Chapter 11
Standard Portals for Intelligent Services

Deborah A. Dahl

Abstract Some multimodal interpretation services natively support W3C multimodal standards, but most still use their own proprietary formats and protocols. This makes it much more difficult for developers to use different systems because they have to learn and program to a new API for each vendor. This paper describes how standards-based servers can wrap proprietary systems in the W3C MMI Architecture and EMMA 2.0 to allow developers to interact with modality interpretation services in a standard way, even if the service that they are using does not natively support the standards.

11.1 Introduction

Multimodal technology that supports forms of input (modalities) such as natural language processing, speech recognition, handwriting recognition, and object recognition from images is becoming increasingly powerful and is being employed in a wide variety of useful applications. However, it is currently typical for each vendor to have its own proprietary application programming interface (API). Because of this, developing multimodal applications requires mastering a different API for each vendor. Furthermore, these API's differ for different vendors' versions of each modality. The result is that developing multimodal applications becomes unnecessarily complex and difficult. Developers require extensive expertise and experience in order to master all of these API's. Acquiring this expertise is especially difficult for developers at small companies. This situation slows down the rate at which multimodal applications can be implemented and makes them more expensive than they would be if API's were uniform.

Standards such as the W3C Multimodal Architecture and Interfaces specification (MMI Architecture) [1–3] define a generic modality API, but the adoption of this standard across many vendors and modalities will take time. In the interim, an

D.A. Dahl (✉)
Conversational Technologies, Plymouth Meeting, PA, USA
e-mail: dahl@conversational-technologies.com

D.A. Dahl (ed.), *Multimodal Interaction with W3C Standards*,
DOI 10.1007/978-3-319-42816-1_11

alternative approach for developers who would like to take advantage of the standards would be to use portals that reformat proprietary results in standard formats.

By providing a standard, generic, modality, and vendor independent API in the form of the MMI Architecture, standard portals greatly simplify the learning process for developers. This approach is also much more extensible to new modalities than proprietary approaches. Furthermore, it makes it easier to change modality vendors if another vendor offers a superior product.

11.2 Overview of a Portal

As stated above, multimodal technology that supports such capabilities as natural language processing, speech recognition, handwriting recognition, and object recognition from images is becoming increasingly powerful and is being used in many applications. There are many products available in this space. Just looking at natural language processing offerings alone, some examples are wit.ai (Facebook) [4], api.ai [5], Microsoft LUIS (Language Understanding Intelligent System) [6], and Amazon Alexa Skills Kit [7], to name just a few of the systems available in 2016. Similarly, there are a number of API's for emotion recognition, including affectiva [8], EmoVu [9], Microsoft Emotion Recognition [10], Kairos [11], and nViso [12].

However, currently all of these systems have their own proprietary API's. Because of this, developing multimodal applications requires mastering a different API for each modality, and each vendor or even multiple API's for one modality, if the application supports multiple modality services.

This problem can be addressed through a standard multimodal web service portal. A standard portal can provide access to many types of modalities through a standard API; specifically, the W3C Multimodal Architecture and Interfaces (MMI Architecture) specification [1–3], as shown in Fig. 11.1. A standard portal serves as a layer of middleware between client applications and modalities. Developers of client applications only need to code to the standard MMI Architecture and the multimodal web service portal will provide the interface to the vendor-specific API, shielding developers from the details of the proprietary API and simplifying development. The standard portal is in fact an MMI Architecture Modality Component, communicating with clients using MMI Architecture Life Cycle events. A

