A GENERAL DIALOGUE MANAGEMENT MODEL FOR DYNAMIC-DOMAIN EXPERT SYSTEMS WITH NATURAL LANGUAGE INTERFACES

Justyna Walkowska
Department of Computer Linguistics and Artificial Intelligence, Faculty of Mathematics and Computer Science
Adam Mickiewicz University, ul. Umultowska 87, 61-614 Poznań, Poland

Keywords: Natural language processing, Natural language interfaces, Conversational agents, Dialogue management.

Abstract: This paper describes a dialogue management model for a dynamic-domain multi-agent expert system with natural language competence. The solutions presented in this paper have been derived in the design and implementation process of Polint-112-SMS, an expert system to be used by security officers overseeing public events (e.g. concerts, football games). This paper presents a modular system architecture and explains the dialogue-oriented features of the modules. The presented problems and solutions include: unification of data obtained from different users, detecting and solving contradictions, pairing questions and answers in an asynchronous mode of communication, deciding when and how to contact the users to obtain more data. The model has been applied in practise and a number of tests have been performed, the results of which are also summarized herein.

1 INTRODUCTION

This paper presents the dialogue-oriented algorithms and solutions derived in the process of design and implementation of Polint-112-SMS (Vetulani et al., 2008), a multi-agent expert system with natural language interface designed to help security officers during mass-event surveillance. It describes the universal aspects of the applied solutions (independent from language, programming language, and the system's domain of operation) that can be applied in a number of expert systems. The model is based on a modular system architecture and describes the mechanisms that support:

- the exchange of questions and answers between the system and a number of human agents,
- incomplete textual data at system's input (inter-sentence anaphora, omissions),
- ambiguous information,
- contradictory information,
- time management,
- merging of the obtained data.

Polint-112-SMS, the system that motivated this research accepts SMS messages written in Polish. The constraint of receiving textual input is not very strong: the model can be applied in speech-based systems too, provided a sufficiently efficient speech-to-text tool exists for a given language.

The system has been subjected to evaluation in a series of field experiments with the participation of security experts. The results of the evaluation are summarized at the end of this paper.

2 SYSTEM ARCHITECTURE

The presented model assumes a modular system architecture. The communication flow between the modules is outlined in figure 1. The roles of the modules are described below.

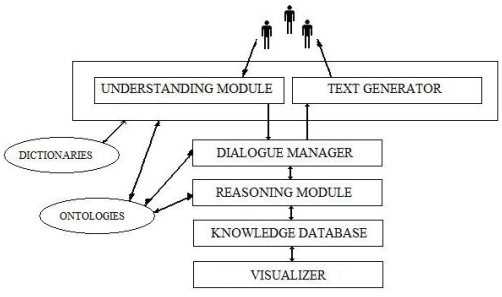

Figure 1: Overall system architecture.

Walkowska J. (2010).
A GENERAL DIALOGUE MANAGEMENT MODEL FOR DYNAMIC-DOMAIN EXPERT SYSTEMS WITH NATURAL LANGUAGE INTERFACES.
In *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence - Artificial Intelligence*, pages 74-81
DOI: 10.5220/0002728200740081

The understanding module's task is to transform the textual data into data structures understood by the core of the system, i.e. by the dialogue manager and the reasoning module. The structures and the method of their creation are described in detail in 3.1.

The dialogue manager will also be described in greater detail in the following section. It is the middleman between the human agents and the system's fact database. It is responsible for obtaining complete information from the agents and for serving to them the data they ask for.

The reasoning module receives the information processed by the dialogue manager. It unifies data coming from different informers and detects contradictions in the system's image of the world, stored in the knowledge database. It is responsible for updating the dynamic data.

The knowledge database is used to store two types of information: the current state of the world, introduced by the informing agents during system operation, and the initial knowledge, describing (if known) the action setting, the map and so on.

The visualizer is an optional module that can be connected to the knowledge database in order to present the data in an additional mode. It is particularly useful in dynamic situations similar to the original Polint-112-SMS domain, where the security commander is able to assess the situation at one glance and to adjust their decisions to the current state of the world.

3 DIALOGUE STRATEGY

Grosz and Sidner (1986) name three *distinct but interacting* discourse components: the linguistic structure, the structure of intentions, and the attentional state.

The linguistic structure describes the sequence of discourse utterances. In our case the incoming information is in the textual form, originally in the form of SMS messages. One message can contain one or more sentences, with the premise that the sentence do not have to be complete. Unlike in many expert or conversational systems, it is not compulsory for the human user to answer all questions asked by the system, nor do they have to answer the questions in exactly the order they have been asked. It is one of the tasks of the dialogue manager (described further in this paper) to connect the respective questions and answers.

