DSML: A Proposal for XML Standards for Messaging Between Components of a Natural Language Dialog System

Dragomir R. Radev, Nanda Kambhatla, Yiming Ye, Catherine Wolf, Wlodek Zadrozny
IBM TJ Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532
{radev,nanda,yiming,cwolf,wlodz}@watson.ibm.com

Abstract

In this paper, we propose using standard XML messaging interfaces between components of natural language dialog systems. We describe a stock trading and information access system, where XML is used to encode speech acts, transactions and retrieved information in messages between system components. We use an XML/XSL based unification approach to display personalized, multi-modal system responses. We are proposing the creation of XML standards for all messaging between components of NLP systems. We hope that the use of XML in messaging will promote greater interoperability of both data and code. The working name of the proposed standard messaging language(s) is DSML ("Dialog System Markup Language".)

1 Introduction

In this paper, we propose using XML based messaging between the components of natural language dialog systems. We describe our dialog system, present an example illustrating the XML messaging, and discuss the advantages of using XML.

XML or eXtensible Markup Language (Bray et al. [1998]) is an emerging worldwide standard (recommended by the World Wide Web Consortium W^3C) for document or message markup. XML is a descendant of the Standard Generalized Markup Language or SGML. XML allows developers to create their own markup languages, the semantics of which enable specific applications. XML parsers are widely available for different platforms. Using XML for messaging facilitates the development of common data abstractions leading to more modularity, and sharing of data and code by researchers.

2 Architecture of NLD system

In our natural language dialog (NLD) processing system, a user can express requests in any modality (e.g., speech, text, graphics, etc.). The NLD system iteratively identifies the communicative act(s) of a user, queries for and fills the parameters of the identified action, executes the action, and displays the results to the user using an appropriate modality.

Our class of NLD systems can be considered to be a sub-class of more general language engineering systems such as the General Architecture for Text Engineering (GATE) (Gaizauskas et al. [1996]) system. Whereas the GATE architecture is intended for general language engineering systems, we are specifically focusing on language engineering applications which involve a dialog with a user to accomplish transaction execution or information exchange.

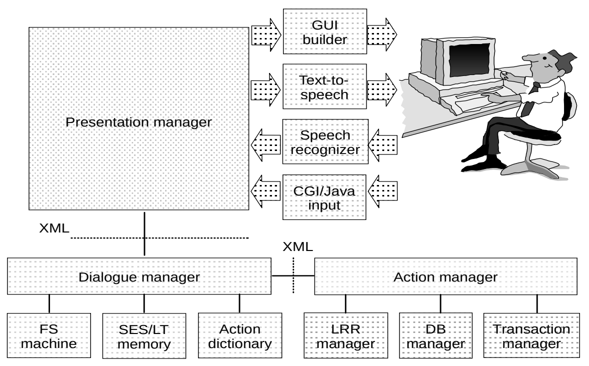

Figure 1: Block diagram of architecture of natural language dialog (NLD) processing system

The natural language dialog (NLD) processing system (see Figure 1) consists of three main modules: the presentation manager (PM), the dialog manager (DM), and the action manager (AM).

2.1 Presentation Manager

The presentation manager is responsible for obtaining any input from the user and for displaying the system's response to the user. The PM pre-processes the user input and sends a modality and channel independent representation of the user input to the DM. For instance, if the user input is spoken, the PM is responsible for executing a speech recognition process to obtain a textual representation of the user's utterance. If the user input is typed in, the PM employs a natural language parser before sending the user text to the DM.

The PM is also responsible for obtaining the system's response from the DM and presenting it to the user using appropriate channels and modalities. For instance, if the user had spoken her request, the PM might decide to present the system's response in an audio format by executing a ”text-to-speech” process. The PM's choice of the specific output format is based on the modality and the channel being used by the user, the bandwidth and display capabilities of the channel, and the personal preferences of the user. Thus, the PM might display a system response from DM as a HTML table, as a textual description, as a spoken summary, etc. The user preferences might either be inferred by the system or explicitly stated by the user (through some mechanism for specifying preferences). After displaying the system response, any user input (e.g. a clarification or a correction or a new request) is again sent to the PM as described above.

