DMML: An XML Language for Interacting with Multi-modal Dialog Systems

Nanda Kambhatla, Malgorzata Budzikowska, Sylvie Levesque, Nicolas Nicolov, Wlodek Zadrozny, Charles Wiecha and Julie MacNaught

IBM T.J. Watson Research Center
30 Saw Mill River Road, Hawthorne, NY 10532
{nanda,sm1,nicolas,wlodz,wiecha,jmacna}@us.ibm.com, slevesqu@ca.ibm.com

Abstract

We present Dialog Moves Markup Language (DMML): an extensible markup language (XML) representation of modality independent communicative acts of automated conversational agents. In our architecture, DMML is the interface to and from conversational dialog managers for user interactions through any channel or modality. The use of a common XML interface language across different channels promotes high cost efficiency for the business. DMML itself has no application or domain specific elements; DMML elements embed elements representing application business logic. DMML captures the abstractions necessary to represent arbitrary multi-agent dialogs and to build cost-efficient, sophisticated natural language dialog systems for business applications.

Introduction

Our goal is to create a framework for building conversational dialog agents for business applications, where users can converse with the agents using any channel of interaction (e.g. web, telephone, PDA, cellular phone, etc.) or modality (speech, text, graphics, etc). Conversational dialog agents are automated software agents that can participate fully in natural dialog (Allen 1995) and whose internal state may include beliefs, desires, and intentions (BDI models; e.g. see (Bratman et al. 1988; Cohen et al. 1990) and references therein). Examples of conversational agents include natural language dialog based telephony banking and stock trading systems (Zadrozny et al. 1998) and planning systems for disaster handling (Ferguson and Allen 1998).

We are building several multi-modal conversational agents for different business applications. In our architecture (see Figure 1), there are several presentation managers (PMs), one for each channel of interaction. A channel can encompass several modalities. For example, users may interact with web sites using speech, text, or graphics (the web presentation manager will handle all these interactions).

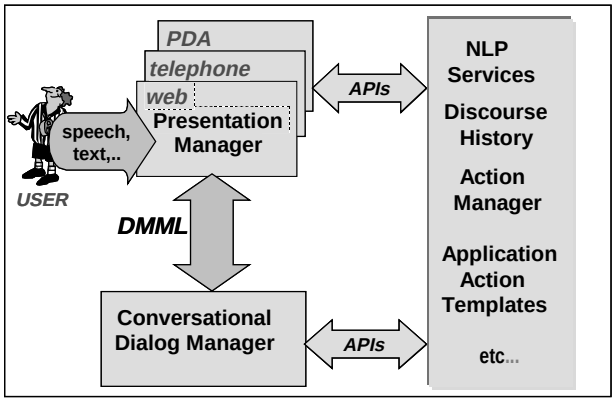

Figure 1: Architecture for multi-modal, multi-channel conversational agents for business applications.

Each presentation manager (PM) is responsible for interacting with the user through its specific channel, sending any user input to a conversational dialog manager (CDM) in a common dialog interaction format (which we call dialog moves markup language or DMML). The PM calls APIs to access natural language understanding and generation modules specific to the modalities in use and to generate or parse DMML messages. For example, a telephony PM might call APIs for accessing a speech recognition module. The PM receives DMML messages (if any) from the CDM and communicates to the user the relevant information though appropriate modalities over the same channel of interaction.

The CDM is responsible for managing the dialog with the user. The CDM interacts with the PMs using a modality- and channel-independent language called DMML. The CDM uses a suite of APIs to interact with specialized services and managers to execute business transactions (through an action manager), to fetch or update the discourse history, to fetch application templates, etc.

The Dialog Moves Markup Language (DMML) is a language designed for representing the communicative acts of conversational agents for communication between such agents. For successful multi-agent communication, the communicative intent of one agent must be recognized by the other agent. DMML attempts to capture such intentions. DMML can be used for messaging between a human (one kind of conversational agent) and a conversational dialog manager (another conversational agent). Notice how this view associates the PM with the human user to result in a software agent communicating with other software agents through DMML (cf. Figure 1). DMML can also be used for communication between any two (or more) agents that have a notion of intentions to communicate, actions they want to achieve through the communication, and a shared view of their environment. DMML can represent a synchronous turn taking dialog between two agents. We believe it is also general enough to represent arbitrary asynchronous multi-party dialogs.

