Reversibility and Re-usability of Resources in NLG and Natural Language Dialog Systems

Martin Klarner
3SOFT GmbH
Frauenweiherstr. 14, D-91058 Erlangen, Germany
martin.klarner@3soft.de

Abstract

Reversibility is a key to efficient and maintainable NLG systems. In this paper, we present a formal definition of reversible NLG systems and develop a classification of existing natural language dialog systems in this framework.

1 Introduction

Reversibility is a key factor in building efficient and maintainable NLG and natural language dialog systems (NLDSs). But previous formal descriptions of reversibility are still lacking in coverage and applicability to existing systems. In this paper, we extend former approaches to this matter by formally defining reversibility in NLDSs and developing a proper classification of such systems in terms of reversibility. After that, existing NLG and generic dialog systems are used as examples for the feasibility and applicability of our classification.

In our point of view, it is useless to consider reversibility for an NLG system alone, because parsing and dialog management are equally important for developing an NLDS. Hence, our classification applies to complete dialog systems and not only NLG systems.

2 A Formal Description of Reversibility

In this section, we will provide a formal definition of reversibility which is based on previous work [Neumann and van Noord, 1994]. To this end, we will first give a short overview of the results obtained there in sect. 2.1. After that, we will present our extended definition in sect. 2.2.

2.1 Previous definitions of Reversibility

In [Neumann and van Noord, 1994], a definition of reversibility for programs is provided. The authors start with a definition for computing a relation r in both directions (def. 1).

Definition 1. (Computing a relation in both directions according to Neumann and van Noord)

A program P **computes a relation r in both directions**, *iff for a given input $\langle dir, e\rangle$ it recursively enumerates the set*

$\{x \mid (\langle e, x\rangle \in r \wedge dir = 0) \vee (\langle x, e\rangle \in r \wedge dir = 1)\}$.

In this definition, the parameter *dir* denotes the direction in which the input is computed, and e represents the content of the input for which the appropriate output has to be obtained.

Let us state a simple corollary to def. 1 which relates the notion of computing a relation in both directions to the notion of inverse relations.

Corollary 1. *A program P computes the relation r in both directions, if P computes r and the inverse relation of r, r^{-1}.*

Proof. According to def. 1, P recursively enumerates the set $\{x \mid \langle e, x\rangle \in r\}$ for $dir = 0$ and the set $\{x \mid \langle x, e\rangle \in r\}$ for $dir = 1$. Hence, it computes r for $dir = 0$ and (using the standard definition of inverse relations) r^{-1} for $dir = 1$. □

Based on def. 1, the following definitions for r-reversibility of programs and relations are provided in [Neumann and van Noord, 1994] (def. 2).

Definition 2. (Reversibility of programs and relations according to Neumann and van Noord)

1. *A program P is* **r-reversible** *if it computes r in both directions.*
2. *A relation r is* **reversible** *iff an r-reversible program exists.*

The notion of reversible programs in def. 1 and 2 is very general: In an extreme case, such a program can consist of two completely independent parts tied together only by an initial conditional statement. This statement decides, depending on the value of the direction parameter *dir*, whether the program part computing relation r (for $dir = 0$) or the one computing r^{-1} (for $dir = 1$) is used. In our opinion, such a program should not be called reversible any more. Hence, definitions 1 and 2 are too general.

On the other hand, they are too specific; this is due to three reasons:

1. Program and data are not distinguished.
2. Thus, different resources and resource types[1] are also not addressed.

[1] such as linguistic and pragmatic resources

3. The availability time for a program or a resource[2] is not considered.

Hence, in the next section we will replace these definitions by a more general description of reversibility for generic program systems before we will describe reversibility in current NLDSs.

2.2 Extended definition of reversibility

In this section, we will present our definition of reversibility. We start with definitions of a generic program system and of system and program relations (def. 3).

Definition 3. (Program system, system relations, and program relations)

1. *A* **program system** S *consists of a triplet* $(COMP_S, PROG_S, RES_S)$ *of*
   (a) *a set of* **preprocessing programs** $COMP_S = \{C_1, \ldots, C_k\}$ *which are executed before system start,*
   (b) *a set of* **runtime programs** $PROG_S = \{P_1, \ldots, P_l\}$,
   (c) *and a set of* **resources** $RES_S = \{R_1, \ldots, R_m\}$.
2. *The set of relations* $REL_S = \{r_1, \ldots, r_n\}$ *computed by the programs of* $PROG_S$ *is called the set of* **system relations** *of* S.
3. *The set of relations* $REL_P = \{r_1, \ldots, r_p\}$ *computed by a single program* $P_i \in PROG_S$ *is called the set of* **program relations** *of* P.