2.2 Dialog Manager

The dialog manager is responsible for determining the specific action requested by the user and filling the parameters of the identified action by way of a dialog with the user. For example, the parameters of a stock market buy request are the ticker symbol of the company, the number of shares, etc. After filling in the parameters, the DM sends the filled action template to the AM for execution. The dialog manager uses the following modules: FS (an augmented finite-state transition table), SES/LT (session/long-term memory), and an action dictionary.

The **finite-state transition table** is used by the dialog manager to track the current state of the discourse in relation to the goal state. For example, if a money transfer transaction with three parameters (A=amount, F=from account, T=to account) is to be performed, the initial state of the finite state machine will correspond to (A=uninstantiated, F=uninstantiated, T=uninstantiated). The final state corresponds to (A=instantiated, F=instantiated, T=instantiated). The dialogue manager is in charge with driving the user from the initial to the final state by asking appropriate questions (which are encoded by the developer for each possible machine node). As an example, if the user asks to “move 100 dollars from my savings account”, the machine would go to the state that corresponds to (A=instantiated, F=instantiated, T=uninstantiated) and appropriately, ask the user “To which account do you want to transfer 100 dollars from your savings account?”

The **session memory** keeps track of the current instantiated values of the various transaction parameters while the **long-term memory** keeps track of previous interactions (e.g., a successful login) and user preferences.

The **action dictionary** specifies the translation from user requests (both transactions and information requests) to action plans for the action manager to satisfy the requests, for example, associating a “buy” request with a particular database engine.

2.3 Action Manager

The action manager is responsible for determining the best mechanism for executing an action, given a filled template. Thus, for a stock news request, the AM decides which information source to look at and the specific information to look for. The AM sends the retrieved information to the DM. The AM is responsible for the coordination of user-initiated transactions, database lookups and information retrieval.

2.4 Use of XML for communication between components

We use XML for three purposes:

- for messaging between the PM and DM, where the messages encode multi-modal information requests by the user and display requests by the DM,
- for messaging between the DM and AM, where the messages encode action requests by the DM and action responses by the AM,
- as a mechanism for generating personalized and modality-specific output response for a user (see Section 4) in the PM.

In the next two sections, we elaborate on the specific XML messaging using a simple stock trading example.

3 XML messaging interfaces

In this section, we will demonstrate the XML messaging interfaces using the example of a dialog between a user and our NLD system for stock trading over the Web. We assume that a user is logged in and can select actions using either a menu of task selections or by typing free form text in a text area. Suppose our user typed in the text ”How is Cisco doing today?”. The PM sends the following XML to the DM.

```
<?xml version="1.0" encoding="UTF-8"?>
<CUSTOMER_REQUEST>
 <INFORMATION_INPUT>
  <TEXT VALUE="How is Cisco doing today?"/>
 </INFORMATION_INPUT>
</CUSTOMER_REQUEST>
```

The DM parses the user text using a natural language grammar, and identifies the speech act(s) in the textual input. The DM uses the identified speech acts (Austin

[1962], Searle [1980]) to determine the intent of the user and formulates a response (e.g. ask user for more information, update session history, send request to AM to satisfy user request, etc.). In this example, the DM identifies it as a news information request, and sends the following XML to the AM.

```
<?xml version="1.0" encoding="UTF-8"?>
<MESSAGE SPEECH_ACT="REQUEST">
 <COMPANY_NEWS TIME="TODAY"
  SYMBOL="CSCO">
 </COMPANY_NEWS>
</MESSAGE>
```

The AM retrieves the latest stock quotes for Cisco from a stock database server and uses ”language reuse and regeneration” (LRR) (Radev [1998]) technology against a set of information servers (e.g., CNNFn.com) to retrieve the latest news about Cisco. The LRR component extracts sentences from text sources to answer questions like the one above without any deep semantic processing of the original question. Note that the type of textual response in the above example can only be obtained using LRR (in the example, no information extraction is required). The AM sends the following XML to the DM:

```
<?xml version="1.0" encoding="UTF-8"?>
<MESSAGE SPEECH_ACT="REPLY">
 <COMPANY_NEWS>
  <TIME DAY="5" MONTH="October"
   HOUR="4:48PM"/>
  <COMPANY SYMBOL="CSCO" CHANGE="-7 7/16"
   PRICE="48 5/16" VOLUME="1,200,000">
    <LRR> Shares of Cisco Systems Inc.
     (CSCO) plummeted 7-7/16 to close
     at 48-5/16 after the company
     confirmed that the FTC is
     investigating the company.
    </LRR>
  </COMPANY>
 </COMPANY_NEWS>
</MESSAGE>
```