The list of possible user intentions in communication with the dynamic-domain expert system is limited by the functional requirements. The human agents may:

- introduce to the system new data about the state of the world,
- ask the system for confirmation of facts,
- ask the system *value* questions to obtain specific data,
- ask the system to inform them every time new information of a specified type arrives.

It is the task of the understanding module to detect the intentions, based on the syntax (indicative, imperative, interrogative) and the use of meta-predicates (*Inform me when...*).

As for the attentional state, the dialogue manager does not explicitly cut the dialogue into segments with the same attention scope. Instead, each time a decision on this matter is needed (mostly when an anaphoric reference occurs) it applies algorithms (presented below) finding the most probable target.

The remaining parts of this section describes the strategies and methods used by the dialogue manager and the reasoning module to respond to the users' needs.

3.1 Frame Completion

One of the most universal data structures for natural-language-based systems is the syntactic frame (or its variation) proposed by Fillmore (Fillmore, 1982). To create such frames the understanding module needs a dictionary (especially in highly inflected languages) and an ontology. The most complete solution is probably to combine a WordNet-like (Fellbaum, 1998; Vossen, 2002) ontology, very convenient for processing nouns and adjectives (although it may contain verbs too), with a verb ontology like FrameNet (Baker et al., 1998). The use of ontologies not only allows for the creation of correct data structures, but also plays a role in disambiguation: when multiple meanings of a given noun are found in the dictionary/ontology, a number of them can be excluded based on the verb's semantic constraints. The understanding module only works on sentence level and is ignorant of the discourse attentional state (as described in (Grosz, Sidner, 1986)). It detects the occurrence of inter-sentence referring expressions, but without looking for their targets. In some cases, like yes/no or proper-name answers, it creates data structures simpler than frames.

The frame (from now on referred to as *object*) obtained from the user's sentence is sent to the

dialogue manager. Let us consider the following example:

X bije się z Y na stadionie.

(X is in a fight with Y in the stadium.)

The understanding module should translates the sentence to an object similar to the one below (all examples are given in the LogTalk notation, native of the original Polint-112-SMS system).

```
obj25:fight
performer(0.9)=obj22:person
    pseudonym(0.4)=Y
performer(0.9)=obj12:person
    pseudonym(0.4)=X
localization(1) = obj23:localization
    space_rel(0)=in
    space_rel_arg(1)=obj24:stadium
```

Information about time and sender has been removed to clarify the listing. The understanding module created a `fight` object that has two filled attributes: `performer` (filled with two different `person` object values) and `localization`. The attributes that are not filled are not shown in the listing. Attribute values may be either atomic values (names `X` and `Y` or relation name `in`) or objects (`person`, `localization`, `stadium`). The numbers in parenthesis are attribute priorities, whose meaning is described below. Every attribute is defined as follows:

```
attrdef(Name,Type,Priority,Arity)
```

`Type` may be either `atomic` or a hierarchy class (e.g. `person`), `Arity` may be either `one` or `many`, depending on whether the attribute may have multiple values. Each attribute is given a priority value, defining the importance of this information in the object.

Having received the parsing result the dialogue manager performs the following operations:

- chooses the most probable semantic interpretation (irrelevant in this example as only one interpretation has been produced, see 3.6),
- updates the sender's profile and session information (see 3.5),
- checks if the received data may represent an answer to one of the previously asked questions (see 3.2),
- tries to merge the object with other objects in the sender's session (see 3.4),
- checks whether the reasoning module possesses more information about this object (e.g. the name and surname of a person with a given pseudonym, or information that the supposed pseudonym is in fact the person's surname) ,
- decides whether the sender should be asked to provide more data concerning the incoming object.

As seen in the example above, all frame attributes have priorities (with defined initial values that can be changed online during the system's operation if necessary). To check whether a question should be formulated and sent back to the human agent, the dialogue manager calculates the highest priority value of an empty frame attribute slot (excluding the slots that have already been question topics). In case of nested objects the priorities (with values ranging from 0 to 1) are multiplied. In the above example, if the frame `person` had an empty slot `surname` with priority `0.8`, the final priority value of the slot would be `0.9*0.8=0.72`. Finally the dialogue manager checks if the maximum priority value is higher than the compulsory attribute threshold value. If so, it creates a *question object* (marking the slot that is the topic of the question) and passes it to the text generator, responsible also for sending the information to the user.

