

Incremental Event Conceptualization and Natural Language Generation in Monitoring Environments

Markus GUHE, Christopher HABEL, Heike TAPPE
Research Group Knowledge and Language Processing (WSV),
Department of Informatics, University of Hamburg
Vogt-Kölln-Straße 30
22527 Hamburg, Germany, D-22527
{guhe, habel, tappe}@informatik.uni-hamburg.de

Abstract

In this paper we present a psycholinguistically motivated architecture and its prototypical implementation for an incremental conceptualizer, which monitors dynamic changes in the world and simultaneously generates warnings for (possibly) safety-critical developments. It does so by conceptualizing events and building up a hierarchical knowledge representation of the perceived states of affairs. If it detects a safety problem, it selects suitable elements from the representation for a warning, brings them into an appropriate order, and generates incremental preverbal messages (propositional structures) from them, which can be taken by a subsequent component to encode them linguistically.

1 Introduction

Systems that generate natural language descriptions of what happens in a dynamically changing world can be improved substantially by working incrementally. Incrementality enhances the overall quality of the systems for three reasons: (1) The dynamic nature of a continuous stream of input information can be handled more directly and, therefore, easier. (2) Incremental systems are capable of producing fluent speech, i.e. speech without artificial auditory gaps. (3) Parallelism that comes with incrementality makes better use of the available resources.

Furthermore, Reiter (1994), who reviews the architecture of some models of natural language generation, shows that psycholinguistic and engineering approaches often result in systems, which are similar in crucial respects. In this paper we ground on two of these common aspects, namely the distinction between *what-to-say* and *how-to-say* (De Smedt, Horacek & Zock, 1996) and the use of a *pipeline* architecture, which divides the

generation process “into multiple modules, with information flowing in a ‘pipeline’ fashion from one module to the next” (Reiter, 1994). Reiter states that these architectures do not require modules to work in parallel; if parallelism is used one has an *incremental* model, cf. De Smedt & Kempen (1987), Ferreira (1996).

The primary research topic of the ConcEv¹ project is the *what-to-say* component, in which the content of utterances is planned (Reiter’s *content planning*, in contrast to the language specific *sentence planning* component). We use the terminology of Levelt (1989), who calls the first component the *conceptualizer*, the second the *formulator*. These modules interact via *preverbal messages*, which are propositional, non-verbal representations of the utterance built up by the conceptualizer. They are transformed by the formulator into linguistic structures for spoken or written output. Besides considering high level communicative goals (*macroplanning*), which are in the focus of most computational approaches to the *what-to-say* component, e.g. De Smedt & Kempen (1987), McKeown (1985), Hovy (1993), Chu-Carroll & Carberry (1998), Radev & McKeown (1998), the type of information to be verbalized also determines the processes of conceptualization on the level of *microplanning*, cf. Levelt (1989). Thus, the traditional top-down approaches have to be combined with bottom-up data-driven approaches of text planning (Marcu, 1997). The conceptualizer that is described in detail in section 3 fits the pipeline architecture on a coarse level, but integrates on finer levels the ideas of functional modules (Cahill et al., 1999).

In the present paper we focus on the task of

¹ ConcEv (Conceptualizing Events) is supported by the DFG (German Science Foundation) in the priority program ‘Language Production’ under grant Ha-1237/10 to Christopher Habel.

generating verbal descriptions of continuously incoming input from a changing physical world (see section 2, for similar settings cf. Neumann & Novak (1983) and André, Herzog & Rist (1988)). This specific task requires an incremental pipeline architecture—as there are certain steps that have to be carried out in a specific order—and, additionally, these steps can be organized in such a way that a sequence of clear-cut modules arises, which can work in parallel. Parallel processing has the advantage that several tasks can be done simultaneously; thus, while utterances are generated for some input, subsequent input can already be taken in and processed.

