Web Application Interface Requirements

Another View

Bill Clare

September, 2009

.

Contents

1. Motivation

HTML is an immensely successful approach to defining a primarily declarative language, with incredible richness, for user interface to the Web. This use has considerable impact on our daily lives. HTML5, and its successors, are long term exercises in extending the basic capabilities of previous versions in a myriad of interesting ways.

This note explores some ideas for long term evolution of these capabilities, primarily from a requirements perspective, rather than design. Secondly, the emphasis remains on capabilities for declarative specifications, with a small assist from procedural code.

Building Up and Designing Down

HTML development appears to start primarily at requirements for a Web based display capability and then to map upwards to satisfy data handling requirements at the application level. Fundamentally HTML is largely based on its heritage as an SGML spinoff "document" specification for text manipulation and a somewhat awkward angle bracket syntax for attributes and elements. (Angle brackets are useful for their original intent of inserting display attributes into text streams, but its other extensions seem extremely cumbersome.) This approach can be systematically extended upward from a display oriented base to meet the requirements of Web applications but it appears, at least to this observer, this is somewhat of a forced fit.

A complementary approach would be to look at the requirements for significant application and enterprise development, and then to map down to display specification requirements. A significant part of this approach views a user application or set of applications as fundamentally interacting with and processing data, through the aid of display capabilities. It leads to fundamental issues of specifying data and data processing, its relation to display processing, and the dynamics and controls for interactions. Here the primary focus is on specification of data manipulation with a display rendering, rather than, a focus on text and display rendering of data to be manipulated.

This note suggests some rough approaches towards general requirements for these and other environments for interactive specifications. This can then lead to an examination of how, and how well, current approaches to HTML, CSS, RDF, XSL, JSON, SVG, ECMA and other scripts, and other technologies can meet or be extended to meet these requirements.

Specification Factoring

In searching for a new and common basis for interface specifications it becomes possible to factor out many common capabilities from higher level specifications.

A guiding principle here is to find commonality among concepts and to separate that commonality as a capability that can be generalized and adapted where needed.

Approach

The object here is a primarily declarative specification language, with minimal scripting required, that:

- Simplifies the Language
    - ♦ XML based with a greatly modified syntax for ease of use.

- ♦ Modular - with small flexible units of specification, that can be easily and systematically combined.

- Integrates Rendering and Manipulation
  - ♦ Integrates and simplifies HTML, CSS, etc. capabilities.
  - ♦ Integrates these uniformly with document (open office, goggle docs, etc.) capabilities.
  - ♦ Is systematically adaptable to devices, environments, preferences, user roles and access control.
- Integrates Data Operations
  - ♦ Supports generic specification for storage, transfer, presentation and interaction with generic data – items, aggregate, sequence, hierarchy, graph, network, etc.
  - ♦ Provides generic data operations to minimize dependence on data location, structure, format, representation, etc..
- Integrates Applications
  - ♦ Uniformly supports dynamic interactions with users, collaborators, data sources, services, applications and systems.
  - ♦ Provides consistent and comprehensive error handling.

Disclaimer

It should be noted, up front, that the author claims only a brief familiarity with the technologies involved here and that these notes are no more than a preliminary sketch. Also I suspect that much of what is here is probably available already, but perhaps in more restricted and cumbersome senses.

As I read further, especially among the W3C documents the complexity seems overwhelming. The nature of the problem here is to distinguish between that which is inherently complex and cannot be simplified, versus what has become complex because of its evolution and relation to other standards.

2. Requirements Drivers

Drivers here are:

- Separation of concerns for:
  - ♦ display content
  - ♦ display logical structure (hierarchy)
  - ♦ display physical structure (screen)
  - ♦ display rendering
  - ♦ application data structure
  - ♦ application data rendering
  - ♦ control.
- Model-View-Controller Paradigm

  This is somewhat motivated by Model-View-Controller concepts. Here the Model can be a separately specified Data Repository, the View is an HTML specified display, and the Controller is an event routing and processing specification.

