The W3C multimodal architecture and interfaces standard

1 author:

Deborah A. Dahl
Conversational Technologies
87 PUBLICATIONS   **964** CITATIONS

The W3C Multimodal Architecture and Interfaces Standard

Abstract: This paper describes the World Wide Web Consortium's (W3C) Multimodal Architecture and Interfaces (MMI Architecture) standard, an architecture and communications protocol that enables a wide variety of independent modalities to be integrated into multimodal applications. By encapsulating the functionalities of modality components and requiring all control information to go through the Interaction Manager, the MMI Architecture simplifies integrating components from multiple sources.

1. Overview

Computer-human interaction originally was based entirely on computer-friendly and often difficult to use commands, which had to be mastered by anyone who wanted to use a computer. More natural interaction became possible with the advent of graphical user interfaces with WYSIWYG displays, which allowed users to use pointing gestures in addition to typed commands. This is still the most common computer-human interaction paradigm. But even in the earliest days of computing [1], it was recognized that computer-human interaction would be improved if the computer could interact with humans in the ways that humans are used to using when they interact with each other. People naturally interact with each other using a combination of spoken language, written language, gestures, and touch, but making natural interaction possible between humans and computers has proven to be much more difficult than expected. However, as technologies such as speech recognition, natural language understanding and gesture recognition become increasingly capable, it is becoming much more feasible to build applications that support natural multimodal interaction through a combination of human-friendly modalities.

2. Why a Standard Architecture?

While the component technologies of multimodal applications are very powerful, they can also be very sophisticated and complex. This makes it very important to have ways of orchestrating their interaction in systems that do not require developers to master each individual modality technology. Consider how difficult it would be to develop applications if each developer, or even each organization, were required to learn such technologies as speech recognition, handwriting recognition, face recognition and natural language understanding, all of which might be used in a single application. In addition, with a standard architecture, experts in specific technologies can develop standalone components that provide specific services such as speech recognition or face recognition without having to also master multimodal application development.

Not only is the set of possible modalities large, it is also continually increasing. Displays, microphones, speakers, cameras, GPS, and accelerometers are nearly always included with current smartphones and tablets. In addition to these common input modalities, the number of hardware additions that can be plugged into devices is also continuing to increase. Medical sensors for blood pressure, heart rate and blood glucose levels are available, as well as environmental sensors like temperature. Defining modality-specific ways of coupling together systems that incorporate several modalities is clearing impractical. All this variety again points to a strong need for a standard way of putting components together.

There is a large research literature on multimodal interaction, including technical work on integrating modalities, for example, [2,3] as well as the human factors of multimodal interaction [4]. In addition, there is a very large literature on individual modalities. However, there have been very few attempts to define an open, non-proprietary, multimodal architecture which can be used by any organization or developer. One example is the Galaxy Communicator project [3,5]; however that system was heavily focused on speech interaction rather than multimodal interaction.

In order to provide an open architecture for multimodal development, and to make the process of developing multimodal applications easier, the World Wide Web Consortium (W3C) has defined a standard for multimodal applications -- the Multimodal Architecture and Interfaces specification [6], based on the work of the W3C Multimodal Interaction Working Group. The goal of the W3C Multimodal

Interaction Architecture (MMI Architecture) is to provide a way to coordinate multiple modalities in a standard way, with standard methods of communication. This enables multimodal applications to be built by developers who are not necessarily expert in every modality that the application uses.

A second advantage of a standard multimodal architecture is that it promotes the development of an ecosystem where companies with expertise in specific modalities can develop reusable components for modalities in which they are knowledgeable. These third-party components can subsequently be plugged into full applications as components. Companies with multimodal application development expertise can then take specific modality components as black boxes and assemble them into applications.

It is important for a standard architecture to be as flexible as possible, in order to accommodate new modalities and new ways of interacting. To that end, important requirements include support for local as well as distributed applications, multi-device applications, and multi-user applications.

This architecture is independent of specific application domains, but is particularly suited for distributed applications that incorporate many modalities. As the number of devices involved increases, it becomes increasingly important to have a structured means of orchestrating their operation so that the overall system is maintainable, extensible, and vendor-independent.

3. Related Work

While this work is focused on the overall integration of components into multimodal systems, there is a great deal of complementary work on such topics as platforms and scripting languages that support multimodal applications.

3.1.　　Platforms

Because multimodal interaction can take place in many different hardware and software environments, it is also important for a multimodal architecture to be platform-neutral. The most obvious platform is the World Wide Web, a distributed platform where a browser runs locally on a computer or mobile device and where the application interacts with one or more remote web servers. However, the web is not by any means the only possible platform for multimodal interaction. Standard desktop or laptop computer applications can also benefit

from multimodal applications. Another important platform is a home network, where devices such as televisions, appliances, lights, and temperature control interact over a Local Area Network. For example, the Digital Living Network Alliance (www.dlna.org) has defined guidelines for using media devices in home networks. Another emerging multimodal platform is the automobile, where voice control and touchscreens are available. In addition, the automobile is also emerging as a multi-device platform. The hands and eyes-busy nature of driving make the car a natural place for multimodal interaction. Although originally automobile systems were all self-contained, in recent years it has become possible for user devices to connect to in-car systems, most notably the audio system, making the car also a multi-device platform.

3.2.   Scripting and Interaction Definition Languages

The W3C Multimodal Architecture does not specify the system's behavior with respect to the user, but rather defines system-internal behavior with respect to the underlying communication among components. There are a number of scripting languages which have been designed to define system behavior regarding the user, and as such are very complementary to the MMI Architecture. For example, SCXML [7] is a high-level language for defining state-based systems that implement specific applications, which could include an interaction between a user and an intelligent agent. Similarly, for voice applications, VoiceXML [8] is a widely-adopted tool for specifying the flow of dialog in voice interactions. Behavior Markup Language (BML) [9] is used to define the behaviors of intelligent agents and the related Functional Markup Language (FML) [10] is used to define what an agent wants to achieve, in terms of actions, goals, and plans. Another markup language, Perception Markup Language (PML) [11] has been defined for representing perceived non-verbal behaviors. Other markup languages include MIML (Multimodal Interaction Markup Language) [12], a dialog description language for dialogs between humans and interactive systems.
Depending on the interaction and modalities of interest, these languages or a combination of these languages could be used to define the behavior of an MMI Architecture Interaction Manager (see Section 5.2. below for a discussion of the Interaction Manager).