Simultaneous conceptualization can be used as the basis of systems producing verbal messages when they detect a (possibly) safety-critical development while monitoring a safety-critical system, like intensive care units, nuclear power plants, or airports. A module for the generation of natural language can be an effective enhancement for monitoring for mainly two reasons: first, in most cases operators are busy observing many displays. Here the auditory presentation of information can make use of idle cognitive resources of the operators and, thus, reduce their workload in directing their attention to a development that may lead to hazardous situations.² Second, the essential piece of information can be extracted from a highly complex set of multimodal information and presented by the system in a crisp way. Language is the best conceivable means to transfer information as pointedly as possible. Moreover, taking the dynamics of the permanently changing world into account has the advantage that safety-critical situations can be anticipated earlier and much more reliably. Conventional systems, in contrast, just compare actual measurements with allowed values and give a warning or an alarm when a violation occurs. But it is more useful, e.g. for a nurse if the system tells her that a patient's blood pressure is rapidly dropping than that his blood pressure is already dangerously low.

We see the proposed implementation of an incremental conceptualizer also as a means to gain

insights into psycholinguistic aspects of natural language processing. Our implementation thus simulates aspects of behavior, e.g. the effects differing time pressure has on verbalizations.

2 Conceptualizing Events

If the system has to produce descriptions about what it 'sees', the main conceptual task is building up conceptual entities representing spatio-temporal constellations of the external world, i.e. *event conceptualization*. Events emerge from dynamic input data, which are segmented by the conceptual system into meaningful units (Avrami & Kareev, 1994). They are therefore internal representations rather than external entities: "[...] events arise in the perception of observers" (Zacks, 1997). Consequently, a language production system designed for verbalizing what the system perceives has to deal with information stemming from multiple modalities, e.g. auditory and spatial. In particular, a continuous multimodal 'perceptual stream' has to be translated into discrete propositional output (preverbal messages) that can be encoded linguistically, cf. Levelt (1989). To meet such demands, three subtasks have to be solved: (1) The input stream has to be subdivided into 'perceptual units'; (2) conceptual representations have to be built up from these 'percepts', which (3) have to be combined to preverbal messages. For the time being, we take the input stream to be strictly sequential, but later versions of our model will compute simultaneous events, as well.

According to Habel & Tappe (1999) the function of the conceptualizer can be subdivided into the following processes: *segmentation & grouping*, *structuring*, *selection*, and *linearization*. The first process operates on the (perceptual) input that is segmented into meaningful basic units (*segmentation*), and—if possible—two or more of these units are grouped together to form more complex entities (*grouping*). The *structuring* process builds up multilevel hierarchical structures from these meaningful basic units.

To exemplify these steps we use the scenario of monitoring the taxiing of an aircraft, viz. the movements of an aircraft from the terminal to its assigned runway and vice versa. Air traffic controllers who guide the movements of aircraft on the ground (surface movement controllers, SMC) have to rely mainly on visual information—either looking out of the window of the control tower or getting information from a airfield control moni-

² The multimodal monitoring environment proposed here reflects the division of labor between the components in working memory (Baddeley, 1986), especially between the visuospatial sketchpad (VSSP) and the phonological loop. Since the observation of multiple display units puts a heavy strain on the VSSP, spoken natural language as input of critical information would use that subcomponent of working memory, namely the phonological loop, which is less strained.

tor—and on communication with the aircraft crews. Yet, in some conditions, e.g. in low-visibility, this method is not failsafe (although reliable). It forces the crews to decrease speed—and such increases number and duration of delays—but it also results in greater safety risks.³ A supporting system that monitors the occurrences on the taxiway can mitigate these effects.

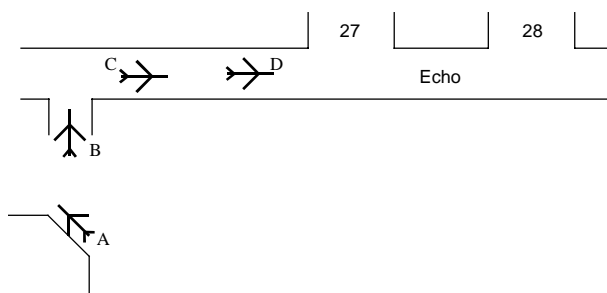


Figure 1. Monitoring the Taxiing of an Aircraft: Phases of a Complex Event

We will demonstrate the workings of our model of an incremental conceptualizer, which produces natural language messages for the SMC, with the example depicted in Figure 1. The flight with the number CK-314 shall taxi from the terminal via taxiway Echo to runway 27. The initial position of the airplane is A. It then starts to move until it reaches position B right before a junction where it has to stop and wait until the way is clear before moving on. It then starts again and continues (C) but its velocity is too high at point D. Consequently the plane might not be able to branch off at the junction, where it is supposed to turn left into runway 27. At that point the monitoring system generates a warning that the plane is in danger of missing the junction. (If the plane indeed misses the junction, an alert is generated, but our example does not include this.)