  In particular:

- Display content, structure and rendering

  CSS provides a significant start for separation of the structuring and rendering of display specifications. However, CSS is a separate language.

- Data Model Specifications

  This introduces separation of concerns for considerations of application data format, structure, rendering, location, and processing functions – independently of how the data is mapped to displays.

- Application Data Rendering

  In particular, common data can be directly mapped to different display entities and entity types and in different ways. For instance depending on the possibly changing size of a display area, an entity might map to an expanded table, a summary table, a description, a title, or an icon, any of which the user could expand dynamically, either in place or in a different display area

- Control and Dynamics

  Here the separate concern is for interactions and control, through a common view of Events, Actions, Commands, Data and Control Routing and Functions, that can be specified for user interactions, application data, display entities and external services.

- Development Scale

  There is a tradeoff between small and large scale applications. For small scale applications, there is a simplicity that is achieved with simple tools and rich semantic entities. For larger scale applications, there is a simplicity with more general and flexible entities.

  There is a bias here for better support of larger scale applications. Needs for more complex applications include flexible data specification and interchange, modularity of components at all levels, multi-paged (or tabbed) applications, simple adaptation to a variety of system, user, application, role and display context environments, succinct syntax, common scoping rules, libraries of specifications, repositories of data values, etc.

- Integration

  Integration is used here in two senses:

  - Integration of multiple technology specifications

    This includes HTML, CSS, XML, RDF and various expression evaluation and scripting capabilities. It also includes other approaches, such as Openoffice.

    This integration can be based on defining the basic presentation elements (such as: glyphs and lines) and building up from them to provide specifications that map to current and new elements. This can eliminate redundancy and conflicts in overlapping specifications and lead to considerable simplification.

  - Integration of multiple capabilities, data resources and services

    There is a need to combine interfaces to multiple system resources, server applications, and databases in a single user interactive application, and to allow these entities to interact at server, data model and display levels.

    This type of interaction can range from informal “mash-ups” to sets of related applications that are integrated at the clients display station.

- Dynamic Controls

Users, Display Entities the underlying Data Entities, and external services need to interact with each other. In particular, this supports user applications based on integration of a variety of service applications(sometimes referred to as "mash-ups").

It should be possible to consistently define events, messages, data exchanges, actions and commands to support interactions among the user, display entities, a data repository, and the back end system and servers, and to build control functions to coordinate these interactions. In particular, a single event can trigger a series of processing steps for display, data and external services.

In particular, an extension of this can support collaboration among end users at the display, data and function levels. For instance, collaboration can cause data change events to be propagated to other users with their own different mappings to the display entities.

- Role based

  It should be easy to partition parts of the resulting application in the context of a user role. This could be a simple security check for the type of administrator exercising an application. Alternatively, it could distinguish among basic, tutorial, intermediate, advanced, and developer modes of execution.

- Textual Language

  The text document and angle bracket heritage leads to awkward, cryptic and verbose specifications.

  For instance compare:

  <tr> <td>1</td><td>2</td> <td>3</td><td>4</td> <td>5</td><td>6</td></tr>

  With

  Row (1,2,3,4,5,6)

- Semantic Content and Generality

  The HTML tradition provides a trade-off between easily used semantic content to be interpreted by the browser and explicit specifications provided by the author. The problem is that semantic entities combine simpler constructs that have evolved in ways that become increasingly complex in their interaction and lack of generality.

- End User Tailoring

  It would be useful for the user/developer to interact dynamically with the specification in a consistent way to tailor it to user, application, department and enterprise needs. This could lead to a full WYSIWYG capability.

- Developer skills

  While much of an advanced capability would be useful primarily to experts, it is expected that the more challenging aspects could be packaged in libraries of modules, that could then be used, tailored and adapted by more novice users.

  A simple example is the ease with which a user can tailor a Yahoo or Google sign-on page.

Note that these requirements are complementary in the sense that taken together they suggest a facility far greater than the sum of their parts.

