feel of the website. In our framework, they design the templates which describe the layout which will be imposed upon manufactured pages, as well as providing additional translation information to turn those layouts into actual target language documents.

- **Content Author.** This role is responsible for inputting information into the website. This information includes both the actual text, images and other multimedia blocks to be displayed as well as the navigational structure between data blocks and the ways in which the data and navigation can be adapted. This role, unlike the above, is particularly ongoing after the website has been developed. (The Layout Designer role can also be considered as ongoing after initial development, but it will require much less frequent use.)

- **User.** The straightforward role representing a user visiting the web site. These users supply the usage information which helps to inform the adaptation process, and consume the final output of the system.

- **Auditor.** This additional role does not play an active part in the operation of the adaptive web system directly, but is responsible for monitoring the system state and assessing its performance. Theory and measurements need to be devised to support this role, as well as tools which allow these observations to be made.

For each of the non-user roles, tools (or tool generators) need to be developed to support them in meeting their responsibilities and accomplishing their duties.

4 Framework Development

4.1 Design Stages

The framework development follows a multiple stage process. It involves different roles for each stage or layer of development, as illustrated in Figure 1.

This layered model is inspired by standard object-oriented software design processes in which object definitions and object interaction structures are inherited from higher levels of abstraction down to lower levels of instantiation.

Each layer makes use and/or produces a package of information about the site. This package includes meta-information, such as the shape of objects within the system, their interaction, and rules governing usage, relationship and interaction of these objects, and data, such as the instantiation of particular objects. The responsibility for managing, restricting and extending the inherited package into the new package is taken on by the designer role of that layer.

The responsibility of taking that package and creating tools to enforce and create instances of objects specified within that package falls upon the tool builder roles. “Tools” here refers not only to GUI-like data-entry tools but also library level and communication-language constructions. These tools will be used both by the designer in the next layer to help in their specialization of the design and by the tool builder of the next layer to form the basis or building blocks to build the tools for the next layer.
Figure 1: Design stages and design roles relationships.

The design activities define the structure of information and the relationships and interactions between pieces of information. The tool building activities provide ways to create, update and use information placed into those structures.

As you progress downward through the layers, there is a gradual transition from emphasis on designing the structure of information to the design of the information using that structure. In particular, once you go below the Site Design layer to Page Layout and Content Creation you begin to focus on using the existing definitions and structures rather than defining new ones. The layers in more detail are:

- **Framework Design.** The top-most layer is the most abstract. At this stage, the general concepts of adaptive web site are established by the Framework Designer role. These concepts include fragment descriptions, rules, the kinds of rules which can be written, and basic relationships such as composition, link, or category.

- **Application Design.** At this stage, a particular application category has been decided upon, such as e-commerce, e-cyclopedia, e-news, etc. Application-specific concepts, rules and rule forms need to be added to the information package. For example, concepts such as “purchase”, “selling item”, or attributes such as “price” are defined for use an e-commerce system. Complex concepts such as “order”, “total price”, “cart” and “purchase history” would also be defined at this layer.

While represented here as a single layer, there could in fact be multiple layers inserted here, each representing a particular specialization of a type of website. For example, a top-most layer defining...
the fundamental concepts of an e-commerce domain might have an additional application design layer describing the specifics of an online book store, which might in turn have a specialized layer below it describing an online used book store. Each layer represents a specialized sub-domain within the application domain.

A single site may actually encompass several application sub-domains as well, each with their particular information and interaction requirements but having a common higher layer on which to communicate. This would allow, for example, a specialized science fiction and encyclopedia sections within the same book selling website. Each would have their special information frameworks and a user would be able to take advantage of specialized contexts for browsing and selecting each type of book, but the user would also be able to purchase both kinds of books in the same general context.

- **Site Design.** In this layer, considerations specific to the site being developed are considered. Specialized rules and rule interactions are defined here which take into account things like locale-specific information (such as taxes, shipping, or regulations), policy information (such as shipping prices, discounts, membership benefits) and tracking/grouping (such as usage tracking, browsing tracking, tracking by genre, tracking by time of purchase, etc.).

- **Page Layout.** At this stage, the physical page layout of the website is considered. Having the elements defined within the previous stages, the Layout Designer role places generic elements into a visible and informational structure. It is at this stage where specific adaptations can be defined, although the ways to describe the information upon which decisions can be made have already been specified, and the ways in which these conditions can be used on a page should have already been built into the tools built in parallel to the design layers.