The DM sends the following XML display request to the PM containing both the stock quote and LRR portions of the above XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<DISPLAY_REQUEST CURRENT_ACTION="NEWS">
 <INFORMATION_DISPLAY>
   <QUOTE_PARAMETERS COMPANY="CSCO"
    PRICE="48 5/16" CHANGE="-7 7/16"
    VOLUME="1,200,000"/>
   <NEWS COMPANY="CSCO">
    Shares of Cisco Systems Inc.
    (CSCO) plummeted 7-7/16 to close
    at 48-5/16 after the company
    confirmed that the FTC is
    investigating the company.
   </NEWS>
 </INFORMATION_DISPLAY>
</DISPLAY_REQUEST>
```

The PM retrieves the above message and presents the information to the user. Note that the actual format and style of the rendered response is determined by the PM based on the available modalities and channels, their bandwidth and display limitations, and the preferences of the user. In this instance, the PM decides to ignore the quote parameters and directly displays the LRR retrieved text to the user. We will further discuss the presentation algorithms in the next section.

The user sees the information displayed and enters the text ”buy 150 shares at market” in the text area. The PM sends the following XML message to the DM.

```
<?xml version="1.0" encoding="UTF-8"?>
<CUSTOMER_REQUEST>
 <INFORMATION_INPUT>
  <TEXT VALUE="buy 150 shares at market"/>
 </INFORMATION_INPUT>
</CUSTOMER_REQUEST>
```

The DM identifies the user request as a ”BUY” transaction request where the ”COMPANY” is ”CSCO” (inferred from history of the conversation). The DM uses system defaults and the user account profile to obtain the values of ”ACCT_TYPE”, ”ACCT_NO”, and ”ALLORNONE”.

The DM sends the following XML to the PM, requesting confirmation from the user for all the parameters of the buy request. In the XML excerpt below, the parameters of the buy transaction that are directly entered by the user (e.g. QUANTITY, ORDERTYPE) are differentiated from the parameters that are inferred by the DM (based on the session history and/or the user preferences, e.g. COMPANY, ALLORNONE, etc.). This potentially enables the PM to display these parameters differently.

```
<?xml version="1.0" encoding="UTF-8"?>
<DISPLAY_REQUEST>
 <INFORMATION_REQUEST  CURRENT_ACTION="BUY">
  <CONFIRM PARAMETER_TYPE="USER">
    <BUY_PARAMETERS QUANTITY="150"
     ORDERTYPE="MARKET"/>
  </CONFIRM>
  <CONFIRM PARAMETER_TYPE="SYSTEM">
    <BUY_PARAMETERS COMPANY="CSCO"
     PRICE="48 5/16" ALLORNONE="YES"
     ACCT_TYPE="MARGIN" ACCT_NO="1234"/>
  </CONFIRM>
 </INFORMATION_REQUEST>
</DISPLAY_REQUEST>
```

The PM displays this information in a graphical tabular format. The user uses the menu item ”QUANTITY” to change the filled in parameter from 150 to 100. The PM sends XML to DM indicating the parameter which changed and the change itself.

```
<?xml version="1.0" encoding="UTF-8"?>
<CUSTOMER_INPUT>
 <INFORMATION_INPUT  CURRENT_ACTION="BUY">
   <BUY_PARAMETERS QUANTITY="100"/>
 </INFORMATION_INPUT>
</CUSTOMER_INPUT>
```

The DM retrieves the changed parameter and sends an updated buy action template with all parameters filled to the user for final confirmation. The user clicks a button to confirm the purchase. The PM sends XML to the DM indicating a confirmed intention to ”BUY” with the stated parameters. The DM sends a filled action template to the AM for execution. The AM completes the transaction (against a transactional database) and sends an action confirmation message (XML) to the DM. The DM then sends an XML message confirming the purchase of shares with the parameters stated. The PM displays the information to the user.