3. Data Entities & Repository

Many end user application can be described and designed as a series of data interactions. The data can then be mapped to display entities in a variety of ways, including content, structure, format and rendering.. The application implementation should be able to first specify the data to be displayed, along with its logical structure and its processing requirements. Secondarily the specification can then concentrate on the display entities that render the data entities and allow interactions with them. With current technologies, this ordinarily requires writing a considerable amount of client oriented software. However much of this could be specified and implemented by extending HTML capabilities.

Data can be accessed through a Data Repository. The Repository can consist of entities that are:

- embedded in display entities,
- local to a display entity specification
- stored in caches and recovery stores
- stored in external services, applications, files or data bases
- shared among users for collaborative processing.

Thus the Data Repository is essentially a data interface that is independent of locations, formats and structures of the actual data. Only basic data access and operations need to be supported by the repository interface.

(XML already provides a full data specification capability and can be usefully exploited. However, just as HTML is a considerable simplification of XML, which is a simplification of SGML, the data model here could be specified with a considerable simplification of XML. I have tried to read *Modularization of XHTML™ 1.0 - Second Edition* – hopefully there should be an easier way..)

Data Entities

A simple data model would be based on entities for values and for specifications treated as values:

- Primitive Data Values
- Structured Data Entities

  Arbitrary Data structures can be built from primitives, aggregates, lists networks and their fundamental relationships.

  Basic structures include arrays, lists, tables, hierarchies and trees, networks and graphs.
- Specifications

  Specifications include those for display entities and their data content.

Name and Value Scopes

It is important to be able to scope data entities at appropriate levels. For instance, application specific entities may need to be local to a particular section or other HTML element. User attributes, particularly color, need to be at a general level. Thus data and specifications need to be defined and clearly separated at enterprise, system, application, module, environment, user, functional, display context scopes.

Library entities can depend on external parameters, particularly to adapt presentation attributes to a consistent display environment. Also library entities can test for environment parameters that are undefined, defaulted, override-able or non-override-able.

An example of external attribute specification for a library module might look like:

AttributeSet(Defaults(…), Overrides(...))

This essentially defines a set of attributes that are needed. If they are specified externally, the externally specification is used, except where there is a need to override or extend the attribute.

Scoping and override rules should be simple, consistent and transparent.

General Operations

Basic operations can be supplied by library functions for access, creation, replace, insert, deletion, and mapping of data content and display entities.

4. Data and Display Manipulation

HTML provides significant capabilities for mapping of data to display presentation. However the concept of mapping and transformation can be significantly generalized.

Basic Mappings

Mappings can involve any or all of changes to format, representation, data transformations, data validation and data locations. Mappings can be to and from HTML data sections, data caches, display entities and external sources and sinks.

Mappings can be based on display context and contained data values.

Semantic and Specific Display Entities

HTML, in general deals, with display entities that have semantic content for a viewer, such as a "h1","h2" headers. These elements bundle specifications for content, rendering and structuring in one entity. This provides a convenient shortcut at the expense the complexity of how such entities interact with each other. This is useful in some applications, for others a more general, flexible and manageable specification capability is important.

Semantic based specifications have considerable developer simplifications, since they specify intent only and much of the details can be left to the browser, or other defaults to determine a rendering. For other developers, greater generality and modularity is useful. Ideally, developers could systematically develop content, style, structure and other properties independently and then combine these in consistent manners to create their own "semantic" entities for elements, attributes and attribute lists.

Display Entity Operations

Emphasis here is on data manipulation as well as textual manipulation.

The user needs to identify display entities to interact with through select commands. Select commands can be for an item, a range of items in a list(i.e., use of the shift key in Window file lists), a group of items (i.e., use of the control key in Window file lists), or a set of items identified by some search criteria.

Operations on selected items include show, hide, modify copy and delete.

Operations on elements of lists and tables include select, find, modify, insert, and delete.

Operations on structures of structures can be applied recursively.