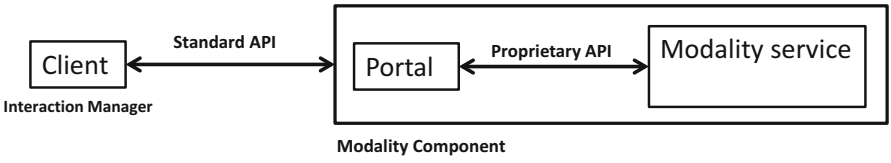

Fig. 11.1 Portal wrapping a standard API around a proprietary API

uniform API also makes it significantly easier to integrate, or fuse, inputs from multiple components. For example, it would be very useful to integrate speech and geolocation inputs in order to respond to user questions such as "where is the nearest Chinese restaurant" or "How far am I from home?" It is easy to see that as mobile devices add capabilities the problem of integrating multiple API's becomes very complex very quickly. While the problem of integrating inputs from multiple device capabilities is to some extent addressed by standard device API's such as the Media Capture and Streams API [13] these API's are still modality-specific, so that cross-modality integration of inputs is still up to the developer.

11.3 The Standard API

11.3.1 MMI Architecture

The standard API discussed in this paper consists of two components:

1. The MMI Architecture Life Cycle events for communication between an Interaction Manager (IM) and the Modality Components (MC's) that support the application.
2. Extensible Multimodal Annotation markup (EMMA 2.0) [14–16] for representing user input and system output.

The MMI Architecture includes both components and events. The components are (1) the Interaction Manager (IM), which coordinates the interaction, and (2) Modality Components (MC's). MC's both interpret multimodal inputs (from users as well as sensors) and create multimodal outputs. Modality Components communicate only with the Interaction Manager, they do not communicate directly with each other.

In addition to the components, the MMI Architecture also includes a set of high level Life Cycle events for communication between the IM and the MC's. Life Cycle events focused on controlling components include **StartRequest**, **PauseRequest**, **ResumeRequest**, and **CancelRequest**. These are messages sent from the IM to MC's. MC's, upon receiving one of these messages, respond with **Response** events, such as **StartResponse** and **PauseResponse**, for acknowledging receipt of the **Request** events and reporting errors. In addition, MC's can send a **DoneNotification** event when the requested processing is completed. Either the IM or an MC can also send an **ExtensionNotification** event at any time. **ExtensionNotification** events can contain arbitrary, application-specific data. No specific syntax is required for Life Cycle events, but XML is used in the examples in the specification, and will be used in this chapter.

Every Life Cycle event can optionally contain a **Data** field with additional information about the event. In the cases where the event pertains to user input or system output, the **Data** field contains Extensible Multimodal Annotation (EMMA) [14–16] data, which represents the user input and/or system output.

11.3.2 EMMA

EMMA is an XML language that is especially appropriate for representing semantically complex information. The semantics of the information itself is contained in the `<emma:interpretation>` element for user input or the `<emma:output>` element for system output. In addition to the actual semantics of the information, EMMA is also able to represent a rich set of metadata related to the context of the input or output. EMMA metadata includes, for example, processor confidence, timestamps, alternatives (nbest), medium and mode, the process that produced the EMMA result, tokens of input and pointers to the original signal (such as an audio file or image), among many other types of metadata.

In effect, the standard API referred to in this chapter consists of MMI Life Cycle events containing EMMA to represent interpreted inputs from users or sensors and system outputs.

11.4 Details of Multimodal Interaction with the Portal

An example architecture of a standard multimodal portal is shown in Fig. 11.2.

Interaction is initiated in the client-side components (1) by the user. Interaction modalities may include, for example, speech, typing, or mouse input, but may potentially include many other forms of input. The client-side components include application logic (2) implemented, for example, in HTML and JavaScript in

