Component Extension (CX) API requirements Version 1.0

W3C Note 11 December 2001

This version:
http://www.w3.org/TR/2001/NOTE-CX-20011211
Latest version:
http://www.w3.org/TR/CX
Editors:
Angel Diaz, IBM
Jon Ferraiolo, Adobe
Stein Kulseth, Opera
Philippe Le Hégaret, W3C
Chris Lilley, W3C
Charles McCathieNevile, W3C
Tapas Roy, Openwave
Ray Whitmer, Netscape/AOL

Abstract

From the early days of the World Wide Web, Web Agents had been extended to support more types of contents. The recent developments of XML and the possibility to mix mupltiple XML Namespaces in the document reiterated the need to extend implementations and relaying on add-on softwares to accomplish tasks not supported by default in the implementation. In other words, we have several XML languages to represent different parts of Web pages (XHTML, SVG, MathML, XForms, etc.), we now need a well defined mechanism that allow different specialized tools to work together and handled these compound documents.

This W3C Note contains a non-exhaustive list of requirements to work on a Component Extension [p.14] API. The goal of this API [p.15] is to extend the ability of a Web application. Note that the Web application can be either on the server side or on a client side, and does not automatically implies interaction with a user or having a Web browser.

Status of this Document

This document is an early draft resulting from the HyperText Coordination Group face to face meeting to discuss standardization of Plug-in and Active Component Architecture for the Web. It is anticipated that this will be published as a W3C note as soon as it has reached an appropriate state of maturity.

This document is a Note made available by the W3C for discussion only. Publication of this Note by W3C indicates no endorsement by W3C or the W3C Team, or any W3C Members. There is no commitment by W3C to invest additional resources in topics addressed by this Note.

Comments on this document are invited and are to be sent to the public mailing list www-component-extension@w3.org. An archive is available at http://lists.w3.org/Archives/Public/www-component-extension/.

W3C Technical reports are published online at http://www.w3.org/TR.

Table of Contents

Appendices

---

1 Requirements

This document contains a description of the Component Extension [p.14] requirements established during the HyperText CG meeting in August 2001. It classifies the Application Programming Interface (API) [p.15] requirements in 3 categories:

[1]

The requirement must be addressed by the Component Extension API.

[2]

The requirement may be addressed by the Component Extension API.

[3]

The requirement is declared out of the scope of the Component Extension API. It might become in its scope in future versions.

The description of a Component Extension must be in terms of content that is handled, not in terms of a specific piece of software. For example, it is appropriate to request a Component Extension that handles SVG [SVG 1.0] [p.15] , including some set of functionalities (for example identified using the SMIL required functionalities mechanism for the switch element, or the SVG equivalent).

The list of requirements provided in this document is not exhaustive and only reflects the requirements from the HyperText CG meeting.

1.1 Formatting [1]

The Host implementation [p.14] must provide a mechanism for an embedded object to request a region or set of regions for displaying content. The formatting interface should be independent of platform specific constraints.

1.1.1 baseline and linehight [1]

The Host implementation and Component Extension need to coordinate the linehight and the baseline.

1.1.2 linebreak, size and rectangle negotiation, multiple rectangles [1]

The Host shall provide means to determine the maximum size of the rectangle into which the embedded object may render. Conceptually, the size of this page may be unconstrained in both dimensions given scrolling. In practice, the Host must somehow constrain this in at least one dimension.

For example, the Host may indicate that this rectangle is scrollable in the line-stacking direction. It must fix the other dimension. Typically, it would report the current width of its renderable area.

The embedded object must be permitted to negotiate rectangles and break into smaller areas across lines if need be.

1.1.3 DOM Views and Formatting [2]

The [DOM Level 3 Views and Formatting] [p.15] should be considered by the Component Extension API.

1.2 Rendering [1]

1.2.1 freeze/unfreeze [1]

A Component Extension [p.14] may wish to suspend rendering momentarily while for example a succession of changes are made to the structure, or the structure is made temporarily invalid, or for some other reason whereby the intermediate state of the document is no to be displayed. Examples of such methods:

SVG DOM: SVGSVGelement

```
unsigned long suspendRedraw(in unsigned long max_wait_milliseconds);
void          unsuspendRedraw(in unsigned long suspend_handle_id)
                raises(DOMException);
void          unsuspendRedrawAll();
void          forceRedraw();
void          pauseAnimations();
void          unpauseAnimations();
```

See also section 5.11 in [SVG10] [p.??] .