For this task two kinds of information have to be available: the *planned* movements of the plane and its *actual* movements. While the former information could be handled directly by the conceptualizer because they are inherently discrete, the latter are information about a continuously changing world. Here the perpetual continuous input stream has to be transformed into discrete items. This process consists, on the one hand, of *segmentations* into discrete units, e.g. a STOP

event, and on the other hand of some *groupings*. The movement from position B to position C, for example, contains—at least—three sub-phases corresponding to a straight, a curved and a second straight section of the trajectory. These three phases can be distinguished by segmentation, but are combined by a grouping. Furthermore, the *structuring* process has to build up the different phases to form the TAXI event, cf. Figure 3.

The third of the above-mentioned sub-processes, *selection*, has two functions: first, it detects that there is a conflict or a (possibly) safety-critical development or situation and decides that a warning has to be generated. Second, it selects the required information for a suitable warning or alert. Since the verbal warning can be given on different levels of detail, it is necessary to select appropriate events from the event hierarchy for further verbalization. On the one hand, it would not be adequate to produce a general warning like “Taxiing problem”—except perhaps when there is not enough time or no more information available at that moment—on the other hand, it would not be suitable to give an in-depth description of each part of the taxiing. Finally, the selected items are brought into an appropriate order by the forth process, *linearization*.

Before we describe the internal structure of the conceptualizer in section 3, we want to discuss the core idea of ‘event conceptualization’ in more detail. The first question to be answered is how a continuous (perceptual) input stream can be segmented into separate events. According to the *cut hypothesis* of Avrahami and Kareev (1994, p. 239), “A sub-sequence of stimuli is cut out of a sequence to become a cognitive entity if it has been experienced many times in different contexts.” This segmentation takes place in the ‘eye of the observer’. Hence, event conceptualization partly depends on individual as well as on contextual factors.

The idea of the cut hypothesis implies the existence of *basic events*, which are the building blocks in our experience used to trigger segmentation. They are minimal conceptual entities human observers ascribe a beginning and an end to. Thus, they are perceptual wholes—although they may have an internal structure—and are therefore the basis for the interface between perception and cognition. Basic events can be grouped together to form *complex events*, e.g. assuming that the four basic events GRIP THE HANDLE OF A WINDOW, TURNING THE HANDLE, PULL, and LET GO THE HANDLE are perceived, the complex event

³ The reports of incidents and accidents of the Australian Bureau of Air Safety Investigation is a rich source of occurrences that should not happen in civil aircraft operations.

OPENING A WINDOW can be built up. Furthermore, subsequent events of opening all windows of a room can be grouped to AIRING. But events can not only be grouped but also segmented: if the event OPENING A WINDOW is perceived, it can be segmented into the respective sub-events. We assume that hierarchical event structures, which are based on knowledge about the internal structure of prototypical events, e.g. in the format of scripts Schank & Abelson (1977), are the representational backbone of event conceptualization, cf. Habel & Tappe (1999).

3 An Incremental Conceptualizer

Incremental processing is the ‘piecemeal’ and parallel processing of a sequential information stream. It is a specific kind of parallel processing in that the processes have a fixed order, which De Smedt & Kempen (1987) describe as a ‘cascade of processes’, in analogy to a water cascade. This metaphor means that, for example, the grammatical encoding—including lexical access—of an utterance segment cannot take place until the information ‘splashes down’ from the conceptual encoding process. Figure 2 sketches such a cascade of dependent parallel processes in our model of the conceptualizer: The cascade consists of the processes *construction*, *selection*, *linearization*, and *pvm-generation* (preverbal-message-generation). These processes also constitute a pipeline in Reiter’s (1994) sense, but they do work in parallel.