Modes, Roles and Security

The user interface needs to respond differently depending on such things as:

- a mode of operation (development, test, training, novice, expert, etc.)
- the users function, capabilities and privileges
- access constraints.

All of this can be supported with a set of global tags that can be applied to most entities. At a minimum, these include user, role and mode of operation.

5. Dynamics

A Web session is fundamentally an interactive activity among an end user, an display interface, the data to be viewed and its processing, and external services and applications. End users, displays, data, services and applications, all need to cooperate through a common framework of event, action, command, function, data and control routing, and controls.

Users, display hardware, display entities, data values, and external interfaces can all act in a cycle of activity. Typically both specifications and data are downloaded from external sources, either directly to a display entity or to a Data Repository. Data is formatted and rendered for user interactions. Such interactions can trigger processing of the display entities, of local data, and/or of external data. Ultimately processed data is typically sent to external sources, with responses provided to the end user.

All of this is part of a control cycle, which can be specified explicitly through a series of Events which can trigger Actions for status update, for query, and for processing either of display, or of repository or remote data entities. Commands can be text or function based. Users, displays, display entities, data objects, and external resources can all receive and send events, commands, data and actions, and can return status and responses.

In addition, Routers can be explicitly specified to receive commands and provide controls to mediate processing and error handling. Sophisticated Routers can act as Controllers for a series of interaction steps to insure synchronization of data, display and application entities, along with appropriate error handling and recovery. In particular, it is often necessary and challenging to synchronize processing across a set of loosely related external systems.

Elements

Dynamics support can be based on elements for:

- Events

  Events are reports from a sender to possibly unknown receivers that signal that a state change has occurred.

- Commands

  Commands are requests for a specific action to a possibly unknown receiver for a need to do something.

  How to accomplish the request is determined by the receiver. Processing occurs in the data scope of the receiver, with parameters from the sender.

- Actions

  Actions specify how processing is to be done.

Actions can be responses to Events or Commands, or steps in a Controller sequence. Actions can raise Events, issue Commands, transfer and transform data, execute scripts, run Controllers and query status and results. Actions can be processed in separate Threads.

- Data Routing

  It should be possible to directly transform, send and receive data to and from different entities.

- Status and Results

  Commands and Actions can report status and results. Actions can query status and results.

- Control Routing

  Routers respond to Events and Commands and can route Events and Commands to appropriate receivers, including lower level Routers

- Controllers

  Controllers augment Routers with a series of Steps, which are Actions. Steps can be specifications for the Controller or they can be executed with scripts.

All of the above can be attached to user interactions, display entities, data entities and external interface drivers. All of these can interact seamlessly.

Macros

A macros is a sequence of operations, possibly parameterized, that are systematically to a presentation or presentation element. Macros can be used to:

- Automate a series of user interactions.
- Perform a series of modifications on data elements.
- Issue any of the dynamic elements above.
- Run series of any of the above operations.

Error Handling

A primary function of a Controller is to coordinate and execute a series of steps. Typically these steps will have error conditions associated with them.

A generic controller might execute a series of Steps, retrieve errors at any level, request ordered backout of previous Steps, and provide an indication of the error to the user – all without specific knowledge of what the Steps actually do, or how they do it. Somewhat more ambitious is a Controller to drive a two-phase commit process.

6. Specification Language

Language here is used generically to encompass:

- Text specifications in files (i.e.!DOCTYPE html)
- Related non-HTML specifications such as RDF, XML, CSS, script languages etc.
- Specifications made in a browser editor; i.e., this uses HTML elements to allow specification of HTML elements.
- User interactions to examine and possibly modify attributes of Display Entities.
- Drag, drop and tailoring of modules onto a display.

Some unified view of what can be specified and the various ways in which it can be specified is essential. In particular, all of this needs to interact with the Data Repository, above, in a common way, for both accessing values, and for defining, modifying and using specifications.

Of course, much of today's HTML is written by tools, not humans, and does not pretend to be readable. Tool "languages" make developing content easier, however they limit the functionality available to the developer. So then the question arises; can HTML language extensions provide the power and convenience of such tools, and still make all the base HTML capabilities available – or even more available ?