Openwave plugin API

PluginSuspend and PluginResume methods.

1.2.2 z-order/painting [1]

It should be possible for content which is rendered by Component Extensions [p.14] to take its correct place in the z-order of the entire document - it can be above other content such as backgrounds, which show through transparent areas, and other content later in the document or with higher z-order should be able to partially overlap content rendered by the Component Extension [p.14] .

1.2.2.1 clipping [1]

Clipping, which addresses removal of parts of display elements that lie outside of a given boundary, must be addrssed by the Component Extension.

1.2.3 sharing colormap [2]

Implementations which require a colormap will need to allow the Component Extension [p.14] to request the allocation of colors in the colormap and to find out what colors are already in the colormap, and to be notified if the color associated with a particular color index has changed.

1.2.4 sharing fonts [2]

The Component Extension [p.14] should have access to system fonts and their properties. The Host implementation provide means for querying metrics of the current font. Metrics of interest could be a subset of those provided by [OpenType] [p.15] . The Host implementation should provide methods for determining glyph bounding boxes, baselines, and nominal font height. It should also provide an indication whether a given character maps to a glyph in that font. The Component Extension [p.14] should be able to find out about downloadable fonts that other content on the same page has made available, and itself to make available fonts that it has downloaded. The font properties in use on the parent of the element whose content is being rendered by the Component Extension [p.14] should be made available by inheritance over the Component Extension [p.14] interface so that the same fonts can be used by the Component Extension [p.14] .

Accessibility information about preferred font sizes should be passed across the Component Extension [p.14] interface.

1.2.5 colors (accessibility issues) [1]

1.2.6 Device dependent drawing [1]

Component Extension [p.14] API should provide a platform dependent (e.g. XWindow Graphic Context, Microsoft Device Context) method for gathering device dependent information for formatting and drawing into each of the rectangles (e.g. for optimisation of drawing, such as window, device context, fonts). However this must not preclude the ability to use device independent methods, and the

specification should recommend that these are available as a fallback.

1.2.7 forcing redraws, invalidate rectangles, invalidate region [1]

In the Component Extension-does-compositing model with direct screen drawing, a Component Extension [p.14] might damage content rendered by something else. This is like the situation in windowing systems where a window is iconised and (portions of) other windows that are now uncovered have to be told to redraw.

The container-does-compositing model, with rendering being an offscreen RGBA pixmap, does not have this problem and need not tell its parents or other Component Extension [p.14] s to redraw. Moving or update of a Component Extension [p.14] merely requires re-composition of the current stack of offscreens without the need for a forced redraw, though this can be limited to a particular invalidated region.

Issue (rectangle-region-1):

These descriptions narrowly miss actually listing a requirement. Make it simpler.

See also [Netscape Plug-ins] [p.15] .

1.2.8 window-less plug-in [1]

Component Extension [p.14] which do not draw at all should be addressed by the Component Extension [p.14] API.

See also [Netscape Plug-ins] [p.15] .

1.3 Style [1]

1.3.1 DOM CSS [1]

This section describes the relation between the Component Extension [p.14] API and the Document Object Model (DOM) Level 2 CSS specification [DOM Level 2 Style Sheets and CSS] [p.15] .

1.3.1.1 specified values [3]

Unlike the DOM Level 2 CSS specification [DOM Level 2 Style Sheets and CSS] [p.15] , this note does not require access to the specified CSS values in the style sheets.

1.3.1.2 computed values [1]

If the Host implementation [p.14] provides a DOM tree to the Component Extension and a CSS Style engine, the Component Extension [p.14] API must have access to the computed CSS values of its nearest DOM Node ancestor (its parentNode) using the DOM Level 2 CSS [DOM Level 2 Style Sheets and CSS] [p.15] ViewCSS interface. Depending on the CSS Style engine, the Host Implementation may also provide a fully decorated DOM tree for the content addressed by the Component Extension [p.14] .

1.3.2 actual values [3]

Actual CSS values, as defined in the CSS specification [CSS Level 2] [p.15] , will not be addressed by the Component Extension [p.14] API.

