ANSI/CMS V1.6-2011
(revision of ANSI/HL7 CMS V1.5-2004 (R2009))
February 7, 2011

HL7 Context Management "CCOW" Standard: Component Technology Mapping: Web/HTTP, Version 1.6, February 2011

Editor: Robert Seliger
Sentillion

IMPORTANT NOTES:

Table of Contents

Figures

Changes from Version 1.4:

- Added support for PKI-based secure binding.
- Added exception names in table 6 for ImproperSignatureFormat and ContextNotActive. These exceptions had been mistakenly omitted.
- Removed interface specification for MappingAgent, as this interface had previously been deprecated.

Changes from Version 1.5:

- None.

1 Introduction

This document specifies the details needed to develop web implementations of applications and components that conform to the HL7 Context Management Specification, Technology- and Subject-Independent Component Architecture, CM-1.6, which shall hereafter be referred to as the CMA. In these systems, context management is primarily (but not necessarily exclusively) between web applications. Using this specification, the resulting applications and service components will be able to communicate with each other per the CMA even if they were independently developed.

The scope of this document is limited to the details pertaining to implementing CMA-conformant applications and components using the common web technologies such as those defined by the Internet Engineering Task Force (IETF), World Wide Web Consortium (W3C) and the European Computer Manufacturers Association (ECMA), as these technologies are pervasive and standard. These technologies include, but are not limited to: HTTP for message transport; Universal Resource Locators (URL) for representing logical addresses of entities located on the web; XML and HTML as necessary for data representation; Java, Java Applets, JavaScript, or other scripting languages for program logic. Other web technologies not explicitly described in this document may also work with the specification defined in this document.

While there is no precise definition for “web application,” the “lowest common denominator” for such applications is assumed to be a 2-tier system in which the user-interface is presented by a web browser and where data is served by a remote web server. Application logic may be physically distributed among the tiers, including the browser.

However, perhaps the most salient hallmark of this model of a web application is that the only software that is assumed to reside on the user’s access device prior to use of a web application is the browser. All elements of the application, including code as well as data, are dynamically loaded into the browser at the time of access.

There are a number of ways to create web applications that are more sophisticated than the least common denominator web application described above. These so-called thin client applications do not execute within a browser, but nevertheless use web technologies such as HTTP to interact with servers. While not the design center for this specification, sophisticated web applications will nevertheless be able to take full advantage of the specification.

1.1 DEFINITION OF WEB APPLICATION

A web application is defined as a set of one or more web pages, whose content and behavior are logically related, and one or more web servers that work in concert to serve these pages to users whose access is mediated by a web browser. The web pages may be static in nature, or may be active in appearance and/or perform computations. Pages that are active in this manner are generally programmed using a scripting language such as ECMA Script and/or programming language such as Java.

1.2 COMPATABILITY

This specification is compatible with at least the following Java-capable web browsers:

- Microsoft Internet Explorer 4.0 SP 1, or later.
- Netscape Navigator Version 4.0, or later.

The specification is likely to be compatible with other implementations of Java-capable web browsers.

The specification also requires the capability to open local HTTP sockets via trusted Java applets for the purpose of sending and receiving HTTP messages between the applets resident in the same host, and which

may reside in the same or different browser instances on the host. It is assumed that any platform that hosts web-based CMA-compliant applications also respects the Internet Port Number Assignment Authority's designation of well-known port numbers.

1.3 DEFINITION OF CLINICAL DESKTOP

A context-enabled web clinical desktop results when a client computing device is used by a particular user to access CMA-compliant web applications that share clinical context. These applications are accessed via one or more web browser instances. The type of each web browser instance may be different (e.g., Netscape Navigator, Microsoft Internet Explorer, etc.).

The means for supporting multiple concurrent clinical desktops on the same client computing device is not specified. The means for supporting a shared clinical desktop across multiple client computing devices is not specified.

1.4 WEB TECHNOLOGY CHALLENGES

Web technologies present unique challenges to implementing the CMA. These challenges include:

- **The difficulties of maintaining state between a web server and each of its clients.** The most wide-spread web communications protocol, Hyper Text Transfer Protocol (HTTP), does not provide an implicit means for maintaining state between messages. (State is maintained for a single message transmission, so that it is possible to associate a reply with the request that elicited the reply.) The maintenance of state between message transmissions requires special design considerations and often the adoption of application-specific state management conventions. This situation presents challenges to implementing the stateful relationship between a context manager and its context participants.
- **The overhead of sending a message using HTTP.** HTTP, which is layered on top of TCP/IP, is designed such that a TCP/IP connection may be established and then torn-down for each message transmission. This situation requires increased sensitivity to the number of messages required to perform a context change transaction.
- **The absence of a standardized means for communicating unsolicited data or state changes from a web server to its clients.** The current state of the art for so-called web-casting is actually based on polling schemes, in which a client periodically polls one or more web servers to determine if there has been a change that is of interest to the client. These schemes lead to a tension between computing and network resource utilization and responsiveness to context changes on the part of the client. This situation complicates the process by which the context manager asynchronously notifies its participants that the context has changed.
- **The strictness of accepted web security mechanisms.** Web security mechanisms, particularly those defined for browsers, greatly limits what the browser-resident portion of an application can do. For example, the default behavior for a Java applet operating in the sandbox security model is that it can only communicate with the web server from whence it came. This situation requires sensitivity to the programming idioms required or implied for context participants.
- **The challenges of determining the identity of the host for the client portion of a web application.** Due to concerns about privacy and security, explicit measures have been taken to make it extremely difficult for the client portion of a web application (i.e., running within a browser) to determine the identity of its host. The situation makes it hard for two CMA-based web applications to determine that they are co-resident on the same "desktop" and should therefore participate in the same context system.

Many of these limitations can be overcome through the use of a distributed object infrastructure as the means for communication between the browser-resident portion of an application and its web server. These infrastructures include: Microsoft's DCOM/ActiveX, Object Management Group's CORBA, or Java's Remote Method Invocation (RMI) capability.

Unfortunately none of these technologies are dominant or pervasive enough to predicate a healthcare standard upon. Instead, a different tact is taken, in which an application architecture and means for communication among an application's constituent components is not assumed. This enables application developers to choose the web technologies and architectures that meet their needs. The only things that are standardized for a web-based CMA are the interfaces and communication technology that applications must use for interaction amongst and between CMA-compliant applications and components. The details follow.

2 Technology Mapping

The CMA is technology-neutral. This means that while an underlying component system is assumed, a specific system is not identified within the architecture. It is the purpose of this document, and its companions for other component technologies, to map the CMA to a specific target technology. For the web, the technology-specific details specified in this document include (but are not limited to):

- HTTP-based messaging
- multiple interfaces
- security
- error handling
- character set
- implementable interface definitions

It is beyond the scope of this document to provide all of the details that are needed in order to fully implement conformant CMA applications and components. The necessary additional details are covered in a series of companion specification documents, starting most notably with the Health Level Seven Context Management Specification, Technology- And Subject- Independent Component Architecture, Version CM-1.6.

As illustrated in Figure 1, these documents are organized to facilitate the process of defining additional link subjects and to accelerate the process of realizing the CMA using any one of a variety of technologies.

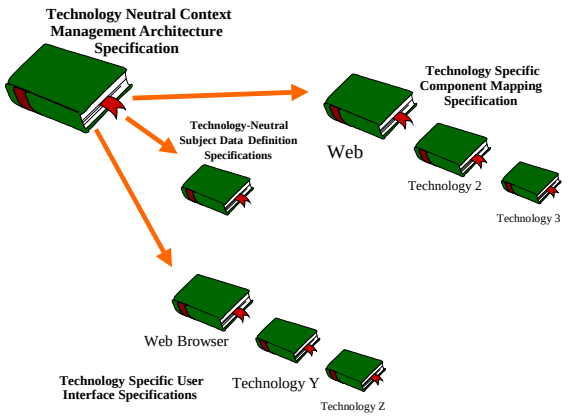

Figure 1: Organization of HL7 Context Management Specification Documents

The context management subjects and technologies that are of interest are determined by the HL7 constituency:

- There is a single HL7 context management data definition specification document for all of the standard link subjects. This document defines the data elements that comprise each link subject. Concurrent with the publication of this document, the following document has been developed:

  Health Level-Seven Standard Context Management Specification,
  Subject Data Definitions, Version CM-1.6

- There is an HL7 context management user interface specification document for each of the user interface technologies with which CMA-enabled applications can be implemented. Each document reflects the user interface requirements established in this document in terms of a technology-

specific look-and-feel. Concurrent with the publication of this document, the following document has been developed:

Health Level-Seven Standard Context Management Specification,
User Interface: Microsoft Windows and Web Browsers, Version CM-1.6

Finally, there is an HL7 context management component technology mapping specification document for each of the component technologies. Each document provides the technology-specific details needed to implement CMA-compliant applications and the associated CMA components, as specified in this document. This document serves the role of specifying the details for a CMA implementation using web technology.

2.1 TECHNOLOGY MAPPING OVERVIEW

In the web technology mapping, per the CMA, each context system is comprised of a context manager, context participant applications (possibly with an authentication repository), and one or more context agents. The roles and responsibilities of these components are unchanged from the CMA, and they each implement the interfaces defined in the CMA.

The CMA does not specify the physical location of these components. In the web technology mapping, which inherently supports distributed computing, the context manager may be co-located or physically distributed relative to the context participants it serves. Similarly, a context agent may be co-located or physically distributed relative to the context manager it serves. In either case, communication amongst and between context participants, the context manager, context agents, and other CMA components is via HTTP, as described next.

2.2 COMPONENT MODEL MAPPING

Each component defined in the CMA specification is implemented as a web-capable program. Component references are represented as absolute dereferencable URLs[1]. A URL contains all of the information necessary to represent the address of a web entity and can be resolved to the network location of the entity it references. The format and content of a URL depends upon the implementation of the component to which the reference refers, and may vary across component implementations.

Each interface defined in the CMA specification is implemented as a set of related HTTP messages, one for each method, that a component may receive. The URL that references a particular web component shall be the only URL necessary for accessing all of the interfaces implemented by the component.

Denotation of the specific interface to which a message is directed shall be encoded in the message. Each message shall also contain a representation of the same parameters and exceptions as the corresponding CMA-defined method. The complete details of how interface methods are represented as HTTP messages is specified in Chapter 7, Representing CMA Methods as HTTP Messages.

In addition to the CMA-defined interfaces, the web technology-specific interface InterfaceInformation, which enables interface interrogation, is also defined (See Section 10.1.1, InterfaceInformation). All web-based CMA applications and components shall implement this technology-specific interface. A client may use this interface to ask a CMA-compliant component whether or not it implements a particular CMA interface.

The mapping of the CMA concept of interface reference management is achieved through a combination of techniques. State-full component explicitly manage references to each other by following the policies defined in the CMA. For example, applications explicitly join and leave the context via the context

[1] In order to keep the web mapping of the CMA as simple as possible, URIs, which are Universal Resource Identifiers, and URNs, which are Universal Resource Names, shall not be used to represent component references.

manager. Stateless components implicitly release references at the end of each HTTP message, due to the stateless nature of HTTP.