By resources we denote every data structure needed by a runtime program for its execution[3]. More precisely, the resource R_{P_i,r_k} provides a necessary (but not sufficient) condition for the runtime program P_i to compute one of its program relations r_k. All of these resources must be available at system start, but they may be generated by preprocessing programs.

Before we can state our definition of reversibility, we have to give a formal description of inverse programs and resources (def. 4).

Definition 4. (Inverse program and inverse resource)

Let S *be a program system,* $R \in RES_S$ *a system resource, and* $P \in PROG_S$ *a program with a program relation* $r \in REL_P$. *Let* R *be a resource needed by* P *for computing* r.

1. *Then every program* P^{-1} *computing the inverse relation* r^{-1} *is called an* **inverse program** *to* P *with respect to* r.
2. *The transformation of a resource* R *needed by* P^{-1} *to compute* r^{-1} *is called* **inverse resource** R^{-1} *to* R *with respect to* r.

A simple corollary relates self-inverse programs to r-reversible programs.

[2] i.e. whether it is available only at runtime or already at compile time

[3] contrary to the terminology used e.g. in operating systems programming

Corollary 2. *If* $P \equiv P^{-1}$ *holds, i.e. if* P *is self-inverse with respect to* r, *then* P *is* r*-reversible.*

Proof. If P computes r, P^{-1} computes r^{-1}, and $P \equiv P^{-1}$ holds, then P computes r^{-1} as well. Then, according to def. 1, P computes r in both directions, and with def. 2 P is r-reversible. □

Algorithmic reversibility

For any program system of def. 3, we define algorithmic reversibility in the following way (def. 5).

Definition 5. (Algorithmic reversibility)

Let S *be a program system,* $P \in PROG_S$ *a program in* S, *and* $r \in REL_P$ *a program relation of* P.

Then S *is* **algorithmic-reversible** *in* P *and* r *if* P *is* r*-reversible.*

Hence, P (and no other program $Q \in PROG_S$ with $Q \neq P$)[4] has to compute r and r^{-1} as well.

Data reversibility

Data reversibility, the counterpart of algorithmic reversibility, can be defined as follows (def. 6).

Definition 6. (Data reversibility)

Let S *be a program system,* $R \in RES_S$ *a system resource of* S, *and* $r \in REL_S$ *a system relation of* S.

Then S *is* **data-reversible** *to* R *and* r *if two programs* $P_1, P2 \in PROG_S$ *exist which both need* R *to be executed and for both of which* $r \in REL_{P_1}$ *and* $r^{-1} \in REL_{P_2}$ *holds.*

Thus, P_1 must compute r using R, and P_2 must compute the inverse relation r^{-1} (also by using R). If $P_1 \equiv P_2 \equiv P$ holds, S is also algorithmic-reversible to P and r.

Static and dynamic reversibility

A different dimension of reversibility dealing with the availability time of a program or a resource can be described as follows (def. 7).

Definition 7. (Static and dynamic reversibility)

Let S *be a program system,* $R \in RES_S$ *a system resource and* $r \in REL_S$ *a system relation of* S.

1. S *is* **static-reversible** *with respect to* R *and* r *if*
   (a) *a program* $P \in PROG_S$ *with* $r \in REL_P$ *exists which needs* R *for its execution,*
   (b) *also* $r^{-1} \in REL_S$, $P^{-1} \in PROG_S$, *and* $R^{-1} \in RES_S$ *holds,*
   and additionally
   (c) *at least one preprocessing program* $C \in COMP_S$ *is needed for the construction of* R^{-1} *from* R *or of* P^{-1} *from* P.
2. *If no such program* C *is needed,* S *is called* **dynamic-reversible** *with respect to* R *and* r.

[4] By $Q \neq P$ we denote syntactic in-equivalence here. This is easily decidable, whereas semantic equivalence of programs is certainly not.

If, under the preconditions of def. 7, the inverse **program** P^{-1} is constructed, S is also algorithmic-reversible with respect to P and r. However, if the inverse **resource** R^{-1} is constructed, S is data-reversible with respect to R and r. Obviously, both algorithmic and data reversibility can occur simultaneously.