1.3.3 Generic Styler [1]

The Host implementation [p.14] should contain a Style engine to handle general properties and to provide extension mechanism in order to support other properties. This Generic Styler engine has the responsibility of decorating the DOM tree with the computed values (see also **1.3.1.2 computed values** [p.6]).

Languages, such as CSS, have mechanism to extend the set of style properties used in the application. As an example, SVG 1.0 [SVG 1.0] [p.15] reused some of the properties listed in the CSS 2 specification [CSS Level 2] [p.15] and adds new ones. It is expected that the Generic Styler of the Host implementation [p.14] supports a specific set of properties and some general style properties such as color, background or fonts are required to be supported. [Definition: A **foreign style property** is a style property that is not recognized internally by the Generic Styler.] The Generic Styler does not know its default value, or if the value could be inherited or not. The Generic Styler is expected to support foreign style properties [p.7] .

The Component Extension [p.14] should declare information on style properties it uses that may be foreign to the Generic Styler such as parse, default values, inheritance, ... The Styler is responsible for maintaining the information and dealing with duplicate declarations between Component Extensions. A Component Extension remove style properties or values associated with the style properties. In other words, the set of style properties stored by the Generic Styler is the union of the set of style properties supported by the Host implementation [p.14] and its Component Extension [p.14] s. If a computed value is not recognized, the Generic Styler can:

- use a default value
- like for colors, always make the computed value have two parts
    1. a value in first set of values
    2. a possible extended value
- have extended value + callback

Issue (generic-style-engine-1):

What to do about style sheets when it is not known what Component Extension will be loaded?

1.3.4 Local Styler [2]

The Component Extension [p.14] can also embedded its own local Style engine. In this case, the Component Extension does not need to declare or get the computed values of the style properties in a DOM tree. This model has the advantage to not assume a DOM API between the Host implementation [p.14] and the Component Extension [p.14] , and only applies to the root node of Component Extension

[p.14] (`svg:svg`, `math:math`, ...). The local Style engine needs to provide:

- access to style sheets.
- access to containing DOM tree (optional).
- access to the `[parent]` information item in order to get the computed values of the style properties: `propname -> value`

  [1]

- access to style properties information: `propname -> inherit/initial value`.

  [1]

1.3.5 Mobile profiles [1]

Mobile profiles are always required to be supported by the Host implementation and its Component Extensions [p.14] .

1.4 Error handling [2]

An error handling mechanism should be provided by the Host implementation to the Component Extension [p.14] in a future revision of the Component Extension [p.14] API. This mechanism may be based on, or inspired from, the error handling mechanism provided by the DOM Level 3 Core specification [DOM Level 3 Core] [p.15] .

1.5 Events [1]

There must be a way to pass events from the Host implementation [p.14] to the Component Extension [p.14] and from the Component Extension [p.14] to the Host implementation. If the Host implementation [p.14] provides a DOM tree to the component extension and contains an event mechanism, the Component Extension API must support the DOM Level 2 Events specification. User Interface events, such as window events or system events (reload, URI notify events), may be addressed.

Namespace-bound events are events whose propagation is limited to elements namespace boundaries, i.e. it propagates until it reaches an element who does not use its ancestor’s namespace name. Namespace bound events are out of scope.

1.6 DOM Core tree [1]

The Component Extension [p.14] API must provide programmatic read-only access to HTML and XML content by conforming to the following modules of the W3C Document Object Model Level 2 Core Specification [DOM Level 2 Core] [p.15] and exporting the interfaces they define:

- the Core module for HTML
- the Core and XML modules for XML.

1.7 Starting point [1]

Features of the existing Netscape Plugin API [Netscape Plug-ins] [p.15] should considered as a starting point with implementation experience for features of the Component Extension [p.14] API, which may be modified, replaced, or eliminated as required to address problems and requirements.

1.8 Connecting with outside/scripting [1]

The Component Extension [p.14] will call a Host implementation [p.14] function to request a connection to one of its features.

This feature may be a generic functionality (eg. "copy/paste", whether this is selected by button, menu item, keyboard shortcut, drag-drop, or ...) or a specific Host implementation [p.14] chrome (eg. toolbar, button, status bar ...).

The request can be to output data through the feature (eg. status bar, title bar, alert, ...), to subscribe to events generated from a browser control, or to add a control to the chrome (eg. toolbar, menubar, menu ...)