DMML is an application of eXtensible Markup Language (XML; (Bray et al. 1998)), a standard for document and message markup. DMML has been designed to be an open markup language with no domain- or application-specific markup tags. It is open to enable developers to use further refinements of the basic speech acts in use. All application- and domain-specific markup is encoded using application-specific tags, which reside *underneath* (i.e. are embedded in) the DMML markup. This concept is explained in greater detail in section 3.

The use of DMML as a standard common interface between conversational agents facilitates tremendous cost-efficiency in developing dialog applications. DMML also enables relatively easy portability to new channels of interaction, since only presentation managers need to be developed for such channels. This is because there is only one conversational dialog manager managing conversations with users across different channels with different modalities. The use of well recognized and emerging standards like XML and XSL is crucial, since the DMML messages can be transformed using XSL (eXtensible Stylesheet Language; (Clark and Deach 1998)) with relative ease to channel or modality dependent languages like VoiceXML (VoiceXML 2000), HTML, or Wireless Markup Language.

In this paper, we present the DMML language (Section 2) and show an example (Section 3) illustrating its components with a dialog between a human and an automated conversational agent for stock trading. We contrast DMML with some related work in section 4, such as the work in philosophy of language, the Elephant 2000 programming language, KQML and agent markup languages. We explain our work in progress with DMML in section 5 and present some conclusions in section 6.

Dialog Moves Markup Language (DMML)

The basic elements of DMML are dialog profiles and dialog moves. Dialog profile elements enable agents to send to each other the constraints on their respective environments or the constraints on the communication itself. For instance, an agent may be unwilling (as a result of its internal decision making process) or unable (due to its environmental constraints) to process requests of a certain kind. Dialog profiles enable agents to communicate such constraints to each other.

A dialog move by agent1 to agent2 represents a set of communicative acts by agent1 directed towards agent2, with the intention of changing agent2's model of the state of the world and/or convince agent2 to take actions based on agent2's revised model of the state of the world.

Dialog profiles

Dialog profiles enable dialog agents to communicate to each other the constraints of their respective environments and constraints on the communication itself. The basic elements of profiles are templates (e.g. XML Schemas (Thompson et al. 2000)) describing these constraints. DMML supports four basic types of templates—assertion templates, command templates, request templates and response templates—corresponding to the four basic types of dialog moves. These templates may contain “schemas” (Thompson et al. 2000) expressing constraints on the corresponding dialog moves. For instance, a response template contains a schema defining the syntax of a valid response. By exchanging such templates, dialog agents can negotiate the parameters of communication before or during a dialog.

Shown below is a profile that defines a template for valid XML elements for representing stock transactions to a stock trading CDM (cf. Figure 1). Due to space constraints, we have only shown the schema for buy transactions. The notion is similar to that of defining valid types for communication. This template specifies that a valid BUY element must contain zero or one instances of a COMPANY element, a QUANTITY element, an ACCOUNT element, and a PRICE element. Each COMPANY element has a text string representing the name of the COMPANY, and so on. The CDM sends profiles like the one below to PMs that are requesting stock trading services. After receiving this profile, each receiving agent (PM) knows the syntax of the XML messages for buying shares, for selling shares, for inquiring about the price of certain shares, etc. Note that profiles are not required to be sent before dialog moves are communicated. DMML allows profiles to be sent at any point in a dialog session. This enables agents to dynamically communicate to each other any changes in their environments or any changes to the constraints of their communication.

```
<profile>
  <template type=”document”>
    <schema id=”stock_transactions”>
      <ElementType name=”COMPANY” model=”closed”
content=”textOnly”
                dt:type=”string”>
        </ElementType>
        <ElementType name=”QUANTITY” model=”closed”
content=”textOnly”
                dt:type=”number”>
        </ElementType>
      <ElementType name=”ACCOUNT” model=”closed”
content=”textOnly”
                dt:type=”string”>
        </ElementType>
      <ElementType name=”PRICE” model=”closed” content=”textOnly”
                dt:type=”amount”>
        </ElementType>
      <ElementType name=”BUY” model=”closed” content=”eltOnly”>
        <element type=”COMPANY” minoccurs=”0” maxoccurs=”1”/>
        <element type=”QUANTITY” minoccurs=”0” maxoccurs=”1”/>
        <element type=”ACCOUNT” minoccurs=”0” maxoccurs=”1”/>
        <element type=”PRICE” minoccurs=”0” maxoccurs=”1”/>
      </ElementType>
      …
    </schema>
  </template>
</profile>
```