Regardless of the final decision, the received object is always sent to the reasoning module.

3.2 Answers and Questions

The exchange of questions and answers in this dialogue model is asynchronous. The questions do not need to be answered in the same order they have been asked, and answering them is not compulsory. The informer is allowed to introduce to the system new data before attempting to answer a question. One of the reasons for allowing this asynchronicity was the original system's channel of communication, that is SMS messages. Early experiments (Walkowska, 2009) proved that it often happens that the informing agent receives a question while typing an unrelated message. The most commonly observed agent's behaviour is to finish typing the message, send it, and only then read the received one (and eventually answer it).

Five types of question/answer pairs are possible in the system:

- the user's question for confirmation and the system's immediate answer (yes/no),
- the user's question for value and the system's immediate answer (text generated from a list of encountered values or information about lack of data),
- the system's yes/no question and the user's answer,

- the system's *value* question and the user's answer,
- the system's question without an answer.

3.2.1 User's Value Questions

There is no question/answer pairing problem when the question are asked by the agents. The parser transforms the question to the very same form of objects (frames) that normal sentences are transformed, marking the sentence type as interrogative and (with value questions) the attribute that is the topic of the question. Let us consider the following question example:

Gdzie jest X?
(Where is X?)

The understanding module creates the following object:

```
question(obj28, [localization])
obj28:person
   pseudonym(0.4)=X
```

`[localization]` is the path to the attribute that is the topic of the question. The dialogue manager fills the questions when it is possible (e.g. solving anaphoric references as described in 3.3) and passes them to the reasoning module. The reasoning module translates the incoming object into a set of queries allowing it to find all possible answers. Simplifying a bit, the knowledge database is searched for objects that have the same (or unifiable) values in all attributes that the *template* (question) object has, but that also have a defined value in the attribute marked as the question topic. All such objects (if any) are passed back to the dialogue manager and the text generator.

3.2.2 User's Confirmation Questions

Confirmation questions (*Are there any fights in sector A?*) are very similar, but there is no additional-filled-attribute constraint. All matching objects are passed to the text generator. The answer to such questions may either be *no* or *yes*. In the latter case the textual representation of the encountered objects is also sent to the asking agent.

3.2.3 System's Value Questions

When the dialogue manager decides to ask a question (as described in 3.1) it saves the information in its memory. The stored data include: the informer's ID, the question object, the path to the slot that is the topic of the question, the time of asking the question, the number of messages exchanged with the agent since asking the question. the information whether the question is a *yes/no* question, and the information whether the agent has answered.

If there are unanswered questions in the dialogue manager's memory, the module checks all incoming messages for the possibility of being answers. Each received object is treated as potential answer for which a question has to be found.

The dialogue manager obtains a list of all unanswered questions that have been asked to the given human agent. Depending on system settings it may decide to reject the questions that have been asked too recently (and in some channels of communication, like SMS messages, it is not possible to create an answer so quickly).

Next, the dialogue manager tests whether it is possible to unify (see 3.4) the potential question and answer pair. If it is possible, it checks (preferably without performing the actual, costly unification, only checking the required paths) if the result of unification has a defined value in the attribute that is the topic of the question. Questions that do not satisfy this constraint are rejected.

If at this moment there are questions left, the dialogue manager chooses the most recent question. The question and answer objects are merged and the question is marked as answered. The resulting object is sent to the reasoning module.

If two similar objects have been passed at once by the parser (resulting from parsing the same sentence/message), and the first one turned out to be an answer to a question, the dialogue manager checks if both values may be merged into the question object. Consider the following dialogue example:

System: *Jakie przedmioty posiada X?*
(What objects does X possess?)
Agent: *Czerwony plecak, nóż.*
(A red backpack, a knife).

The code below presents: the question object, the two incoming potential-answer objects, the resulting object in which the required `[article]` path is filled with two object values.

```
question(obj100, [article])
obj100:person
pseudonym(0.4)=X

obj110:article
name(1)=plecak:1
colour(1)=obj120:colour
  name(0)=czerwony
```

```
obj130:article
name(1)=nóż:1

obj100:person
pseudonym(0.4)=X
article(0.1)=obj130:article
 name(1)=nóż:1
article(0.1)=obj110:article
 name(1)=plecak:1
     colour(1)=obj120:colour
      name(0)=czerwony:1
```

There are cases in which the dictionary and ontology used by the understanding module are not enough, and the module has to let a simple text string in to the system. A common case are *proper-name answers*: answers with only the names of the entities.