The Host implementation [p.14] should return a value indicating whether it allows the connection or not.

The Component Extension [p.14] API should define how the features are selected, and specify a minimal list of connections that the browser *should* allow if it does have the requested feature, so that the Component Extension [p.14] can reasonably expect such a connection request to be granted.

Subscribed-to events can be passed to the Component Extension [p.14] using the normal event-passing mechanism (see **1.5 Events** [p.8]). If the connection needs to pass large amounts of data, the Component Extension [p.14] ’s stream APIs should be used.

- chrome *[1]*
  - adding new items/removing new items
  - "right-click drop down menu"/main menu bar
  - status line
- copy/paste
- find

- drag/drop
- focus
- alert *[1]*
- query user agents capabilities?
- cross plug-in streams
- properties *[2]*
  - properties dialog
  - system properties: hostname, IP, ...

1.9 Nesting/reetrance [1]

It must be possible for a Component Extension [p.14] to discover, where available, and integrate the formatting and rendering of externally supported content types inserted into the Component Extension [p.14] DOM Node’s subhierarchy that are not internally supported by the Component Extension [p.14] .

1.10 Abstraction level [1]

Much like the W3C DOM IDL descriptions and language bindings, the Component Extension API must be described independent of programming language, operating system and platform.

1.11 timeline [2]

SMIL 2.0 [SMIL 2.0] [p.15] is a specification for synchronized multimedia and SMIL Animation [SMIL Animation] [p.16] is a specification for how the timing and animation aspects of SMIL 2.0 can be integrated into other languages, such as SVG (see chapter 19 in [SVG10] [p.??]).

If the Host implementation [p.14] supports synchronized playing of media such as audio, video or animations, then the Host implementation must support the ability for time-based Component Extensions [p.14] to play media on a portion of a Host canvas, and the Host-played media and the Component Extension-played media must be synchronized. The API for achieving this synchronization must be rich enough to permit accurate implementation of the ’syncBehavior’ and ’syncTolerance’ attributes defined in the section 10.3.1 in [SMIL 2.0] [p.15] .

SMIL 2.0 and SMIL Animation allow content to be started and stopped via interactivity, such as User Interface events (e.g., mouse and keyboard). Host-supplied event propagation APIs must be such that time-based Component Extensions [p.14] can implement the interactivity capabilities defined in the section 10.3.1, "Basic time support", in [SMIL 2.0] [p.15] .

If the Host implementation [p.14] supports streaming media, then the Host implementation [p.14] must supply APIs that allow Component Extensions [p.14] to receive streaming content.

1.12 network/HTTP [1]

Networking Support will include an API to request the data from a URI reference.

The Component Extension [p.14] will call a Host implementation [p.14] function and pass the URI reference, any additional headers (for example, in the case of HTTP, maybe an additional accept type), the method (in case of HTTP, GET, PUT, POST or HEAD) and a pointer to a notify data. The Component Extension [p.14] may tell the Host implementation [p.14] to use the cache or to override it.

When the request is complete, the Host implementation [p.14] will call the Component Extension [p.14] 's notify function, and pass the data to the Component Extension [p.14] using the Component Extension [p.14] 's stream APIs.

Example:

```
kError HostURIRequest(kURIMethod method,
                      const char *uri,
                      const char *headers,
                      const char *entity,
                      unsigned long entitySize,
                      void* notifyData,
                      boolean useCache);
void   CXURINotify(const char *uri,
                   kStatus status,
                   void* notifyData);
```

Issue (networking-1):

In the Netscape model [Netscape Plug-ins] [p.15] , the data corresponding to a network request are sent to the Component Extension [p.14] by the Host implementation [p.14] by creating a stream: it is a callback mechanism.

Should we reused this mechanism or do we want to tie all asynchronous requests with one (probably DOM) event model ?

1.13 Editing [2]

Many Component Extension [p.14] s will involve editing functions, so the API must provide a method for access to editing functionalities - character input, drawing interfaces, etc. These must be available in a device independent manner. This requirement also means that the available rendering space may need to be dynamically re-sized (see also **1.1.2 linebreak, size and rectangle negotiation, multiple rectangles** [p.4]).

1.13.1 Editing mode [1]

The Component Extension API must provide an interface that allows the notification of the toggle between "edit" and "view" modes. This API must be present on both the Host implementation and the Component Extension.