Finally, a web-based Context Management Registry serves as the web realization of the CMA-specified Interface Reference Registry. The Context Management Registry provides a well-known service for obtaining interface references to web-based CMA component instances. For example, the registry provides the means for locating the context manager instance that is responsible for managing the context for a particular clinical desktop.

The web context management components and the interfaces that they support are illustrated in Figure 2. Web Interfaces in a Common Context System.

The means by which the URLs for these interfaces are obtained is summarized in Table 1: How URLs Are Obtained.

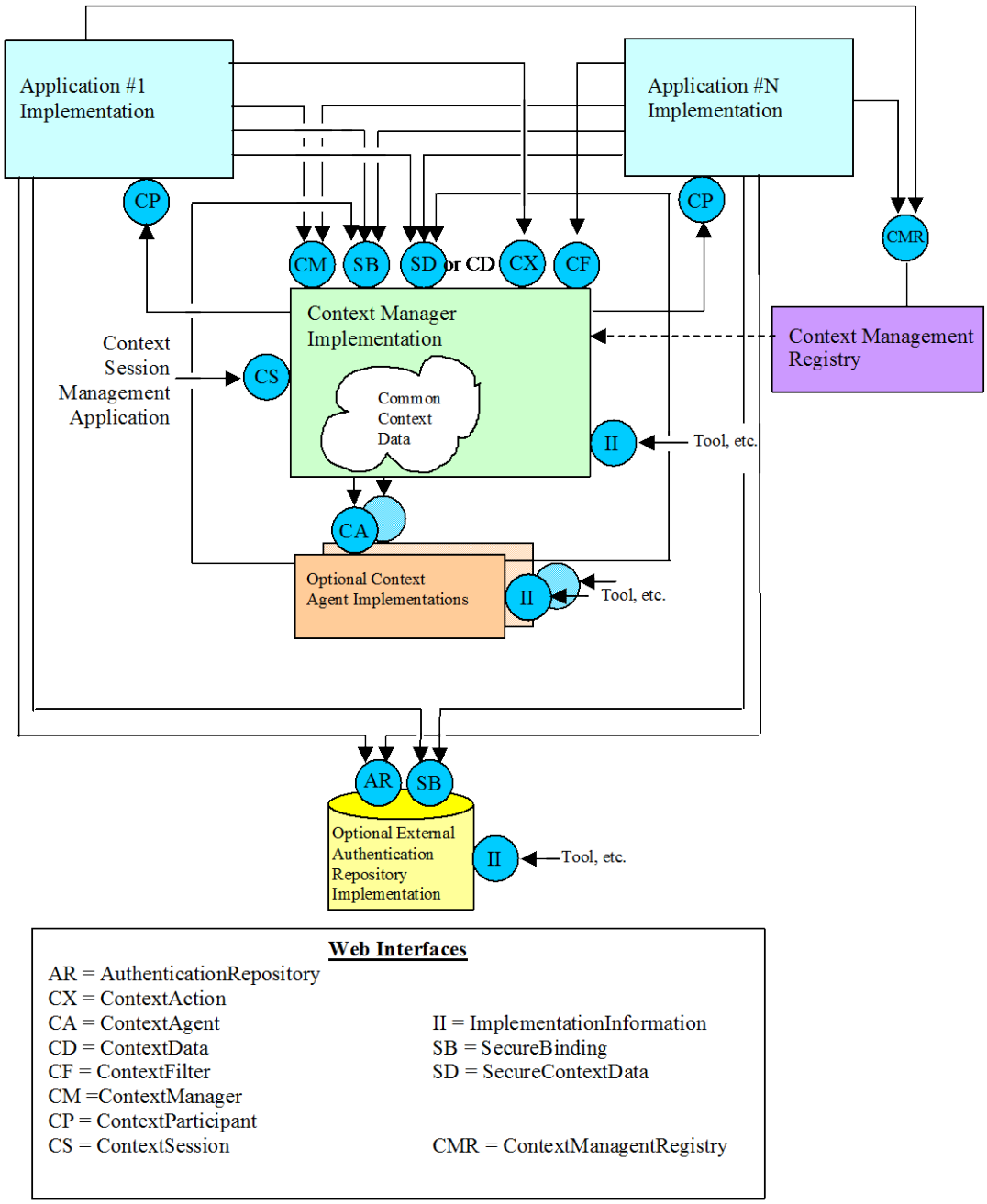

Figure 2: Web Interfaces in a Common Context System

Table 1: How URLs Are Obtained

	Client's means for obtaining server's URL …	
Server	**Client**	**Means for obtaining reference**
Context	Context	A context participant uses the URL http://localhost:2116/ to

	Client's means for obtaining server's URL …	
Server	**Client**	**Means for obtaining reference**
Management Registry	Participant	communicate with the context management registry, an instance of which is present on any desktop capable of hosting CMA-compliant application clients.. The well known port 2116 was assigned by the Internet Port Number Assignment Authority to CCOW.
Context Manager	Context Participant	A context participant obtains the URL for a context manager from the context management registry that is resident on the desktop from which the participant application was launched.
Context Manager	Context Agent	The context manager provides its URL the context agent when the context manager calls MappingAgent::ContextChangesPending or ContextAgent::ContextChangesPending.
Context Agent	Context Manager	A context manager is configured with the URLs for the site-specific context agents that it is to use.
Context Participant	Context Manager	A context participant provides its URL to the context manager when the context participant calls ContextManager::JoinCommonContext.
Authentication Repository	Context Participant	A context participant is configured with the URLs for the site-specific authentication repository that it is to use.

2.3 CONTEXT MANAGER

A context manager for web applications may reside on the clinical desktop within the browser (e.g., as an applet), on the clinical desktop but outside of the browser, or on a server. In all of these cases, conceptually there is only one context manager instance for each clinical desktop.

If the context manager is hosted on a server, then it is a context manager implementation decision as to how the abstraction of one context manager per desktop is achieved.

If the context manager is hosted within a web browser, then the browser must be capable of sending context management-related HTTP messages to external sources. The browser must also be capable of receiving HTTP messages from these sources and dispatching the messages to the context manager.

2.4 CONTEXT PARTICIPANT

In the CMA, the context participant capability of a web application may be implemented within the application's web client or web server.

2.4.1 Implementation Considerations

If the context participant capability is implemented in the application's client, then special attention to security may be necessary. For example, with Java 1.1, a context participant implemented as an applet must be a signed applet. This is because the applet needs to have access to desktop resources, such as sockets. In Java 1.1 this is not possible when the sandbox security model is applied, but can be achieved when the signed applet model is used.

If the application's context participant capability is implemented in the application's web server, then the server presumably is capable of serving multiple concurrent web clients. If this is the case, then it is an

application implementation decision as to how the correspondence between the application's client portion and its server-based context participant portion is maintained.

2.4.2 ContextParticipant Interface

A context participant is a client of the context manager's interfaces. In addition, a context participant must implement the ContextParticipant interface. See Chapter 4, Context Participant Implementation Responsibilities, for a description of this interface.

A context participant implements this interface as a set of HTTP messages that it receives, each of which corresponds to a method defined for the ContextParticipant interface. The approach for mapping CMA methods to HTTP messages is the same as defined for the context manager. (See Chapter 7, Representing CMA Methods as HTTP Messages.)

It is through the ContextParticipant interface that the context manager communicates with a participant to conduct context change surveys and to notify a participant of the result of each survey. A participant that implements this interface must be capable of receiving unsolicited HTTP messages.

A context participant provides a URL to its ContextParticipant interface to the context manager when it joins the common context, is represented as a URL. The content and format of this URL depends upon the implementation of the context participant. The URL may contain context session-specific and/or context participant instance-specific information. The URL may be for an HTTP or HTTPS connection.

It should be noted that if the URL includes the name of the host upon which the context participant is executing, then the host shall be accessible by the Domain Name Server (DNS). To avoid the dependence upon the DNS, the URL should incorporate the host's IP address instead of its name.

2.5 WEB SYSTEM COMPONENT DISTRIBUTION OPTIONS

There are six variations of component distribution options for web based context management solutions. Two of these variations – a server-centric approach and a client-centric approach – are illustrated in Figure 3.

All six variations are summarized in Table 2: Web Component Distribution Options, which assumes two CMA-compliant web applications, each with a client portion and a server portion and the context manager that they share. The table indicates the location of each applications context participant implementation, and the location of the context manager.

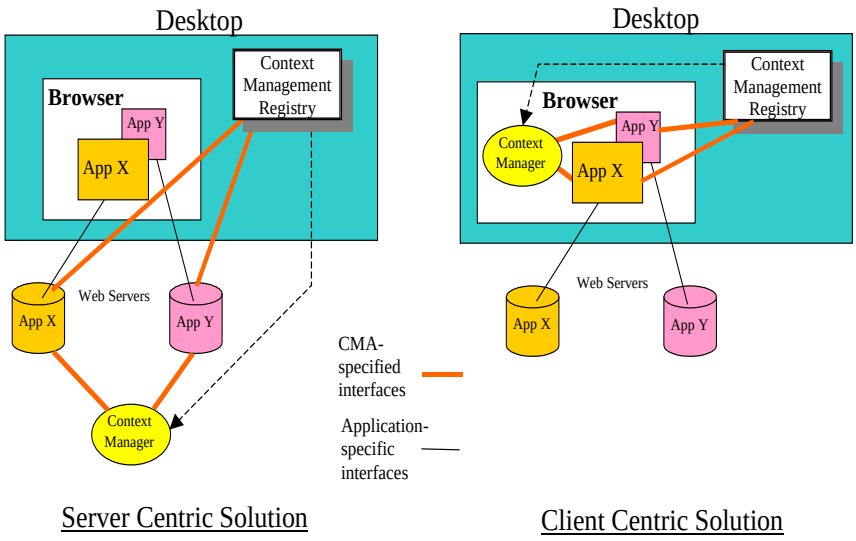

Figure 3: Example Component Distribution Options

Table 2: Web Component Distribution Options

Application 1	Application 2	Context Manager
Client	Client	Client
Client	Client	Server
Client	Server	Client
Client	Server	Server
Server	Server	Client
Server	Server	Server

2.6 IMPLEMENTING THE PING METHOD

Several CMA interfaces, such as ContextParticipant and ContextAgent, define the method Ping. This method has no inputs or outputs. The implementation of this method is supposed to return immediately. The successful return of this method indicates to the caller the presumed liveliness of the callee. The lack of a return, or an error instead of a return, indicates that the callee was not able to respond.

For certain web architectures, a surrogate such as a web server, receives CMA messages on behalf of a collection of context participants or context agents. When this is the case, there are circumstances under which the web server may be able to explicitly indicate via an error response that a pinged participant or agent is no longer live. A standard HTTP reply header with the standard HTTP response code 503 (Service Unavailable) shall be returned by the web server to the caller for this purpose.

3 Context Management Registry

The web technology mapping for the interface instance registry defined in the CMA Chapter 6, Component Model, is a local service, the address of which is well-known, that is resident on each desktop that hosts CMA-compliant application clients. The means by which this service is installed upon a desktop is not specified, and depends upon the implementation of the service.

