

Dialogue Management Tool

Camilla Gustavsson, Linda Strindlund, Emma Wiknertz

Linköping University, Sweden

Abstract

This paper describes a tool that can be used to simplify creating dialogues within, for example, an interactive *Talking Head* (TH) application or an ordinary question and answer file. What does the word *dialogue* within this area actually mean? Let us use the TH example; a dialogue occurs between the user and the TH when, for example, the user asks a question and the TH responds to that particular question. The answer given by the TH should be dependent on earlier questions and responses within that dialogue, i.e. which *state* the dialogue is in. A *Dialogue Manager* keeps track of the dialogue state and determines the responses to each question. But to be able to do this, the structure of the dialogue should be created in advance, i.e. all the different questions that the TH can answer should be defined, and these questions should be connected to the correct answer. To simplify the preparation of a dialogue, the *Dialogue Management Tool* has been developed. By using the tool the construction of the dialogues becomes easier, since it, among other things, prohibits incorrect references.

Keywords: Dialogue Management, Talking Head, FAQ, XML and Markup Language.

Introduction

In an interactive *Talking Head* (TH) application, there is a need for the TH to be able to converse with the user in some way. For example, a virtual salesperson has to be able to answer the user's questions about certain products. An information provider must answer questions about a certain domain. Furthermore, both have to actively ask questions or at least notify the user when it is unclear what the user really means.

Developing a dialogue includes creating *stimuli* and *responses*. When the user input matches a stimulus this should trigger the correct response. Depending on the stimulus the dialogue should traverse into different *states*. This is a well-known trick to make an application seem more intelligent. By handling this, the application will know the context of the dialogue and will therefore be able to respond correctly. The trick has been used by, for example, *Julia* and *Colin*, who are two chatterbots developed by Mauldin (1994). They seem somewhat intelligent to the user even though the structure of their knowledge is an ordinary network with a number of states.

Managing the dialogue is a very important issue in order to create an interesting and interactive TH application. By using network structures for the dialogue it is possible to create a more intelligent conversation since it gives the possibility to keep track of the conversation's state. Since the dialogues might become very large and complex, it can take a great amount of time to construct correct network structures. The aim of the *Dialogue Management Tool* (DMT) is to simplify the construction and maintenance of the dialogue.

Representation of a dialogue

The TH in the following dialogue between a TH and Anna uses the same trick as *Julia* and *Colin*, i.e. moves the dialogue into different states depending on Anna's input:

TH says, "How are you?" to Anna.
Anna says, "Not so good." to TH.
TH says, "Why is that?" to Anna.
Anna says, "I have a terrible headache." to TH.
TH says, "Have you taken aspirin?" to Anna.
Anna says, "I have to go. Goodbye!" to TH.

Figure 1 represents a fragment of the rules used by the TH in the discussion. In the diagram, S represents the stimulus, written in a regular expression, and R represents the response.

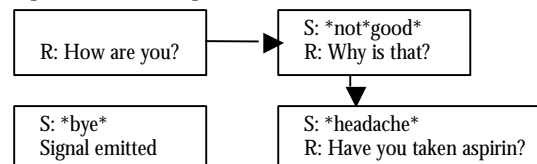


Figure 1. A diagram of the greeting example.

The first question is an active prompt from the TH and does not have to be triggered by a stimulus. Anna's answer, "Not so good." is a stimulus that moves the dialogue to a different state. In this new state the TH knows that Anna is not feeling good. The TH then asks: "Why is that?", which is a response that only can take place because of the fact that the TH "remembers" the previous questions and answers. Anna's answer about the headache is yet another stimulus that moves the dialogue into a new state and a responding question is posed. Anna's end phrase moves the dialogue into a final state, which also is an entry state and therefore can be entered at any time during the dialogue.

This short example points out the importance of dividing the dialogue into different states. The question "Why is that?" can not be posed without a

known context, since it would not have a meaning if the context is missing. Furthermore, to pose the question “Have you taken aspirin?” the TH has to know that Anna suffers from a headache. It is also important to point out that the TH can keep track of a whole sequence of stimuli and responses. This means that the TH can produce a response that relates to a discussion that appeared earlier in the conversation.

