Memo Concept-to-Speech

Knowledge Resources for the Text Planner: The Domain Model and Plans for Discourse

Elizabeth J. Garner

1995

Contents

Abstract

The task of the text planner is to convert an input specification into an output suitable for manipulation by the tactical generator; a task which involves both content selection and content organisation. This paper takes a look at some of the resources needed for this. Using the results of a corpus analysis I first show how both the underlying domain and communication knowledge may be modelled in the Knowledge Representation Language LOOM. Having looked at possible methods of specifying input, I go on to discuss how certain types of variation may be expressed by discourse plans. Then, taking examples from the corpus, I demonstrate how these may be implemented by the use of LOOM production rules. Finally, I look at the form of the output of the production rules and suggest what further resources are necessary in order to arrive at the desired output for the tactical generator.

1 Introduction

Generation is usually divided into two parts, text planning and grammatical realisation. Grammatical realisation consists of a complex mapping process from one form (some kind of fixed logical propositions) into another (natural language sentences), maintaining the same informational content at both levels. Text planning is more complex, involving at least two tasks; *content selection* i.e. choosing what to say, and *text organisation*, choosing how to say it (how to order information, how to map semantic information onto lexical, where to place sentence boundaries, what the thematic structure of the text should be). Here we have no linguistic theory to guide what we choose as our input and how this input may be modified — the whole process is much more intuitive.

A popular method of carrying out text planning is the use of an AI planning mechanism Moore & Paris (1992). Input to the plans is some kind of specification of the message. A library of plans applies to this input until an output is produced which may be passed to the tactical generator for realisation. This method combines the two tasks of text planning, since the discourse plans produce as output speech acts already ordered and carved up into sentence sized chunks ready for the tactical generator.

However, there is no reason why planning should end with the output of plans. Provided the output is specified in a manner which may be further manipulated by other mechanisms we may use a planner to perform content selection and the initial stages of text organisation (e.g. rough ordering of information) but leave remaining mechanisms to perform finer organisation (thematic structuring, fixing of sentence boundaries, etc.). In this way we may make use of a very powerful AI mechanism for capturing types of variation, without limiting ourselves to the types of operations plans are able to perform.

This paper represents the first stage in an attempt to construct this kind of text planner. I present here the sources of information needed for the planner as a whole, i.e. a domain model (DM) which describes the real-world domain we are dealing with as well as modelling the types of text we wish to generate. I then move on to look at the planning process itself, looking at the possible means of providing input and the types of variation we can capture by means of a planning mechanism, i.e. the use of productions in the Knowledge Representation Language LOOM. Both the construction of the domain model and the text planner rely very heavily on an analysis of a suitable corpus, which is also presented in this paper.

2 Corpus Analysis

The intuitive nature of text planning makes it all the more important to rely on a close analysis of an available corpus to guide the choices made. For the purpose of providing

information relevant to the development of the text planner an analysis has been carried out on two sets of data, made available by the Labour Inspectorate of the provincial government of South Tirol in Bozen. The first set consists of official correspondence dealing with employment contracts. The second set consists of a number of application forms for various benefits, e.g. family benefit, retirement pensions. The analysis carried out on each of these sets of data entailed:

1. The identification of the discourse entities used in the texts, paying particular attention to how domain-specific terms are referred to. Here we must distinguish between the first occasion and object is referred to in the text (which is properly speaking a problem of lexicalisation) and all subsequent occurances (a problem of reference).
2. An analysis of the rhetorical structures of the text, closely based on the principles of Rhetorical Structure Theory (RST) (Mann & Thompson (1987)). The analysis differs from RST in that:
   (a) the text is not analysed at all levels, but begin at paragraph level. (In general this corresponds to the units where RST relations MAY be made explicit.)
   (b) the crossing of branches in the introduction and later elaboration of sequential topics is permitted.
   (c) following Wiebe (1993), more than one relation between individual clauses is allowed.
   (d) where appropriate, non-clausal constituents may form part of an RST schema.
3. An analysis of the intentional structure of the texts. This has as starting point the work of Grosz & Sidner (1986), interpreted through a model of *domain communication knowledge*, Kittredge & al. (1991). Domain communication knowledge is knowledge which is required to communicate about a domain, and is based upon the idea that certain text-types exhibit stereotypical ordering of content.