The purpose of this local service, known as the context management registry (CMR), is to enable CMA-compliant component instances to be located. Specifically, a CMA-compliant web application may obtain from the registry the URL for the specific context manager instance that it is to use when participating in a context management session.

The registry is capable of responding to HTTP messages that employ URL-encoded arguments, per Chapter, 7, Representing CMA Methods as HTTP Messages. The registry listens for messages on the well-known port `2116`, which has been assigned by the Internet Port Number Assignment Authority to Health Level Seven expressly for use by implementations of the registry. Independent of the implementation of the registry, its URL is always `http://localhost:2116/`.

3.1 SECURITY CONCERNS

The proper use of the context management registry is necessary to defend against security attacks. The specific attack of concern is one in which rogue user A launches a valid CMA-compliant application from their desktop, but coerces the application to join unsuspecting user B's context session. This attack could be implemented by providing a compromised context management registry that returns URLs for CMA components associated with desktops other than the one on which it is running.

There are two parts to the defense against this attack: firstly ensuring that a client cannot obtain a valid URL for a context management component for another desktop; and secondly ensuring that the URL obtained by a local client is not compromised.

To ensure the first, the registry shall only service local requests to locate a CMA component, such as the context manager. The only way in which a client may issue such a request is for it to have a component co-resident on the same host as the registry it wants to communicate with. For a client component to be co-resident, it either needs to have been installed on the host, or downloaded in the form of a trusted applet. In either case, explicit user action is required. This means that the user is empowered to maintain the integrity of the desktop by being careful as to which clients they install or trust.

To ensure the second, any application that is architected such that its context participant is hosted other than on the desktop, e.g. on a web server, must implement a means to communicate the URL it obtains from a desktop-resident registry instance to its server in a secure fashion.

Such an application generally entails the use of a desktop-resident applet that is part of the application, and whose purpose is to obtain the necessary URL(s) from the registry. This applet then communicates the URL(s) to the server-resident portion of the application. In so doing, this communication needs to be protected so that the server-based context participant can be assured that the URL it receives came from its own desktop-resident program. Otherwise it would be possible for rogue user to coerce a server-based context participant to join the wrong context. For example, the rogue user might reverse-engineer the URL needed to communicate with the server and include the URL for the context manager as an argument.

When such an application is used over public networks, then an SSL connection shall be used to communicate the URL(s) from the client to the server. When used over a private network, in which case the

likelihood of such an attack is much less, the use of an SSL connection for this purpose is recommend but not required.

Though this section describes the implementation using applets, applications may use any other appropriate mechanism that supports the same level of security.

3.2 CONTEXT MANAGEMENT REGISTRY RESPONSIBILITIES

The context management registry shall support the interface ContextManagementRegistry, which in turn contains the method Locate. This method accepts as an input parameter the name of the component instance to locate, the CMA version of the component, the URL of the calling application's ContextParticipant interface, and an optional additional data element that further describes the component of interest. The use and content of this data element may vary across different types of components.

3.2.1 Locating the Context Management Registry

An application shall use the URL http://localhost:2116/ to address the registry. Each desktop that is capable of hosting CMA-compliant applications shall have a local instance of the registry listening on this port.

For security reasons (see Section 3.1), the registry will only accept invocations of the method ContextManagementRegistry::Locate that originate from the same host as the registry instance resides upon. Therefore, the portion of the application that invokes this method must be co-resident on the same host as the registry instance.

This generally means that embedded within the application's web page is code, typically in the form of a signed applet, that is capable of formulating the invocation of the method and then performing the method via an HTTP socket. When an applet is used, it needs to be signed so that it can perform the socket call within the framework of browser-based security mechanisms.

3.2.2 Locate Method

The Locate method returns three outputs. The first output is the URL for the desired component. This output may be used to communicate with the component.

The second output is a string that contains parameters that must be included in the HTTP GET or POST messages that are used to send method invocations to the located component. (See Section 10.1.3.1, Locate for details.)

The third output is the domain name for the organization or site being served by the located component (e.g, www.duke.edu might denote Duke University Medical Center, whereas www.mayo.edu might denote The Mayo Clinic). This output enables a registry client whose functionality varies depending upon the site or organization that it is serving to determine the site or organization on whose behalf the located component is operating. For example, an application served via a multi-customer application service provider might use this parameter to determine which customer site it is serving.

3.2.3 How the Registry Locates a Component

The means by which the context management registry determines which URL to return to the caller is not specified. The content and format of this URL depends upon the implementation of the located component. The URL may contain context session-specific and/or component instance-specific information. It may be the case that the registry has implementation-specific knowledge about the components that it locates and that this information is used by the registry to create the necessary URL's.

3.2.4 Locating the Context Manager

Currently, the registry may only be used to locate a context manager[2]. Using the Locate method, the name for the component to be located shall be the case insensitive string `CCOW.ContextManager` and the CMA version for this component shall be `1.5`. Both of these strings are case insensitive. Further, an additional input is used to specify the URL for an application's ContextParticipant. If this input represents an HTTP connection, then the URL returned by the registry shall also be for an HTTP connection. If this input represents an HTTPS connection, then the URL returned by the registry shall also be for an HTTPS connection.

3.2.5 Special Support for Applications Run In Citrix or Windows Terminal Server Remote Sessions

It may be the case that applications accessed from a clinical desktop are actually executing within a remote Citrix or Windows Terminal Server session, which in turn displays the application on the desktop. In order for these remote applications to join the context session for the desktop, it is necessary that they locate the same context manager as is located by the applications accessed directly from the desktop.

To enable this, an optional data element defined for the Locate method shall be used when the caller is an application running in a remote Citrix hosted or Windows Terminal Server (WTS) hosted session. This data element makes it possible for the context management registry that is resident on a Citrix or WTS server to locate the URL for the context manager associated with the remote desktop at which the application will be displayed. (See Section 10.1.3.1.2, Usage of the Parameter *descriptiveData*, for the specific usage of this data element by Citrix of WTS hosted applications.) The means by which a registry implementation locates the appropriate context manager URL this is not specified.

Note that it is always allowable to use this optional data element. When the data element is provided but it is not needed by the registry, then the registry shall ignore it.

3.3 USING THE CONTEXT MANAGER URL

An application shall obtain the URL for the context manager instance that it shall use solely from the context management registry resident on the desktop from which the application was launched. An application shall use the method ContextManagementRegistry::Locate to obtain the necessary URL whenever it joins the common context system (via ContextManager::JoinCommonContext) for a particular clinical desktop. For security reasons (see Section 3.1) an application shall never accept a context manager URL from another application.

When the application's context participant is implemented on the application's web server, then it will be necessary for the application to communicate the context manager URL it obtained from the registry to its server-based context participant. The means by which this is achieved depends upon the application's implementation. However, when the application is used over public networks, than for security reasons it shall be the case that this URL is communicated via an SSL connection.

An application shall not assume that the context manager URL it obtained is valid after it leaves a common context session (via ContextManager::LeaveCommonContext).

3.4 LOCATING AUTHENTICATION REPOSITORIES

An application that uses an authentication repository shall be configurable such that the URLs for the authentication repository that it is to use can be defined.

[2] In the future, the Context Management Registry may support the capability to locate other types of CMA components.

3.5 CONTEXT MANAGER RESPONSIBILITIES

A context manager shall be configurable such that the URLs for the site-specific context agents that it is to use can be defined.

4 Context Participant Implementation Responsibilities

Developers of CMA-compliant web applications have a great deal of flexibility in terms of how they implement their applications. However, there are CMA-imposed constraints that must be respected, as described below. Additional constraints are specified in the document HL7 Context Management Specification, User Interface, Microsoft Windows and Web Browsers, CM-1.6.

4.1 CONTEXT CHANGE NOTIFICATIONS

A context participant shall be capable of changing its data display whenever a context change transaction is initiated by the user via another application that is a participant in the same context session. Depending upon the architecture of a CMA-compliant web application, the implementation of the context system of which it is a member, and the network in which this system resides, it may be difficult to affect the change of the application's data display due to a context change.

The reason is that knowledge of the context change may reside on a server (i.e., one that hosts the context participant portion of one or more of the web applications and/or that hosts the context manager), and communication from server to client can be difficult or impossible in certain networks. This is particularly true for public networks wherein it may not be possible for a server to obtain the IP address of a client, or send it an unsolicited message, due to intervening firewalls, routers, or gateways.

4.1.1 Notification Protocol

To accommodate these restrictions, a protocol involving client-only HTTP messages is defined. This protocol is used for notifying the web pages belonging to CMA-compliant applications whenever a context change transaction that affects these pages is committed. Only the web pages for applications that do not have an alternative means for detecting the completion of a context change transaction need to participate in this protocol.

4.1.1.1 Listener Responsibilities

When first displayed, a web page that requires a local notification of a context change must open an HTTP socket, register the URL, including the port number, for the socket with the context manager, and listen on the socket for a context change notification. The socket shall be created such that its port number is dynamically assigned by the underlying operating system. The method Register, defined for the context manager's ListenerRegistrar interface, shall be used to register the URL for a web page's listener socket (See Section 10.1.2, ListenerRegistrar).

The method ListenerRegistrar::Register also requires the participant coupon for the application that is displaying this web page. This enables the context manager to associate a listener URL with a specific context participant. This means that the application must first join the common context via ContextManager::JoinCommonContext before any of its web pages can register their URLs with the context manager.

An application may have multiple web pages, each of which is a listener. Each page may create its own listener socket and register this socket with the context manager.

When a web page receives a notification on its listener socket it shall resynchronize the context-sensitive portions of its display with the current context. The means by which the web page resynchronizes with the context depends upon the implementation of the underlying web application.

The listener socket shall be closed and unregistered from the context manager when the page is no longer displayed. The method Unregister, defined for the context manager's ListenerRegistrar interface, shall be used to unregister the URL for a web page's listener socket (See Section 10.1.2, ListenerRegistrar). The context manager shall unregister all listener URLs for a context participant when the participant leaves the common context and has not already unregistered its listener URL(s).

4.1.1.2 Notifier Responsibilities

At the conclusion of a committed transaction, the web page displayed by the application that was used to instigate a context change transaction shall send an HTTP GET message to each local socket registered with the context manager. The list of URLs for these sockets shall be returned by the context manager as an output parameter for the method PublishChangesDecision defined for the interface ContextManager. This is a technology-specific parameter (i.e., it is not defined in the CMA definition for this method). Adding this parameter to this method provides a simple means for an application to obtain the necessary port numbers when it completes a transaction.

4.1.1.3 Notification Message

The notification message shall contain a single URL-encoded parameter with the name *contextChangeCoupon*, the value of which shall be the long integer value representing the coupon for the committed context change transaction. The representation of this parameter shall be as defined in Chapter 7, Representing CMA Methods as HTTP Messages, Section 7.3, Named Arguments.

4.1.2 Avoiding Race Conditions

A window exists in which a context change transaction instigated by another application in the same context session may occur before the listener socket is ready to receive notifications. This is due to the fact that an application's web page cannot register its listener port until after the application has joined the common context. The application that joined the context system may miss the necessary notification and therefore be out of synchrony with the current context.