The user input might contain grammatically incorrect stimuli, but it should still trigger a response. Using pattern matching for the stimulus input solves this. Furthermore, a certain response might be considered the “correct” one for more than one stimulus. In the previous example, the stimuli “Not so good.” should trigger the same response as for example “I’m not feeling very well today.” and hence give the same answer, “Why is that?”. By forming regular expressions or word graphs for the *Dialogue Manager* (DM) to parse, it is also possible to create a stimulus that matches a great number of user interactions. For example, the stimulus “*not*good*” matches both “Not so good” and “I’m not feeling that good”.

Dialogue Management Tool

The *Dialogue Management Tool* (DMT) is a tool that aims to simplify the construction and maintenance of dialogues significantly. When constructing a dialogue, the tool makes crosschecks regarding types, names and quantity. It also maintains the consistency when updating the dialogues at a later state. Furthermore, it provides a time efficient way of creating dialogues, since the underlying structure does not have to be considered.

The DMT uses the new markup language *Dialogue Management Tool Language* (DMTL) in order to represent the dialogue and its states as a network (Gustavsson, Strindlund & Wiknertz, 2001).

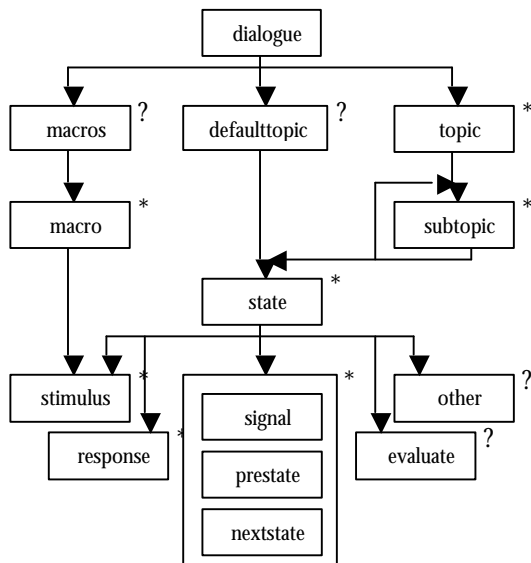


Figure 2. The structure of DMTL.

DMTL is an XML-based language and uses a *Document Type Definition* (DTD). A DTD is a set of rules that defines the grammar of an XML document. A document that fulfills the grammar rules in a specific DTD is called a *valid* document (Navarro, White & Burman, 2000). The output from the DMT is a valid DMTL document to be parsed by a DM. The structure of the DMTL DTD is shown in figure 2.

In order to give an overview of a dialogue, the previous conversation example between a TH and Anna will be expanded and step-by-step marked up according to the DMTL DTD.

The root element in DMTL is *dialogue*, which includes zero or one *macros*, zero or one *defaulttopic* and zero or more *topics*. A *macros* element includes zero or more *macro* elements that will be described later. The *defaulttopic* contains zero or more *states*, which cater for all the user inputs that do not match any other stimulus.

```

<dialogue>
  <defaulttopic> ... </defaulttopic>
  <topic name="greeting"> ... </topic>
</dialogue>
  
```

A *topic* includes zero or more *subtopics*.

```

<topic name="greeting">
  <subtopic name="casual"> ... </subtopic>
  <subtopic name="polite"> ... </subtopic>
</topic>
  
```

A *subtopic* in turn includes zero or more *subtopics* and zero or more *states*.

```

<subtopic name="casual">
  <subtopic name="swedish"> ... </subtopic>
  <state name="initial" type="active"> ... </state>
</subtopic>
  
```

A *state* includes *stimuli*, *responses*, *prestates*, *nextstates*, *signals*, *evaluate* and *other*.

The *stimuli* can be of several different types depending on the application; text, audio, visual and haptic, although text is the default value. For example, Anna might only look sad instead of saying “Not so good” in order to give a corresponding visual stimulus.