3 Reversibility in Dialog Systems

Consider the special relation $s_{p\to s}$ between phonetic and semantic structures. This is the relation computed by the analysis part of a natural language dialog system (NLDS). By applying our definitions of reversibility presented in sect. 2.2 on $s_{p\to s}$, we face an important question of natural language processing: To what extent is a given NLDS reversible? But before we consider this question in more detail, we have to define our notion of an NLDS first. Based on def. 3, we formally describe an NLDS as follows (def. 8).

Definition 8. (NLDS)
Let $r_{p\to s}$ be the relation between phonological and semantic structures and $r_{p\to s}^{-1}$ the inverse relation of $r_{p\to s}$.[5]
An **NLDS** *is a program system S with $r_{p\to s} \in REL_S$ and $r_{p\to s}^{-1} \equiv r_{s\to p} \in REL_S$.*

Hence, an NLDS must contain both the relations $r_{p\to s}$ and $r_{s\to p}$ as system relations. This is quite obvious, since natural language dialog requires both natural language understanding (NLU) and natural language generation (NLG).

4 Classification of Reversibility Types

As we have seen in the previous sections, generic program systems and NLDSs in particular can be reversible in two independent dimensions: On the one hand, they can be static or[6] dynamic, and on the other hand, algorithms and/or data can be reversible. Given that a system may also be not reversible at all in both dimensions just mentioned, we obtain a classification of nine possible reversibility types.

[Neumann, 1994], however, describes just four types of reversibility in dialog systems and takes only the grammar as a linguistic resource into account: Type A has static reversibility (in terms of data and algorithms), while type B has dynamic data reversibility. Type C has statistically reversible data and dynamically reversible algorithms, while type D has dynamic data and algorithmic reversibility.

By further exploring the notions of algorithmic and data reversibility introduced above, both of which can be realized in three different variants (none, static, and dynamic), we are able to extend the classification in [Neumann, 1994] by two more types: Type E is statically reversible in terms of data and algorithms, and type F has dynamic data and static algorithmic reversibility. Our extended classification of reversible dialog systems is depicted in fig. 1.

There are three more possible types in our classification, all of them without data reversibility: Type G has statically and type H dynamically reversible algorithms, whereas type I does not have any reversibility at all. While types G and H are just not desirable for real-world NLDSs, type I is even unacceptable. Hence we decided to exclude types G, H, and I from fig. 1 and depict them separately in fig. 2. However, the legend displayed there applies to fig. 1 as well.

It has to be pointed out here that any classification of reversible dialog systems must not be restricted to the grammar, but has to be extended to the other resources used in an NLDS as well. Apart from the grammar, we distinguish five additional system resources: Lexicon and morphology component are linguistic resources (together with the grammar), whereas discourse memory, domain model, and user model are pragmatic system resources. Hence, the reversibility of an NLDS can be classified depending on (at least) six different resource categories. Together with the six reversibility types introduced above, these six resources form a 6-tuple which enables us to describe the reversibility of an NLDS formally and completely.

Let us take the CONALD dialog system [Ludwig, 2003] as an example. The system lexicon is precompiled into an NLG lexicon at development time, hence we have static reversibility of type E here. On the other hand, the morphology component is used by both the parser and the generator at runtime in a uniform way (cf. [Klarner and Ludwig, 2004]), resulting in dynamic reversibility for this component. Discourse memory and domain model are used in the dialog manager for pragmatic integration and by the NLG component. The data structures are identical, but the algorithms are different. Thus, we have type B reversibility for these two resources. The user model, however, is not used for parsing, only for generation, hence the system is not reversible with respect to the user model.

In table 1 the reversibility types of the different resources are put together. They form a tuple (E, D, A, B, B, none) completely describing reversibility in CONALD.

Resource	**Type**
Lexicon	E
Morphology	D
Grammar	A
Discourse Memory	B
Domain Model	B
User Model	none

Table 1: Reversibility of CONALD.

The AMALIA system [Gabrilovich *et al.*, 1998] is a typical example for PROLOG-based reversible NLG systems. The system grammar is first inverted and then compiled into two different versions, one for parsing and one for generation. Thus, we have type C reversibility here. The

[5] Henceforth, we will denote $r_{p\to s}^{-1}$ just $r_{s\to p}$ for obvious simplicity reasons.