To avoid this situation, the method ListenerRegistrar::Register, which is used to register with the context manager the port number for a listener socket, returns the context coupon for the most recently committed context change transaction. A web page shall compare this coupon to the coupon it believes to represent the most recent transaction. The means by which the web page determines the value of the coupon it believes to represent the most recent transaction depends upon the implementation of the underlying web application.

If the value of the coupon returned by ListenerRegistrar::Register is greater than the coupon held by the web page, then the web page shall resynchronize the context-sensitive portions of its display with the current context. The means by which the web page resynchronizes with the context depends upon the implementation of the underlying web application.

4.1.3 Listener and Notifier Applets

A web page may use a signed applet or equivalent technology to implement its listener and/or notifier responsibilities. A third-party applet may be used as a listener, or as a notifier, simply by embedding the applet into the necessary web pages.

4.2 BROWSER-RESIDENT CONTEXT PARTICIPANTS

The context participant portion of a web application may reside on the client within the browser, or on the application's server. In either case the context participant must be able to open an HTTP connection to the context manager, and it must also be able to accept incoming HTTP messages that emanate from the context manager.

This requirement may pose certain complexities when the participant is browser resident because the default security model for most browsers prevents a web page from communicating with servers other than the one that provided the web page. One way to avoid this restriction is to use a signed applet. A signed applet enables users to decide whether they trust the source of an applet that is subject to less restrictive security constraints.

4.3 CACHED WEB PAGES

Most web browsers support the caching of web pages that the user has previously accessed. This enables the browser to redisplay the page without necessarily accessing the server from whence the page came. A redisplay might occur if the URL to a cached page is opened by the user via another page, the URL is explicitly entered by the user as an address, or the user pages back to view previously viewed pages.

It is possible that the context has changed since the last time the page was displayed. A web application that is an active context participant shall show data that is consistent with the new context, or it may attempt to set the context to reflect its internal state.

In order to implement either of these behaviors, a page must be capable of detecting that it is being redisplayed. This can be accomplished, for example, by embedding an applet or a scripted function whose job is to perform the necessary actions upon a display event.

Alternatively, a page can be marked as not cacheable. This means that the page will be freshly retrieved by the browser from the page's web server each time the page is accessed. A page marked as not cacheable will, by definition, always reflect the current context.

4.4 CONTEXT CHANGE USER DIALOGS

The CMA specifies that if a surveyed application votes to conditionally accept a context change, or if an application is busy, then the instigating application must present the user with a dialog through which the desired course of action can be indicated.

The appearance and wording of this dialog for CMA-compliant web applications shall be the same as defined in the document HL7 Context Management Specification, User Interface, Microsoft Windows and Web Browsers, CM-1.6.

There are many ways in which this dialog can be implemented, including as a Java applet, using Java Script, or HTML.

4.5 WHEN TO LEAVE THE COMMON CONTEXT

A web application joins the common context the first time it is accessed from a particular clinical desktop. However, it is not as easy to for a web application to decide when to leave the context. This is because the nature of the web is such that applications are not so much opened and closed, but rather are visited, often periodically over extended periods of time.

In general, an application should leave the common context when its web pages are no longer resident on the desktop from which the application is being accessed. This includes cached pages as well as pages in view.

There are several ways that an application can accomplish this. First, it can assert that its pages should never be cached. This way, an application maintains participation in the context system only when its pages are visible.

Alternatively, an application can implement its pages so that they are cacheable, but with the inclusion of an applet or scripted function that detects when the browser purges the page from the cache. The application ceases its participation in the context system when it no longer has any cached or visible pages. (A variation of this approach is for the application to designate an expiration time for its pages, ensuring

that they will be discarded after some period of time, thereby enabling the application to leave the context system when all of its pages have expired.)

5 Context Action Continuations

A context action continuation entails the gathering of user input by an action agent when it services a context action so that it can complete the action and/or the display of information to the user in order to communicate the result of performing an action. In a web-based implementation of the CMA, the execution of a continuation involves the cooperation of the context participant that requested the action, the context manager, and the action agent [3].

The goal is to enable the seamless transfer of user focus in a manner that also ensures that all of the necessary context-sensitive operations can still be performed. Accomplishing this entails the routing of the context action methods through the context manager just as would be the case if a continuation was not involved. This enables the context manager to ensure that the necessary security checks are performed and policies are enforced, and to facilitate the mapping of the method inputs and outputs by a mapping agent.

Continuations are optional and depend upon each action agent's implementation. However, in order to ensure implementation independence between context participants and action agents, a context participant that requests an action, as well as the context manager, must always be prepared to handle a continuation. This enables the decision to employ a continuation to rest entirely within the implementation of each action agent.

5.1 POLICY WITHOUT CONTINUATIONS

When a continuation is not used by an action agent, the process for performing a context action involves a call by a context participant to the context manager's ContextManager::Perform method, followed by a call by the context manager to the appropriate context agent's ContextAgent::ContextChangesPending method.

In the absence of the need for additional user input, the context agent performs the action and returns the action result to the context manager. The context manager relays this response to the context participant. The method calls and responses are implemented using the standard technique defined in this document, in which each method call and associated response is mapped into a single HTTP request/reply message pair.

However, because the context participant does not know whether or not a continuation will be necessary, it shall always provide a value for the web technology-specific parameters named "CPCallBackURL" and "CPErrorURL" to the method ContextAction::Perform. The value for "CPCallBackURL" parameter represents the URL that the context manager should call-back to as the final step in processing a continuation, if one was to occur, at which point the flow of control is returned to the context participant. See Section 5.2 for more information about the use of this parameter. The value for "CPErrorURL" is also a call-back URL, but is called if the context manager or context agent experiences an exception when attempting to perform the context action. This URL facilitates the creation of applications that have different entry points for handling normal versus exceptional processing. (Note that the values of these URLs can be the same.)

The constituent steps of this flow of events are illustrated in Figure 4. Message Flow for Context Action Without Continuation.

[3] A continuation is the coordinated transfer of the user's focus from the context participant application (with which the user's interaction resulted in a request to perform a context action) to the application that services the context action request. This transfer of focus is facilitated by the context manager in order to ensure that there are no implementation dependencies between context participants and context action applications.

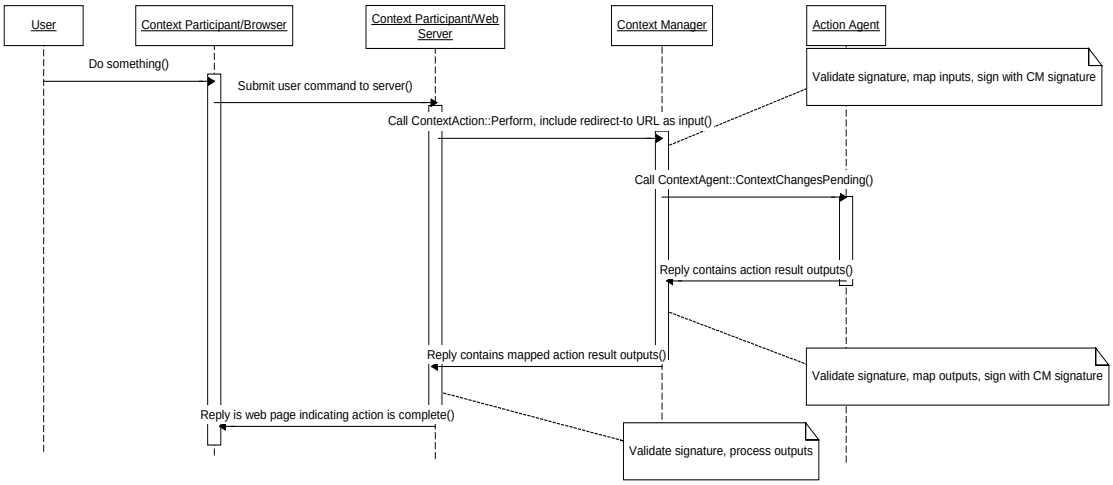

Figure 4: Message Flow for Context Action Without Continuation

5.2 POLICY WITH CONTINUATIONS

In order to support continuations in a web system, it is necessary to restructure the flow of HTTP messages as compared to the case in which a continuation is not involved. The objective is to provide a seamless means for redirecting the user's focus from an application to an agent and then back to the application. The established web paradigm for implementing this type of flow of control is the redirection of the web browser. A somewhat elaborated version of a typical browser redirection scheme is employed as the basis for continuations.

In the architecture for web-based continuations, the first redirect is used to transfer control of the browser from the context participant application that the user is interacting with to the context agent. A second redirect transmits the context action result via the browser to the context manager. This enables the context manager to validate the authenticity of the message and to see that the outputs are mapped as necessary. It also shifts control of the browser from the agent to the context manager. The final redirect is the used by the context manager to transfer control of the browser back to the original context participant application.

Given this background, the following policy shall be followed in order to support web-based continuations, wherein the context manager and context participants shall process all redirect messages per HTTP norms. The constituent steps of the policy are also labeled in Figure 5. Message Flow for Browser-Initiated Context Action With Continuation and Figure 6. Message Flow for Server-Initiated Context Action With Continuation:

Step 1: The user performs a gesture via the context participant's user interface, the process of which requires that the participant perform a context action.

Step 2: The context participant shall call the method ContextAction::Perform on the context manager, issuing the call as an HTTP request. The method call is URL-encoded in the request per the conventions established in this document. In so doing, the context participant shall include a web technology-specific parameter that represents the URL that the context manager should call-back to as the final step in processing the continuation, at which point the flow of control is returned to the context participant. This parameter shall be named "CPCallBackURL".

Step 3: The context manager processes the call by performing the necessary security checks and by facilitating the mapping of the inputs. It then calls the method ContextAgent::ContextChangesPending upon the appropriate action agent, providing the mapped inputs as the inputs to this call. The method call is URL-encoded in the request per the conventions established in this document.

Step 4: The context agent shall initiate a continuation by responding to the context manager's call to ContextAgent::ContextChangesPending with an HTTP redirect message. This message transfers control of the context participant's browser to the agent so that the agent can gather additional inputs from the user. (The context participant will regain control of the browser when the context action has been performed.) The URL to redirect and its format and contents to is not specified, but rather depends upon the agent's implementation.

Step 5: Consistent with the point above, when a continuation is involved, the context manager does not receive a normal HTTP reply message when it calls ContextAgent::ContextChangesPending, within which the method outputs have been encoded in the reply message body. Instead, it receives an HTTP redirect message from the action agent, which it shall pass on to the context participant that initiated the context action request. (The context manager will eventually receive the method outputs that represent the response to its call to ContextAgent::ContextChangesPending via an HTTP request message that is the result of another redirect, as described below.)

Step 6: When the context manager processes the redirect it receives from the action agent, it will append a URL-encoded call-back URL as an argument to the redirect URL established by the agent. The name of the appended call-back URL shall be "CMCallBackURL". The context manager then replies to the context participant's call to ContextAction::Perform with an HTTP redirect message, in which the redirect URL includes the redirect URL established by the action agent with the context manager's call-back URL appended as an argument.