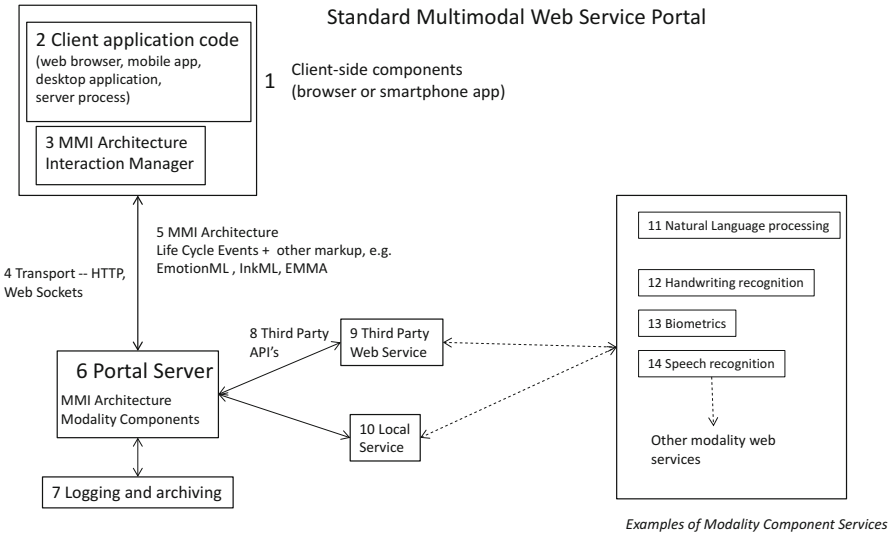

Fig. 11.2 Architecture of an MMI portal

browser-based implementations. The MMI Architecture Interaction Manager (3) sends over a transport mechanism such as HTTP (4) an MMI Architecture compliant Life Cycle event (5).

The Life Cycle event instructs the Portal Server (6) to process the user's input as required by the nature of the input (natural language understanding for language input, for example). (7) Logging and archiving of Life Cycle events may optionally occur at any point in processing.

Most critically, once the Portal Server has determined which third party services (if any) are required to process the event, it creates an API call (8) to that service (9). Although these API calls themselves may be proprietary, knowledge of any proprietary details is restricted to the Portal Server and is therefore isolated from the application developer, who only has to be concerned with sending and receiving standard MMI Architecture Life Cycle events. Within this architecture, it is also possible for services to be provided locally, within the portal (10).

Examples of possible (remote or local) modality services include but are not limited to natural language processing (11), handwriting recognition (12), biometric processing (13), and speech recognition (14). Once the appropriate service is contacted, its result is transmitted back to the Portal Server (6), reformulated into standard Life Cycle events (5), and sent back to the client-side components (1), specifically to the client Interaction Manager (3). Finally, application-specific code in the client (2) executes the appropriate action as determined by the processing result.

11.5 Implementing a Portal

Developing a standard portal requires developing several components. Going back to Fig. 11.2, the first component (2) is an application using client-side code (running in a web browser or as native code) which captures user input in modalities that are appropriate to the application. For example, a hand-held translation system requires speech to be captured for speech to speech translation, or keystrokes to be captured for translation from typing.

In addition, the client-side code will include functionality that controls the components with standard MMI Architecture Life Cycle events (that is, it will include an Interaction Manager (3)). The Interaction Manager can be implemented as a reusable library (for example, a Javascript library for browser clients) that can be used in many applications. SCXML [17, 18] is a suggested choice for Interaction Managers in the MMI Architecture. SCXML as a choice for Interaction Managers is especially efficient because the SCXML interpreter itself need only be implemented once for each platform, with the Interaction Managers for specific applications being implemented in SCMXL markup.

The Portal Server, which processes the Life Cycle events receives events using a standard transport such as HTTP [19] or Web Sockets [20, 21] (see Fig. 11.3 for an example of an actual HTTP POST message). The Portal Server is the key to the portal, because it serves to isolate proprietary API's (8) from the developer and enables the developer to access modality component services (11–14) entirely

```
POST https://proloquia-nlservice.rhcloud.com/rest/processmessage HTTP/1.1
Host: proloquia-nlservice.rhcloud.com
Connection: keep-alive
Content-Length: 725
Origin: https://proloquia-nlservice.rhcloud.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.106 Safari/537.36
Content-Type: text/xml
Accept: */*
Referer: https://proloquia-nlservice.rhcloud.com/understanding.html
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8

<mmi:StartRequest xmlns:mmi="http://www.w3.org/2008/04/mmi-arch" mmi:Context="nlClient0395"
mmi:RequestID="requestID0249" mmi:Source="ctNLClient"
mmi:Target="ctNLServer"><mmi:Data><function>understanding</function><emma:emma
xmlns:emma="http://www.w3.org/2003/04/emma" version="1.1"><emma:interpretation id="initial1"
emma:function="understanding" emma:tokens="put a couple cans of tomato soup on the
shopping list" emma:medium="tactile" emma:mode="keys" emma:verbal="true" emma:device-
type="keyboard" emma:end="1452629301869" emma:lang="en-US" emma:expressed-
through="text"><emma:literal>put a couple cans of tomato soup on the shopping
list</emma:literal></emma:interpretation></emma:emma></mmi:Data></mmi:StartRequest>
```