One central parameter of incremental processing, which is highly relevant for the format of preverbal messages, is the size of the increments. Assume that a description (no warning this time) of the turning of the flight number CK-314 into taxiway Echo shall be given. This could be done by a proposition like *turn(ck314, goal(tw-echo))*, which is a potential increment for a preverbal message. Yet, such a proposition would have to be built up completely, before the subsequent components can begin forming it into a sentence like ‘Flight CK 314 turns into taxiway Echo.’ Hence the formulator could not start processing the first element, say *turn*, as soon as it is received from the conceptualizer. In contrast to this, we opt for an architecture, in which the selection of appropriate lemmas from the lexicon can start for parts of a preverbal message, before other entities are built up on the preverbal message level.

As a consequence, the dynamics in incremental processing demands a modified notion of preverbal messages. We conceive of them no longer as

complete propositions—as is mostly the case in approaches combining Levelt’s ideas with conceptual semantics—but as sequences of well-formed propositional structures on a sub-propositional level; in logical terminology: predicate symbols, functional expressions, terms, etc. The incremental formulator SYNPHONICS, which takes specific well-formed parts of propositions as input, follows these principles (Abb et al. 1996).

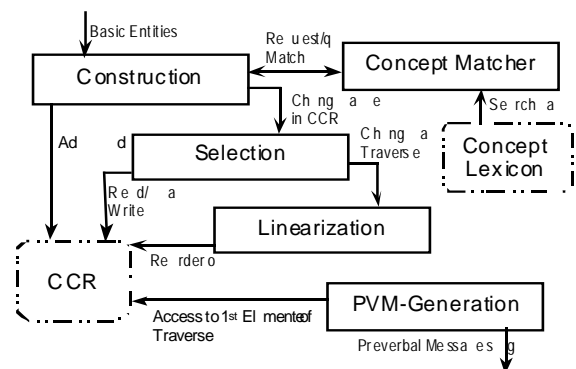


Figure 2. Model of an Incremental Conceptualizer

3.1 Coarse Architecture

In short, our conceptualizer performs the task ‘Give warnings about (possibly) safety-critical developments and situations!’ It operates on two different input streams: a discrete one, which contains the plans for the movements of the aircraft on the ground, and a continuous one, which originates in the sensors distributed over the taxiway. Since the conceptualizer cannot directly operate on the continuous input stream, these input information must be converted into a stream of discrete *basic entities*, which are *basic events* in this case. In our example a basic event is induced by sensoric data sent to the monitoring system, e.g. that a particular aircraft passes its position. Since the other input stream is already discrete, it simply has to be adapted to the required input format of the conceptualizer, i.e. it has to be converted into basic events, as well. We will neglect this process and concentrate on the continuous input stream.

Based on Habel & Tappe (1999) we propose a model of the conceptualizer as depicted in Figure 2. It consists of incremental (cascaded) processes that work on the blackboard-like *current conceptual structure* (CCR). At first sight, the use of a data structure, to which more than one process has access, seems to collide with the notion of a cascaded information stream. These pro-

cesses are interdependent in such a way, however, that they indeed behave incrementally; e.g. the selection process cannot select anything that has not been inserted into the CCR (constructed). The CCR can be seen as a shared memory unit with a common data structure. A third kind of information is needed for a representation of the state of affairs: the constellation of the terminals, taxiways, runways, and the participating object(s), or, more generally: the spatial arrangement of the world and information about objects in it. For example, there is one node that stands for flight CK-314, and all the nodes shown in Figure 3 are linked to it via an *actor* relation. Since this type of information is not in the focus of the present paper, we will not discuss it the following.

In addition to the cascaded processes there is a *concept lexicon*, accessible via a *concept matcher*: these modules, which are called by the construction process, find best matches for structures that can either be subsumed by a more complex concept or may represent still incomplete concepts. The first is necessary to build up hierarchical structures at all. The second is needed for the generation of expectations about developments in the near future. When, for example, flight CK-314 is at position D, the expectation is generated that it will go on straight at the next junction or that it will be unable to turn left at the next junction when keeping the current velocity.⁴ On the other hand, after the two nodes $START_1$ and $CHPOS_1$ (Figure 3) are constructed, these are given to the concept matcher for a subsumption test, which consists of trying to match the nodes onto more complex concepts. This yields that they can be joined together to a MOVE node ($MOVE_1$). Thus, it informs the construction process that a STOP event ($STOP_1$) will probably occur in the near future, which illustrates the second function of the matcher: the generation of expectations. (Even the last MOVE of a sequence of MOVE events contains a STOP event, because aircraft stop at the beginning of the runway, which is the last event of the taxiing, before they commence the takeoff.) The construction process inserts these two new nodes together with the information that the $STOP_1$ node is just a hypothesis up to now, nothing actually perceived.