If the dialogue manager receives a proper-name string, it checks whether it has asked for data that can be answered in this way. If so, there are two possibilities:

- it is enough to paste the string as the question object's attribute value,
- an object needs to be created (e.g. a person object with a given pseudonym, surname etc.) and this object has to be merged into the question object.

When question and answer objects are merged, the procedure described in 3.1 is applied.

3.2.4 System's Confirmation Questions

The system asks confirmation questions only in one case: when it encounters contradictory data and wants the responsible agents to confirm or deny their versions. However, there is a group of questions that are not asked to confirm anything, but still the answer to them is *yes* or *no*. The group consists of *binary* questions, asked when the system needs to fill a binary attribute (*Does X have a beard?*).

The method applied by the dialogue manager to pair *yes/no* answers with their questions is very simple: the 'too recent' questions (as described in 3.2.3) are rejected, and then the newest *yes/no* question is chosen.

The method may seem error-prone, but corpus studies (Walkowska, 2009) prove that the agents, while trying to be as concise as possible, easily recognize 'dangerous' portions of dialogue that can lead to ambiguities. If for some reason the system asked two yes/no questions very close to each other it is probable that the agent will answer 'Yes, X has a beard.' or 'X has a beard.' instead of just 'Yes'.

3.2.5 Unanswered System's Questions

It may happen, and it is allowed in the dialoguing system's model, that human agents do not answer some of the questions they have been asked. They may be unaware of the fact of asking the question (problems with communication, noise, too many messages) or they may decide not to answer it, especially when they do not know the answer and sending a message is too costly.

The dialogue manager does not wait infinitely for an answer. After a set amount of time it assumes that the user is not going to answer a question. The question is then marked as answered, even though the corresponding object attribute remains empty. The procedure described in 3.1 is applied to check if there are more empty attribute slots worth asking for.

The dialogue manager executes the same behaviour after receiving an *I don't know* answer.

The mechanism applied when the unanswered question concerns a contradiction in the knowledge database is described in 3.6.

3.3 Anaphora

Anaphoric references are present in many languages, but the methods of detecting them differ. It is assumed here that the understanding module, working on sentence level, is able to properly recognize the intra-sentence references, as in:

Mężczyzna, który wszedł do sektora 5, ma wąsy.
(The man that entered sector 5 has a moustache.)

It is the understanding module's task to carry on the information that the man in sector 5 and the man with the moustache are in fact the same person.

The presence of inter-sentence backward anaphoric references can be detected by observing: articles (in the languages in which they exist, Polish not being one of them), pronouns, sentence subject omissions, cue words and others (Dunin-Keplicz, 1983, Walkowska, 2009). In this model, the parser is responsible for informing the dialogue manager that there is such reference, and the dialogue manager has to find the target. Let us investigate the following basic example of a message received from an agent:

Uderzył X.
[He] hit X.

Here is the corresponding object:

```
obj192:battery
patient(0.9)=obj192:person
```

```
    pseudonym(0.4)=X
performer(1)=obj191:person(ref)
```

The dialogue manager has to decide which one of the objects in the given agent's session is referenced as `obj191`. To do so, it retrieves all objects from the agent's session and rates them, awarding points for: time proximity, frame class concordance (classes close in the hierarchy), nesting concordance (in the example, a session `person` object will be given more points if it is also nested in a `battery` object), role concordance (active/passive, in the above case it is `performer` versus `patient`), and content concordance (if, what seems to happen rarely, the reference object contains data). Finally the object with the highest score is chosen. If there is more than one object that has been awarded the maximum score, the most recently modified (newest) object is chosen. It is merged with the reference object. If no similar objects are found in the session, the dialogue manager ignores the reference.

3.4 Data Unification

The unification and merging of data is not a dialoguing feature, but as the word unifiable appears often in this paper, a short explanation is in order.

Apart from some special-case rules, the idea is as follows. Two frame objects are unifiable if:

- they are objects of the same class, or one of the classes is a specialization of the other (`person`/`policeman`),
- the one-value attributes, if filled, contain identical or unifiable (for nested objects) data,
- the multiple-value attributes do not contain implicitly opposite values (e.g. the same value, but negated in one of the objects).

It is important to note that the word 'identical' in the above list has been used for clarity reasons. In fact, a more advanced checking mechanism is applied even for atomic values. In particular, the ontology is consulted for values being ontology items in order to detect synonyms or a hyponym/hypernym pair.