- **Content Creation.** Although listed after Page Layout, this is actually more parallel with that stage than sequential, as indicated in the diagram. At this stage, content is added to the site by the Content Author role according to the fragment descriptions and Site Design model. Associations between individual fragments can be added here (categorical associations between types of items are already part of the model, added at site design time, and need not be here), as can conditional relationships and actions. Note that “content creation” need not mean that fragments to go in pages are always created by someone with this role; they could simply be modifying or adding rules to the this system. In this way, this role continues on into the next stage, and serves as the point of regular maintenance.

- **Live usage.** Finally, the system is used in real-time to answer requests provided by the User role. By combining the appropriate content and layout, a page is created to be returned to the user. (Obviously, much more happens at this stage, but from a design standpoint this is a sufficient description. For more detail on this stage, see Section 4.4 below.)

### 4.2 Modular Decomposition

A high-level functional decomposition can be seen in Figure 2.

In this diagram, there are four major modules:

1. **Application (Site) Description.** The application description contains all the information about the data that the system uses which is determined by the authors of the system. This is a collection of site building blocks (to encompass both web-related and domain-concept data), relationships and rules. All of the mentioned items are application- (even site-) specific.

   The building blocks include domain entities (abstract or concrete) as well as blocks used to realize those entities in page generation process. The relationships between the building blocks can be composition, association or linkage. Any type of relationship may have a few sub-types. For example, ‘association’ relationship may have ‘is-a’, ‘realized-by’, ‘requires-the-existence-of’ sub-types. ‘is-a’ relationship can be used to shape the concept hierarchies, and ‘realized-by’ can be used to map between domain concepts and web components like pictures and/or description text about a product item, for instance.
Rules, in general, specify the actions that take place during the system activity. They provide the system with instructions for page construction, updating profiles, running offline processes, and so forth.

As a clarifying example, consider an online retail toy store. The first page has a ‘seasonal’ section as a constant part. So the first page (as an entity) is ‘composed’ of a seasonal section plus other sections. The seasonal section can be ‘realized-by’ any page component which ‘is-a’ seasonal component (here the component can be, for example, a page frame). In the description of website, there is a rule that specifies on Christmas season, the seasonal component is a Christmas component, that in turn is ‘composed’ of toys that are related to Christmas. There is a concept hierarchy of toys in the system, and based on rules, the Christmas toys are going to be selected. Each toy is connected to its picture, descriptive text, price information and so on through ‘realized-by’ and other relationships. The process of realizing (synthesizing) a page starts with abstract domain/web concepts and ends when all participating components of a presentation are populated by concrete entities.

2. **Adaptation References.** This module contains all of the dynamic information, such as the information known about users, which can be used to make decisions.

The user profile, as an example, is one of the most important ‘adaptation reference’, because personalization is an important aspect of adaptive websites. Here, the ‘adaptation reference’ is any system process that is recognized as a provider of information used in condition clauses of system rules. For
instance, in an e-learning system, a rule may specify that for students with a particular level of credit history, the ‘course recommendation’ page should include a course from other educational groups. At run-time, the user profile module will be inquired about credit history of students that the course recommendation page is going to be generated for them.

During the system activity, other reference modules might be inquired as well, like environmental agents that are in charge of providing the system with environmental information (e.g. momentary available bandwidth) or a system profile module that provides information like the average server load during the last one minute.

3. **Adaptation Engine.** This module contains the functional components which transform an initial request into a final page. This module makes use of the Application Description to retrieve data (and information about data) and the Adaptive References for information about users, the system, etc. Besides adaptive references, System-Activity Trackers and Author CGIs might be used as well.

Creation of a web page starts with interpreting the received request. The Core Finder module is in charge of interpreting the user’s request and finding a ‘core’ component around which the actual page is going to be built. The core finder may inspect the properties of building blocks and/or refer to system rules to identify the proper core component. The Rule Resolver module would help here.

In the next step of page building, the skeleton of page is constructed, i.e. the (abstract) components that are going to be a part of the page are determined. The Page Constructor submits this ‘potential page’ to the Page Realization module, that populates the target page with web components and data. These web components are technology-independent; for example, at the Final-Form Generation step, a ‘paragraph initiator’ is going to be replaced by an actual HTML or SMIL tag; a ‘teddy bear icon’ may remain intact, be resized, or be replaced by a text phrase, based on the limitations of client’s browser.

4. **System Activity Tracking.** This module contains those functional components which update system state data based on ongoing activity.