Dialog moves

Currently, DMML includes four basic types of dialog_moves: assertions, commands, requests and responses.

- *Assertions* represent unsolicited information by an agent that does not necessarily require a response: greetings, warnings, reminders, thanks and welcome messages, offers, etc. Assertions also include statements of fact by agents.
- *Commands* represent speech acts where the commanding agent's expectation is of unconditional action execution: e.g. help, exit, cancel and operator commands.
- *Requests* represent requests for information, confirmation, clarification, identification, action execution, notification, etc. Note that most request elements will have business/application elements as their descendents describing the specific things being requested for the given application. Request elements may also contain response template elements describing valid responses to the request and validation scripts (in a standard language like ECMAScript) that can verify the validity of responses with respect to some other (semantic) constraints.
- *Responses* represent responses to request moves. Thus, a response element always contains an attribute referring to the id of the request move to which it is the response. DMML does not mandate response elements to immediately follow request elements; i.e. the responses can be asynchronous and out of turn. Responses may include notifications, clarifications, confirmations (confirmed or rejected), action_results, information, answer_lists and descriptions. Answer_lists include a list of answers. A description may include a summary, identity of responder, a rationale, and a list of suggestions. Suggestions are lists of alternative answers. Note that most response elements will have business/application elements as their descendants describing the specific things being sent as a response to the request for the given application.

Currently DMML represents mostly directive and assertive communicative acts, due to the limitations of the conversational agents for which it is an interface language. However, in future we plan to also support commisive (e.g. *promise*), permissive (e.g. *permit*), prohibitive (e.g. *forbid*), declarative (e.g. *declare*), and expressive (e.g. *wish*) communicative acts (cf. Singh 1998).

DMML is an open markup language. Thus, a user query “Can you tell me the price of IBM” could be represented as

```
<REQUEST>
  Can you tell me the price of IBM?
</REQUEST>
```

Or as

```
<REQUEST>
  <TELL>
    the price of IBM?
  </TELL>
</REQUEST>
```

Or as

```
<REQUEST REQUEST_TYPE=”INFORMATION”>
  <PRICE_INFO>
    <COMPANY>IBM</COMPANY>
  </PRICE_INFO>
</REQUEST>.
```

All the above fragments are valid DMML fragments. Allowing open markup enables developers to extend the pre-specified dialog moves by specifying their own sub-categories of requests, commands, assertions, responses, notifications, etc.

In DMML, all the application specific markup is embedded within the DMML markup to support heterogeneous agents and to ensure that DMML is not domain or application dependent. DMML can encapsulate any content as long as it is provided in XML compliant format. For example, in the last excerpt above, all the business application markup (within the request element) is shown in bold and is not part of DMML. This enables separate XSL transformations

that can transform the 'conversational style' independent of the specific application or even the domain of the application. Thus, the same DMML message can be used for completely different applications or for different implementations of the same application.