Step 7 - 8: The context participant processes the redirect, which has the effect of transferring control of the browser to the action agent. This happens automatically if the portion of the context participant that issued the call to the context manager's ContextAction::Perform method is browser-resident. Otherwise, if the portion of the context participant that issued the call is server-resident, then the server must reply to the browser with the redirect, leaving the URL in the redirect in tact. The browser will then automatically redirect to the action agent.

Step 9 - 13: The action agent uses the browser to gather additional user inputs via web pages provided by the agent. How this is accomplished is not specified and depends upon the agent's implementation.

Step 14 - 16: When the action agent completes its continuation and performs the context action, it creates a response to the context manager's previous call to ContextAgent::ContextChangesPending. This response is represented as an HTTP redirect message that is the reply to the final submission via the browser of the user's inputs for the continuation. The redirect URL is comprised of the context manager's call-back URL (i.e., "CMCallBackURL") with the URL-encoded outputs to ContextAgent::ContextChangesPending appended as arguments. The redirect will result in an HTTP request message being sent via the browser to the context manager.

Step 17: The context manager shall process the HTTP redirect that contains the URL-encoded outputs to its previous call to ContextAgent::ContextChangesPending as if it had received the outputs as a direct reply to the HTTP request message that initiated the call.

Step 18 - 20: When the context manger completes its part of the continuation (e.g., validates the action agent's digital signature, facilitates the mapping of the context action outputs, etc.), it creates a response to the context participant's previous call to ContextManager::Perform. This response is represented as an HTTP redirect message that is the reply to the HTTP request message received in the previous step. The redirect URL is comprised of the context participant's call-back URL (i.e., "CPCallBackURL") with the URL-encoded outputs to ContextManager::Perform appended as arguments. The redirect will result in an HTTP request message being sent via the browser to the context participant's web server.

Step 21: The server-resident portion of the context participant shall process the HTTP redirect that contains the URL-encoded outputs to its previous call ContextManager::Perform as if it had received the outputs as a direct reply to the HTTP request message that initiated the call.

Step 22: The server-resident portion of the context participant replies to the browser with the web page that represents the completion of the execution of the context action. The appearance and content of this page depends upon the context participant's implementation.

Note that variations in Steps 21 and 22 are possible. For example, the necessary processing could occur solely with the browser-resident portion of the context participant. The decision as to where to perform this processing depends upon the context participant's implementation.

Also note that a context participant is able to match the message it receives in Step 21 with the method call it performed in Step 2 by encoding context participant-specific data in the call-back URL it provided to the context manager in Step 2. Similarly, the context manager is able to match the message it receives in Step 17 with the method call it performed in Step 3 by encoding context manager-specific data in the call-back URL it provided on the redirect message in Step 6.

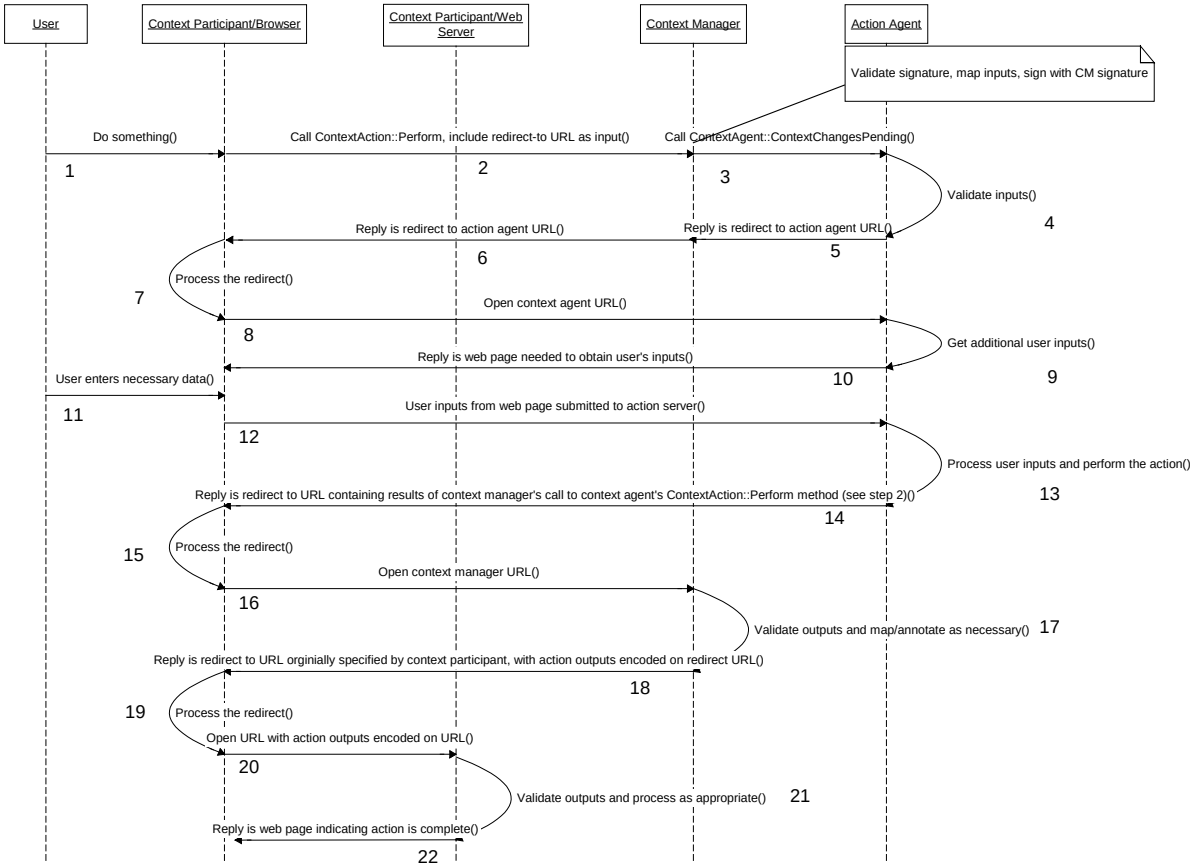

Figure 5: Message Flow for Browser-Initiated Context Action With Continuation

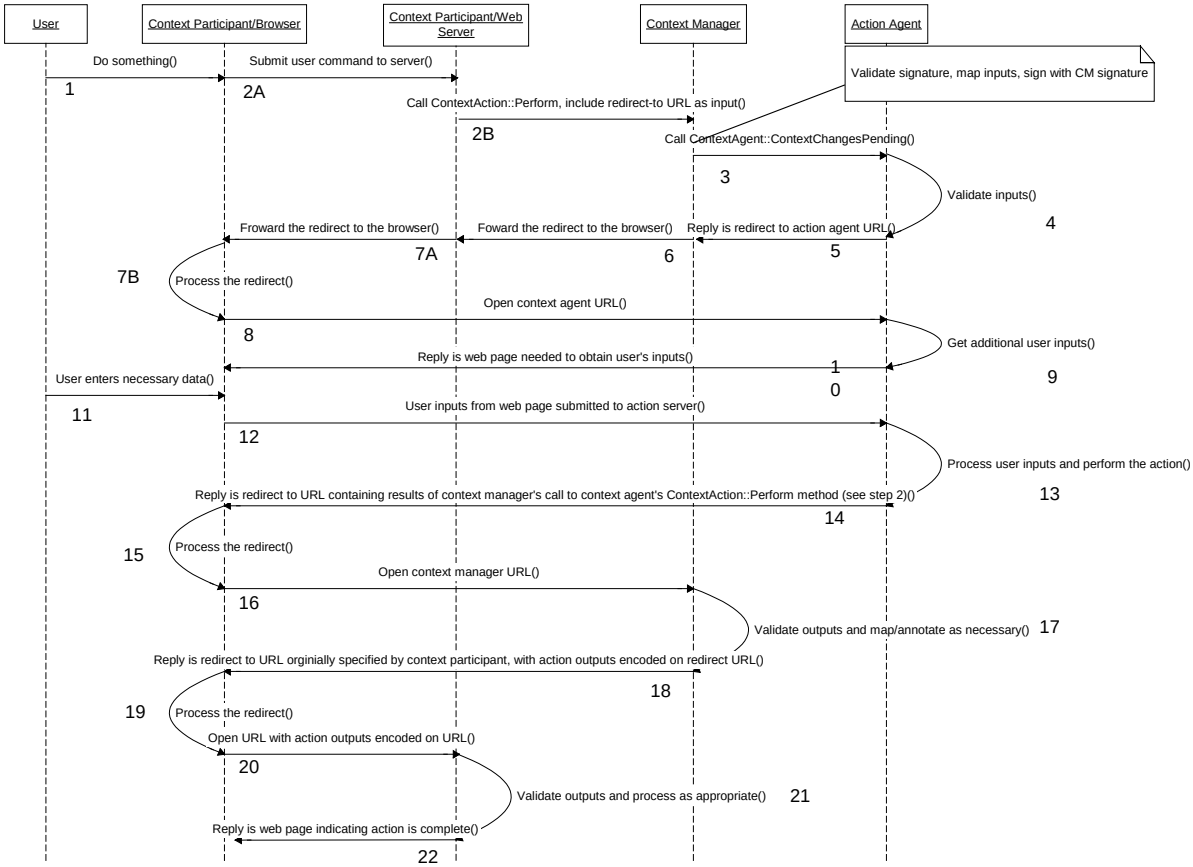

Figure 6: Message Flow for Server-Initiated Context Action With Continuation

6 Security

The web technology mapping for securing communications amongst and between CMA-compliant applications and components employs a two-tier approach.

The required tier implements the chain of trust and associated secure interfaces specified in the CMA using web technologies. This ensures that all CMA-related communications pertaining to the chain of trust can be authenticated and message integrity can be verified, enabling an authentication repository and context manager to respectively enforce client access privileges (where clients are programs, not people).

The second tier, which is required when CMA-related communications take place over public networks, additionally ensures that all such communications are encrypted while also providing an added degree of message authentication. Second tier encryption prevents unwanted parties from viewing context data as it is communicated across the network. Second tier message authentication ensures that only known parties (where a party is a program, not a person) may participate in a context system.

The second tier protections are achieved by channeling all CMA-related HTTP communications through an inherently secure network connection using upon the Secure Socket Layer v3 standard.

The details for implementing tier 1 security and tier 2 security are described below.

6.1 TIER 1 SECURITY: CMA INTERFACES

Web implementations of the CMA-defined secure interfaces shall use the RSA public key / private key scheme and shall use the MD5 one-way hash algorithm. This ensures that all CMA-related HTTP messages can be authenticated and message integrity can be verified using robust and proven algorithms that can be implemented with readily available commercial technologies.

6.1.1 Secure Binding Properties

The CMA-defined interface SecureBinding requires that the bindee indicate to the binder various security properties that depend upon the bindee's implementation. The properties that must be indicated, and the allowed value or values for each property, depend upon the underlying implementation technology.

For a web implementation, the property names and values defined in Table 3. Properties for Passcode-Based Secure Binding shall be used for establishing a passcode-based secure binding.