3.5 Informer Profiles

A number of users are allowed to contact the system to feed or retrieve data. The system does not need to know the list of agents in advance, but it has to be able to identify them (e.g. by means of their phone numbers, IP addresses – depending on the channel of communication) in order to manage the dialogues properly. The system may treat the users equally, but it does not have to. Some user data may be introduced in advance, for example if many users are allowed to ask for information, but only a smaller subgroup can introduce data. The system can also collect online statistics: the number of sent messages, the percentage of answered question, the average answer delay time, and so on, and use them to better adapt the dialogue to each agent.

3.6 Contradictions

One of the consequences of allowing multiple users to introduce information to the system is the possibility of data contradictions. There are two types of such contradictions: contradictions caused by one agent and contradictions in data introduced be two different people. In the former case, the system simply overwrites the data with newer values, assuming the situation has changed or the agent corrected themselves. In the latter, the system (the reasoning module, as the dialogue manager only works on one-dialogue levels) performs a more complicated action.

When the reasoning module detects a contradiction in data coming from two different agents (e.g. the differences in the physical description of a person), it temporarily sets the newest data as valid. Then it prepares two version of the information and sends it (through the dialogue manager and text generator) to the agents responsible for the clash. They are presented both versions and are asked to confirm whether they are still convinced of theirs. The table below presents the possible scenarios. *New* is the answer of the agent responsible for the newer (current) version, *Old* is the other agent's answer. It is the task of the dialogue manager to properly assign the *yes/no* answer to the contradiction question and pass it on to the reasoning module, but it is the latter that performs the operations on contradictory data.

Table 1: Possible contradiction scenarios.

Old	New	System's reaction
-/no	-/no/yes	Leave the current (new) version.
yes	-/no	Restore the previous version.
yes	yes	Set the version of the agent with higher credibility. If credibility values are equal, or are unknown, keep the new version.

An alternative solution for the situation with two *yes* (*I am sure*) answers is to un-merge the objects coming from the two agents, assuming they are not really talking about the same entity.

The *credibility* value mentioned in the table may be a part of the informer profile. It may be provided at system's initialization and/or calculated online (e.g. lowered every time the agent's data causes an unresolved contradiction).

3.7 Time Management

As the paper describes a *dynamic* expert system, some decisions about time management have to be explained. The knowledge database has to contains facts about the current situation. Two questions need to be posed here:

- Should the system also 'remember' information that has become obsolete?
- If nobody informs about the expiration of some data, when should the system conclude they are obsolete?

The decision about remembering obsolete information depends on the system's definition. If it has to answer questions about facts from the past, the information has to be kept. In this case the older objects (frames) can be transferred to a different data pool or simply be marked as expired. Old attribute values may be kept as values with negative certainty together with the time they expired or were overwritten. If this solution is applied (as in Polint-112-SMS), the expired values should not be taken into consideration when counting the number of values of an attribute (so an attribute whose definition only allows one value may have 0 or 1 normal value and an unlimited number of expired values).

If there are types of data (frames) that can expire in a dynamic situation, it is the reasoning module's task to make the correct decisions. It may apply rules according to which a frame of a given type (e.g. a frame representing a dynamic event, like a fight) is automatically expired by an internal thread when nobody has mentioned it for a given amount of time. It might also decide to ask the agents responsible for introducing the possibly-expired data for confirmation.

3.8 Desambiguation

There are situations in which the understanding module cannot decide on the proper semantic interpretation of a sentence. In some cases the semantic requirements of a frame are not enough to choose the correct meaning of a word. The module then proposes more than one interpretation to the dialogue manager. Depending on the importance of disambiguation (in a security expert system setting: both possible meanings are insignificant articles, or one of the meanings is a dangerous weapon) the dialogue manager may decide to ask for clarification (e.g. presenting other words from the same WordNet synset) or to wait for more information.

Another ambiguous case is that of short answers to system's questions. Short, proper-name answers are described in 3.2.3. However, it is not always possible for the parser to decide that the string it has received at its input is a proper-name answer. Some proper names, like nicknames and pseudonyms, often consist of common nouns or adjectives. The understanding module, operating at the level of one sentence, is unable to reach a conclusion when it is presented a one-word string that is present in the dictionary or ontology. Because of this, it creates two (or possibly more) possible interpretations. The dialogue manager chooses the correct one, checking the dialogue history (Does this information make sense when combined with other data?) and the questions asked to the given informer (Can the proper-name interpretation be an answer to one of the questions?).