Fig. 11.3 HTTP POST request with MMI StartRequest event for "put a couple cans of tomato soup on the shopping list"

through standard mechanisms. Implementing the Portal Server requires developing code that can (1) interpret MMI Life Cycle events, (2) determine what services are being requested, (3) translate the user's request to the native API used by the service, (4) call the required services, and (5) reformat the results back into standard MMI Life Cycle events. In addition, a Portal Server can optionally perform other useful functions such as logging and archiving the event traffic, providing information as to what services are available, acting as a security gateway, format conversions, and managing user credentials.

The Interaction Manager (3) and the Portal Server (6) are essential parts of the portal. The Interaction Manager creates the Life Cycle events from the user's input and interprets the Life Cycle events sent back from the Portal Server. A transport mechanism (4) is required for the portal, but it is not necessary for the developer to implement the transport mechanism because a number of standard transport mechanisms are already available and are appropriate for use in this architecture, including HTTP or WebSockets. The Portal Server can be used on its own, without the ability to access third party components (8, 9, 11–14), just using local services (10); however, the portal is far more useful if translation from standard Life Cycle events to third party API's (8) is implemented for accessing existing third party services. In addition, logging and archiving services (7), while not necessary, are extremely useful in production systems for monitoring usage and debugging problems. Another aspect of a portal that would be very useful, although not required, is a way for clients to query the Portal Server in order to discover available services. Discovery and Registration functionality of this kind could be implemented using the W3C Discovery and Registration approach discussed in [22–24].

11.6 An Example: Home Control

The Internet of Things (IoT) has enormous potential for adding convenience, comfort, safety, and efficiency to everyday life as well as for supporting larger scale enterprise applications. However, there will soon be too many items in the IoT to realistically expect conventional graphical interfaces to support all the ways in which users might want to interact with them. For this reason, natural language using a standard API will become very important for these types of interactions. This section discusses an IoT example in the area of home control.

Home control is a common use case for the IoT. Home control includes control of lighting, appliances, heating and air conditioning, entertainment and security, among many other possibilities. Even limiting consideration to items that users will want to interact with in the home still leaves the possibility of interaction with hundreds of devices. If each device, or even each vendor of a connected home system, has its own API, this will quickly become unmanageable for developers who wish to integrate many devices into an application. Here we will describe an MMI Architecture approach for controlling lighting with a standard portal.

Figure 11.4 shows a web page with a user interface for natural language control of lighting. The user can click "Start Recognition" to start recognition and speak, or the user can type the request into a text box. In this case the user has typed "It's dark in here." The web page Javascript wraps the input in EMMA and the **StartRequest** LifeCycle event to produce the event shown in Fig. 11.5. Application-specific information is contained in "`<mmi:Data>`". In this example there is an application-specific field "`function`" which determines which function the data pertains to, in this case "`lightControl`". The user input itself, expressed in EMMA, is also contained in the "`<mmi:Data>`" field.