Table 3: Properties for Passcode-Based Secure Binding

Property Name	Allowed Value	Meaning
Technology	Web	Web technology.
PubKeyScheme	RSA	RSA public key / private key scheme.
PubKeySize	*N*	N is a positive integer representing the number of bits contained in the public key.
HashAlgo	MD5	MD5 secure hash algorithm (creates 128 bit hash).

Property names and values are not case sensitive. Property values shall be character-encoded per the convention stated in the CMA specification.

For a web implementation, the following secure binding property names and values defined in Table 4. Properties for PKI-Based Secure Binding shall be used.

Table 4: Properties for PKI-Based Secure Binding

Property Name	Allowed Value	Meaning
Technology	Web	Web technology.
Certificate	X.509	The binding is PKI-based and requires the use of X.509-compliant certificates.
PubKeyScheme	RSA	RSA public key / private key scheme.
PubKeySize	*N*	N is a positive integer representing the number of bits contained in the public key.
HashAlgo	MD5	MD5 secure hash algorithm (creates 128 bit hash).

6.1.2 Creating Digital Signatures

A digital signature is created from a hash of the data that is to be signed. Per the CMA, only the characters that represent data values are used in computing the signature. The special characters used for HTTP-encoding messages shall not be used in computing the hash. (See Chapter, 7, Representing CMA Methods as HTTP Messages.)

6.1.3 Signature Format

Digital signatures passed via any of the CMA-defined web interfaces shall be MD5 signatures with block padding as defined in PKCS#1[4]. The bytes in a signature are in big-endian order. This binary data shall be character-encoded per the convention defined in the CMA specification.

6.1.4 Certificate Format

Certificates shall be represented as a DER encoded string per the ISO/IEC X.509 v3 standard. This string contains binary data that has been character-encoded per the convention defined in the CMA specification.

6.1.5 Public Key Format

Public keys passed via any of the CMA-defined web interfaces shall be represented as a DER encoded string per the ISO/IEC X.509 v3 standard. The bytes in a public key are in big-endian order. This string contains binary data that has been character-encoded per the convention defined in the CMA specification.

6.1.6 Hash Value Format

Hash values passed via any of the CMA-defined web interfaces shall be MD5 hashes as defined in RFC1321[5]. The bytes in a hash are in big-endian order. This binary data shall be character-encoded per the convention defined in the CMA specification. Hash values shall be compared for equality by comparing their character-encoded string representations. Character case shall not be considered when comparing these strings.

6.2 TIER 2 SECURITY: SECURE SOCKETS LAYER

When communicating over public networks, CMA-compliant web applications and components shall use SSL. Mutual authentication and 112-bit triple DES symmetric keys shall be used, wherein applications and

[4] PKCS #1 v2.0: RSA Cryptography Standard. RSA Laboratories, October 1, 1998.

[5] R. Rivest. RFC 1321: The MD5 Message-Digest Algorithm. Internet Activities Board, April 1992.

context agents are clients, an authentication repository and the context manager are servers. This requires that each CMA application and component implementation shall have an X.509 v3 compatible certificate.

6.2.1 Certificate Creation

The certificate shall be issued by a certificate authority that the site trusts. The signature algorithm shall be RSA. CMA applications and components shall use the names shown in Table 5 as one of the organizational unit (ou) fields within the certificate subject name.

Table 5. Certificate Subject Names

Component	Field
Application	ou=CCOW.ApplicationName where ApplicationName is the same name that application uses when it performs ContextManager::JoinCommonContext, less the instance suffix
Authentication Repository	ou=CCOW.AuthenticationRepository
Context Manager	ou=CCOW.ContextManager
Mapping Agent	ou=CCOW.MappingAgent_Subject where Subject is the name of the standard or custom mapped subject
Annotation Agent	ou=CCOW.AnnotationAgent_Subject where Subject is the name of the standard or custom annotated subject
Action Agent	Ou=CCOW.ActionAgent_Subject where Subject is the name of the standard or custom action subject

The field names in are not case sensitive. The characters used to represent an application name may only include alphanumeric characters, the period (.) character and the underscore (_) character.

6.2.2 Certificate Authentication

CMA-compliant web applications and CMA components shall be configurable such that the implementations of these components can be instructed by a systems administrator as to which other CMA application and/or component implementations they should trust when presented with a valid certificate. Further, these applications and components shall be configurable such that they can instructed by a systems administrator as to which certificate authorities they are to trust for the purpose of authenticating a certificate presented by a CMA component.

6.3 ADDITIONAL SECURITY CONSIDERATIONS

It is recommended that users of CMA-compliant applications should not be able to see or otherwise easily access the cipher text, which is data that has been either digitally signed and/or encrypted using digital keys. A malicious user could use cipher text in combination with the corresponding plain text to discover private signature and encryption keys.

7 Representing CMA Methods as HTTP Messages

The general schema for representing CMA methods as HTTP messages is to URL-encode the method name and associated parameters as a non-cached HTTP message. The authoritative source for URL-encoding is IETF RFC 1738, which can be found at http://www.ietf.org/rfc/rfc1738.txt.

All CMA web-based components (e.g., context manager, context agent, etc.) shall be capable of receiving CMA methods encoded as HTTP POST or GET messages. The context manager, which is the only component that communicates directly with an application, shall only send HTTP GET messages to applications.

CMA-compliant web applications shall only need to receive CMA methods encoded as HTTP GET messages, but may send HTTP POST or GET messages to CMA components.

7.1 COMPONENT REFERENCE

A component reference is represented as a URL. The path for a component (e.g., Context Manager) shall be encoded in the file name portion of the URL, for example:

```
www.mcis.duke.edu/CCOW/ContextManager
```

7.2 MIME HEADER

The HTTP MIME header for both request and reply messages shall define the value for the standard Content-Type header using the standard name:

```
Content-Type: application/x-www-form-urlencoded
```

The header shall also indicate that the message should not be cached. In HTTP 1.0, the Expires header should be set to a time that is earlier than when the request was issued (an arbitrarily early time may be used):

```
Expires: Mon, 01 Jan 1990 00:00:00 GMT
```

In HTTP 1.1, the Cache-Control header shall be set to indicate that the maximum time that the message should be considered as fresh is zero seconds, and that this information must be respected (the net effect is that messages will not be cached):

```
Cache-Control: max-age=0, must-revalidate
```

7.3 NAMED ARGUMENTS

Named arguments shall be encoded as an argument name followed by the equal sign character (=) followed by a character-encoded representation of the argument's value. If the argument is not the first in the list, then it shall be preceded by the ampersand character (&), for example:

```
&name=value
```

The order in which named arguments are encoded shall not matter. Applications and components shall ignore any named argument whose name is not recognized. Argument names are not case sensitive. The case sensitivity of argument values is specified in the CMA.

7.3.1 Interface Name

The method name shall be encoded as the second named argument, for example:

```
&method=SetItemValues
```

If the requested interface is not implemented by the component, then it shall return an HTTP reply header with the standard HTTP response code 404 (Not Found).

7.3.2 Method Name

The method name shall be encoded as the second named argument, for example:

```
&method=SetItemValues
```

If the requested interface is not implemented by the component, then it shall return an HTTP reply header with the standard HTTP response code 404 (Not Found).

7.3.3 Input Parameters

The method input parameters as defined in the CMA specification shall be encoded as the remaining arguments using the same names that appear in the CMA.

If a parameter is not recognized by the component, then the component shall ignore the parameter. If a necessary parameter is missing, then the component shall return an HTTP reply header with the standard HTTP response code 404 (Not Found).

7.3.3.1 Data Value Representation

All data values shall be converted to string representations per the following CMA Specification Document Sections: 17.2.7, Character Encoded Binary Data; 17.2.8, Representing Message Authentication Codes, Signatures and Public Keys; Section 17.2.9, Representing Basic Data Types as Strings.

7.3.3.2 Arrays

Arrays are encoded as a vertical bar (|)-separated list of elements[6]. For example:

```
&itemNames=Patient.Id.MRN.medical_center|Patient.Co.Name
&itemValues=123-81283-JMDH-79|Marchant^Kyle^^^^
&contextCoupon=27
&appSignature=0BC12D890913E9C98182808CD00BB983288A81238
```

7.3.3.3 Null Value

A method input parameter whose value is null shall not have any value encoded. For example:

```
&contextParticipant=
```

7.3.3.4 Empty String

A method input parameter whose value is an empty string shall not have any value encoded. For example, a parameter whose value is an empty string and a parameter whose value is an array with two elements, the first of which is an empty string, is shown below:

```
&appSignature=
&itemValues=|Marchant^Kyle^^^^
```

[6] A vertical bar (|) is the default HL7 field delimiter, whereas the carrot (^) character is the default HL7 component separator

7.3.3.5 Empty Array

A method input parameter whose value is an array with no elements shall not have any value encoded. For example:

```
&itemNames=
```

7.4 HTTP CHARACTER ENCODING CONVENTIONS

All characters used in representing an argument value (i.e., to the right of the equal sign (=)) shall be encoded per HTTP conventions, defined in IETF RFC 2396, Section 2.4, which can be found at http://www.ietf.org/rfc/rfc2396.txt. A conservative summary of these conventions is as follows:

- The ASCII characters '**a**' through '**z**', '**A**' through '**Z**', and '**0**' through '**9**' remain the same.
- The space character ' ' is converted into a plus sign '**+**'.
- All other characters are converted into the 3-character string "**%*xy***", where *xy* is the two-digit hexadecimal representation of the lower 8-bits of the character.

7.5 OUTPUTS

Method output parameters are encoded in the body of an HTTP reply message. These parameters are encoded using the same scheme as for encoding input parameters. The HTTP reply header shall include the standard HTTP response code 200 (OK) unless otherwise noted in the interface definitions in Chapter 10, Interface Listing.

7.6 EXCEPTIONS

CMA-specified exceptions are encoded in the same manner as outputs. However, in lieu of outputs, the exception shall be encoded in the body of the reply message as a pseudo output parameter whose name is `exception` and whose value is the name of the exception, as follows:

```
exception=ExceptionName
```

If the exception includes data members, then these members shall be encoded in the body of the reply message following the exception name. These members shall be encoded using a same scheme as for encoding input parameters. If members are encoded, then the first member shall be preceded by an ampersand (&), and subsequent members shall be delimited by an ampersand (&), for example:

```
exception=BadItemValue&itemName=Patient.Co.Sex&itemValue=G&reason=Must be F, M, O or U
```

The optional pseudo output parameter whose name is exceptionMessage and whose value is an explanation of the exception may also be encoded in the body of the response message:

```
exceptionMessage=explanation
```

This parameter is intended for diagnostic purposes. The content of the explanation is not specified and is implementation-dependent.

7.7 REDIRECTS FOR CONTEXT ACTION CONTINUATIONS

Special conventions are necessary when HTTP redirect messages are used to send a method call or a method response as part of the processing associated with a context action continuation. (See Chapter 5.)

When a method call is performed via a redirect message the call shall be encoded in the redirect URL using the same scheme as described above for a conventional call encoded in an HTTP request message URL. This URL is contained in the redirect message header in the field named *location*. The HTTP response code shall be set to 302 (Moved Temporarily).