[6] The "or" here must be read as an "exclusive or".

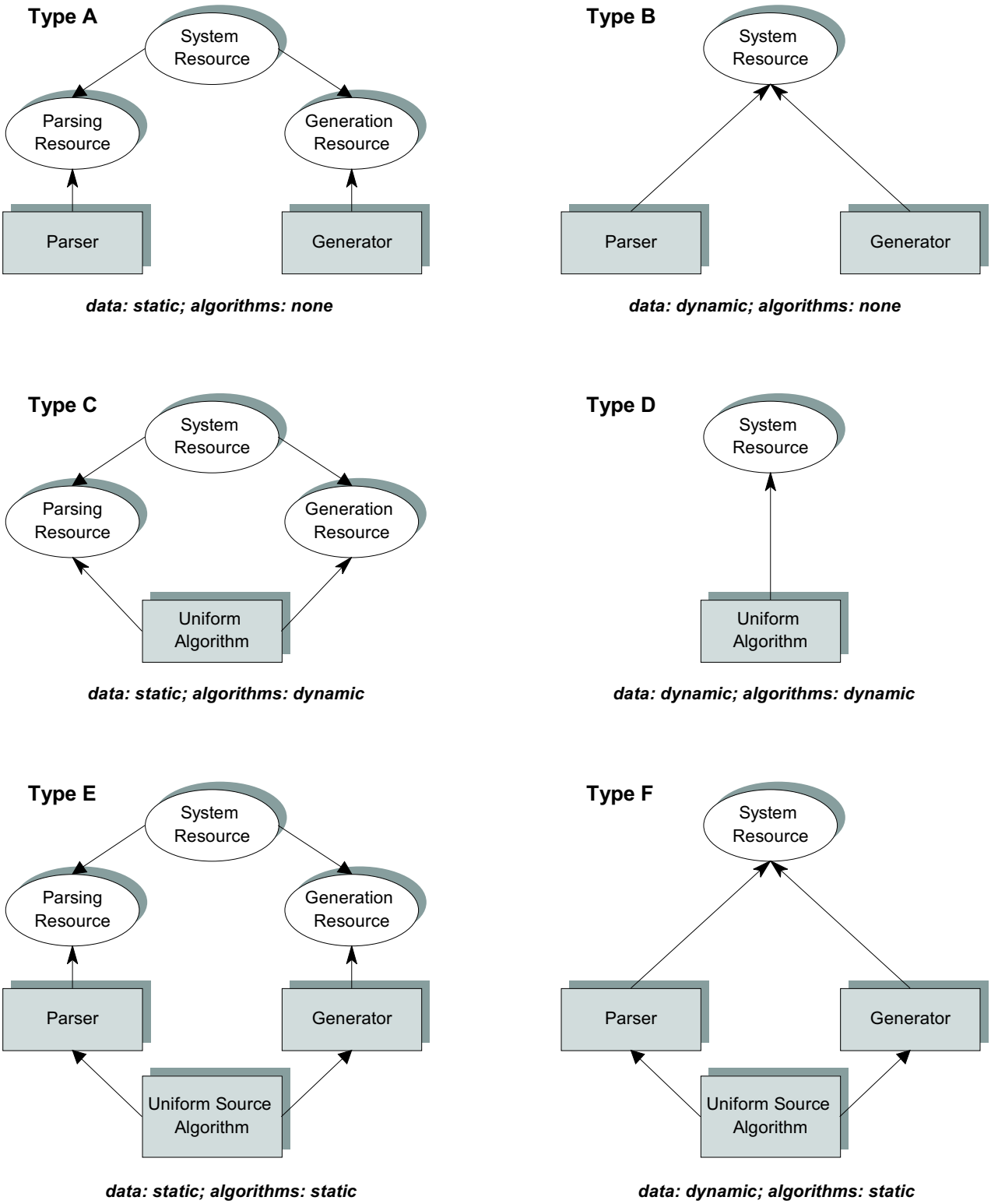
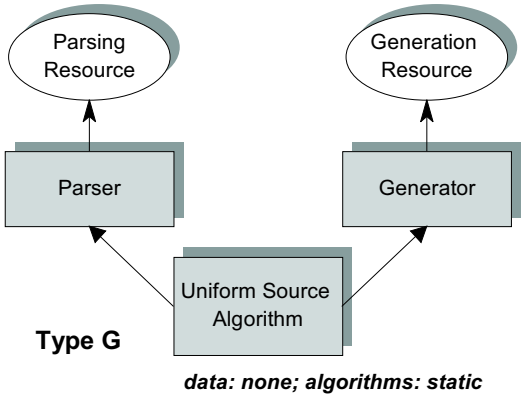
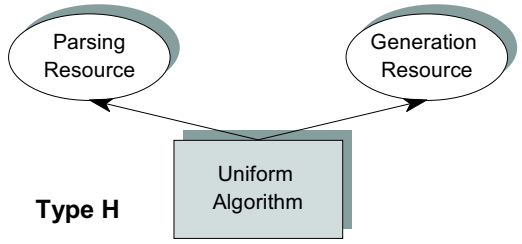
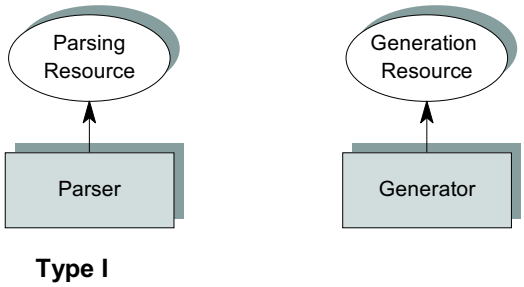