An example: use of DMML for a stock trading application

In this section, we present an example of a dialog between a user and a conversational agent for stock trading. We show a sequence of user utterances, the corresponding DMML messages sent by the PM (presentation manager) to the CDM (conversational dialog manager; see Figure 1), the DMML messages sent by the CDM to the PM, and the corresponding content of the user screen. While this example does not illustrate all the elements of DMML, it gives a flavor of the use of DMML elements and elements representing the embedded business logic. All application specific business logic encoding is shown in bold type and is not part of DMML. Note that the references to the schema "stock_transactions" in response-templates are to the example schema shown in the previous section that defines the syntax of valid XML messages for communication for this domain.

USER: Hi! I want to buy one hundred shares of IBM

```
<dmml version="1.0">
  <move from="PM1" to="CDM "
        channel="web" id="m1">
    <assertion type="greet" id="a1">
      <xml schema=”assert_schema”>
        <MESSAGE>Hi</MESSAGE>
      </xml>
    </assertion>
    <assertion type="info" id="a2">
      <xml  schema="logon_schema" >
          <NAME>Steve</NAME>
          <PASSWORD>adroit</PASSWORD>
          <ACCOUNT>12345</ACCOUNT>
      </xml>
    </assertion>
    <request type="action" id="r1">
      <xml  schema="stock_transactions">
        <BUY>
          <COMPANY>IBM</COMPANY>
          <QUANTIY>100</QUANTITY>
        </BUY>
      </xml>
    </request>
  </move>
</dmml>
```

SYSTEM: Hello, Steve!

You said you wanted to buy 100 shares of IBM.

Do you want to use your checking account?

At what price do you want to buy these shares?

```
<dmml version="1.0">
   <move from="CDM" to="PM1"
        channel="web" id="i2">
      <assertion type="greet" id="a3”
        previousmoveid="a2">
        <xml schema=”assert_schema”>
          <MESSAGE>
              Hi Steve!!!
          </MESSAGE>
        </xml>
      </assertion>
      <request type="info" id="r2" previousmoveid="r1">
        <xml schema="stock_transactions">
          <BUY>
            <COMPANY origin=”user_specified”>
                IBM</COMPANY>
            <QUANTIY origin=”user_specified”>
                100</QUANTITY>
            <ACCOUNT origin=”system_inferred”>
                12345</ACCOUNT>
          </BUY>
        </xml>
        <responsetemplate>
          <schema id=”stock_transactions/>
        </responsetemplate>
      </request>
   </move>
</dmml>
```

USER: What is the current price of IBM?

```
<dmml version="1.0">
   <move from="PM1" to="CDM" channel="web" id="i3">
      <request type="info" id="r3">
        <xml  schema="stock_transactions">
          <PRICE_INFO>
            <COMPANY>IBM</COMPANY>
          </PRICE_INFO>
        </xml>
        <responsetemplate>
          <schema id=”stock_transactions”/>
        </responsetemplate>
      </request>
   </move>
</dmml>
```

SYSTEM: The current price of IBM is 110 1/2.

Do you want to buy 100 shares at this price from your checking account?

```
<dmml version="1.0">
   <move from="CDM" to="PM1" channel="web" id="i4">
      <response type="info" id="rp1" previousmoveid="r3">
        <xml schema="stock_transactions"> >
          <PRICE_INFO>
            <COMPANY origin=”user_specified”>
              IBM</COMPANY>
            <PRICE origin=”system_specified”>
              110 ½</PRICE>
          </PRICE_INFO>
        </xml>
      </response>
      <request type="confirmation" id="r4" previousmoveid="r1">
        <xml schema="stock_transactions">
          <BUY >
            <COMPANY origin=”user_specified”>
              IBM</COMPANY>
            <QUANTITY origin=”user_specified”>
              100</QUANTITY>
            <ACCOUNT origin=”system_inferred”>
```

```
            12345</ACCOUNT>
          <PRICE origin="system_inferred">
            110 ½</PRICE
        </BUY >
      </xml>
      </responsetemplate>
        <schema id="stock_transactions"/>
      </responsetemplate>
    </request>
  </move>
</dmml>
```

USER: Make that 75 shares.

```
<dmml version="1.0">
  <move from="PM1" to="CDM" channel="web" id="i5">
    <request type="correction" id="r5" previousmoveid="r4">
      <xml schema="stock_transactions">
        <BUY >
          <QUANTITY origin="user_specification">
            75</QUANTITY>
        </BUY>
      </xml>
      </responsetemplate>
        <schema id="stock_transactions"/>
      </responsetemplate>
    </request>
  </move>
</dmml>
```

SYSTEM: Do you want me to execute a buy order of 75 shares of IBM at 110 1/2 from your checking account?