An example of an activity tracker is a ‘group clustering module’. This module is responsible for creating and maintaining groups of users that can be clustered based on their ‘stereotypical’ interests or behavior. This module might later be inquired to classify a new user based on his/her recent activities, so the system becomes able to come up with some product recommendations based on the known interests of the corresponding group. Another example of an activity tracker module is user-state updater, through which system keeps track of the user’s current state like logged-in/browsing/logged-out.

4.3 **Module Interdependence**

The interdependence of the modules is shown in Figure 3. This view also gives some hints about the system implementation by grouping the modules into five subsystems:

1. **Site Description.** All modules within this subsystem are in fact data repositories that store different kinds of website data. While the system might use a single format (and maybe a single database) to store/retrieve the data of website description, each module is only in charge of a particular subsection of the site data, for both storage/retrieval operations and interaction with other modules.

2. **Web Component Libraries.** These are in fact parts of the application description. These libraries contain shared web components that can be used to build multiple web applications from the same domain.

3. **Synthesis Engine.** This incorporates all modules of Adaptation Engine, and can be more than that. This subsystem is the heart of the system, and it interacts with data modules as well as active modules. By gathering the required information from both site description modules and conceptual tasks, the synthesis engine chooses target page components, determines the layout of the page and then populates the page by fetching appropriate web building blocks and data.
4. **Conceptual Tasks.** These are pluggable active components of the system. Conceptual tasks include adaptation references, system activity trackers, and any third-party plugin that is compatible with the ‘Unified Interface’ for data interchange. Based on the instructions either built in synthesis engine modules or expressed by designers as ‘Rules’, the synthesis engine subsystem will inquire from the conceptual tasks about a certain condition or value, or issues a request to the modules of this subsystem.

5. **System Repository.** This is used to store the ‘system data’, which is data produced during the system activity and is not a part of the site’s permanent data. This includes user/groups profiles, current environmental information, system configuration, recent average server load, web server log, etc.

Table 1 shows a summary of how modules of different sub-systems use each other to carry out their functions.

Figure 3: High-level view of module interdependence in the AWS framework.
<table>
<thead>
<tr>
<th>Module</th>
<th>Uses</th>
<th>In order to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Finding</td>
<td>Domain/Web Building Fragments, Web Server</td>
<td>Receive the user’s request and find the core of the to-be-built page</td>
</tr>
<tr>
<td>Rule Resolution</td>
<td>Rules</td>
<td>Resolve which rules are applicable</td>
</tr>
<tr>
<td>Page Construction</td>
<td>Core Finding, Rule Resolution, Composition Relationships, Linkage Relationships, User Profile Manager, Environmental Agents, System Profile Manager, CGI Applications, Usage Groups Handler, User State Manager</td>
<td>Construct the skeleton of the page</td>
</tr>
<tr>
<td>Page Realization</td>
<td>Page Construction, Domain/Web Building Fragments, Domain Associations, CGI Applications, User State Manager</td>
<td>Populate the skeleton with actual web components</td>
</tr>
<tr>
<td>Final-Form Generation</td>
<td>Page Realization, Web Building Fragments, Web Composition Relationships, Web Linkage Relationships</td>
<td>Generate the browser-ready page considering the target web technology (HTML, SMIL, WML, ...)</td>
</tr>
<tr>
<td>User Profile Manager</td>
<td>Domain Building Fragments, Rules, System Repository, Environmental Agents</td>
<td>Instantiate the profile from user model and update the profile fields</td>
</tr>
<tr>
<td>Environmental Agents</td>
<td>System Repository, rules</td>
<td>Store the current environmental situations specified by rules</td>
</tr>
<tr>
<td>System Profile Manager</td>
<td>System Repository, Rules</td>
<td>Profile system information specified by rules</td>
</tr>
<tr>
<td>Usage Groups Handler</td>
<td>System Repository, Rules</td>
<td>Create the user groups based on specified rules</td>
</tr>
<tr>
<td>Association Miner</td>
<td>System Repository, Domain Associations</td>
<td>Generate new domain relationships</td>
</tr>
<tr>
<td>User State Manager</td>
<td>System Repository</td>
<td>Keep track of users’ states</td>
</tr>
<tr>
<td>CGI Applications</td>
<td>Compiler</td>
<td>Fulfill author’s site-customization needs</td>
</tr>
<tr>
<td>Domain/Web Building Fragments</td>
<td>Author’s User Interface, Compiler</td>
<td>Provide the building blocks of web site, including domain as well as web components</td>
</tr>
<tr>
<td>Composition/Linkage Relationships</td>
<td>Author's User Interface, Compiler, Association Miner</td>
<td>Provide information about how domain/web fragments are related together</td>
</tr>
<tr>
<td>Domain Association</td>
<td>Author’s User Interface, Compiler, Association Miner</td>
<td>Provide new data from usage mining in order to change the domain associations as well as navigational relationships</td>
</tr>
<tr>
<td>Rules</td>
<td>Author’s User Interface, Compiler</td>
<td>Define which actions in system must take place upon realization of certain conditions</td>
</tr>
<tr>
<td>Web Server</td>
<td>Final-Form Generation, System Repository</td>
<td>Deliver the synthesized page back to user’s browser and update the server log</td>
</tr>
</tbody>
</table>