4. Components of a Multimodal System

At a high level, multimodal systems are typically based on architectures that include the following components:

1) Modality components for processing any of an open-ended set of possible specific input modalities such as speech, GUI input (pointing device), typed input, camera input and electronic ink

2) Fusion components for integrating inputs from multiple modalities that represent a unified user intent; for example speech combined with a pointing gesture

3) Dialog or interaction managers that
   a) use the fused multimodal inputs to determine the user's intent
   b) take into account the user's intent, the interaction context, the task to be performed and any relevant system information, determine the next step in the interaction

4) Fission components that determine how to best present the interaction manager's response to the user – for example, whether to speak, display typed input, or display graphics

5) Presentation components that present the system's output to the user using a combination of graphics, text, audio, or other forms of output as appropriate to the application

5. Components of the Standard

The W3C Multimodal Architecture focuses on Modality Components (MC's) for input and presentation and on the Interaction Manager (IM) for coordinating the MC's and user interaction. Fusion and fission components can be conceptualized as independent MC's or as part of the IM. The MC's and the IM are together referred to as *constituents*.

Encapsulation is a central principle of the MMI Architecture. That is, the internal workings of each constituent are not visible to the other constituents. All communication is carried out through standard messages, called *lifecycle events*, which may carry application- or modality-specific information as a payload. In addition, messages are only sent between the IM and MC's, never directly between MC's. This principle makes it possible for third-party modality

components to be developed independently of any one system. Fig. 1 shows an example of the W3C MMI Architecture with an IM and multiple MC's. It is unlikely that any one system would include all of the MC's shown, but they are included to show the variety of possible MC's in a multimodal system.
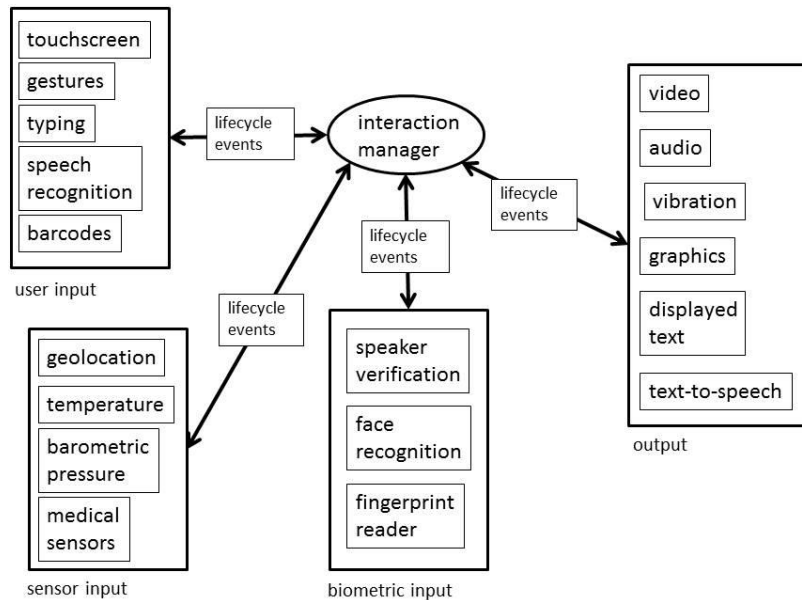
Fig. 1 the IM and MC's in the W3C Multimodal Architecture

5.1. Modality Components

MC's commonly encapsulate modality-specific input capabilities such as speech recognition or handwriting recognition, but they actually can have broader functions. For example, MC's can encapsulate multiple functionalities, but present themselves to the IM as a single component. This is called a *complex modality component* (discussed in more detail below), and is useful for tightly coupled functionalities such as coordinating text to speech output with an avatar's facial expressions and lip movements. In many cases an input can also require processing by multiple MC's before the user's intent is fully determined. For example, a speech recognition MC could send its results back to the IM, which then might in turn send the speech results to a natural language processing engine. If, as is often the case, speech recognition and natural language processing are tightly integrated, the speech recognition and natural language functionalities can be combined into a complex MC which communicates with the IM as a single component.

MC's in the MMI Architecture can also encapsulate broader functionalities than intentional user inputs such as speech and GUI actions. An example is the biometric MC's in Fig. 1. Identifying users through biometric properties for the purposes of security and/or personalization is a natural part of many multimodal applications. For this reason, it makes sense to include biometrics such as speaker verification or face recognition components in multimodal systems as MC's. Sensor inputs such as medical and environmental sensors and geolocation devices are also possible components of multimodal applications, and fit well into the MMI Architecture as MC's. Finally, devices like TV's and home appliances can also function as MC's in multimodal applications. Interesting applications such as an integrated "home interface" could be supported this way, enabling users to do things like ask for the current temperature, have it displayed on the TV they're watching, and then seamlessly ask the home interface to raise or lower the temperature. A similar application would be a unified "car interface" that would let users control the audio system, interact with the GPS, and find out how long it will be before they need to buy gas by interacting with a single IM.

5.2. Interaction Manager

The IM controls the interaction between the user and the system by taking into account the user's input, the task to be performed, the context of the interaction, and other data as needed. The MMI Architecture does not specify any particular standard for implementing the IM, but an example of a standard language that would be suitable for an IM is State Chart Extensible Markup Language (SCXML) [7]. Examples of other options for implementing an IM are Javascript for an IM running in a web browser or a suitable server-side programming language for server-based IM's.