…

As suggested by the example above, the use of DMML for messaging between PM and CDM enables tremendous cost-efficiency for the development of multi-modal and multi-channel stock trading systems. The development costs are greatly reduced since we need to build one dialog manager instead of one for each channel. Also, enabling conversational access through new channels is relatively easy, since it entails only the building of another PM (e.g. using XSL) and not another dialog engine. Moreover, DMML gives us a mechanism to specify business specific syntactic and/or some semantic constraints using XML Schemas and ECMAScript.

Related Work

The act of uttering a sentence (through speech, typed in text, etc.) is called a speech act. When a speech act occurs, the following acts are preformed (Austin 1962):

- locutionary act: the act of the utterance being produced,
- illocutionary act: the act the speaker performs in uttering the words, and
- perlocutionary act: the act that actually occurs as a result of the utterance.

In DMML, we attempt to capture the illocutionary acts of human users when they interact with a conversational agent and the illocutionary acts of conversational agents when interacting with other conversational agents or a human user.

The Elephant 2000 programming language (McCarthy 1998) is a language sharing some of the design goals of DMML. However, while Elephant is intended to be a declarative programming language (based on speech act theory) for building intelligent agents, DMML is intended to be an interface language between such agents.

DMML can also be used for representing human-to-human dialogs. When used for this purpose, DMML is similar in spirit to the SGML annotation scheme (Isard et al. 98) used in the new version of the Edinburgh Map Task corpus[1] for providing abstract annotations for sophisticated human-to-human task-oriented dialogs. The annotations include dialog moves, dialogue games, dialog transactions, POS tagging etc. DMML offers additional constructs (like profiles) that facilitate the annotation of environmental (and other) constraints that the above work lacks.

DMML can also be used as a general agent communication markup language (e.g. KQML (Finin et al 1994)). DMML provides a standard (e.g. XML, XSL, XML Schemas, etc.) interface for interacting with dialog agents with certain characteristics. DMML satisfies Singh's (Singh 1998) criteria for a flexible and powerful agent markup language. DMML is an open language since its syntax can be extended (e.g. by defining different kinds of *requests*) and it allows open application markup embedded within the speech acts. DMML also satisfies the heterogeneity criterion since it allows agents of different design to talk to each other by exchanging their respective constraints. However, DMML does not satisfy the requirement that it's semantics be based on social agency, because (currently) DMML does not allow for the specification of norms of interpretation. For DMML-based languages, such norms have to be developed independently of specifying communication protocols (cf. Singh 1998). The DMML design assumes that the automated agents using DMML for communication are working together--without necessarily having knowledge of each other's environment--to achieve common business goals. Thus, (currently) there is an implicit assumption that the agents are benign, honest, cooperative, and share the semantics of the business transactions. These assumptions alleviate the need for a more rigorous semantic specification.

Work in progress

We are planning to use DMML in our architecture for building multi-modal multi-channel software applications. The goal is to build an architectural framework and tools that empower an application developer to author a business

[1] Currently the annotation scheme is being converted to XML.

application that can scale to multiple channels of interaction and modalities with relative ease. We intend to use this broad architecture as a platform to developing a series of conversational systems that can be accessed by multiple channels and modalities, e.g. web, telephone, etc.

In our architecture, DMML provides a modality- and channel-independent mechanism for describing communicative intent and environmental constraints of agents. The use of XML and related standards like XSL, ECMAScript, XML Schemas promotes inter-operability across heterogeneous platforms and software vendors.

We are currently finalizing a specification of version 1 of DMML, and implementing a series of APIs and accessory modules—as suggested in Figure 1—for use in the architecture described above. We plan to have a prototype system ready by fall 2000.