```
<?xml version="1.0" encoding="UTF-8"?>
<DISPLAY_REQUEST>
 <ACTION_EXECUTED CURRENT_ACTION="BUY"
  CONFIRMATION_NO="4567">
   <BUY_PARAMETERS COMPANY="CSCO"
    QUANTITY="100" ORDERTYPE="MARKET"
    PRICE="48 5/16" ALLORNONE="YES"
    ACCT_TYPE="MARGIN" ACCT_NO="1234"/>
 </ACTION_EXECUTED>
</DISPLAY_REQUEST>
```

In the above example, using XML enables us to encode semantic abstractions in a platform (or language) independent manner. This in turn enables us to separate content generation from content presentation and drive the response to the user through multiple modalities. It is obvious that the separation of form and content is enabled by the modular architecture more than the specific language used. We advocate the use of XML primarily for convenience (standardized language, existing generation and parsing tools, etc.). In the next section, we discuss an XML based scheme for generating personalized responses on varied modalities in the presentation manager.

4 Personalized, Multi-Modal Content Presentation

As mentioned earlier, the presentation manager (PM) receives modality independent responses from the dialog manager (DM), and generates a personalized response for a specific modality based on the user's preferences and display capabilities and time and/or space constraints. Several mechanisms exist for transforming the modality independent responses to a specific output for a given modality. In this section, we briefly discuss a mechanism based on XML/XSL and functional unification for achieving the transformation. XSL or eXtensible Style Language ((Clark and Deach [1998])) is an emerging standard for transforming an XML document into formatting objects or into other XML documents.

Using the XML/XSL unification approach for presentation, we match the XML sent by DM to PM against a set of IF/THEN rules. Whenever XML fragments match the ”IF” portion of any of these rules, the corresponding rule is activated and the ”THEN” portion of the rule contains the transformed XML (e.g., an HTML table). Modality specific output and personalization is achieved by storing a set of rules (e.g., as an XSL document) for each mode of output (e.g., text paragraph, HTML table, etc.) and for each user (to encode personalized choices of presentation). The matching of rules occurs via a unification mechanism which can be used to fill in personalized parameter values.

We illustrate this approach with a few examples. In these examples, we assume that the PM uses the following algorithm:

```
1. GET MESSAGE %M FROM THE DM
2.   SELECT USER MODEL %U
3.   SELECT OUTPUT MODE %C
4.   %M <--- PERSONALIZE (%M,%U)
5.   %M <--- RENDER_FOR_MODALITY (%M,%C)
6.   SEND %M TO BROWSER
7. GOTO 1.
```

We assume that the PM executes steps 2. and 3. using some algorithm based on the display constraints and system defaults. We further assume Steps 4. and 5. are performed using unification (e.g., using Michael Elhadad's FUF system (Elhadad [1991])) in an XSL or other standard styling platform.

Please note that currently, the XSL syntax and specifications are in flux. So, we are using our own syntax for the examples below. However, we are certainly advocating the use of a standard ”pattern/action” rule matching mechanism (provided by XSL) by the PM. In the examples below, the ”IF” portion of the rules is specified using the ”<INPUT>” element and the ”THEN” portion of the rules is specified using the ”<OUTPUT>” element. We use ”&” to denote function invocation and ”%” to denote variable names.

Text Generation: In this mode of output, any language reuse components (i.e. <LRR> components in XML sent by DM) are ignored and text is generated from scratch using the parameters of the action. A sample rule might look like the following:

```
<INPUT>
 <MESSAGE>
   <TIME DAY="%D" MONTH="%M"
         HOUR="%T"/>
   <COMPANY_NEWS>
   <COMPANY SYMBOL="%S"
         CHANGE="%C" PRICE="%P"/>
   </COMPANY_NEWS>
 </MESSAGE>
</INPUT>
<OUTPUT>
 <MESSAGE>
   <HTML><P>&text_generate(COMPANY_NEWS,
            %D,%M,%T,%S,%C,%P)</P>
   </HTML>
 </MESSAGE>
</OUTPUT>
```

In this example, the rule uses variables to store the values of the parameters (e.g., DAY, MONTH, etc.) and calls an external function &text_generate() to generate text and format it in HTML based on the parameters.

Language Reuse: In this mode of output, all elements other than <LRR> are ignored. The rule activates only if an <LRR> element is present in the XML sent by DM. The output HTML contains the value of the <LRR> element.

```
<INPUT>
 <MESSAGE>
   <COMPANY_NEWS>
    <COMPANY><LRR> %L </LRR></COMPANY>
   </COMPANY_NEWS>
 </MESSAGE>
</INPUT>
<OUTPUT>
 <MESSAGE><HTML><P>%L</P></HTML>
 </MESSAGE>
</OUTPUT>
```

HTML table: In this mode of output, the parameters are extracted, any <LRR> elements are ignored, and an HTML table is directly generated from the parameters in the ”THEN” portion of the rule.