Semantics

There seem to be several types of issues:

- Vocabulary

  Many words have historical meanings for a specialized environment, that doesn't adapt well to current software usage. For instance, "***action***" is a URL, "***object***" and "***type***" have meanings that conflict with other usages in software development. More fundamentality, the specification should be for a ***display***, not a ***document***.

- Generality

  Some capabilities seem to have evolved to meet a specific need and do not apply to more general cases. For example, "***select***" and "***selectness***" have a specialized meaning only for "***choice***" elements rather than allowing almost any display element to be selected for further operations (e.g. show/hide).

- Implied semantics

  There is considerable tension between including considerable specific semantic value in terms as opposed to more flexible capabilities. This allows the browser to do more of the work, but limits the developer in providing similar capability.

  For instance, for simple applications, it is useful to use a built-in element, such as ***small,*** and rely on browser defaults for presentation. For other applications, developers would want to specify their own categories for "size" and to use these with the same syntax.

  In particular, developers should be able to define their own constructs, with their own semantic names from language primitives for character string styles, for paragraph styles, for widgets, for divisions, for events, and for other display elements. These specifications can be given with flexibility, precision (i.e. they work the same in different browsers), and modularity.

  More generally, it seems useful for HTML to be directly extensible in the sense that new elements and attributes can be defined in HTML with appropriate content and rendering rules.

- Legacy Interactions

  As the capabilities have evolved the interactions of semantic driven specifications have become complex – and this can only get worse. For example, consider the subtle distinctions of attribute groups such as; core, common, global core, global common, with these then compounded by "extra" extensions. Similar issues arise in 'flow" content specifications.

  The issue here is not only complexity for the developer, but the lack of simplicity and generality for the capabilities.

- Modularity

It should be possible to build specifications systematically from smaller units to higher components. This needs to be true for both display and data content specifications.

A fix for these issues may be insurmountable. A clear, intuitive, simple and consistent set of terms, for all present HTML concepts, would be a significant challenge at best. An obvious mapping of this to current software terminology would be even more difficult.

However continued evolution of the current state of affairs can only become increasingly arcane, except to specialists. There is a precedent here however in that the current approach involves use of several languages and even has HTML and XHTML variations in the basic syntax. Would an additional language option only add to this confusion ?

On the other hand, it even took Stroustrup a whole book to explain C++.

Syntax

Angle bracket notation is useful in some places, but it is generally clumsy and verbose. In the small it is easy to read, in the large it is hard to navigate and understand.

Other alternatives are obviously possible. Here the XML concept of an info set, where the content is independent of the syntax is useful. In particular, the content can be specified and edited as data entities.

There needs to be standard techniques to avoid excessive levels of indentation. Use of a high level outline, with separate specifications for its elements is one technique. Use of Skeletons (below) is another.

For software development, there is a common mantra for "picture on a page", i.e. a complete development in a short space of a single aspect of the larger problem to be addressed. This seems sorely lacking from typically HTML documents.

Attributes and Entities

The evolution of XML syntax seems to have created a considerable confusion between the roles of attributes and of nested elements. It would seem that nested elements should represent nested data, and not the characteristics or processing of that data.

Specifications that might have been provided as attributes seem to be provided as nested elements, probably based on long standing XML restrictions that attributes can only specify single values and do not allow even simple structures, like aggregates and lists.

It would seem that consistent handling of attributes and elements could simplify specification of actual content, and could allow for consistent handling of attribute specification, access and overrides. Use of "refid" seems to be a hack around this, applicable in some cases.

From the W3Schools web site:

> "There are no rules about when to use attributes and when to use elements. Attributes are handy in HTML. In XML my advice is to avoid them. Use elements instead."
>
> "Some of the problems with using attributes are:
>
> - attributes cannot contain multiple values (elements can)
> - attributes cannot contain tree structures (elements can)

- attributes are not easily expandable (for future changes)

Attributes are difficult to read and maintain."