5.3. Runtime Framework

The Runtime Framework refers to all the infrastructure services that are necessary for successful execution of a multimodal application. For example, these include starting modality components, handling communication, and logging. The standard leaves the specifics of these functions to be defined in a platform-specific way, although the standard discusses an *Event Transport Layer* which is part of the Runtime Framework. The Event Transport Layer includes one or more

transport mechanisms linking the IM to the various MC's. Various protocols can be used for event transport, but the specification does place two requirements on the Event Transport Layer -- the events are required to be delivered reliably and they are required to be delivered in the order they were generated. Appendix F of the specification [6] describes an Event Transport Layer based on HTTP in detail. The Runtime Framework is also where functions such as discovery and registration of modality components occur, as discussed in [13].

5.4.   Standard Messages: The Lifecycle Events

The messages defined in the MMI Architecture are referred to as lifecycle events. Most of the lifecycle events come in request/response pairs. Most commonly the requests are sent from the IM to an MC and the responses are returned from the MC to the IM, but the request for a new context is sent by an MC and a request for status can be sent in either direction.

The events generally fall into two categories. The first is generic control events, such as starting and stopping components and the corresponding acknowledgement of these requests. This category includes the events NewContextRequest/Response, StartRequest/Response, PrepareRequest/Response, CancelRequest/Response, PauseRequest/Response, Resume Request/Response, and StatusRequest/Response.

The second category includes modality- or application-specific events that are used to set parameters in a component and report processing results. These are the ExtensionNotification and DoneNotification events. The ExtensionNotification and DoneNotification events are the only events that convey user inputs to the IM. Clearly, the semantics of user inputs can be very complex, especially in the case of language input modalities such as speech or text input. In order to provide a standard representation of the semantics of user inputs, a companion standard to the MMI Architecture has been developed, Extensible Multimodal Annotation (EMMA) [14]. EMMA defines a modality-independent framework for representing the application-specific semantics of user inputs as well as additional metadata about the inputs, and is discussed in more detail in Section 8. below. Table 1 summarizes the lifecycle events and their functions.

Name	Function
Paired messages initiated by an MC	

NewContextRequest	Request to the IM to initiate a new context of interaction
NewContextResponse	Response from the IM with a new context ID
Paired messages initiated by an IM	
PrepareRequest	Request to the MC to prepare to run, possibly including a URL pointing to markup, such as a grammar or VoiceXML document that will be required when the StartRequest is issued. This allows the MC to fetch and compile markup, if necessary.
PrepareResponse	Response from the MC to a PrepareRequest. The MC is not required to take any action other than acknowledging that it has received the PrepareRequest, although it would be desirable to send back error information if there are problems preparing.
StartRequest	Request to the MC to start processing.
StartResponse	Response from the MC to the IM to acknowledge the StartRequest. The MC is required to send this event in response to a StartRequest.
CancelRequest	The CancelRequest message is sent to stop processing in the MC. In this case, the MC must stop processing and return a CancelResponse.
CancelResponse	Response from the MC to the IM to acknowledge the CancelRequest.
PauseRequest	A PauseRequest is a request to the MC to pause processing.
PauseResponse	MC's return a PauseResponse once they have paused, or if they are unable to pause, the message is sent when they determine that they will be unable to pause.
ResumeRequest	ResumeRequest is a request to resume processing that was paused by a previous PauseRequest. It can only be sent to a currently paused context.
ResumeResponse	MC's return a ResumeResponse after receiving a ResumeRequest. The Status field is "Success" if the implementation has succeeded in resuming processing and "Failure" otherwise.
ClearContextRequest	A ClearContextRequest may be sent to an MC to indicate that the specified context is no longer active and that any resources associated with it may be freed. MC's are not required to take any particular action in response to this command, but are required to return a ClearContextResponse.
ClearContextResponse	A response from the MC acknowledging the ClearContextRequest. Note that once the IM has sent a ClearContextRequest to an MC, no more events can be sent for that context.
Paired messages that can be sent in either direction	
StatusRequest	The StatusRequest message and the corresponding StatusResponse provide keep-alive functionality. This message can be sent by either the IM or an MC.
StatusResponse	Response to the StatusRequest message. If the request specifies a context which is unknown to the MC, the MC's behavior is undefined.
Unpaired Messages	
DoneNotification	Sent from an MC to the IM to indicate completion of a

	task; contains any data from the MC action, such as an EMMA message representing user input. (EMMA is described in Section 8.
ExtensionNotification	Sent in either direction to convey application-specific data, typically sent from the MC to convey user inputs in EMMA format or report intermediate statuses. Typically sent from the IM to set modality-specific parameters such as speech recognition grammars, timeouts or confidence thresholds. This is the point of extensibility for the lifecycle events. If no other lifecycle event is suitable for a message between the IM and MC's, this message is used.

Table 1 Lifecycle events in the MMI Architecture

Although lifecycle events do not have to be represented in a specific format, the specification describes an XML [15] format. If the events are represented in XML, the specification defines a required XML syntax in Appendix C.

5.5.   Fields of the Lifecycle Events

The lifecycle events make use of a set of common fields used by most events, as well as event-specific fields that are required by certain events.

Fields Common to Most Events

Context

A URI that is unique for the lifetime of the system. The Context field identifies the interaction. All events relating to a given interaction use the same Context URI. Events containing a different Context URI are interpreted as part of other, unrelated, interactions. Unlike all other events, the NewContextRequest event does not contain a Context field, since it is sent before the context is initiated. The Context field is optional for the StatusRequest/StatusResponse event. If it is absent the request is interpreted as a request for the status of the underlying server rather than a request for the status of a particular context.

Source

A URI that is the address of the sender of the event. The recipient of the event has to be able to send an event back to the sender by using this value as the "Target" field of a message.

Target

The Target is a URI that represents the address to which the event will be delivered.