The *responses* could be plain text or marked up in any language. For example, the question/answer structure in a FAQ file could be maintained by using stimuli and responses. The response could also be marked up to direct or control the way in which the response is presented, for example, by using HTML anchors.

Prestate specifies the states from which the dialogue could have come and *nextstate* the states to which the dialogue can move.

The *signal* element enables the match to generate or emit a signal or notification to the DM, which it may choose to ignore or handle in some way. In the case

example given, when Anna says: "I have to go. Goodbye!" the DM may simply close the connection.

The *evaluate* element can be used for defining a condition that has to be fulfilled before the dialogue is able to move into this particular state, hence this will increase the efficiency when searching the dialogue structure. For example, a variable can be set to imply that a state is visited and this can then be used as a condition for traversing another state. *Other* can be used for specifying any additional application specific information necessary or simply for adding comments. Though, the simple dialogue with Anna does not require *evaluate* or *other*.

The DMTL dialogue below describes the example given about the TH and Anna, thus it only constitutes a fragment of the whole dialogue.

```
<dialogue>
  <topic name="greeting">
    <subtopic name="casual">
      <state name="initial" type="active">
        <response>How are you?</response>
        <nextstate name="greeting.casual.bad"/>
        <nextstate name="greeting.casual.good"/>
      </state>
      <state name="bad" type="linked">
        <stimulus>*not*good*</stimulus>
        <response>Why is that?</response>
        <nextstate
          name="greeting.casual.headache"/>
      </state>
      <state name="headache" type="linked">
        <stimulus>*headache*</stimulus>
        <response>Have you taken aspirin?
        </response>
      </state>
      <state name="bye" type="entry">
        <stimulus>*bye*</stimulus>
        <signal name="exit"/>
      </state>
      ...
    </subtopic>
  </topic>
</dialogue>
```

In the current version of DMT there are four different state types; *linked*, *entry*, *visitswitch* and *active*. An *active* state is a state that invokes a question, without having to be triggered by a stimulus. An *entry* state is a state that can be invoked any time during the dialogue if the stimulus matches. A *linked* state is connected to other states by using *nextstate* or *prestate*. A *visitswitch* state points to several other states and works in a similar way as a case statement in C or Java. Which state the dialogue should move into depends on, for example, if the state has been visited before.

Dialogues tend to grow fast and become large and complex, with many topics, subtopics and states. This becomes an efficiency problem when a dialogue manager has to parse all the different paths in the dialogue when searching for a suitable stimulus. To avoid this an attribute for the *subtopic* element was introduced, *keyword*. This makes it possible to specify a number of keywords for each subtopic and only if

any of these match the user input the subtopic is parsed to find a state with suitable stimulus.

Further, when creating stimuli all different ways of giving a specific stimulus must be considered. Since the natural language is complex, there are many different ways to express the same question. In order to facilitate for the user of the DMT, *macros* can be created to match the semantic of a certain stimulus. For example, the macro "WHATIS" can be used as "WHATIS VHML". This matches "What is VHML?", "What does VHML mean?" and so on.

Responses can be any text, but the current version of the DMT supports the *Virtual Human Markup Language* (VHML, 2001) within the text. Though, any markup language can be used in the dialogue. VHML is an XML-based language and is used for controlling the characters in a Virtual Human application, regarding sounds, emotions and movements of the body and in the face. Therefore, VHML can be useful when controlling the output of a TH application.

Since VHML, as well as DMTL, is an XML-based language, a problem exists in that the DMTL documents include VHML elements inside the *responses*. Because the VHML elements are not, and should not be, included in the DMTL DTD, the DMTL document will not be valid if the VHML elements remain inside the responses. The solution to this was to implement a transform function that transforms the VHML elements into plain text by using the standard entities for XML, i.e.:

Character	Entity
&	&
<	<
>	>
"	"
'	'

For example:

```
<response>
  <vhml xml:lang="en">
    <p>
      <sad> Why is that? </sad>
    </p>
  </vhml>
</response>
```

is transformed into

```
<response>
  &lt;vhml xml:lang="en"&gt;&gt;
  &lt;p&gt;
    &lt;sad&gt; Why is that? &lt;/sad&gt;
  &lt;/p&gt;
  &lt;/vhml&gt;
</response>
```

However, inside the *vhml* element these standard entities may already be used, which shows another problem. If, for example, a greater than sign is needed in the response, the user has to type in the standard entity *>*; instead of *>*, as in any other XML document. The *>* is then transformed into plain text.