The four cascaded processes that constitute the ‘heart’ of the conceptualizer and that will be described in more detail in the following sections

are:

- (1) construction
- (2) selection
- (3) linearization
- (4) pvm-generation

The first process comprises the processes segmentation & grouping as well as structuring of Habel & Tappe (1999), apart from the segmentations that are already done in the pre-processing step. The selection and the linearization processes correlate to the ones in Habel & Tappe (1999), thus, the first selects nodes for verbalizations, while the second brings them into an appropriate order. The pvm-generation is an additional process and guarantees that the selection as well as the linearization have some time to change (the order of) the selected nodes, before they are passed on to the formulator. We call this time span the *latency time*.

For the implementation of this architecture and (a first version of) the algorithms we use a formalism called *referential nets* (Habel, 1986), which was developed to represent linguistic as well as common sense knowledge. Entities are represented by *referential objects* (*refOs*), which can be connected via *relations*, so that a network structure arises. The basic entities the pre-processing component produces already contain some information about what attributes (e.g. which sort) have to be ascribed to a refO. In the following we use symbolic constants to refer to refOs. These are just arbitrary labels; the important point is that the refOs can be related to suitable refOs of subsequent processes, which, for example, stand for lexical items.

3.2 Construction

The *construction* process takes basic entities as input and builds up a hierarchical knowledge representation of the perceived states of affairs in the CCR. In the domain we discuss here, three relations are especially relevant for the representation of events: (temporal) inclusion (\sqsubseteq), temporal precedence (\prec), and the match of planned events onto actual events (μ). For the example described above the sub-net of the referential net that contains the actual events (the ones that have sort A-Event) is depicted in Figure 3 (the velocity problem is just detected). $MOVE_2$, for example, is temporally included in the event TAXI ($MOVE_2 \sqsubseteq TAXI$), the event $MOVE_1$ is the temporal predecessor of $MOVE_2$ ($MOVE_1 \prec MOVE_2$), a matching between a planned and an actual event is $\mu(MOVE_1, MOVE_{p1})$, where $MOVE_{p1}$ stands for the planned

⁴ The computation of the velocity is easily done from the sensoric data.

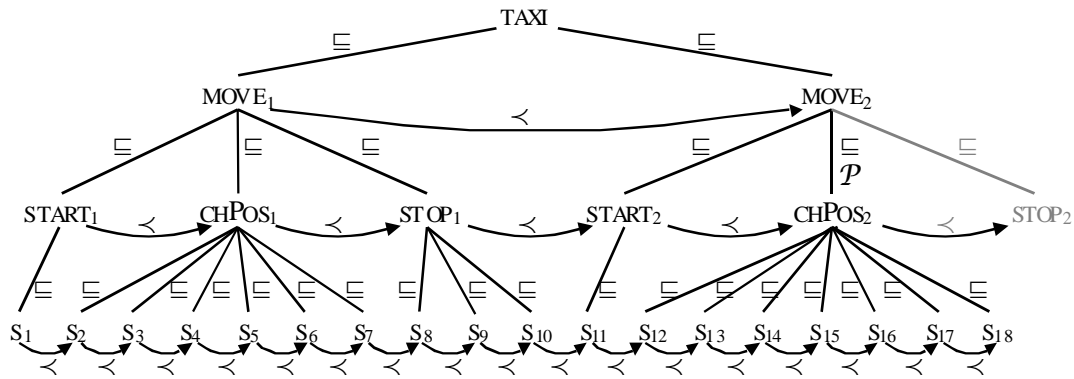


Figure 3. The Knowledge Representation for the Example (STOP₂ is only expected)

movement from position A to B.