RequestID

The requestID is a unique identifier for a Request/Response pair. Most lifecycle events come in Request/Response pairs that share a common RequestID. For any such pair, the RequestID in the Response event is required to match the RequestID in the request event. The RequestID for such a pair will be unique within the given Context.

Status

An enumeration of "Success" and "Failure". The Response event of a Request/Response pair uses this field to report whether it succeeded in carrying out the request. Details are provided in the StatusInfo field (below).

StatusInfo

This field is available in the Response events of a Request/Response pair for use in providing additional, application-specific, status information. The StatusInfo field is used for providing detailed error information.

Data

This field contains arbitrary data. When a DoneNotification or ExtensionNotification event containing user inputs is sent from an MC to the IM, the data field contains the user input, represented in EMMA if appropriate.

Event-specific Fields

Content

The PrepareRequest and StartRequest events include a Content field which contains inline content which the MC will run. An example of inline content would be text marked up with Speech Synthesis Markup Language (SSML) [16] for rendering by a text-to-speech engine.

ContentURL

The PrepareRequest and StartRequest events include a ContentURL field which contains a pointer to content which the MC will run. An example of content that would be naturally referred to with a ContentURL would be a relatively large

document that would be unwieldy if included inline in the lifecycle event, such as a large speech recognition grammar.

Note that Content and ContentURL are mutually exclusive. If both Content and ContentURL are empty the component reverts to its default behavior. This is appropriate for a non-scriptable component, such as a camera. Sending a camera a StartRequest message would result in the camera taking a picture, no scripting would be necessary.

Status

The StatusResponse message includes a Status field, which is different from the Status field common to all Response messages. While the common Status field is used by an MC to report whether it succeeded in carrying out a Request, and can take on the values of either "Success" or "Failure", the Status field of StatusResponse refers to the status of the Context or of the constituent. This field can take on the values "Alive" or "Dead". The meaning of these values depends on whether the "context" parameter is present. Specifically:

1) If the Context field is present
    a) If the specified context is still active and able to handle additional lifecycle events, the sender of the StatusResponse message sets the value of this field to "Alive".
    b) If the context has terminated or is otherwise unable to process new lifecycle events, the sender sets the Status to "Dead".
2) If the Context field is not present, the Status refers to the underlying server.
    a) If the sender is able to create new contexts, it sets the Status to "Alive",
    b) Otherwise, the Status is set to "Dead".

6. Simple, Complex, and Nested Components

Although all MC's are black boxes from the point of view of the rest of the system, based on their internal organization they can be classified into one of three categories; simple, complex or nested. A simple modality component provides a single functionality and does not have any subcomponents that communicate among themselves. For example, audio capture, handwriting recognition, speech recognition, or face identification from an image would all be examples of simple modality components.

A complex MC includes functionality from one or more simple components. For example an HTML page might include functionality for capturing typed input as well as spoken input. Implemented as a complex MC, the HTML page would send messages to the IM for each user input, whether spoken or written. The IM would send lifecycle events to the HTML page appropriate for either spoken or written input. EMMA results sent back to the IM would contain information about the medium and mode of the input, so that the IM can distinguish between spoken and written inputs.

A nested MC is similar to a complex MC in that it includes internal MC's. In addition, a nested MC essentially contains its own internal IM that manages interaction among the internal MC's. It can be thought of as an instantiation of the MMI Architecture that presents itself as an MC to a higher-level IM. An example of a nested MC would be a VoiceXML component [8,17] used to provide voice services such as speech recognition and text to speech while also handling dialog management for form-filling tasks with its internal IM. These three types of MC's are discussed in detail in Appendix H of the specification.

7. Defining Modality Components

Because the standard MMI Architecture is modality-independent, many details of how a particular modality component would be implemented are not defined by the architecture.

This raises the question of how developers can know how to use third-party components. Some components, such as speech recognition, can have fairly complex API's which are very modality-specific. If there is an existing API for a type of component, the developer can use that API and map the modality-specific API calls to the MMI Architecture lifecycle events[1]. In some cases there are also standard API's that can be used. For example, there are W3C standard API's for geolocation [18] and camera [19] information. There is no standard API for speech recognition, although several have been proposed, for example, [20-23] and more recently [24]. If there is not yet a standard API for the modality

---

[1] For example, the StartRequest event might be mapped to a "startListening" method used by a modality-specific API.

component, it becomes especially important for the developer to describe the API for other users of the component.

Most notably, the need for description applies to the ExtensionNotification event where modality-specific and application-specific information is conveyed. Suggestions for the information to be included in defining modality components is discussed in a W3C Note on "Best practices for creating MMI Modality Components" [25].

This note includes the following eight guidelines for defining a modality component for use within the MMI Architecture.

Guideline 1: Each modality component must implement all of the MMI life-cycle events. This is a basic requirement for the component to be MMI Architecture-compliant. Note that not all events make sense for all components. For example, a speech recognition component may not be able to pause once it has started processing. Nevertheless an MMI Architecture-compliant speech recognition MC must be able to respond to a PauseRequest event with a PauseResponse event. In this case, the PauseResponse event might contain a status message that informs the IM that it cannot pause.

Guideline 2: The definition of the component must identify other functions of the modality component that are relevant to the interaction manager. That includes any functions that can be controlled externally, such as starting, stopping, pausing and setting parameters. As with any API, functionality that is not exposed externally does not need to be described. Functionality that requires events outside of the standard events must be defined with ExtensionNotification events. Most commonly, this additional functionality involves ExtensionNotification events that are sent from the IM to set parameters within the MC and ExtensionNotification events that are sent from the MC to the IM to return results or report intermediate statuses of the processing.

Guideline 3: If the component handles media for either capture or processing, acceptable media formats must be specified. For example, audio formats that a speech recognizer accepts must be specified. A handwriting recognition component could specify, for example, that the input it expects must be in the form of the InkML standard [26].