Figure 1: Reversible dialog systems.







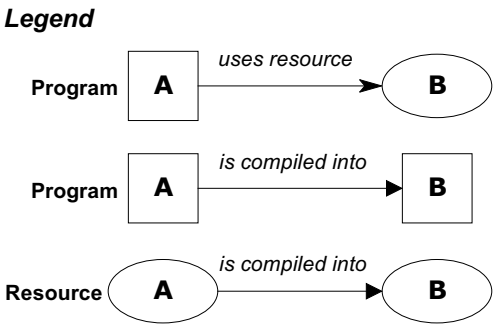


Figure 2: Not-so-reversible dialog systems.

same applies to the lexicon. As there are no pragmatic resources and no morphology component, we can skip their analysis here. Hence, AMALIA can be characterized by the reversibility tuple (C, n/a, C, n/a, n/a, n/a); cf. table 2.

Resource	Type
Lexicon	C
Morphology	n/a
Grammar	C
Discourse Memory	n/a
Domain Model	n/a
User Model	n/a

Table 2: Reversibility of AMALIA.

Our third and final example is TRIPS [Ferguson and Allen, 1998]. In this system, the Discourse Context and the Reference component are shared between the Interpretation Manager (which is used for parsing) and the Generation Manager (cf. [Allen *et al.*, 2001]). This results in type B for the discourse memory. The same holds for the ontology of TRIPS (cf. [Stent, 2001], p. 139): Its domain model is of type B as well. As there is no specific user model contained in the system, there is also no degree of reversibility to be found there. For various reasons, the Generation Manager uses its own grammar and morphology component (cf. [Stent, 2001], p. 180 & 182). The NLG lexicon of TRIPS is obtained semi-automatically from various system resources and off-line extraction (cf. [Stent, 2001], p. 180). Hence, we have type A reversibility here. We therefore conclude that TRIPS can be described by the reversibility tuple (A, none, C, B, B, n/a); cf. table 3.

Resource	Type
Lexicon	A
Morphology	none
Grammar	none
Discourse Memory	B
Domain Model	B
User Model	n/a

Table 3: Reversibility of TRIPS.

5 Re-usability as Static Reversibility of Resources

Given our definitions of reversibility in sect. 2, we can view re-using resources in an NLDS as static or dynamic reversibility of the system for these resources. Compared to the definition in [Neumann and van Noord, 1994] referred in sect. 2.1, this is a more general definition which can be applied to a lot of existing NLDSs.

Let us again use the CONALD system as an example, this time only taking the data structures into account, in order to search for possible re-use of resources. Two core linguistic resources of its parsing branch are re-used

in its NLG component HYPERBUG [Klarner and Ludwig, 2004]: The system lexicon and the morphology component are both used by the parser and the generator, with static reversibility for the system lexicon and dynamic reversibility for the morphology component. As mentioned in sect. 4, re-use is also done for the pragmatic resources, namely discourse memory and domain model.