MOVE is a label for complex events that consists of maximally three sub-events, namely START, CHPOS (CHANGE OF POSITION), and STOP, where the first and the last sub-event are optional and the middle event can be any kind of movement along a trajectory. START, CHPOS, and STOP nodes contain the sensor data nodes *s*. Temporal inclusion relates also TAXI and the basic events and MOVE and basic events, but are left out in the figure to keep it readable. The precedence relation, though, exists explicitly only between nodes of the same granularity, i.e. between the basic events and between the nodes of each intermediate level, e.g. CHPOS₁ < STOP₁ < START₂; the implicit precedences have to be derived from these. The μ relation is not included in Figure 3. We use four sorts to subdivide the CCR into four sub-nets, each consisting of refOs of one sort: planned event (P-Event), actual event (A-Event), problem (Problem), and real-world object (R-Object).

The construction process, which builds up the representation of the actually registered events and to detect problems, can be realized by the following algorithm

1. Generate a new node for the basic entity that is provided by the pre-processing unit. Link it to other nodes according to the information (i.e., attributes) coming with the basic entity.
2. Take the new node, possibly with related nodes—especially those that stand in the μ relation to this one—and hand them over to the concept matcher to find the best matching complex concept that contains these nodes—if there is any—and to find (possible) problems.
3. In case there is a problem, create a new node for it, and link it to the involved nodes. Continue with step 2. (In step 2 the ‘new’ node is the ‘old’ one, not the problem node!) In case of a complex concept, create a new node for it, together with the simpler nodes that are still lacking (generation of expectations), and link it to the basic nodes it subsumes.
4. Try to find relations to other complex nodes with

which links can be established.

5. Continue with step 2 to try to find more complex concepts and problems, until the next new basic entity enters the system or until there are no more complex concepts. Then proceed with step 1.
- In the following application of this algorithm to the example, the μ relation is left out until the problem is identified. Up to that point, each node, except for the *S* nodes, has a corresponding one in the sub-structure of planned events, related by μ .
- a. *S*₁ is read by the construction process as a basic event and inserted into the (up to now empty) knowledge representation. (step 1)
 - b. This node is handed over to the concept matcher (there are no related nodes, yet), which responds that this is a START event. (2)
 - c. START₁ is generated and linked via \sqsubseteq to *S*₁. (3)
 - d. Steps 4 and 5 yield no results.
 - e. *S*₂, the next basic event, is inserted. A < relation between the two *S* nodes is established. (1)
 - f. Because of their temporal vicinity the two *S* nodes are taken and given to the concept matcher, which computes a CHPOS event. (2)
 - g. CHPOS₁ is generated and linked via \sqsubseteq to *S*₂. (3) (It would equally be possible to take *S*₂ and *S*₁ to be parts of CHPOS₁. The only reason we chose this possibility is that it preserves the tree structure—at least for the moment.)
 - h. START₁ and CHPOS₁ are linked by <. (4)
 - i. Now START₁ and CHPOS₁ are given to the concept matcher, which joins them together in a complex event MOVE (5, 2).
 - j. MOVE₁ and the ‘expectation’ node STOP₁ are generated and MOVE₁ is linked via \sqsubseteq to its elements (3).
 - k. Step 4 yields no result.
 - l. MOVE₁ is just part of a more complex event TAXI. (5, 2)
 - m. TAXI is inserted and linked via \sqsubseteq to MOVE₁. (3)
 - n. Node *S*₃ is read and subsumed to CHPOS₁. In this step the attributes of CHPOS₁ are updated. (Especially interesting for our example: the velocity attribute.)
 - o. Nodes *S*₄ to *S*₇ are integrated and subsumed to CHPOS₁.

- p. S_8 is read and subsumed to $STOP_1$, which is now changed from an ‘expectation’ node to a ‘normal’ one. $STOP_1$ comprises the span of time, where flight number CK-314 is waiting at position B.
- q. Nodes S_9 and S_{10} are integrated and subsumed to $STOP_1$.
- r. Nodes S_{11} to S_{17} are read in and the $START_2$, $CHPOS_2$, and $MOVE_2$ are built up analogously.
- s. S_{18} is read in, inserted, and the velocity attribute of $CHPOS_2$ is updated while it is linked to $CHPOS_2$. (1)
- t. $CHPOS_2$ together with its corresponding planned node is given to the concept matcher, which detects the too high velocity. (2)
- u. Problem node $PROBLEM_v$ is created. (3)

The rest of the representation is built up analogously.