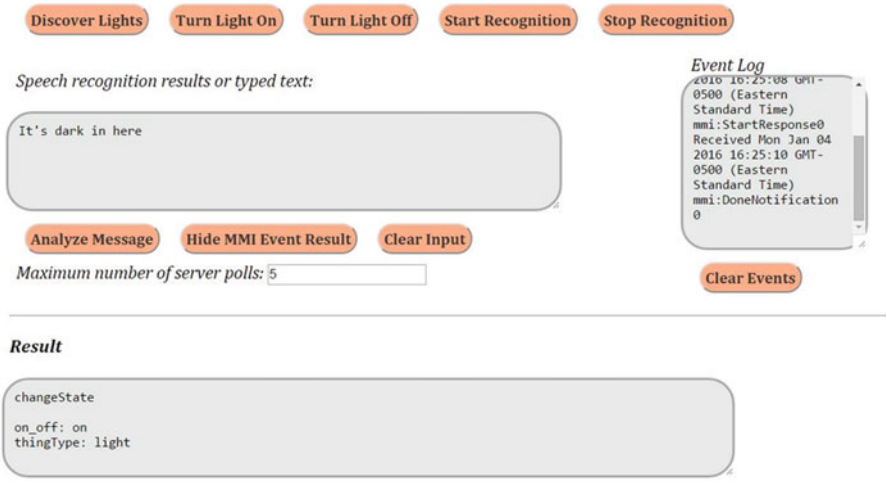

Fig. 11.4 Web page for home control

```
<mmi:StartRequest xmlns:mmi="http://www.w3.org/2008/04/mmi-arch"
                  mmi:Context="nlClient0515"
                  mmi:RequestID="requestID1841"
                  mmi:Source="ctNLClient"
                  mmi:Target="ctNLServer">
  <mmi:Data>
    <function>lightControl</function>
    <emma:emma xmlns:emma="http://www.w3.org/2003/04/emma" version="2.0">
      <emma:interpretation
        id="initial2"
        emma:function="lightControl"
        emma:tokens="It's dark in here"
        emma:medium="acoustic"
        emma:mode="voice"
        emma:verbal="true"
        emma:device-type="microphone"
        emma:end="1451946835723"
        emma:lang="en-US"
        emma:expressed-through="text">
        <emma:literal>It's dark in here</emma:literal>
      </emma:interpretation>
    </emma:emma>
  </mmi:Data>
</mmi:StartRequest>
```

Fig. 11.5 StartRequest Life Cycle event for "it's dark in here"

The **StartRequest** event is sent to the portal via HTTP POST and the portal is polled using AJAX [25] for information returned in response to the **StartRequest**. The first event returned from the portal is a **StartResponse** which simply acknowledges that the **StartRequest** was received. The portal then creates an API request to a wit.ai [4] natural language processing endpoint which has been trained to understand home control requests. It then sends the native request to the wit.ai service endpoint. Wit.ai interprets "it's dark in here" to mean that the user wants to turn on the light. The wit.ai endpoint returns natural language understanding results in a its own proprietary JSON format, as shown in Fig. 11.6. However, since the web client Interaction Manager expects MMI Architecture Life Cycle events, the portal will reformat the proprietary result into standard EMMA, and place the EMMA into the Data field of a Life Cycle event. The resulting **DoneNotification** event which is sent back to the client is shown in Fig. 11.7, with the actual interpretation boxed and in bold (see [14, 16] for details of the EMMA XML format).

Comparing the native API result in Fig. 11.6 with the MMI Architecture/EMMA result in Fig. 11.7, we can note that the semantic information contained in the result is the same—"it's dark in here" is interpreted as "turn the light on." Both formats also include confidence information. The EMMA result contains additional metadata, including timestamps, the language of the input, the process that produced the result, and information about the modality of the input (`emma:mode="keys"`). While some of this information is optional in EMMA, including the richer metadata can become very important for debugging and tuning large-scale, enterprise

```
{
 "msg_id": "a81ffcef-4606-4a93-8f63-0ccf9d8a5b05",
 "_text": "it's dark in here",
 "outcomes": [
  {
   "_text": "it's dark in here",
   "confidence": 0.782,
   "intent": "changeState",
   "entities": {
    "thingType": [
     {
      "type": "value",
      "value": "light"
     }
    ],
    "on_off": [
     {
      "value": "on"
     }
    ]
   }
  }
 ]
}
```


Fig. 11.6 Native wit.ai JSON output

applications. It is also possible to retain the complete EMMA data on the server (where it can be used in debugging and tuning) while sending only the minimum amount of data to a client (for use in interactive dialogs), using mechanisms that have been newly introduced in EMMA 2.0 [16]. While in this case the native format of wit.ai is JSON and the MMI/EMMA format is in XML, there are many software tools available for converting between these formats.