1.14 associations/registrations/negociations [1]

In order for the Component Extension [p.14] to be able to extend the Host DOM implementation, there must be a way for the Component Extension [p.14] to register its own DOM implementation in the Host DOM implementation. In that case, the DOM implementation Node factories in Document must create DOM Nodes using the registered Component Extensions [p.14] Node for the corresponding types. Only the construction of the DOM nodes is affected by the registrations, not the building of the DOM tree. If the Component Extension [p.14] does not provide a DOM implementation, the Host implementation [p.14] must build the entire DOM tree itself, including for the content addressed by the Component Extension [p.14] .

If the Host implementation [p.14] provides support for XML 1.0 [XML] [p.16] documents, it must also implement the namespaces support defined in the Namespaces recommendation [XML Namespaces] [p.16] . Namespace conflicts resolution between Component Extension [p.14] s must be resolved by the Host implementation [p.14] . There must be a way for the user to choose between Component Extension [p.14] implementations in case of conflicts.

The set of functionalities provided by the Host implementation [p.14] must be accessible to the Component Extension. The same applies for the Component Extension [p.14] : the Host implementation [p.??] must have a way to query the set of functionalities provided by the Component Extension [p.14] .

There are severals ways to extend the Host implementation. Technologies such as XBL or IE behaviors must be considered.

The group might decide to come up with a modular architecture for an XML parser that will permit some of the XML content to be handled by other component. A general stream API should be addressed by the Component Extension to gain access to their data.

1.15 Accessibility [2]

The Component Extension [p.14] API must provide for accessibility requirements. Substantially, this reinforces the need for several of the requirements listed already, such as the ability to specify a content-type rather than a plugin, the provision and use of device-independent interaction interfaces.

The User Agent Accessibility Guidelines [UUAG] [p.??] require that access is provided for DOM interfaces, and "platform-specific interfaces" (for example MSAA, the Java Accessibility API, etc) where they exist.

1.16 Versioning [1]

The Component Extension [p.14] API should have an identifiable versioning mechanism. The version information must change each time the functionality is changed.

I am not sure it is necessary to discuss techniques here, as is done in the following paragraph.

One technique for achieving this is to provide the naming for functionalities, and use a mechanism like CC/PP for identifying the available functionalities. Although this seems a little heavy for this purpose, and seems to be better suited to using various unstandardized systems rather than for a standardized system.

1.17 storage/persistence [2]

The Host implementation [p.14] will allow the Component Extension [p.14] to save data in persistent storage or a file system. This ability will be governed by the security model associated with the Component Extension [p.14] .

For example, a web page has the ability to store cookies in the file system of the host. But a sandbox model exists there. Furthermore, the user may block web pages from setting cookies.

The storage functionality can be divided into:

1. Reading and Writing data specific to the Component Extension [p.14] . (’Private’ Storage) Example, saving high scores in a game. Storing cookies is an example in the context of a web page.
2. Reading and Writing data from the global data. (’Public’ Storage). Example, reading list of addresses (or a specific address) from an addressbook. *[3]*

For the first item, the Host implementation [p.14] may provide the Component Extension [p.14] APIs to get and set values.

For example, to set:

```
HostSetAttribute("newscore", itoa(newScore));
```

to read,

```
length = HostGetDataSize ("highscores");
if (length > 0)
{
    scoreString = memAlloc (length);
    HostGetAttribute ("highscores", scoreString);
}
```

If the security settings of the Component Extension [p.14] do not allow these operations, then these operations will fail. Or, the user maybe prompted for advice.

See also [WAP Persistent Storage] [p.16] .

1.18 Memory management [1]

Memory management would include APIs for

- Allocating a block of memory
- Resizing the block of memory
- Releasing the block of memory and
- Flushing memory from the host's space.

Issue (memory-1):

how much memory should a plugin be allowed ? Probable answer: System dependent. If the allocation function fails, an exception will be generated.

Issue (memory-2):

What about read/write? If the allocate APIs return a pointer, (as the Netscape APIs do), what will prevent a plugin to do something like

```
char* s = MemAlloc (100);
s[2000] = 'a';
```

On a system that doesn't have a memory management unit, this may freeze the system.

1.19 Security

Security issues must be considered for each of the functionalities of the Component Extension API.