3.3 Selection

The *selection* process chooses the nodes that will be verbalized. In the exemplifying domain, only nodes of the sort Problem are selected for verbalization. These are linked together to form a (partial) path—the *traverse*—through the network. We use the notion *traverse* in two readings: (i) it denotes the *resulting* path, i.e. the concatenation of all nodes that are actually chosen and verbalized; (ii) it means the *current* traverse that contains a limited number of selected elements that are not yet taken by the pvm-generation at a given point of time. Elements that are contained in the current traverse may be removed and are not contained in the resulting traverse anymore.⁵

The selection strategy for the purpose at hand takes new nodes that are of sort Problem and considers them for insertion into the traverse. The selection process can manipulate the traverse by means of two operations: *appending* and *replacing* elements. While the first one is more fundamental, the second especially serves the purpose to substitute already chosen nodes by better candidates. Consider a situation in which CK-314 is taxiing too fast ($PROBLEM_v$ / velocity too high). Because of this critical velocity, it also comes too close to a second aircraft (MG-209); an additional problem node is constructed ($PROBLEM_c$ / aircraft too close). If the selection process simply chose both nodes, it would lead to an unclear, redundant, and cognitively more complex warning. Thus, if $PROBLEM_c$ is recognized first, but the selection

process determines $PROBLEM_v$ to be more urgent and, additionally, to be essentially the same as $PROBLEM_c$, the best solution is to generate only one warning based on $PROBLEM_v$.

3.4 Linearization

The *linearization* process has the function of bringing the selected nodes into an appropriate order. For a general application of a psycholinguistic model it might be required, for example, to move the more prominent (more informative) item to the beginning of the sentence and, therefore, to reorder the events. In our architecture for monitoring systems the linearization process is the means for pushing the most urgent problems to the beginning of the traverse.

In general it is even very probable that changing the order of selected nodes is necessary more often than keeping the (randomly) arising one. This is so, because the sequence of information is a crucial factor for any speech producing system to be easily understandable, e.g. it can be essential to follow the principle ‘Known information first, unknown last’.

3.5 PVM-Generation

The *pvm-generation* observes the traverse and generates preverbal messages from the selected nodes. There is a latency time between the time when a node is selected and the moment when the pvm-generation uses it as preverbal message.

The latency time can be either fixed or variable. The model would be easier with a fixed time interval, but since we also want to be able to simulate differing processing loads and differing time pressure in cognitive systems we favor the variable variant. Consider, for example, that the selection process chooses so many nodes that the traverse is in danger of reaching its maximal length or, even worse, that selected elements are lost, because more elements than possible would need to be stored. An obvious solution would be to change the selection strategy in such a way that more complex nodes are selected. But due to other restraining factors this will not always be possible, e.g. a certain level of detail might be required, and so preverbal messages will be generated with a higher frequency by reducing the latency time. This has the effect that all selected nodes are verbalized. On the other hand, increasing the latency time can be a suitable measure against the production of too many utterances that are corrections of previous utterances. Such corrections arise when the selection process is unable to make nec-

⁵ The limited capacity of the current traverse can be varied for simulations of cognitive behavior with different processing (memory) capacities, leading to different verbalizations. But, for an application oriented monitoring system, a limitation of the size is undesirable as all identified problems should be reported.

essary changes (replacements) in the traverse, because the pvm-generation already took the elements that would otherwise have been replaced.

After the latency time of a node is elapsed the pvm-generation passes it on to the subsequent process of the formulator. This means—as we argued above—that propositions exist only ‘piece-wise’. Complete propositions exist, then, on a higher level of abstraction. When the PROBLEM_v node is selected as a preverbal message, information about what is the problematic event (moving too fast), the object involved (the aircraft with the flight number CK-314) and about the location of the movement (taxiway Echo) have to be conveyed. This is done by handing on not only the PROBLEM_v node but also further nodes that contain information about the event (in the CHPOS₂ node), the flight (in a FLIGHT node), and about the location (in a TW-ECHO node). Thus, the pvm-generation considers some important relations to other nodes before the PROBLEM_v node is passed on, and checks what other nodes are needed for the verbalization and passes them on, as well.