Guideline 4: Specify protocols for use between the component and the IM (e.g., SIP or HTTP) in the Event Transport Layer. The architecture allows events to be

transmitted between the IM and the MC using any protocol that guarantees reliable delivery in the order the events are sent. Consequently, in defining a specific modality component, the developer must specify which protocols can be used with the component. HTTP is one common example, although other protocols are possible. Alternatives include Session Initiation Protocol (SIP), Server-sent Events [27], Web Sockets [28] or local sockets over a LAN, for example.

Guideline 5: Specify supported human languages, e.g., English, German, Chinese and locale, if relevant. Some modality components are language-specific. These include speech recognition, text to speech, handwriting recognition and natural language processing. Other components, such as video capture, are not language-specific. If the component is language-specific, the supported languages must be listed.

Guideline 6: Specify supporting languages required by the component, if any. Some components make use of markup languages. For example, a TTS component might use SSML to describe the pronunciation of text sent to the component. Similarly, grammars used by a speech recognition component might be defined using the Speech Recognition Grammar Specification (SRGS) [29] and the Semantic Interpretation for Speech Recognition specification [30].

Guideline 7: MC's sending data to the IM must use the EMMA format where appropriate. The EMMA format was designed to represent user inputs, particularly user inputs with complex semantics. MC's that generate representations with complex semantics include speech recognition, handwriting recognition, or natural language processing, for example. Simpler inputs, such as mouse clicks or selection of a choice with radio buttons can be represented with EMMA, (especially in applications where spoken and mouse input might be integrated) but EMMA is not as important for simple inputs.

Guideline 8: Specify error codes and their meanings. Error messages specific to the component should be documented as well. Standard errors, such as HTTP errors or XML errors, do not need to be specified in the component definition. Examples of descriptions of three components (graphical and speech recognition MC's and an SCXML IM) can be found in [31].

8. Representing and Communicating the Semantics of User Input (EMMA)

As briefly mentioned in Section 5.4. above, the EMMA specification was developed to represent the semantics of user inputs. EMMA defines a modality-independent framework for representing the application-specific semantics of user inputs as well as additional metadata about the inputs. Because EMMA is modality-independent, the meaning of an input is represented in the same way regardless of the original input modality. Fig. 2 shows partial EMMA representations for the semantics of "I want to go from Boston to Denver on November 28, 2012". The semantics is the same, whether the input was spoken, entered with a GUI interface, or handwritten, so the application-specific semantics contained in the <emma:interpretation> element is the same. Only the medium and the mode (as highlighted) differ to reflect the different modalities of input.

```
Speech
<emma:interpretation medium="acoustic" mode="voice" id="int1">
    <origin>Boston</origin>
    <destination>Denver</destination>
    <date>11282012</date>
 </emma:interpretation>
GUI
<emma:interpretation medium="tactile" mode="gui" id="int1">
    <origin>Boston</origin>
    <destination>Denver</destination>
    <date>112812</date>
 </emma:interpretation>
Handwriting
<emma:interpretation medium="tactile" mode="ink" id="int1">
    <origin>Boston</origin>
    <destination>Denver</destination>
    <date>11282012</date>
 </emma:interpretation>
```

Fig. 2 Partial EMMA representations for speech, GUI, and typed input for "I want to go from Boston to Denver on November 28, 2012

In addition, metadata such as confidence, timestamps and alternative interpretations is also represented in the same way, regardless of the input modality. When an MC represents its results in EMMA, the DoneNotification or the ExtensionNotification lifecycle event Data fields contain the EMMA results

which are sent to the IM. An example of a DoneNotification event with an EMMA payload can be seen in Fig. 4.

9. Example – A Personal Assistant Application

Fig. 3 shows the IM and several MC's for a personal assistant application. This is a common type of spoken dialog application that provides assistance to the user in performing such tasks as making phone calls, sending text messages and sending email. It can also provide useful information such as news, weather, and sports scores. This example is essentially a command and control application where the user makes a request and the system carries it out. Dotted lines within the IM show state transitions through the interaction. Solid lines represent lifecycle events between the IM and the MC's. In order to simplify the diagram, not all lifecycle events involved in an interaction are explicitly shown.

The IM is shown as a state chart that begins by identifying the user. The IM may also be responsible for starting the MC's that collect the user's input (speech recognition, typing, handwriting, and touch events), although this is not shown in the diagram. The IM identifies the user by starting a speaker identification MC with a StartRequest event. The speaker identification component responds with a StartResponse event. If the MC is unable to start, it can send back error messages or other status information in the StatusResponse, but if the MC is able to start, it begins its operation. When the MC is finished with its operation, it returns the data (in this case an identification of a speaker) with a DoneNotification event. In this example the MC responsible for identifying the user is a speaker identification component, but identification could just as well be done through face recognition from a camera or through the use of a fingerprint reader.

Interaction Manager for a personal assistant application
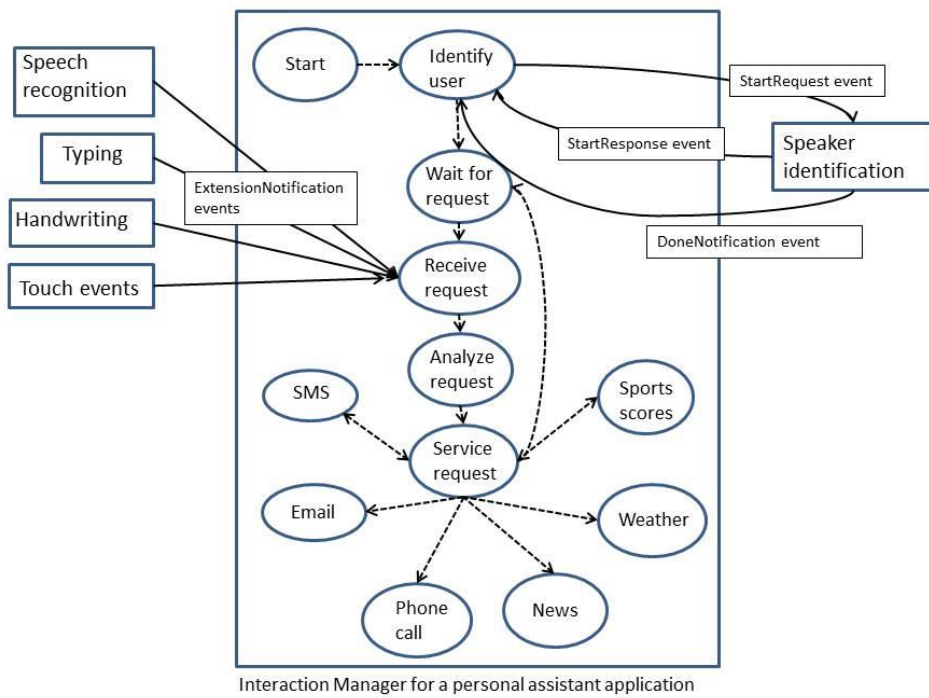
Fig. 3 IM and MC's for a personal assistant application