Generally speaking, the more linguistic and pragmatic resources are re-used in an NLDS, the higher its degree of reversibility becomes, and the more efficient the system will be to develop and maintain.

6 Conclusion and Further Work

We have developed a formal description of reversibility for NLDSs, using definitions for program systems, system relations, and system resources. Based on these definitions, we have presented a classification of reversible NLDSs in general and NLG systems in particular. Our classification extends previous approaches in three dimensions: First, it covers static and dynamic reversibility, second, it considers algorithmic and data reversibility, and third, it takes the different resources of a dialog system into account.

The 6-tuple used in our classification can, of course, be extended to incorporate different linguistic and pragmatic resources, should they prove useful for an NLDS. However, we identified the set of resources mentioned above by thorough investigation of existing systems based on the results presented in [Maier, 1999] for text planning; presently, we do not think we need additional ones.

Unfortunately, our definition of reversibility does not yet completely reflect all aspects of current NLDSs: For example, it does not cover systems where preprocessing and runtime programs cannot be clearly separated, because such systems allow a flexible choice for a given resource and/or algorithm to be computed beforehand (by preprocessing) or at runtime.[7] This extended degree of dynamic has yet to be taken into account in our definitions.

The obvious practical application of our classification is twofold: First, using it in a descriptive way to analyze existing systems. Second, and more practical, using it in a *normative* way to further develop one's one NLDS to be as reversible as possible (i.e., to obtain a "D" in all six positions of the 6-tuple of reversibility types). Both applications are important, but the second is the one we are going to pursue in the near future.

Acknowledgments

Most of the work described here was done while completing my PhD thesis [Klarner, 2005] at the Chair for Artifical Intelligence of the University of Erlangen-Nuremberg. This is why I want to thank my former colleagues there, especially Bernd Ludwig and Peter Reiß, for their enduring cooperation and support. Many thanks to the anonymous reviewers as well for providing very helpful comments to the initial version of this paper.

[7] While such systems are certainly an attractive theoretical possibility, we are not aware of real-world existing ones so far.

References

[Allen *et al.*, 2001] J. Allen, G. Ferguson, and A. Stent. An architecture for more realistic conversational systems. In *Proc. 6th Int. Conf. on Intelligent User Interfaces (IUI-2001)*, pages 1–8, Santa Fe, 2001.

[Ferguson and Allen, 1998] George Ferguson and James Allen. Trips: An intelligent integrated problem-solving assistant. In *Proc. AAAI-98*, Madison, WI, 1998.

[Gabrilovich *et al.*, 1998] Evgeniy Gabrilovich, Nissim Francez, and Shuly Wintner. Natural language generation with abstract machine. In *Proc. INLG-98*, Niagara-on-the-Lake, 1998.

[Klarner and Ludwig, 2004] Martin Klarner and Bernd Ludwig. Hybrid natural language generation in a spoken language dialog system. In Susanne Biundo, Thom Frühwirth, and Günther Palm, editors, *Proc. KI-2004*, pages 97–112, Ulm, 2004.

[Klarner, 2005] Martin Klarner. *Hybride, pragmatisch eingebettete Realisierung mittels Bottom-Up-Generierung in einem natürlichsprachlichen Dialogsystem.* PhD thesis, Erlangen-Nürnberg, 2005.

[Ludwig, 2003] Bernd Ludwig. *Ein konfigurierbares Dialogsystem für Mensch-Maschine-Interaktion in gesprochener Sprache.* PhD thesis, Universität Erlangen-Nürnberg, 2003.

[Maier, 1999] Elisabeth Maier. *Entwurf und Implementierung von Wissensquellen für die Textplanung – eine modulare Architektur.* Berlin, 1999.

[Neumann and van Noord, 1994] Günther Neumann and Gertjaan van Noord. Reversibility and self-monitoring in natural language generation. In Tomek Strzalkowski, editor, *Reversible Grammars in Natural Language Processing.* Boston, Dordrecht, London, 1994.

[Neumann, 1994] Günter Neumann. *A Uniform Computation Model for Natural Language Parsing and Generation.* PhD thesis, Universität des Saarlands, 1994.

[Stent, 2001] Amanda J. Stent. *Dialogue Systems as Conversational Partners: Applying Conversation Acts Theory to Natural Language Generation for Task-Oriented Mixed-Initiative Spoken Dialogue.* PhD thesis, University of Massachusetts Amherst, 2001.