For example:

```
<response>
  <vhml>
    <p>
      5 &gt; 3
    </p>
  </vhml>
</response>
```

is transformed into

```
<response>
  &lt;vhml&gt;
  &lt;p&gt;
    5 &amp;gt; 3
  &lt;/p&gt;
  &lt;/vhml&gt;
</response>
```

To process an XML document, like the DMTL document, an API has to be used. There are two major types of XML APIs, tree-based APIs and event-based APIs. A tree-based API compiles an XML document into an internal tree structure and then allows an application to navigate that tree. The *Document Object Model (DOM)* is a standard tree-based API for XML and HTML documents, developed by the World Wide Web Consortium. An event-based API, on the other hand, reports parsing events, such as the start and end of elements, directly to the application through callbacks, and does not usually build an entire tree. The *Simple API for XML (SAX)*, is an event-based API (SAX 2.0, 2001). SAX requires less memory than DOM and tends to run faster. However, with SAX, the application only sees the XML elements once and has to figure out what to do with the data right away, do it and then get ready to handle the next item. DOM, on the other hand, is more memory-intensive than SAX, since the entire document must be kept in memory at once. The advantage of DOM however, is that the application can go back and forth in the document and make changes to it (Navarro, White & Burman, 2000).

The input to the DMT is both saved as a DMTL document and stored as a DOM tree. The reason why DOM is used is that changes are made dynamically in a tree to update information at all times. The DMTL document keeps a static status of the DOM tree.

Future work

During the development of the DMT some issues have arisen that, if solved, will make the tool even more useful.

- The current version of DMT supplies VHML support by providing a list with VHML elements that can be inserted into the responses. To internationalize the DMT, this list should be written in the user's language of choice.

- One useful feature would be to be able to import a file with another dialogue structure, not just DMTL, into the DMT. After updating, the file could be exported back to the original structure.
- In the DMT GUI, the states in a subtopic are presented in a list. When the user activates a state the information within this certain state is presented. It would be an advantage to be able to see the whole network or parts of the network graphically as well. This feature would provide the user with an even better overview of the dialogue.

Conclusions

The DMT makes construction of dialogues easier and keeps track of the state traversing in a conversation. Currently the DMT is based on responses marked up in VHML. An interactive detective story has been marked up in VHML using the DMT (Gustavsson, Strindlund & Wiknertz, 2001). This is only a small application, thus it constitutes a dialogue with approximately 500 states. Keeping track of these states is a complex task and shows the advantages of using a tool as DMT. Further, the current version of DMT has been found adequate with two other applications, the Mentor System developed by Marriott (to be published) and the FAQBot by Beard (1999). Other applications may require alteration, but the current work shows a convenient means of constructing dialogues.

References

- Beard, S. (1999), *FAQBot*. Honours thesis, Curtin University of Technology, Perth, Australia.
- Gustavsson, C., Strindlund, L. & Wiknertz, E. (2001), *Verification, Validation and Evaluation of the Virtual Human Markup Language (VHML)*, Master thesis, Linköping University, Sweden.
- Marriott, A. (to be published), 'A Java Based Mentor System'. In *Java in the Computer Science Curriculum*. Editor Greening, T. LNCS Springer.
- Mauldin, M. L. (1994), 'Chatterbots, Tinymuds, And The Turing Test: Entering The Loebner Prize Competition'. In the proceedings of *AAAI-94*, AAAI Press, Seattle.
- Navarro, A., White, C. & Burman, L. (2000), *Mastering XML*. SYBEX Inc., Alameda, CA.
- SAX 2.0 (2001), *The Simple API for XML*. Available: <http://www.megginson.com/SAX/index.html>, [2001, August 10].
- VHML (2001). *VHML*. Available: <http://www.vhml.org>, [2001, September 26].