A Glossary

Component Extension

> The term **Component Extension** (also well-known as **plug-ins** in Web browsers) refers to any software in charge of providing the client-side part of the Component Extension API. It is a program that runs as part of the Host implementation [p.14] and that is not part of content.

Host implementation

The term **Host implementation** refers to any software in charge of providing the server-side part of the Component Extension API. Softwares may include Web browsers, media players, Component Extensions [p.14] , and other programs (including assistive technologies) that help in retrieving and processing (this includes rendering) Web content.

Application Programming Interface (API)

An application programming interface (API) defines how communication may take place between applications. It is a set of functions or methods used to access some functionality.

B References

CSS Level 2
W3C (World Wide Web Consortium) Cascading Style Sheets, level 2 Specification, May 1998. Available at http://www.w3.org/TR/1998/REC-CSS2-19980512

DOM Level 2 Core
W3C (World Wide Web Consortium) Document Object Model Level 2 Core Specification, November 2000. Available at http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113

DOM Level 3 Core
W3C (World Wide Web Consortium) Document Object Model Level 3 Core Specification, September 2001. Available at http://www.w3.org/TR/DOM-Level-3-Core

DOM Level 2 Style Sheets and CSS
W3C (World Wide Web Consortium) Document Object Model Level 3 Views and Formatting Specification, November 2000. Available at http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113

DOM Level 3 Views and Formatting
W3C (World Wide Web Consortium) Document Object Model Level 3 Views and Formatting, November 2000. Available at http://www.w3.org/TR/2000/WD-DOM-Level-3-Views-20001115

Netscape Plug-ins
Netscape Plug-in Guide, 1998. Available at http://developer.netscape.com/docs/manuals/communicator/plugin/index.htm

OpenType
Microsoft OpenType Specification, April 2001. Available at http://www.microsoft.com/typography/otspec/default.htm

RFC2396
IETF (Internet Engineering Task Force) RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, eds. T. Berners-Lee, R. Fielding, L. Masinter. August 1998. Available at http://www.ietf.org/rfc/rfc2396.txt

SVG 1.0
W3C (World Wide Web Consortium) Scalable Vector Graphics (SVG) 1.0 Specification, September 2001. Available at http://www.w3.org/TR/SVG

SMIL 2.0
W3C (World Wide Web Consortium) Synchronized Multimedia Integration Language (SMIL 2.0), August 2001. Available at http://www.w3.org/TR/smil20

SMIL Animation
W3C (World Wide Web Consortium) SMIL Animation, September 2001. Available at http://www.w3.org/TR/smil-animation
User Agent Accessibility Guidelines 1.0
W3C (World Wide Web Consortium) User Agent Accessibility Guidelines 1.0, September 2001. Available at http://www.w3.org/TR/UAAG10
WAP Persistent Storage
WAP Forum (Wireless Application Forum) WAP Persistent Storage Interface , May 2001. Available at http://www1.wapforum.org/tech/documents/WAP-227-PSTOR-20010530-a.pdf
XML
W3C (World Wide Web Consortium) Extensible Markup Language (XML) 1.0, October 2000. Available at http://www.w3.org/TR/2000/REC-xml-20001006
XML Information set
W3C (World Wide Web Consortium) XML Information Set, August 2001. Available at http://www.w3.org/TR/2001/PR-xml-infoset-20010810
XML Namespaces
W3C (World Wide Web Consortium) Namespaces in XML, January 1999. Available at http://www.w3.org/TR/1999/REC-xml-names-19990114

C Contributors

The people who contributed to this document are the members of the HyperText Coordination Group and the participants of the Oslo face-to-face meeting:

Jonny Axelsson (Opera), Bert Bos (W3C), Angel Diaz (IBM), Jon Ferraiolo (Adobe), Max Froumentin (W3C), Rick Graham (Bitflash), Stein Kulseth (Opera), Dean Jackson (W3C), Philippe Le Hégaret (W3C), Håkon Lie (Opera), Rune Lillesveen (Opera), Chris Lilley, (W3C), Charles McCathieNevile (W3C), Steven Pemberton (CWI/W3C), Vincent Quint (W3C, HyperText CG Chair), Tapas Roy (Openwave), Peter Stark (Ericksson), Ray Whitmer (Netscape/AOL), Steve Zilles (Adobe)