When a method response is sent via a redirect message, the method outputs or exception shall be encoded as URL arguments using the same scheme as described above for a conventional method response that is encoded in the body of an HTTP reply message. This URL is contained in the redirect message header in the field named *location*. The HTTP response code shall be set to 302 (Moved Temporarily).

8 Error Handling

The CMA specifies a set of exceptions that can be raised by CMA components. (Context participant applications do not currently throw exceptions). The names for these exceptions as they shall appear in HTTP reply messages are listed in Table 6. Additional information may be included in the reply message when an exception is raised. This information shall be encoded as specified in Section 7.6, Exceptions.

Table 6: Exception Names

Exception	Explanation
NotImplemented	Method not implemented.
GeneralFailure	A context management failure, not represented by any of the other exceptions, has occurred.
ChangesNotEnded	Attempt to publish context changes before ending the context change transaction.
InvalidContextCoupon	A context coupon does not match most recently committed coupon or current transaction coupon.
NameValueCountMismatch	A name array and its corresponding value array do not have the same number of elements.
NotInTransaction	Attempt to perform a context management transaction method when a transaction is not in progress.
TransactionInProgress	Attempt to perform a context management method when a transaction is in progress.
UnknownItemName	An item name not known.
UnknownParticipant	Participant coupon does not denote a known participant.
TooManyParticipants	Attempt to join a context that can't accommodate another participant.
AcceptNotPossible	Attempt to publish an "accept" decision but there were participants for which it was not possible to obtain a survey response (e.g., these participants were blocked)
BadItemNameFormat	An item name does not conform to format rules.
BadItemType	An item data type does not conform to data definition for the item.
BadItemValue	An item value does not conform to the allowed set of values as defined by the data definition for the item.
InvalidTransaction	A transaction has been invalidated and aborted because it violates one or more semantic integrity constraints.
UndoNotPossible	Attempt to undo context changes after the transaction has ended.
ChangesNotPossible	Attempt to set or delete context data after the transaction has ended.
ChangesNotAllowed	Mapping agent attempts set or delete a context data item that has been set by the participant that instigated the transaction.
AuthenticationFailed	A signature could not be authenticated.
SignatureRequired	A signature is required to perform the method.
UnknownApplication	An application name is not known.
UnknownConnection	A connection is not known to the authentication repository.
LogonNotFound	The desired user logon is not found in the authentication repository.
UnknownDataFormat	The format of user authentication data requested could not be found in the authentication repository.

Exception	Explanation
UnknownBindee	A secure binding coupon does not denote a known bindee.
ImproperKeyFormat	A public key is not properly formatted.
BindingRejected	The identity of a bindee could not be verified.
ImproperMACFormat	A message authentication code is not properly formatted.
UnknownPropertyName	A property name is not known.
BadPropertyType	A property data type does not conform to specification.
BadPropertyValue	A property data value does not conform to specification.
AlreadyJoined	The application has already joined the context.
ActionNotPossible	There is presently a context change transaction in progress or another action is being performed.
ActionNotAllowed	Application attempts to set items other than for the context action to be performed.
FilterNotSet	Application attempts to get the names of subjects that are being filtered on its behalf, but a filter is has not yet been set by the application.
UnableToLocate	The context management registry could not locate the specified component instance.
ImproperSignatureFormat	A digital signature code is not properly formatted.
ContextNotActive	A context manager does not represent an active session.

9 Character Set

CMA methods are mapped to HTTP messages, and may be partially or entirely encoded as part of a URL. URLs are currently required to be in US-ASCII, the character set referred to as ISO-8859-1, according to RFC2396. Therefore, only the ASCII character set shall be used to represent any data that is transmitted as part of a CMA-defined method.

However, it may be necessary to represent the data values for certain CMA method parameters using a local character set (i.e., not US-ASCII). When this is the case, the parameter value may be represented using Unicode (see http://www.unicode.org).

In doing so, each Unicode codepoint within the Unicode string shall be encoded as a two-character US-ASCII string representing the hex value of the Unicode codepoint. Each such two-character string shall be preceded by the percent (%) character to signal the message recipient that the following two characters are to be interpreted as hex value for a Unicode codepoint. The high byte of the Unicode codepoint shall be encoded lexically before the low byte.

10 Interface Listing

10.1 TECHNOLOGY-SPECIFIC INTERFACES

10.1.1 InterfaceInformation

This interface is not defined by the CMA. This interface is a technology-specific interface that enables any CMA-compliant web component to implement the interface interrogation capability that is defined in the CMA. This interface enables a component to interrogate another component to determine if it implements a particular interface.

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"InterfaceInformation"
method	string	"Interrogate"
interfaceName	string	The name of the interface of interest. Case Sensitive.
HTTP Reply Message		
implemented	boolean	True if the component implements the requested interface, false otherwise.
Possible Exceptions		
None		

10.1.2 ListenerRegistrar

This interface is not defined by the CMA. This interface is a technology-specific interface that enables the context manager to maintain a list of client-side URLs for use in notifying web pages about committed context change transactions. Note that the methods for this interface fail silent – they do not raise any exceptions.

10.1.2.1 Register

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ListenerRegistrar"
method	string	"Register"
url	string	The listener's URL.
participantCoupon	long	The coupon denoting the participant whose listener URL is to be registered.
HTTP Reply Message		
contextCoupon	long	The context coupon for the most recently committed context change transaction.
Possible Exceptions		
None		

The default formulation of the listener URL shall be

```
http://localhost:pn/
```

where *pn* is an integer representing the port number for the listener's socket.

10.1.2.2 Unregister

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ListenerRegistrar"
method	string	"Unregister"
url	string	The listener URL that is to be unregistered.
HTTP Reply Message		
empty		
Possible Exceptions		
None		

10.1.3 ContextManagementRegistry

This interface is not defined by the CMA. This interface is a technology-specific interface that enables a host to obtain a URL to the web implementation of a CMA component instance.

10.1.3.1 Locate

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextManagementRegistry"
method	string	"Locate"
componentName	string	The name of the component to locate. Currently, only the value CCOW.ContextManager shall be used. (In the future, the registry may be capable of locating other types of components.) The value for this parameter is not case sensitive.
version	string	The version of the CMA specification that the component implements. Currently, only the value `1.5` shall be used. The value for this parameter is not case sensitive.
descriptiveData	string	This optional parameter is additional data used to describe the component of interest. The allowed values and interpretation of this parameter depends upon the type of component to be located. See Section 10.1.3.1.2 for the currently supported uses of this parameter.
contextParticipant	string	Participant's URL. If this URL is for an HTTPS connection, the output componentURL will also be for an HTTPS connection, otherwise, the output URL will be for a normal HTTP connection.
HTTP Reply Message		
componentUrl	string	The URL of the desired component.
componentParameters	string	Optional parameters that if present shall be included with every call to the located component. If the call is sent using an HTTP GET, a "?" shall be appended to *componentURL* followed by the *componentParameter* string and any other method specific parameters. If the call is sent using an HTTP POST, the parameters shall be sent as part of the body of the message along with any other method specific parameters. The component parameters are opaque and should not be interpreted by a client. See Section 10.1.3.1.1for examples.
site	string	The domain name (e.g., `www.duke.edu`) of the site or organization that is being served by the located component (i.e., referenced by *componentURL*).
Possible Exceptions		
UnableToLocate	The registry could not locate the specified component.	

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
UnknownPropertyName	The format of the parameter *descriptiveData* is not recognized.	
BadPropertyValue	The value for the parameter *version* or *descriptiveData* is not valid.	

10.1.3.1.1 Examples of Using the Output from Locate

As an example of a call to Locate, assume that the output componentURL has the value http://www.sentillion.com/duke/contextManager/, and that the output componentParameters has the value id=987654321&domain=duke.edu.

A subsequent call to the context manager's JoinCommonContext method using an HTTP GET message would use a URL of the form:

```
http://www.sentillion.com/duke/contextManager/?
id=987654321&domain=duke.edu&interface=ContextManager&method=JoinCommonContext& … other parameters …
```

Similarly, a call to this method using an HTTP POST message would use the URL:

```
http://www.sentillion.com/duke/contextManager/
```

while the body of the message would include:

```
id=987654321&domain=duke.edu&interface=ContextManager&method=JoinCommonContext& … other parameters …
```

10.1.3.1.2 Usage of the Parameter descriptiveData

Per Section 3.2.5, Special Support for Applications Run In Citrix or Windows Terminal Server Remote Sessions, the parameter *descriptiveData* may be used to provide the context management registry with data that can be used to locate the context manager for the remote Citrix or WTS client desktop at which the application will be displayed.

Several formats are possible for this parameter, where each format is represented as a name/value pair:

clientHostName:*name* where *name* is the network host name of the client machine[7].

clientIPAddress:*address* where *address* is the IP address of the client machine.

`citrixSessionId:`*`id`* where *id* is the Citrix session identifier.

`wtsSessionId:`*`id`* where *id* is the WTS session identifier.

Only one format may be used for *descriptiveData* at a time. The name and value for each format are not case sensitive. The context management registry shall accept any of these formats.

10.2 CMA INTERFACES

The use of the interfaces described below, the meaning of the methods defined for each interface, and the interpretation of each method's input and output parameters are as specified in the CMA and are not further described below. Further, the exceptions that may be raised by each method are also specified in the CMA. However, additional method parameters that are unique to the web technology mapping are described below.

10.2.1 AuthenticationRepository

10.2.1.1 Connect

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"AuthenticationRepository"

[7] This name is not necessarily the same as the Citrix client name.

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
method	string	“Connect”
applicationName	string	
HTTP Reply Message		
bindingCoupon	long	

10.2.1.2 Disconnect

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	“AuthenticationRepository”
method	string	“Disconnect”
bindingCoupon	long	
HTTP Reply Message		
empty		

10.2.1.3 SetAuthenticationData

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	“AuthenticationRepository”
method	string	“SetAuthenticationData”
coupon	long	
logonName	string	
dataFormat	string	
userData	string	
appSignature	string	
HTTP Reply Message		
empty		

10.2.1.4 DeleteAuthenticationData

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	“AuthenticationRepository”
method	string	“DeleteAuthenticationData”
coupon	long	
logonName	string	
dataFormat	string	
appSignature	string	
HTTP Reply Message		
empty		

10.2.1.5 GetAuthenticationData

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	“AuthenticationRepository”
method	string	“GetAuthenticationData”
coupon	long	
logonName	string	
dataFormat	string	
appSignature	string	
HTTP Reply Message		
userData	string	
repositorySignature	string	

10.2.2 ContextAgent

10.2.2.1 ContextChangesPending

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextAgent"
method	string	"ContextChangesPending"
agentCoupon	long	
contextManager	string	URL for the context manager that is informing the context agent of the pending context changes.
itemNames	string[]	
itemValues	string[]	
contextCoupon	long	
managerSignature	string	
HTTP Reply Message (*)		
agentCoupon	long	
itemNames	string[]	
itemValues	string[]	
contextCoupon	long	
agentSignature	string	
decision	string	
reason	string	

* Note that per Chapter 5, Context Action Continuations, and Chapter, 7, Representing CMA Methods as HTTP Messages, this message may be encoded in the header of an HTTP redirect message.