Conclusions

In this paper, we have presented Dialog Moves Markup Language (DMML), an XML language for interaction between conversational dialog agents. The elements of DMML are dialog moves representing communicative acts like requests, responses, assertions and commands, and dialog profiles representing constraints on dialog moves. DMML is designed to be an open markup language that allows agents to define and use their own sub-categorizations of the basic dialog moves. Hence DMML supports multiple levels of granularity/abstractions in representing natural language and multi-modal dialogs. Moreover, DMML supports communication between heterogeneous agents, because of the use of shared, common speech acts for communicating intentions and goals, rather than procedures for achieving these goals. All application specific markup is embedded within DMML elements and is in itself not part of DMML.

DMML supports multi-party, multi-modal, multi-channel interactions with a single dialog engine. The constructs of DMML are modality- and channel- independent by design and represent the abstract intentions of communicating agents. The use of XML enables relatively easy transformation to modality specific presentation languages such as VoiceXML and HTML. Thus, DMML enables tremendous cost efficiency in building conversational systems.

We believe DMML can be the basis for developing a standard XML based language for representing the communicative acts of arbitrary multi-modal multi-agent dialogs. However, currently DMML represents mainly assertive and directive communicative acts. In future work, we plan to evolve DMML to include other categories of communicative acts and representations of the social context shared by all communicating agents in the environment.

Acknowledgments

We thank all of our colleagues in the conversational machines group and in the DMML project team for many valuable insights and for their constant help.

References

Allen J. 1995. *Natural Language Understanding*. The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, USA. Second Edition.

Austin, J. L. (1962). *How to do things with words*. Oxford: Clarendon Press.

Bratman, M. E.; Israel, D.; and Pollack, M. E. 1988. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349—355.

Bray, T.; Paoli, J.; and Sperberg-McQueen, C., M. 1998. *Extensible Markup Language (XML) 1.0. Technical Report* http://www.w3.org/TR/REC-xml, World Wide Web Consortium Recommendation.

Clark, J. and Deach, S. 1998. *Extensible Stylesheet Language (XSL) 1.0. Technical Report*, http://www.w3.org/TR/WD-xsl-19980818.html, World Wide Web Consortium Working Draft.

Cohen, P. R.; Morgan, J.; and Pollack., M. E. eds., 1990. *Intentions in Communication*, The MIT Press, Cambridge, Massachusetts, USA.

Ferguson, G.; and Allen, J. 1998. TRIPS: An Integrated Intelligent Problem-Solving Assistant, Proceedings of AAAI-98 and IAAI-98, AAAI Press/ MIT Press, pp 567-572.

Finin, T.; Fritzson, R.; McKay, D.; and McEntire, R. 1994. KQML – A Language and Protocol for Knowledge and Information Exchange. Technical Report CS-94-02, Computer Science Department, University of Maryland and Valley Forge Engineering Center, Unisys Corporation.

Isard, A.; McKelvie, D.; and Thompson, H. 1998. Towards a minimal standard for dialogue transcripts: A new SGML architecture for the HCRC Map Task Corpus. Proceedings of ICSLP'98. Sydney.

McCarthy, J. 1998. Elephant 2000: A Programming Language Based on Speech Acts, Stanford University, http://www.formal.stanford.edu/jmc/elephant/elephant.html.

Singh, M.P., (1998). Agent Communication Languages: Rethinking the principles. IEEE Computer, 31(12), pp 40-47.

Thompson, H. S.; Beech, D.; Maloney, M.; and Mendelsohn, N. (2000). XML Schema Part 1: Structures, http://www.w3.org/TR/xmlschema-1/.

VoiceXML (2000). Web site: www.voicexml.org, v1.0 specification: http://www.voicexml.org/specs/VoiceXML-100.pdf

Zadrozny, W.; Wolf, C.; Kambhatla, N.; and Ye, Y. (1998), Conversation Machines for Transaction Processing, Proceedings of IAAI'98, AAAI Press/MIT Press, pp 1160-1166.