"What I'm trying to say here is that metadata (data about data) should be stored as attributes, and that data itself should be stored as elements."

These restrictions appear to be hereditary rather than fundamental, and therefore it would be unfortunate to continue them.

Modules

It is useful to be able to specify libraries of modules at the standards, enterprise, application and user levels. Typically, a module might be defined at the section level along with a declaration of its dependencies on its environment, both textual and system. Also modules can be parameterized to allow tailoring to their specific use.

Modules could be typically characterized as primarily base modules that are either data, display or control oriented, or else they are final modules that combines these separate specifications for data, how the data is rendered, and how it is processed. Thus a "Display" element might bind a series of data and control items to a display instance or type.

For instance:

- Sets of character styles with rows of values for font, size, color, bolding, etc., as well as possibly other elements that are either similar or to be contrasted.
- Sets of related text styles with rows of display character styles, along with layout, spacing and alignment characteristics for each portion of text.
- Sets of related framing styles with rows of values for positioning, shape, border, background, and grid styles.
- Sets of display widgets with specified characteristics of the above types.

  Elements can be built up systematically, for instance, from list to column to table presentations.
- Sets of elements with actions for specified events.
- Nested tables can be used to provide different display specifications for a common entity in different contexts.

Reflection, Transparency, Discovery and Specification

It should be possible to examine and possibly modify all entities in a presentation. The mechanism for this could be implemented, for instance, through special modes of operation, where right clicking on any entity reveals information about it.

Information that should be generally accessible includes:

- All attributes of the entity, flagged as explicit, inherited or default.
- All descriptive data in the entity's specification.
- The hierarchy in which the specification is imbedded.
- Application meta-data included with the underlying data entity.
- Source data for the entity.

- Related data entities, using both forward and backward Relationships.

Name Scopes

It is important to be able to scope data entities at appropriate levels and provide consistent rules for inheritance, override, extension and modification. Data and specification values need to be defined and clearly separated at enterprise, system, application, environment, user, functional, display context scopes.

Templates and Skeletons

Current HTML tends to produce highly elaborate and verbose specifications, with considerable repetition. There are probably few uses of HTML, other than "tool" generators, that do not use cut-and-paste at some level of scaling.

Repetition is not so much a problem for small applications. For larger applications and entire series of applications, lessons from software development teach that simplicity and maintainability can be increased by avoiding repetition, through various means of packaging common types, functionality and expressions.

A blunt approach to this would be templates that are simple skeletons for parameter substitution and abbreviations for name element substitution (as in URLs). Parameters can be used positionally or as name-value pairs, and by reference.

Skeletons are of two types:

- Character string replacement
- Syntactic element substitution.

Skeletons can applied at increasing levels of scope, such as attribute lists, sets of list and table elements, complete widgets, and entire modules. Skeletons can be nested, especially to exploit an inheritance capability. Skeleton parameters can take a variable length set of values, for instance, to specify a set of list or attribute items.

Skeleton specification can be enhanced with the ability to include simple counters, and some decision and iteration capability. Within the skeleton, values can be conditionally based on defaults, external context specification state, immediate actual context state (e.g. size), interaction state (e.g. selection, mouse-over) and parameters.

Skeletons and scripts are invoked with a common syntax. Skeletons can be overloaded on data types (similar to C++ functions)., so that a skeleton can be invoked with different parameter lists (and corresponding different defaults). Also different implementation instances of a skeleton can be selected based on the data types to be handled.

Statements that define a skeleton could be labeled (on the left), to avoid "***id=***" and "***refid=***". This can considerably ease reading of larger specifications. The statement labels can be used with parentheses to access its value (thus avoiding getElementById).

An approach to invoking a Skeleton, might be to use angle brackets for attributes and parentheses for values. For instance:

Rows<attributes>(1,2,3,4,5,6) (a,b,c,d,e,f) (U,V,W,X,Y,Z) ;

can create three rows with common attributes for the row and each item element.