10.2.2.2 Ping

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextAgent"
method	string	"Ping"
HTTP Reply Message		
empty		

10.2.3 ContextData

10.2.3.1 GetItemNames

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextData"
method	string	"GetItemNames"
contextCoupon	long	
HTTP Reply Message		
names	string[]	

10.2.3.2 SetItemValues

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextData"
method	string	"SetItemValues"
participantCoupon	long	
itemNames	string[]	
itemValues	array of various types	An item's value type is a function of the item's data definition, as defined in the CMA data definition specifications.
contextCoupon	long	
HTTP Reply Message		
empty		

10.2.3.3 GetItemValues

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextData"
method	string	"GetItemValues"
itemNames	string[]	
onlyChanges	boolean	
contextCoupon	long	
HTTP Reply Message		
itemValues	array of various types	An item's value type is a function of the item's data definition, as defined in the CMA data definition specifications.

10.2.3.4 DeleteItems

This method is deprecated in the CMA, and is not supported in the web technology mapping.

10.2.4 ContextFilter

10.2.4.1 SetSubjectsOfInterest

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextFilter"
method	string	"SetSubjectsOfInterest"
participantCoupon	long	
subjectNames	string[]	
HTTP Reply Message		
names	string[]	

10.2.4.2 GetSubjectsOfInterest

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextFilter"
method	string	"GetSubjectsOfInterest"
participantCoupon	long	
HTTP Reply Message		
subjectNames	string[]	

10.2.4.3 ClearFilter

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextFilter"
method	string	"ClearFilter"
participantCoupon	long	
HTTP Reply Message		
empty		

10.2.5 ContextManager

10.2.5.1 GetMostRecentContextCoupon

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextManager"
method	string	"GetMostRecentContextCoupon"
HTTP Reply Message		
contextCoupon	long	

10.2.5.2 JoinCommonContext

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextManager"
method	string	"JoinCommonContext"
applicationName	string	
contextParticipant	string	Participant's URL.
survey	boolean	
wait	boolean	
HTTP Reply Message		
participantCoupon	long	

10.2.5.3 LeaveCommonContext

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextManager"
method	string	"LeaveCommonContext"
participantCoupon	long	
HTTP Reply Message		
empty		

10.2.5.4 StartContextChanges

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextManager"
method	string	"StartContextChanges"
participantCoupon	long	
HTTP Reply Message		
contextCoupon	long	

10.2.5.5 EndContextChanges

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextManager"
method	string	"EndContextChanges"
contextCoupon	long	
HTTP Reply Message		
noContinue	boolean	
responses	string[]	

10.2.5.6 UndoContextChanges

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextManager"
method	string	"UndoContextChanges"
contextCoupon	long	
HTTP Reply Message		
empty		

10.2.5.7 PublishChangesDecision

The output *listenerURLs* is a web technology-specific parameter. See Section 4.1.1.2, Notifier Responsibilities.

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextManager"
method	string	"PublishChangesDecision"
contextCoupon	long	
decision	string	
HTTP Reply Message		
listenerURLs	string[]	The URLs for listeners that need to be notified about the conclusion of a context change transaction. Empty if the value of the input *decision* is not "accept". See Section 4.1.1.2, Notifier Responsibilities

10.2.5.8 SuspendParticipation

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextManager"
method	string	"SuspendParticipation"
participantCoupon	long	
HTTP Reply Message		
empty		

10.2.5.9 ResumeParticipation

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextManager"
method	string	"ResumeParticipation"
participantCoupon	long	
wait	boolean	
HTTP Reply Message		
empty		

10.2.6 ContextParticipant

10.2.6.1 ContextChangesPending

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextParticipant"
method	string	"ContextChangesPending"
contextCoupon	long	
HTTP Reply Message		
decision	string	
reason	string	

10.2.6.2 ContextChangesAccepted

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextParticipant"
method	string	"ContextChangesAccepted"
contextCoupon	long	
HTTP Reply Message		
empty		

10.2.6.3 ContextChangesCanceled

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextParticipant"
method	string	"ContextChangesCanceled"
contextCoupon	long	
HTTP Reply Message		
empty		

10.2.6.4 CommonContextTerminated

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextParticipant"
method	string	"CommonContextTerminated"
HTTP Reply Message		
empty		

10.2.6.5 Ping

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextParticipant"
method	string	"Ping"
HTTP Reply Message		
empty		

10.2.7 ContextSession

10.2.7.1 Create

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextSession"
method	string	"Create"
HTTP Reply Message		
newContextManager	string	URL of the context manager that represents the newly created context session. If the context manager that implements this method is called via an HTTPS URL, then this output will also be an HTTPS URL. Otherwise, this output will be a normal HTTP URL.

10.2.7.2 Activate

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextSession"
method	string	"Activate"
participantCoupon	long	
cmToActivate	string	
nonce	string	
AppSignature	string	
HTTP Reply Message		
empty		

10.2.8 ContextAction

10.2.8.1 Perform

The inputs *cpCallBackURL* and *cpErrorURL* is a web technology-specific parameter. See Section 5.2, Policy With Continuations.

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ContextAction"
method	string	"Perform"
cpCallBackURL	string	URL that the context participant should be redirected to once the context action is complete.
cpErrorURL	string	URL that the context participant should be redirected if the context manager or context agent experiences an error while performing the context action.
partcipantCoupon	long	
inputNames	string[]	
inputValues	string[]	
appSignature	string	
HTTP Reply Message *		
actionCoupon	long	
outputNames	string[]	
outputValues	string[]	
managerSignature	string	

* Note that per Chapter 5, Context Action Continuations, and Chapter 7, Representing CMA Methods as HTTP Messages, this reply may be encoded in the header of an HTTP redirect message.

10.2.9 ImplementationInformation

10.2.9.1 ComponentName

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	“ImplementationInformation”
method	string	“ComponentName”
HTTP Reply Message		
componentName	string	

10.2.9.2 RevMajorNum

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	“ImplementationInformation”
method	string	“RevMajorNum”
HTTP Reply Message		
revMajorNum	string	

10.2.9.3 RevMinorNum

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	“ImplementationInformation”
method	string	“RevMinorNum”
HTTP Reply Message		
revMinorNum	string	

10.2.9.4 PartNumber

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	“ImplementationInformation”
method	string	“PartNumber”
HTTP Reply Message		
partNumber	string	

10.2.9.5 Manufacturer

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	“ImplementationInformation”
method	string	“Manufacturer”
HTTP Reply Message		
manufacturer	string	

10.2.9.6 TargetOS

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ImplementationInformation"
method	string	"TargetOS"
HTTP Reply Message		
targetOS	string	

10.2.9.7 TargetOSRev

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ImplementationInformation"
method	string	"TargetOSRev"
HTTP Reply Message		
targetOSRev	string	

10.2.9.8 WhenInstalled

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"ImplementationInformation"
method	string	"WhenInstalled"
HTTP Reply Message		
whenInstalled	string	

10.2.10 SecuringBinding

10.2.10.1 InitializeBinding

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"SecureBinding"
method	string	"InitializeBinding"
bindeeCoupon	long	
propertyNames	string[]	
propertyValues	array of various types	A property's type depends upon the property's definition. See Chapter 6, Security.
HTTP Reply Message		
binderPublicKey	string	
mac	string	

10.2.10.2 FinalizeBinding

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"SecureBinding"
method	string	"FinalizeBinding"
bindeeCoupon	long	
bindeePublicKey	string	
mac	string	
HTTP Reply Message		
privileges	string[]	

10.2.11 SecureContextData

10.2.11.1 GetItemNames

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"SecureContextData"
method	string	"GetItemNames"
contextCoupon	long	
HTTP Reply Message		
names	string[]	

10.2.11.2 SetItemValues

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"SecureContextData"
method	string	"SetItemValues"
participantCoupon	long	
itemNames	string[]	
itemValues	array of various types	An item's value type is a function of the item's data definition, as defined in the CMA data definition specifications.
contextCoupon	long	
appSignature	string	
HTTP Reply Message		
empty		

10.2.11.3 GetItemValues

The input *paricipantCoupon* is defined in the CMA as a parameter whose use is optional depending upon the implementation technology. This parameter is required in the web technology mapping of the CMA.

HTTP Request Message		
Argument Name	**Data Type**	**Comment**
interface	string	"SecureContextData"
method	string	"GetItemValues"
participantCoupon	long	
itemNames	string[]	
onlyChanges	boolean	
contextCoupon	long	
appSignature	string	
HTTP Reply Message		
itemValues	array of various types	An item's value type is a function of the item's data definition, as defined in the CMA data definition specifications.
managerSignature	string	

11 Appendix: Web Use Cases

The following use cases are representative of the ways in which web applications are used. These use cases form the basis for a set of canonical situations that a web-based context management solution needs to address:

11.1 USE CASE 1

The user has a single web browser and is interacting with an application. The user establishes the context for the application, for example, by selecting the patient of interest. The user then opens a URL presented by the application to get to a second application. The second application is served by a different web server than the first application. The second application automatically displays data for the patient selected via the first application.

11.2 USE CASE 2

The user has a single web browser and is interacting with an application. The user establishes the context for the application, for example, by selecting the patient of interest. The user then opens a URL presented by the application to get to a second application. The second application is not context enabled. However, the second application contains a URL to a third application that is context enabled. The third application automatically displays data for the patient selected via the first application.

11.3 USE CASE 3

The user has a web browser and pages backwards, such that an application formerly in view is once again in view. However, the user has changed the context since the application was in view. The application may (a) automatically display data representing the current context or (b) may attempt to set the current context such that it reflects the application's internal state. The application may allow the user to decide how it should behave.

11.4 USE CASE 4

The user has a single web browser and is interacting with an application, with which the user has established a context. The user then opens a second web browser and opens another application. The second application is served by a different web server than the first application. The second application automatically displays data for the context as established by the user via the first application.

11.5 USE CASE 5

The user has a single web browser and is interacting with an application, with which the user has established a context. The user then performs a gesture on a web page being viewed within the browser which causes a second web browser to be automatically opened to another application. The second application is served by a different web server than the first application. The second application automatically displays data for the context as established by the user via the first application.

11.6 USE CASE 6

The user has a single web browser and is interacting with an application whose displayed page is composed of multiple frames, each served by a different web server. The user changes the context via one of the frames. All of the frames automatically display data for the new context.

11.7 USE CASE 7

The user has a single web browser and is interacting with an application. The user establishes the context for the application, for example, by selecting the patient of interest. The user enters data into the application but does not save this work before following a URL to another application. In the new application the user changes the context, for example by selecting a new patient. The first application only conditionally accepts the context change. The second application presents a dialog to the user asking whether or not they want to apply the context change, or break the second application's link with the context.

11.8 USE CASE 8

The user has two web browsers open, and each browser is presenting an application such that both applications are linked to the same common context. The user then attempts to change the context using the first application, for example, by selecting a different patient. For some underlying technical reason, the first application is unable to inform the second application about the pending context change (i.e., in the parlance of the CMA, the second application is "busy.") The second application presents a dialog to the user asking whether or not they want to cancel the context change, or break the second application's link with the context.