11.7 Existing Portals

A very experimental MMI Architecture client and portal has been implemented by the author. Please contact the author for access to the portal. This portal includes demos of emotion recognition from language, natural language understanding, and part of speech tagging, among others. The portal accepts MMI Architecture Life Cycle events over HTTP with user inputs represented in EMMA. The examples in this chapter were produced by this portal.

For emotion recognition, an EmotionML wrapper for the Microsoft Project Oxford Emotion Recognizer is also available [26]. While not a full MMI Architecture portal, it does wrap a proprietary API with a standard, EmotionML [27, 28], which is very much in the spirit of providing standard API's to otherwise proprietary services.

```
<mmi:mmi
   xmlns:mmi="mmi">
  <mmi:DoneNotification mmi:Context="nlClient0515" mmi:RequestID="requestID0217" mmi:Source="ctNLServer"
mmi:Status="success" mmi:Target="ctNLClient">
    <mmi:Data>
      <emma:emma
   xmlns:emma="http://www.w3.org/2003/04/emma"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0"
xsi:schemaLocation="http://www.w3.org/2003/04/emma  http://www.w3.org/TR/2009/REC-emma-
20090210/emma.xsd">
        <emma:interpretation emma:confidence="0.744" emma:process="wit.ai" emma:tokens="It's dark in here"
id="interp12">
          <emma:derived-from composite="false" resource="#initial1"/>
          <nlResult>
            <_text>It's dark in here</_text>
            <msg_id>6a24c2e5-a904-4f9d-95fd-a0dc892d9460</msg_id>
            <outcomes>
              <e>
                <_text>It's dark in here</_text>
                <confidence>0.744</confidence>
                <entities>
                  <on_off>
                    <e>
                      <value>on</value>
                    </e>
                  </on_off>
                  <thingType>
                    <e>
                      <type>value</type>
                      <value>light</value>
                    </e>
                  </thingType>
                </entities>
                <intent>changeState</intent>
              </e>
            </outcomes>
          </nlResult>
        </emma:interpretation>
        <emma:derivation>
          <emma:interpretation emma:device-type="keyboard" emma:end="1451942708899" emma:expressed-
through="text" emma:function="lightControl" emma:lang="en-US" emma:medium="tactile" emma:mode="keys"
emma:tokens="It's dark in here" emma:verbal="true" id="initial1">
            <emma:literal>It's dark in here</emma:literal>
          </emma:interpretation>
        </emma:derivation>
      </emma:emma>
    </mmi:Data>
  </mmi:DoneNotification>
</mmi:mmi>
```

interpretation

Fig. 11.7 DoneNotification event for the interpretation of "it's dark in here" as "turn the light on"

11.8 Integrating Portals with Other MMI-Standards Compliant Components

As the standards become more widely integrated into modality services, there will be increasing native support for EMMA and the MMI Architecture. This development will be completely compatible with the portal model. Components supporting the standards natively will be fully interoperable with a standard portal. For example, an application for emotion recognition might fuse results from language