Tabular Specifications

The number of properties for any display element can be quite large. Succinct, manageable and modular specifications can in many cases be based on tables of specifications that systematically build on each other. Grouping sets of parameters is useful to avoid repetition and to provide building blocks from lower to higher entities.

For instance:

- Sets of character styles with rows of values for font, size, color, bolding, etc., as well as possibly other elements that are either similar or to be contrasted.
- Sets of related text styles with rows of display character styles, along with layout, spacing and alignment characteristics for each portion of text.
- Sets of related framing styles with rows of values for positioning, shape, border, background, and grid styles.
- Sets of display widgets with specified characteristics of the above types.

  Elements can be built up systematically, for instance, from list to column to table presentations.
- Sets of elements with actions for specified events.
- Nested tables can be used to provide different display specifications for a common entity in different contexts.

More generally, any set of attributes or other entities can be packaged as parameters to a skeleton and then used to specify multiple expansions of the skeleton.

Statement Syntax

Use of data and display items, as well as macros, can be uniformly specified by common syntax for the use of name, value and type. In many instances, this can avoid the use of ***id*** and ***refid,*** with increases in clarity and visibility.

In particular, it should be possible for an interpreter to translate the following to and from XML.

A common syntax, for instance, for the basic data operations specified above, could be:

- Value specification

  Name Type Parm, . . .

  Or

  Type Name(Parm, . . .), . . .

  Where

  - ***Name*** is used to reference the specified item or its components.
  - ***Type*** is an element type or macro. Inheritance can be implicit, based on nested contest or explicit using:

    Type: Reference, . . .
  - ***Parm*** are used to specify components of the type or attributes of the type or component..

    e.g. ***Attribute***

    Component(Parm, . . .)

(Parm, . . .)
(Condition, Parm)
(Count, Parm)

Parms can be positional, as above, or keyword (***Parm* = . . .**)

WYSIWYG Display Construction

One of the principles here is that the elements of a presentation of a specification are editable. This is an extension of the current capabilities of HTML browsers to "View Source". Here, with the appropriate user role, both tabular and text views of source can be viewed and edited.

In that specifications for data entities, display entities and controls are all treated as data, there is a continuous range of possibilities for user interactions ranging from changing a visible display element attribute to modifying it, or even defining a new display element. Some of these capabilities can be inherent in the language, some can be provided by the developer. Typically, a single specification might provide various modes of operations such as developer, adapter, expert, advanced, basic and tutorial modes for different clients.

At the simplest extreme is the ability of a user to click on an entity, observe its attributes, and perhaps modify them. This could be for individual base elements or for applying a set of styles to a higher level context. At the other end is a toolbar whereby the user can drag and drop skeletons and styles onto the running display. Libraries of system and application modules and attribute sets can be parameterized, saved and then added to an interface document, with appropriate parameters. With the addition of a few controls, these elements can be made to interact with each other and with back-end systems.

Application Development

Application development consists of the following steps, each of which can produce specification parts in an integrated library. The emphasis here is on separation of concerns for development and maintenance and for integration of application suites for enterprise operations.

- Local Data
    - Data types
    - Local data values
- Remote Data
    - Connections
    - Local store and structures
    - Data operations
    - Mapping between remote and local data
- Control operations
    - Mapping of presentation events to data operations and other presentation elements
    - Mapping of external events to data operations and presentation elements.
- Presentation
    - Parameters
    - Attributes

- Parts
- Containers of attributes and parts
- Layout of containers

7. Direction

It would seem possible to have a complete and consistent language syntax, different from but compatible with, HTML, CSS, RDF, XML, XUL, XSLT, open doc, and other specifications (other than scripts, which follow different language rules). The language should be complete in that it consistently includes presentation specifications, a data engine and a control engine. Consistency with WYSIWIG and other editing and tool generation styles should also be possible. Such a language should be clear, simple, flexible, general, powerful, succinct, adaptable and have all the attributes of ease of use and ease of maintenance.