4.4 Operational Environment

Once the system built with the framework is completed, it will have an operational environment as shown in Figure 4. It is important to understand this environment so that any implications about systems built with the framework can be included in the framework design.

Broadly, the system activities can be divided into online activities which are performed in response to a particular request, and offline activities, which maintain the data, the relationships and the rules. These
two sets of activities interact indirectly with each through a set of databases, which by themselves do not have any (high-level) activities.

Another way to divide these two groups of activity is by looking the responsibility for updating information and the timeframe of a response. Online activities are responsible (generally) for updating very little information; the request goes into the activity log, and the user’s profile is updated based on what patterns or groupings have been deduced. Also, the request must return to the user within a reasonably short period of time. On the other hand, offline activities are responsible for updating all parts of the system (except the history log, but possibly including the user profile), and operate in a considerably longer time frame. Indeed, since grouping mechanisms require multiple users and requests in order to begin discovering groups, they in fact cannot act on any less a time frame than multiple requests. This separation also allows the two kinds of activities to run in parallel without regards to the other, allowing for synchronization and locking to be handled by the database containers themselves.

The cycle of activity in the online activities is as follows:

1. A user makes a page request, using their browser.
2. The request triggers an update to the user activity log.

Figure 4: Activity diagram of operational environment of an adaptive website.
3. The request may be used by group matcher which seek to determine what groups of users or interests this user might belong to. In order to do this, it might make use of the user’s profile. If the user has no profile yet, one would be created for them. Once the group matcher has done it’s work, it should update the user profile with the list of groups and their respective strength of association that it had discovered.

4. The request might be used to look for patterns in the individual user’s activity, such as time-related patterns (“the user always looks at a certain page after 8am and before 9am”) or interest-related patterns (“the user is browsing a lot of science fiction-related pages”). This kind of miner needs to consider the user’s history (stored in their user profile) as well as the content and structure of the data. While shown as an online activity, this could also be done as an offline activity. However, if done offline the period between updates is likely to be too long to catch short-term patterns, and the user is likely to be left too conclude that the system is not responsive.

5. The request may be seen as the confirmation or declination of a previous response, and may be used to update some potential notion of the user (such as a conclusion reached by a group miner or usage miner). For example, spending a lot of time reading a response or following a link from a previous response page may be seen as positive indicators that the response was appropriate. On the other hand, taking a very small amount of time or following a link entirely unconnected to the previous response can be seen as indicators of disinterest.

6. Finally, the system combines information about the request, the user’s profile, knowledge about the content at the site, rules governing the output format, rules governing page linking, rules governing page access and other information. This combination must satisfy the informational needs of the user and the style requirements of the site to create a response (page) for the user to consume. At this point, the user evaluates the page and may provide direct feedback to the response to update their user profile.

In contrast, the offline activities can all occur in parallel:

- A usage miner might read the activity that has happened and attempt to find users with activity in common. This activity may be considered a simple sequence (or set) of page requests or may examine the content of the pages viewed to discern common interests on pages viewed by different users.

- Other miners might consider other ways in which groups of users might be formed. These miners may consider factors such as demographic information (age, sex, income, etc.), geographic information (same city, state or country), feature-related information (such as enabling or disabling the same sets of features), bandwidth, display, or a number of other things.

  It is important to include a place where any designer-specified process may query multiple sources of information (both within the system and outside it) and produce labelled groups of users. One such “miner” may be a designer themselves who, for example, arbitrarily specifies that users who behave in certain ways may be considered part of the same group, such as those who start from one particular page or who browse at the same time relative to their local time zone.