The formulator can establish interrelations between the refOs by the ascribed relations, e.g. the PROBLEM_v node contains an *actor* relation to the FLIGHT node, which enables the formulator to look up all necessary information at the relevant nodes. Taken together they are now equivalent to a proposition like *problem(ck-314, velocity(tooHigh), tw-Echo, goal(rw27))*.

4 Conclusion

We presented a psycholinguistically motivated architecture of an incremental conceptualizer together with some remarks on its prototypical implementation and how this implementation can be used for monitoring purposes. The conceptualizer watches dynamic changes in the world and generates on-line propositional, preverbal structures that can serve as input to a subsequent component, which encodes these structures linguistically.

References

Abb, B.; Günther, C.; Herweg, M.; Lebeth, K.; Maienborn, M. & Schopp, A. (1996) Incremental grammatical encoding—an outline of the SYNPHONICS formulator. In G. Adorni & M. Zock, eds., *Trends in natural language generation: An artificial intelligence perspective*, 277–299, Berlin: Springer.

André, E.; Herzog, G. & Rist, T. (1988) On the simultaneous interpretation of real world image sequences and their natural language description: The system

SOCGER, 449–454. *Proc. of the 8th ECAI*, Munich.

Australian Bureau of Air Safety Investigation. <http://www.basi.gov.au>

Avrahami, J. & Kareev, Y. (1994) The emergence of events. *Cognition*, 53, 239–261.

Baddeley, A. (1986) *Working Memory*. Oxford: Oxford University Press.

Cahill, L.; Doran, Ch.; Evans, R.; Mellish, Ch.; Pava, D.; Reape, M.; Scott, D. & Tipper, N. (1999) In search of a reference architecture for NLG systems. *EWNLG-1999*, Toulouse.

Chu-Carroll, J. & Carberry, S. (1998) Collaborative response generation in planning dialogues. *Computational Linguistics*, 24, 355–400.

De Smedt, K.; Horacek, H. & Zock, M. (1996) Architectures for natural language generation: Problems and perspectives. In G. Adorni & M. Zock, eds., *Trends in natural language generation*, Berlin: Springer.

De Smedt, K. & Kempen, G. (1987) Incremental sentence production: Self-correction and coordination. In G. Kempen, ed., *Natural language generation*, 365–76, Boston: Martinus Nijhoff.

Ferreira, F. (1996) Is it better to give than donate? Syntactic flexibility in language production. *Journal of Memory and Language*, 35, 724–755.

Habel, Ch. (1986) *Prinzipien der Referentialität: Untersuchungen zur propositionalen Repräsentation von Wissen*. Berlin: Springer.

Habel, Ch. & Tappe, H. (1999) Processes of segmentation and linearization in describing events. In R. Klaunder & C. von Stutterheim, eds., *Representations and Processes in Language Production*, 117–153, Wiesbaden: Deutscher Universitäts-Verlag.

Hovy, E. H. (1993) Automated discourse generation using discourse structure relations. *Artificial Intelligence*, 63, 341–385.

Levelt, W.J.M. (1989) *Speaking: From intention to articulation*. Cambridge, MA: MIT Press.

Marcu, D. (1997) From local to global coherence: A bottom-up approach to text planning. *Proc. AAAI 97*, 629–635.

McKeown, K. (1985) *Text generation*. Cambridge: Cambridge University Press.

Neumann, B. & Novak, H. -J. (1983). Event models for recognition and natural language. *IJCAI-83*, 724–726.

Radev, D. R. & McKeown, K. (1998) Generating natural language summaries from multiple on-line sources. *Computational Linguistics*, 24, 469–500.

Reiter E. (1994) Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? *IWNLG-1994*, 163–170, Kennebunkport, ME.

Reithinger, N. (1992) The Performance of an incremental generation component for multi-modal dialog contributions. In: R. Dale, E. Hovy, D. Rösner & O. Stock, eds., *Aspects of automated natural language generation*, 263–276, Berlin: Springer.

Schank, R. C. & Abelson, R. P. (1977) *Scripts, plans, goals and understanding: An inquiry into human knowledge structures*. Hillsdale: Lawrence Erlbaum.

Zacks, J. (1997) *Seeing the structure in events*. Manuscript, Stanford University.