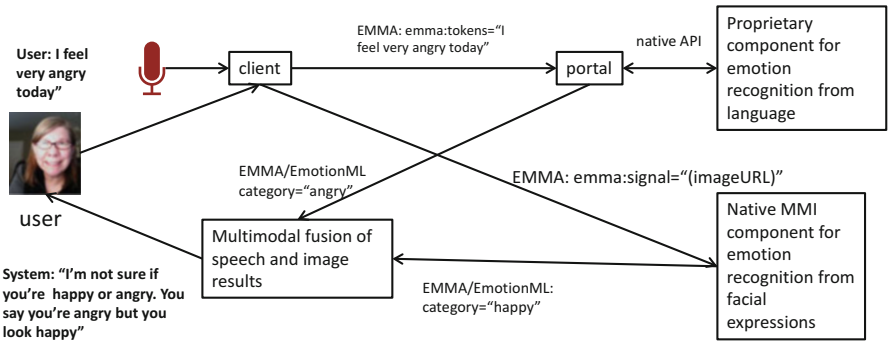

Fig. 11.8 Mixing portals with MMI native components

and facial expressions to improve the accuracy of the emotion recognition result. The language recognition could come from a service provided by a portal, while the facial expression analysis could come from a service that supports the MMI Architecture natively. Integration of information from different modalities (fusion) would be provided by a fusion component, as shown in Fig. 11.8. Of course, systems can include more than one standard portal, where each portal provides different modality services.

11.9 Developing Standard Modality Components and Portals

Given an existing modality processor (for example, handwriting recognition, speech recognition, object recognition, or emotion recognition) developing a standard component is straightforward. Following the requirements and documentation guidelines in [29], the developer provides access to the native capabilities of the component through MMI Life Cycle events. Thus, the user of the component will use a standard API call such as the one shown in Fig. 11.5, rather than the corresponding native call, the HTTP GET message https://api.wit.ai/message?v=20141022&q=it%27s%20dark%20in%20here.

Clearly, the native call is less verbose, but much of the detailed information in the standard API call is optional. In addition, the additional standard information, if used, can provide a great deal of detail that is valuable for logging, archiving, and tuning applications. This kind of information is especially important in large scale commercial applications.

Multiple modality components can be aggregated into a portal by providing a single REST endpoint and including information in the `mmi:Data` field to indicate which modality component is being requested.

11.10 Conclusions

In summary, standards-based multimodal portals can provide standard interfaces to otherwise proprietary services, providing a way for developers to use standards with proprietary systems.

In doing so, they provide the following advantages over proprietary approaches:

1. They reduce the need for developers to learn proprietary API's.
2. They can foster the adoption of standards by supporting a phased implementation approach.
3. They increase vendor-independence.
4. They can simplify logging and analysis of inputs for debugging and tuning because processing results from different vendors' services will be in the same format.
5. They simplify adding new modalities to an existing application because inputs from different modalities will be in the same format.
6. They simplify integration of inputs from components using the MMI Architecture API's natively with information produced by proprietary systems.

References

1. Barnett, J., Bodell, M., Dahl, D. A., Kliche, I., Larson, J., Porter, B., et al. (2012). Multimodal architecture and interfaces. World Wide Web Consortium. http://www.w3.org/TR/mmi-arch/. Accessed 20 Nov 2012.
2. Dahl, D. A. (2013). The W3C multimodal architecture and interfaces standard. *Journal on Multimodal User Interfaces*, 1–12 (2013). doi:10.1007/s12193-013-0120-5.
3. Barnett, J. (2016). Introduction to the multimodal architecture. In D. Dahl (Ed.), *Multimodal interaction with W3C standards: Towards natural user interfaces to everything*. New York, NY: Springer.
4. wit.ai (2015). wit.ai. https://wit.ai/. Accessed 17 Mar 2015.
5. api.ai (2015). api.ai. http://api.ai/. Accessed 17 Mar 2015.
6. Microsoft (2015). Language Understanding Intelligent Service (LUIS). Microsoft. http://www.projectoxford.ai/luis. Accessed 5 June 2015.
7. Amazon (2016). Alexa Skills Kit. Amazon. https://developer.amazon.com/public/solutions/alexa/alexa-skills-kit. Accessed 6 Jan 2016.
8. affectiva (2016). Affdex emotion sensing and analytics. affectiva. http://www.affectiva.com/solutions/apis-sdks/. Accessed 11 Jan 2016.
9. EmoVu (2016). EmoVu Cloud API. Eyeris. http://emovu.com/e/developers/api/. Accessed 12 Jan 2016.
10. Microsoft (2016). Project oxford emotion recognition. Microsoft. https://www.projectoxford.ai/demo/emotion. Accessed 12 Jan 2016.
11. Kairos (2016). Emotion analysis API. Kairos. https://www.kairos.com/emotion-analysis-api. Accessed 11 Jan 2016.
12. nViso (2016). nViso emotion recognition. nViso. http://www.nviso.ch/index.html. Accessed 11 Jan 2016.
13. Burnett, D., Bergkvist, A., Jennings, C., & Narayanan, A. (2015). *Media capture and streams* (14th ed.). Boston, MA: World Wide Web Consortium.