- Making use of the groups that have been discovered, the long-term structure miner acts in much the same way as a recommendation system, in that it tries to group together pages which are commonly seen requested together, or which logically belong together. By contrast with recommendation systems, however, the long-term structure miner seeks to permanently change the way the website is laid out. Often, this type of system would only make suggestions to an actual designer, who would consider the combination and possibly give a label to it (something which an automated system might have considerable difficulty doing).

- Similar to the usage miner, the content miner looks to create groups of related pages of information based on their content. In this way, pages which are never browsed in the same session but are related informationally can be recommended for linking.
As with most of these miners, there also exists the possibility of a designer making these groupings using meta-information that is not apparent from the page’s physical content. For example, a set of pages related to supporting an on-call system are related informationally, but there may not be a sufficient amount of information on each physical page for a system to discover the association.

- Leveraging the knowledge about the contents of the data and the structure of the pages provides a powerful way to mine for associations between pages.
- Finally, the author and designer play the most significant role in providing the actual data itself, and in producing a logical structure according to a design plan. Throughout the design of the framework it is important to consider not only how the resulting system can service the user or the goals set out for the framework, but also how such a system integrates with the site authors and designers, and how it extends and enhances their capabilities.

The online activities can be seen in more detail in Figure 5. In this diagram, data components have been grouped together into three types models: a model of how the user is perceived, a model of how the information is associated, and a model of how pages are constructed.

![Figure 5: Detailed examination of the online activities in an adaptive website.](image-url)
A Framework Scenarios

In order to understand the system, it is important to consider it from several different perspectives. In this section, the framework is considered from the following perspectives: from the site builder developing a particular instance of an application; from a content author adding information to the site; from within the system performing the adaptation; from the presentation generator taking an adapted page and modifying it for delivery to a user; from of a user using a website developed using the framework.

A.1 Builder

The Builder sees the system as a structured information database, and is responsible for taking a set of application requirements and using the tools and the framework to describe the structure, how the structure can interact and to build the particular application desired. For example, the application requirements might describe an e-commerce application in which the primary product consists of books and book-related items such as books on tape, writing materials, and magazines. The builder needs to design the data structures necessary to describe each of these items, so that an application author can later add new items for sale to the system.

These data structures in this case would include (for example): the definition of an author (name, country of origin, birth date, list of books written, primary genre, image(s), etc.); the definition of a book (author(s), date published, publishers, number of pages, language, cover image(s), subject heading(s)/genre(s), UPC code, sales information, etc.); the definition of sales information (price(s), discount(s), bundle(s), etc.); the definition of a genre (name(s), description, examples); and so on.

In additional to data structures describing the items for sale, the builder also needs to decide how a user might be described. In this case, the user information might include: name, shipping address, browsing habits, purchase history, credit information, or demographic information (age, income, race, birthplace). Some information might be derived from this information, such as liked/disliked genres (derived from browsing habits and/or purchase history, for example), like/disliked authors, and the number of books likely to purchase at once.

The builder may also decide what kinds of usage tracking will be available. In this case, both individual and group purchase tracking provides valuable information to form groups. Other groups may be formed by the arbitrary categories of books, and a similarity between two customers can be estimated based on the similarity between their purchase patterns (or the purchase groups to which they belong), their demographic information and the subjects of the books which they have purchased or browsed.

The builder may also customize the types of rules that can be created. More accurately, they may define the conditions under which rules might be fired and the actions which a triggered rule can perform. They may also place restrictions on when rules can be used, or what condition-action combinations are allowed.

Once these definitions have been completed, the builder will use the tools provided by the framework and these definitions to create customized tools to be used by the application authors and the page designers.

A.2 Author

Note: All over the scenario, the author could use developed tools for defining and editing software instead of editing raw XML.

1. Author uses a description language (probably XML and/or HTML) to describe and define the individual fragments to be available in the system. This includes defining the various attribute of those fragments and any resource that they use (image, sound, text, ...). The most important attribute of a fragment is its name which must be unique in the system and is used for reference later on.

For instance, the author needs to define a fragment for a specific type of “Jacket” with all the properties it has and their values.

Note: Although in this stage, author is supposed to define the fragments individually regardless of any possible relation with other fragments, it is evident that almost in every domain we have a hierarchical representation of the domain, which author can capture when defining the fragments. In this specific
scenario, author could have defined a fragment for “Clothing” which has different sub-fragments, and this is easily done using XML. This is in fact the “Composition” relation between fragments. Author may decide to add, remove or change these relations later in the authoring process.