4 EVALUATION

Polint-112-SMS, the expert system that originated this research, is functional and has been subjected to different series of tests and field experiments. All data exchanges between the users and the system have been saved and analysed, and additionally the users (including security professionals) have been asked to assess their experience of working with the system.

The system has been generally described as 'useful', but two interesting conclusions have been drawn after the analysis of the dialogues and user surveys. The first one is that there are some situations that change so rapidly that the users are not able to keep describing them. If, however, they do send the information, even after an event occurred, the system may still be used to recreate the event sequence. The other one is that the system must not ask too many questions, because they may become annoying for a user who cannot answer them all on time. The implementation conclusion is as follows: the frame attribute priorities have to be chosen carefully not to flood the agent with too many questions. Limits might be introduced on a number of questions that the system is allowed to send in a given amount of time. In some cases it might be advisable for the system to risk slight

errors in processing but keep from asking for confirmation or disambiguation of facts.

ACKNOWLEDGEMENTS

This work has been supported by Polish Ministry of Science and Higher Education, grant R00 028 02 (within the Polish Platform for Homeland Security).

Polint-112-SMS design and implementation team is comprised of Zygmunt Vetulani (chief), Piotr Kubacki, Marek Kubis, Jacek Marciniak, Tomasz Obrębski, Jędrzej Osiński, Justyna Walkowska, and Krzysztof Witalewski.

REFERENCES

Baker, C. F., Fillmore, C. J., Lowe, J.B., 1998. The Berkeley FrameNet Project. In *Proceedings of COLING-ACL'98* (pp. 86-90), Montréal: Association for Computational Linguistics.

Blaylock, N., Allen, J., 2005. A Collaborative Problem-Solving Model of Dialogue. In Dybkjær, L., Minker, W. (Eds.) *Proceedings of the SIGdial Workshop on Discourse and Dialog* (pp. 200-211), Lisbon: Association for Computational Linguistics.

Dunin-Keplicz, B., 1983. Towards better understanding of anaphora. In *Proceedings of 1st Conference of the European Chapter of ACL, E-ACL'83* (pp. 139-143), Pisa: Association for Computational Linguistics.

Fairon, C., Paumier, S., 2006. A translated corpus of 30,000 French SMS. In *Proceedings of LREC 2006*, Genova.

Fellbaum, C. D., 1998. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA.

Fillmore, C.J., 1982. Frame semantics. In *Linguistics in the Morning Calm* (pp. 111-137). Seoul, Hanshin Publishing Co.

Grosz, B. J., Sidner, C. L., 1986. Attention, Intentions and the Structure of Discourse. In *Computational Linguistics*, 12(3), 175-204.

Milward, D., Beveridge, M., 2003. Ontology-based dialogue systems. In *3rd Workshop on Knowledge and Reasoning in Practical Dialogue Systems IJCAI03* (pp. 9-18), Cambridge, MA: MIT Press.

Minker, W., Bennacef, S., 2004. *Speech and Human-Machine Dialogue. The Kluwer International Series in Engineering and Computer Science.* Massachusetts: Kluwer Academic Publishers.

Vetulani, Z., 2002. Question answering system for Polish POLINT and its language resources. In *Proceedings of the Question Answering - Strategy and Resources Workshop, LREC 2002*, Las Palmas de Gran Canaria.

Vetulani, Z., Marciniak, J., 2000. Corpus Based Methodology in the Study and Design of Systems with Emulated Linguistic Competence. In Christodoulakis, D. N. (Ed.), *Natural Language Processing - NLP2000, Second International Conference Proceedings* (pp. 346-357), Lecture Notes in Artificial Intelligence 1835. Springer Verlag.

Vetulani, Z., Marciniak, J., Konieczka, P., Walkowska, J., 2008. An SMS-based System Architecture (Logical Model) To Support Management of Information Exchange in Emergency Situations. POLINT-112-SMS project. In *Intelligent Information Processing IV. 5th IFIP International Conference on Intelligent Information Processing* (pp. 240-253). Springer Boston.

Vossen, P. (ed.), 2002. *EuroWordNet General Document*, Version 3 (Final) , University of Amsterdam.

Walkowska, J. 2009., Gathering and Analysis of a Corpus of Polish SMS Dialogues. In Kłopotek, M. A., Przepiórkowski, A., Wierzchoń, S. T., Trojanowski, K. (Eds.), *Challenging Problems of Science. Computer Science. Recent Advances in Intelligent Information Systems* (pp. 145-157). Academic Publishing House EXIT, Warsaw.