Fig. 4 shows the full DoneNotification event in XML format from the speaker identification MC, including the EMMA data. The lifecycle event information includes the source component, the target component, the context ID, the requestID, and the status, as described in Section 5.5. The EMMA result of the identification process is included in the data field. The identification component has identified the user as "Mary Smith" with a confidence of .8. After the user has been identified, the application waits for requests from the user.

```
<mmi:mmi  xmlns:mmi="http://www.w3.org/2008/04/mmi-arch"
version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma">
<mmi:doneNotification mmi:source="someURI"
mmi:target="someOtherURI" mmi:context="someURI"
mmi:status="success" mmi:requestID="request-1" >
    <mmi:data>
    <emma:emma version="1.0">
      <emma:interpretation
        id="int1"
        emma:confidence=".80"
        emma:medium="acoustic"
        emma:mode="voice"
        emma:verbal="false"
        emma:function="identification">
          <person>12345</person>
          <name>Mary Smith</name>
      </emma:interpretation>
    </emma:emma>
    </mmi:data>
    </mmi:doneNotification>
</mmi:mmi>
```

Fig. 4 DoneNotification event from a speaker identification MC

At this point the user input MC's are all running, and like the IM, are waiting for events from the user. When the user provides an input through speech, typing, handwriting or tapping on the touchscreen, the MC that processes the action sends an ExtensionNotification event to the IM with the user's input represented in EMMA. For example the user might say something like "What's the weather going to be like today?" and the speech recognition MC would send the message to the IM with that meaning[2]. The IM analyzes the request and determines how it should be serviced. In this case it invokes a weather component to find out what the weather will be like, presents the information to the user, and returns to the wait state, where it awaits a new input from the user.

---

[2] In this example, we assume that the speech recognition component provides an interpretation of the input, in addition to the literal tokens of input, to allow for the user to express this request in other words, such as "Tell me about today's weather", or even "Will I need my umbrella?" However, the architecture supports interpreting the user's input with a separate natural language understanding MC.

10. Related Issues and Future Directions

There are several areas where it would be useful to extend the MMI Architecture to either provide additional important functionality, or to improve interoperability among components developed by different vendors.

10.1. Encapsulation and Modality Interaction

Although encapsulation is essential for making it possible to easily add new modality functionality by incorporating components from different vendors into a system, it does make close interaction between related modalities more difficult, especially when low latency is required. One good example of this is coordination of text-to-speech (TTS) output with graphical displays of human faces, where timing must be very precise so that the speech and visual components stay synchronized. The architecture provides for this case with the concepts of nested and complex components that allow closely related functionalities to be contained in a single component; however, as nested and complex components include more and more functionalities, we begin to lose the advantages of encapsulation and modularity.

10.2. Communication Protocols

The MMI Architecture does not define a required communications prototcol for transporting lifecycle events between MC's and IM's. This enables it to be flexible and usable for many different use cases. For web applications, HTTP is often appropriate, but newer protocols such as web sockets [32,28] and server-sent events [27] can also be used.

Appendix F of the specification documents in detail how HTTP can be used as a communications protocol for an MMI Architecture application. One configuration of IM's and MC's where HTTP would be suitable is where the IM resides on a web server and the MC's are HTTP clients. Because only clients initiate exchanges in HTTP, in order to support the MMI Architecture, the server has to have some means of sending events to the client. One way to achieve this is to have the client MC poll the server for events by sending an HTTP/GET request to the server to obtain any IM-initiated lifecycle events. Events can be sent from the client to the server with normal HTTP/POST requests.

10.3. Discovery and Registration

The MMI Architecture specification describes clearly how system components behave and interact, but it does not address how components are discovered and made available in the first place. For this reason, the Multimodal Interaction Working Group has begun a work item on discovery and registration [13] which specifies methods for dynamically combining and controlling modalities through the use of a registry based on user-experience data and modality states. Although many multimodal applications can be supported by interaction through fixed, known components, there are a number of use cases for dynamically configured systems where discovery is very important. Some of these are discussed in detail in [13], including smart homes, personalized in-car experience, multiple users interacting in public spaces, and interaction with health sensors.

10.4. Media

A number of MC's make use of media such as audio, images, or video. The MMI Architecture does not define how a component obtains media or what media formats can be used. Audio data, for example, could be sent to a server using multipart form data sent by HTTP/Post. Session Initiation Protocol (SIP) and Real Time Streaming Protocol (RTSP) are other options. Web Real-Time Communications (WebRTC) [33] is an emerging W3C standard that will also be useful for transmitting media as it matures.