**Note:** This stage is similar to the Concept Editing in some systems like AHA. It seems that in those systems, there is a more systematic definition of “Concepts” “Fragments” “Atoms” “Composition” ...

2. Author now needs to define three different relations defined in the framework specification. The “Composition” relations could be defined in the previous stage since they are relatively static relations, however, the author can change them or define rules for changing them. For instance, the fragment “Jacket” is always in the “Clothing” section, however, due to dynamic and adaptive nature of the system, it might become a sub-fragment for other sections too.

For defining the “Linking” relation, the author could use a different specification file to write the fragments linked to each other with the type of the linking (one-way, bidirectional, ...). The linking could be specified in fragments definition through attributes or apart from the fragments definition in a file:

“Jacket” links:bidirectional “Pants”

**Note:** This is the default linking relations, however, due to adaptiveness of the system, the linking relations at runtime may be different from what the author determines here. So, it is the author’s responsibility to specify the links that are strong enough.

“Association” relations seem to be a very dynamic one, and might just be specified through rules. If the author knows the associations at design time, so he/she can decide on a more concrete relation (e.g. linking) then.

3. Author defines user model attributes. These are the attributes that are important in this application. The system should be able to integrate the author-defined user model into the adaptation process.

In this example, in addition to general user model attributes like name, we may need “age” “section-of-interest” ...

4. When all the fragments are defined, their attributes set and their relations specified, the author may define the dynamics of the systems. The author can write the rules to do some actions on user model and domain model based on the current model of the system (user, domain, ...).

```java
if user.current_section != user.section_of_interest then user.current_section.Addlink(user.section_of_interest)
```

A.3 Adapter

A general adaptive website normally contains three main components: adaptation model, user model and domain model. Adaptation engine is the core of the system. It is responsible for processing user request according to her profile, and generating a response to user.

To explain the process of adaptation more clearly, we give an example. Suppose our adaptive web system is built for a car sales company. Once a user comes to the web site and wants to find a suitable car. Suppose this user selects the used car section. This request is handled by the system. The system will find out all the information of used cars in its database and only present part of it to the user according to user’s interests. For example, if the user’s age is between 20 to 30, the system will present sports cars on the top of the page. If the user’s age is above 30, the system will present Toyota family cars on the top of the page. Once their interests are determined, the final page will be generated and sent to the user.
A.4 Presentation Generator

The input of the Presentation Generator is a Potential Page, which consists of the content of the page and the page type. The supporting information includes User Context, User Preferences, and Restriction Rules. User Context contains user’s environment, such as bandwidth, platform. User Preferences include the user’s presentation preferences, such as fonts or colors. Restriction rules put limitations on the activities and data used to generate a response to the request that must be satisfied. The Page Generator combines the content and page template to construct the display page.

![Presentation Generator Diagram]

Figure 6: Presentation generator/designer scenario.

A.5 User

Since an Adaptive Web System is developed to cater to users’ interests, goals, and knowledge levels, then what should be provided and how to provide it based on users’ queries are the main concern of the system design. As we have known, Adaptive Web System could provide adaptations in content, presentation and navigation. Therefore, in user scenario, the final page will be described in three views: content, presentation, and navigation.

A.5.1 Content

- **Optional explanations.** The final web page could provide explanations of concepts to the users based on their background.

- **Optional detailed information.** The final web page could provide detailed information about some products base on users’ interests.

- **Page recommendation.** Aside from providing appropriate content, the final web page could recommend a list of sorted page links based on users’ interests and background.
• **Optional opportunistic hints.** The final web page could provide appropriate hints based on users’ interests and current circumstances.

• **Content annotation.** The final web page could use different colors or icons to identify which items are more important based on users’ background or interests.

• **Content ordering.** The same content of the final web page could be provided in different orders to identify the importance of the fragments.

• **Link hiding, disabling and removal.** The appearance or function of some links are changed or disabled inside the final web page based on users’ interests or background.

**A.5.2 Presentation**

• **Language.** The final page could provide appropriate the language to the users based on their nationality.

• **Different formats for different users.** The final web page could be provided in different formats with the same content to different people. For example, provide graphics for sighted people but a text file that can be easily converted using text-to-speech tools or an audio clip for the blind.

• **Different formats for different machines.** The final web could be provided in different formats, such as Adobe Acrobat Portable Document Format (PDF) file or a plain ASCII text file.

**A.5.3 Navigation**

• **Direct link.** A user might be redirected directly to a more appropriate page, skipping several steps in the hierarchy of the web site.