14. Johnston, M. (2016). Extensible multimodal annotation for intelligent interactive systems. In D. Dahl (Ed.), *Multimodal interaction with W3C standards: Towards natural user interfaces to everything*. New York, NY: Springer.
15. Johnston, M., Baggia, P., Burnett, D., Carter, J., Dahl, D. A., McCobb, G., et al. (2009). EMMA: Extensible MultiModal Annotation markup language. W3C. http://www.w3.org/TR/emma/. Accessed 9 Nov 2012.
16. Johnston, M., Dahl, D. A., Denny, T., & Kharidi, N. (2015). EMMA: Extensible MultiModal Annotation markup language Version 2.0. World Wide Web Consortium. http://www.w3.org/TR/emma20/. Accessed 16 Dec 2015.
17. Barnett, J. (2016). Introduction to SCXML. In D. Dahl (Ed.), *Multimodal interaction with W3C standards: Toward natural user interfaces to everything*. New York, NY: Springer.
18. Barnett, J., Akolkar, R., Auburn, R. J., Bodell, M., Burnett, D. C., Carter, J., et al. (2015). State Chart XML (SCXML): State machine notation for control abstraction. World Wide Web Consortium. http://www.w3.org/TR/scxml/. Accessed 20 Feb 2016.
19. Fielding, R. T., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., et al. (1999). RFC 2616 hypertext transfer protocol—HTTP/1.1. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc2616. Accessed 12 Jan 2016.
20. Fette, I., & Melnikov, A. (2011). RFC 6455 The WebSocket Protocol. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc6455. Accessed 12 Jan 2016.
21. Hickson, I. (2012). The WebSocket API. The World Wide Web Consortium. http://www.w3.org/TR/websockets/. Accessed 20 Nov 2012.
22. Rodríguez, B. H., Barnett, J., Dahl, D., Tumuluri, R., Kharidi, N., & Ashimura, K. (2015). Discovery and registration of multimodal modality components: State handling. World Wide Web Consortium. https://www.w3.org/TR/mmi-mc-discovery/.
23. Rodriguez, B. H., & Moissinac, J.-C. (2016). Discovery and registration—finding and integrating components into dynamic systems. In D. A. Dahl (Ed.), *Multimodal interaction with W3C standards: Toward natural user interfaces to everything*. New York, NY: Springer.
24. Rodriguez, B. H., Wiechno, P., Dahl, D. A., 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 26 Nov 2012.
25. Garrett, J. J. (2005). Ajax: A new approach to web applications. Adaptive Path. https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php. Accessed 14 Jan 2016.
26. Hilton, A. (2015). EmotionAPI 0.2.0. Coolfire solutions. https://github.com/Felsig/Emotion-API. Accessed 11 Jan 2016.
27. Schröder, M., Baggia, P., Burkhardt, F., Pelachaud, C., Peter, C., & Zovato, E. (2014). Emotion Markup Language (EmotionML) 1.0 World Wide Web Consortium. http://www.w3.org/TR/emotionml/.
28. Burkhardt, F., Pelachaud, C., & Schuller, B. (2016). Emotion markup language. In D. Dahl (Ed.), *Multimodal interaction with W3C standards: Toward natural user interfaces to everything*. New York, NY: Springer.
29. Kliche, I., Dahl, D. A., Larson, J. A., Rodriguez, B. H., & 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 20 Nov 2012.