10.5. Streaming

Many user inputs can be sent in discrete messages that contain the entire input, for example, when the mouse is clicked or the "submit" button is pressed the complete user input is conveyed in that message. On the other hand, some inputs are continuous and can be sent for processing while the input is being collected. For example, a speech input can be sent as a complete audio file after the user has finished speaking or it can be sent continuously while speech is occurring. In most cases the user experience is improved if a continuous input is streamed because latency is reduced. In addition, for applications like dictation, the user experience is also improved if the user can see the speech recognition results incrementally, while he or she is still speaking. Intermediate results of processing streams can be returned as multiple incremental ExtensionNotification events with EMMA data

fields. Although there is no standard yet for incremental EMMA results, a proposal is described in [34], using the new attributes "emma:streamId" which identifies a stream, "emma:streamSeqNr" indicating the current input's position in a stream, and "emma:streamProgress" to indicate the beginning, middle and end of streams.

10.6. Implementations and Evaluation

As part of the W3C standardization process, the MMI Architecture standard was implemented by five organizations which submitted formal implementation reports. The implementation reports are summarized in [35]. As examples of the kinds of applications that have been built on MMI Architecture implementations, Openstream has implemented a number of types of applications, including financial services, healthcare, and mobile workforce solutions [36]. A research application of multimodal services for multimedia is described in [37]. Outside of the implementations included in the formal report, the specification has also been implemented by others, for example [38]. An evaluation of the ability for implementations of multimodal components from different vendors to interoperate was performed by a group consisting of representatives from Openstream, France Telecom, and Deutsche Telekom [31], who were able to successfully demonstrate interoperability between a voice MC developed by Openstream, a graphical MC developed by Deutsche Telekom, and an IM developed by France Telecom. The test application was a multimodal math quiz. Feedback on the standard is always welcome on the Multimodal Interaction Working Group's mailing list, www-multimodal@w3.org.

11. Summary

The W3C Multimodal Architecture and Interfaces specification, along with the Extensible Multimodal Annotation specification, provides a generic, flexible and extensible standard for integrating a broad variety of system components into multimodal applications. Whether the application runs on a traditional browser, on a mobile device, or in a home or a car, the multimodal architecture provides the infrastructure for a wide variety of innovative applications, accessed through natural, multimodal interfaces.

The W3C Multimodal Architecture and Interfaces and EMMA specifications represent the work of many individuals who have participated in the Multimodal Interaction Working Group. In particular, I would like to acknowledge the work of the following authors of the MMI Architecture and EMMA specifications and related documents.

Kazuyuki Ashimura	Nagesh Kharidi	B. Helena Rodriguez
Jim Barnett	Ingmar Kliche	Muthuselvam Selvaraj
Paolo Baggia	Jim Larson	Andrew Wahbe
Michael Bodell	Raj Tumuluri	Piotr Wiechno
Daniel C. Burnett	Brad Porter	Moshe Yudkowsky
Jerry Carter	Dave Raggett	
Michael Johnston	T.V. Raman	

12. References

1. Turing A (1950) Computing machinery and intelligence. Mind 59:433-460
2. Johnston M, Bangalore S, Vasireddy G, Stent A, Ehlen P, Walker M, Whittaker S, Maloor P MATCH: an architecture for multimodal dialogue systems. In: Proceedings of the 40th Annual Meeting on Association For Computational Linguistics, Philadelphia, 2001. Association for Computational Linguistics, pp 376-383
3. Bayer S (2005) Building a Standards and Research Community with the Galaxy Communicator Software Infrastructure. In: Dahl DA (ed) Practical Spoken Dialog Systems, vol 26. Text, Speech and Language Technology. Kluwer Academic Publishers, Dordrecht, pp 166-196
4. Oviatt SL (1999) Ten myths of multimodal interaction. Communications of the ACM 42:74--81
5. Seneff S, Lau R, Polifroni J Organization, communication, and control in the Galaxy-II Conversational System. In: Proceedings of Eurospeech 1999, Budapest, 1999.
6. Barnett J, Bodell M, Dahl DA, Kliche I, Larson J, Porter B, Raggett D, Raman TV, Rodriguez BH, Selvaraj M, Tumuluri R, Wahbe A, Wiechno P, Yudkowsky M (2012) Multimodal Architecture and Interfaces. World Wide Web Consortium. http://www.w3.org/TR/mmi-arch/. Accessed November 20 2012
7. Barnett J, Akolkar R, Auburn RJ, Bodell M, Burnett DC, Carter J, McGlashan S, Lager T, Helbing M, Hosn R, Raman TV, Reifenrath K, Rosenthal Na (2012) State Chart XML (SCXML): State Machine Notation for Control Abstraction. World Wide Web Consortium. http://www.w3.org/TR/scxml/. Accessed November 20 2012
8. McGlashan S, Burnett DC, Carter J, Danielsen P, Ferrans J, Hunt A, Lucas B, Porter B, Rehor K, Tryphonas S (2004) Voice Extensible Markup Language (VoiceXML 2.0). W3C. http://www.w3.org/TR/voicexml20/. Accessed November 9 2012
9. Kopp S, Krenn B, Marsella S, Marshall A, Pelachaud C, Pirker H, Thórisson KR, Vilhjálmsson H Towards a Common Framework for Multimodal Generation: The Behavior Markup Language In: International Conference on Intelligent Virtual Agents, Marina del Rey, California, USA, August 21-23, 2006 2006.
10. Heylen D, Kopp S, Marsella S, Pelachaud C, Vilhjalmsson H (2008) The Next Step Towards a Functional Markup Language. Paper presented at the Proc. of Intelligent Virtual Agents (IVA 2008), Tokyo, Japan, September 1-3
11. Scherer S, Marsella S, Stratou G, Xu Y, Morbini F, Egan A, Rizzo A, Morency L-P (2012) Perception Markup Language: Towards a Standardized Representation of Perceived Nonverbal Behaviors. In: Nakano Y, Neff M, Paiva A, Walker M (eds) Intelligent Virtual Agents, vol 7502.

Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp 455-463. doi:10.1007/978-3-642-33197-8_47

12. Araki M, Tachibana K Multimodal Dialog Description Language for Rapid System Development. In: 7th SIGdial Workshop on Discourse and Dialogue, Sydney, Australia, July 2006.

13. Rodriguez BH, Wiechno P, Dahl DA, Ashimura K, Tumuluri R (2012) Registration & Discovery of Multimodal Modality Components in Multimodal Systems: Use Cases and Requirements. World Wide Web Consortium. http://www.w3.org/TR/mmi-discovery/. Accessed November 26 2012

14. Johnston M, Baggia P, Burnett D, Carter J, Dahl DA, McCobb G, Raggett D (2009) EMMA: Extensible MultiModal Annotation markup language. W3C. http://www.w3.org/TR/emma/. Accessed November 9 2012

15. Bray T, Jean Paoli J, Sperberg-McQueen CM, Maler E, Yergeau F (2004) Extensible Markup Language (XML) 1.0 (Third Edition). World Wide Web Consortium. http://www.w3.org/TR/2004/REC-xml-20040204/. Accessed November 9 2012

16. Burnett DC, Walker MR, Hunt A (2004) W3C Speech Synthesis Markup Language (SSML). W3C. http://www.w3.org/TR/speech-synthesis/.

17. Oshry M, Auburn RJ, Baggia P, Bodell M, Burke D, Burnett DC, Candell E, Carter J, McGlashan S, Lee A, Porter B, Rehor K (2007) Voice Extensible Markup Language (VoiceXML) 2.1. http://www.w3.org/TR/voicexml21/. Accessed November 9 2012

18. Popescu A (2012) Geolocation API Specification. World Wide Web Consortium. http://www.w3.org/TR/geolocation-API/. Accessed November 27 2012

19. Kostiainen A, Oksanen I, Hazaël-Massieux D (2012) HTML Media Capture. World Wide Web Consortium. http://www.w3.org/TR/capture-api/. Accessed November 27 2012

20. Microsoft (2007) Microsoft Speech API 5.3 (SAPI). http://msdn2.microsoft.com/en-us/library/ms723627.aspx.

21. Java Speech API. (1998) Sun Microsystems. http://java.sun.com/products/java-media/speech/.

22. SALT Forum (2002) Speech Application Language Tags (SALT). http://www.saltforum.org.

23. IBM (2003) X+V 1.1. http://www-3.ibm.com/software/pervasive/multimodal/x+v/11/spec.htm.

24. Bodell M, Bringert B, Brown R, Burnett DC, Dahl DA, Druta D, Ehlen P, Hemphill C, Johnston M, Pettay O, Sampath S, Schröder M, Shires G, Tumuluri R, Young M (2011) HTML Speech Incubator Group Final Report. World Wide Web Consortium. http://www.w3.org/2005/Incubator/htmlspeech/XGR-htmlspeech-20111206/. Accessed November 27 2012

25. Kliche I, Dahl DA, Larson JA, Rodriguez BH, Selvaraj M (2011) Best practices for creating MMI Modality Components. World Wide Web Consortium. http://www.w3.org/TR/2011/NOTE-mmi-mcbp-20110301/. Accessed November 20 2012

26. Watt SM, Underhill T, Chee Y-M, Franke K, Froumentin M, Madhvanath S, Magaña J-A, Pakosz G, Russell G, Selvaraj M, Seni G, Tremblay C, Yaeger L (2011) Ink Markup Language (InkML). World Wide Web Consortium. http://www.w3.org/TR/InkML. Accessed November 27 2012

27. Hickson I (2012) Server-sent Events. World Wide Web Consortium. http://www.w3.org/TR/eventsource/. Accessed November 20 2012

28. Hickson I (2012) The WebSocket API. The World Wide Web Consortium. http://www.w3.org/TR/websockets/. Accessed November 20 2012

29. Hunt A, McGlashan S (2004) W3C Speech Recognition Grammar Specification (SRGS). W3C. http://www.w3.org/TR/speech-grammar/. Accessed November 9 2012

30. Van Tichelen L, Burke D (2007) Semantic Interpretation for Speech Recognition. W3C. http://www.w3.org/TR/semantic-interpretation/. Accessed November 9 2012

31. Kliche I, Kharidi N, Wiechno P (2012) MMI interoperability test report. World Wide Web Consortium. http://www.w3.org/TR/2012/NOTE-mmi-interop-20120124/. Accessed November 27 2012

32. Fette I, Melnikov A (2011) RFC 6455: The WebSocket Protocol. Internet Engineering Task Force. http://tools.ietf.org/html/rfc6455. Accessed November 20 2012

33. Bergkvist A, Burnett DC, Jennings C, Narayanan A (2012) WebRTC 1.0: Real-time Communication Between Browsers. World Wide Web Consortium. http://www.w3.org/TR/webrtc/. Accessed November 28 2012

34. Johnston M, Dahl DA, Kliche I, Baggia P, Burnett DC, Burkhardt F, Ashimura K (2009) Use Cases for Possible Future EMMA Features. World Wide Web Consortium. http://www.w3.org/TR/emma-usecases/.

35. Wiechno P, Kharidi N, Kliche I, Rodriguez BH, Schnelle-Walka D, Dahl DA, Ashimura K (2012) Multimodal Architecture and Interfaces 1.0 Implementation Report. World Wide Web Consortium. http://www.w3.org/2002/mmi/2012/mmi-arch-ir/. Accessed November 27 2012

36. Openstream I (2013) Solutions. http://www.openstream.com/solutions.htm. Accessed March 15 2013

37. Rodriguez BH, Moissianc J-C, Demeure I Multimodal Instantiation of Assistance Services. In: iiWAS '10 Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services Paris, France, 2010. ACM, pp 934-937

38. Pous M, Ceccaroni L Multimodal Interaction in Distributed and Ubiquitous Computing. In: Fifth International Conference on Internet and Web Applications and Services (ICIW), Barcelona, Spain, 2010.