```
<INPUT>
 <MESSAGE>
    <TIME DAY="%D" MONTH="%M"
      HOUR="%T"/>
    <COMPANY_NEWS>
      <COMPANY SYMBOL="%S"
         CHANGE="%C" PRICE="%P"/>
    </COMPANY_NEWS>
 </MESSAGE>
</INPUT>
<OUTPUT>
 <MESSAGE>
   <HTML>
     <TABLE>
       <TR>
         <TH> Day </TH>
         <TH> Month </TH>
         <TH> Time </TH>
         <TH> Stock Symbol </TH>
         <TH> Change </TH>
         <TH> Price </TH>
       </TR>
         <TR>
         <TD> %D </TD>
         <TD> %M </TD>
         <TD> %T </TD>
         <TD> %S </TD>
         <TD> %C </TD>
         <TD> %P </TD>
       </TR>
     </TABLE>
   </HTML>
 </MESSAGE>
</OUTPUT>
```

Personalization: We assume personalization information is also encoded as rules. For example, suppose a user prefers to read text that is summarized to 30% of its original length. The rule is expressed as

```
<INPUT>
 <MESSAGE>
   <COMPANY_NEWS> <LRR>%L</LRR>
   </COMPANY_NEWS>
 </MESSAGE>
</INPUT>
<OUTPUT>
 <MESSAGE>
   <LRR> &text_summarize (%L, .3) </LRR>
 </MESSAGE>
</OUTPUT>
```

Here is another example of a user model (the ”identity” user model which doesn’t indicate any user-specific preferences):

```
<INPUT> %I </INPUT>
<OUTPUT> %I </OUTPUT>
```

The ”identity” user model I has the following property:

```
PERSONALIZE (%X, I) = %X
```

Both &text_generate() and &text_summarize() are external functions that implement the semantics of XML.

5 Discussion

In this paper, we have proposed to standardize the languages for messaging between the different components of a dialog processing system that uses the dialog system markup language (DSML). We also described our natural language dialog (NLD) system and presented the XML messaging interfaces between the components of the NLD system as an illustration of the proposed standard. Using XML enables us to:

- move towards a plug-and-play architecture. For example, we plan to build XML interfaces to enable different text summarization and machine translation modules to connect to the PM, grammar parsers to connect to the DM, and information retrieval modules to connect to the AM.

- have platform-independent components. For example using Java for coding DM on NT does not constrain any other modules in any way.
- encode semantic abstractions such as transactions, actions, speech acts, etc. as data.
- separate content from presentation, which in turn enables us to generate personalized modality specific content using unification.
- build facilitators for existing systems.

6 Related Work

Our research is related to previous work on agent communication languages and in architectures for natural language generation.

6.1 Agent communication languages

An overview of agent communication languages is available in (Singh [1998]). Here we will mention some of the most relevant languages that have influenced our design, namely KQML and Arcol.

KQML (Knowledge Query and Manipulation Language) (Finin et al. [1994]) is a language and protocol for exchanging information and knowledge and is part of the ARPA Knowledge Sharing Effort (Patil et al. [1992]). KQML defines an extensible set of performatives which specify the actions that intelligent agents can perform or attempt to perform on themselves and on each other's knowledge bases. KQML has been used in concurrent engineering, intelligent planning, and scheduling.

Arcol, developed at France Télécom is an agent-based communication language, which unlike KQML, has primitives that can be composed and uses belief information to achieve communicative goals.

6.2 Architectures for Natural Language Engineering

Several large-scale architectural projects have similar goals to ours:

The **General Architecture for Text Engineering** (GATE; Gaizauskas et al. [1996]) is a general software environment for research and development in natural language engineering. The GATE architecture can be used both to develop language engineering modules or to develop large scale, end-to-end, real world applications of natural language techniques.

Our NLD system presented in section 2 is specifically designed for supporting the design and development of natural language *dialog* systems, where the system is involved in an iterative dialog with the user to satisfy her transactional or information requests. Thus, our system is less general in scope than the GATE system.

XML at the LTG in Edinburgh The MUC-7 System built by the Language Technology Group at the University of Edinburgh (Mikheev et al. [1998]) used a set of reusable text handling tools, which interacted with each other using XML input/output streams in a UNIX pipeline. The whole system was designed to enable inter-operability of tools and data for different tasks and applications. The tools do not convert the XML into internal representations, but instead just operate on the elements and attributes of interest, potentially creating new elements or modifying existing ones. Our NLD system is designed to be similarly flexible, but for the specific domain of natural language dialog processing.

Finally, **Tipster** (`http://www.fas.org/irp/program/process/tipster.htm`) is a DARPA effort to build text applications for areas ranging from Document Detection to Information Extraction using a set of general-purpose building blocks. Tipster's architecture has a wide acceptance among the research community, however its focus on batch-mode applications makes it unsuitable for natural language dialog.

7 Conclusion and Future Work

We are proposing the use of XML/XSL technologies as a standard messaging format for the components of natural language dialog systems. The XML can be used at several levels. For instance, we can use XML for representing the logical forms output by NLP parsers and also to represent the dialog acts for dialog processing. Similarly, XML can be used to represent contextual information needed for utterance interpretation (e.g., background knowledge, previous discourse, etc.).

We envision a phased approach to building XML based standards for NLP systems:

Phase I There is a broad usage of XML for NLP/dialog processing systems leading to more modular and easily exchangeable code and data. However, each group uses their own custom XML. The groups communicate with each other.

Phase II Standard XML languages are identified for different domains (e.g., brokerage systems, ATIS, directory services, etc.). Thus, we agree both on the syntax and the vocabulary of the representations. We foresee potential development of transducers to transform messages in other languages such as KQML to XML. Like XML, our XML-based standard will facilitate the encoding of speech acts such as ”request”, ”assert”, ”reply”, etc.

Phase III Standard *semantics* are identified for interpreting the standard XML languages, leading to interoperable data and code. This stage will involve the use of content languages such as KIF(Genesereth and Fikes [1992]) or Lisp as well as standardized domain ontologies.

Acknowledgments

We are extremely grateful to David Epstein, Kathy Bohrer, and Scott McFaddin for their review of this paper and helpful comments. We also thank Sylvie Levesque and Joyce Chai for participating in several key design discussions and for their very helpful comments. Hongyan Jing made some very useful remarks which we incorporated in the paper.

References

J. Austin. *How to do things with words*. Boston, Harvard University Press, 1962.

Tim Bray, Jean Paoli, and C.M. Sperberg-McQueen. Extensible markup language (XML) 1.0. Technical Report http://www.w3.org/TR/REC-xml, World Wide Web Consortium Recommendation, 1998.

James Clark and Stephen Deach. Extensible stylesheet language (XSL) 1.0. Technical Report http://www.w3.org/TR/WD-xsl-19980818.html, World Wide Web Consortium Working Draft, 1998.

Michael Elhadad. FUF: The universal unifier - user manual, version 5.0. Technical Report CUCS-038-91, Columbia University, 1991.

Tim Finin, Rich Fritzson, Don McKay, and Robin McEntire. 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, 1994.

Robert Gaizauskas, Hamish Cunningham, Yorick Wilks, Peter Rodgers, and Kevin Humphreys. GATE: An environment to support research and development in natural language processing. In *Proceedings of the 8th IEEE International Conference on Tools with Artificial Intelligence*, Toulouse, France, 1996.

Michael Genesereth and Richard Fikes. Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, Stanford, California, 1992.

Andrei Mikheev, Claire Grover, and Marc Moens. Description of the LTG system used for MUC-7. Technical Report http://www.ltg.ed.ac.uk/papers/muc.ps, HCRC Language Technology Group, University of Edinburgh, 1998.

R. S. Patil, R. E. Fikes, P. F. Patel-Schneider, D. McKay, T. Finin, T. Gruber, and R. Neches. The DARPA knowledge sharing effort: Progress report. In C. Rich, W. Swartout, and B. Nebel, editors, *Knowledge Representation*, pages 777–788, 1992.

Dragomir R. Radev. *Language Reuse and Regeneration: Generating Natural Language Summaries from Multiple On-Line Sources*. PhD thesis, Department of Computer Science, Columbia University, New York, 1998.

J. Searle. The background of meaning. In J.R. Searle, F. Kiefer, and M. Bierwisch, editors, *Speech act theory and progmatics*. 1980.

Munindar P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, 1998.