Examples of the RST and intentional structures of some of the texts in the corpust are given in the appendix.

3 LOOM

LOOM 2.0 (Brill (1993), MacGregor (1988)) has been chosen to represent the domain model and form the basis of the text planner. There are two main reasons for this:

1. LOOM is a well-established knowledge representation system that has been widely used in the AI community.

2. LOOM offers a wide range of closely integrated facilities enabling us to cover the whole transformation from input specification to the output required for the tactical generator within a single framework.

LOOM is a member of the KLONE family of knowledge representational systems (Brachman & Schmolze (1985)). It makes the classic distinction between concepts (terminology) represented in the **T**(erminological)-**Box** and instances (assertions) represented in the **A**(ssertional)-**Box** providing for classification-based inference as well strict implication and default reasoning by means of a well-integrated collection of specialised reasoning components. In addition, data-driven programming is made possible by the incorporation of production rules, which are triggered by changes in the domain model. These capabilities of LOOM lie within a framework of query-based assertion and retrieval.

In the following sections we will look at how these facilities have been utilised for modelling the domain and the development of a text planner.

4 The Domain Model

The domain model (DM), as already stated, is made up of two parts, the **T-box** and the **A-box**.

The **T-Box** introduces the terminology necessary for a particular domain. It consists of a set of concept and relation descriptions. Concepts represent abstract classes of individuals, relations correspond to abstract relationships between (classes of) individuals. Concept descriptions are potentially complex structures, formed by composing a limited set of description forming operators.

The **A-box** introduces the assertions into the domain. Assertions are represented as instances of a concept and correspond to particular real-world entities.

The objects contained in the DM represent the domain as conceived by the system. In this conceptualisation, in addition to concepts and instances there exists a third class of objects. These are, like instances, single objects, but their identity is not known to the system. So, for example, in the application forms there exists the individual referred to by means of the phrase *Ehepartner des Antragstellers*. While there potentially exist many spouses of many different applicants, in any particular form-filling event it is one particular individual who is meant. To distinguish such instances I have adopted the term **generic instance** from Vander Linden & al (1994) and signalled all objects of this kind as being instances of the concept **generic-instance**.

LOOM is designed not only to define models but also to dynamically match object instances in the database against defined models, i.e. to *classify* instances. By means of classification

we can find out further information about the instances in the domain, recognising them as instances of additional concepts, and placing further restrictions on them.

4.1 Representing the Domain

In this section we take a close look at two parts of the DM hierarchy which have already been implemented. The first is that part of the hierarchy which deals with the representation of contracts and forms part of the subject-matter description of the domain. The second is that part of the hierachy which deals with how information is encoded, i.e. which deals with the representation of communicative knowledge.

4.1.1 The contract hierachy

The highest-level concept in our contract hierarchy is **contract**:

(1)
```
(defconcept contract
    :is-primitive (:and
                   (:exactly 1 is-governed-by)
                   (:at-least 2 contract-party)))
```

Firstly, then, **contract** is a *primitive* concept in our hierachy, i.e. it includes necessary, but not sufficient conditions for membership in the given class.[1] Primitive concepts may be contrasted with *defined* concepts, which contain all the necessary and sufficient conditions for membership in the class. The concept has at least three roles, one is an **is-governed-by** relation; there are at least two fillers of the role **contract-party**.

The role **is-governed-by** is defined as being the inverse of the **governs** relation. The definition of **governs**

(2)
```
(defrelation governs
    :domain law)
```

states that it is a legal role of the concept **law**, i.e. (an instance of) a **law** governs something, and (an instance of) a **contract** is governed by a **law**.

A subtype of **contract** is **work-contract**:

[1] Primitive concepts can be thought of as *natural kinds*. They are the kinds of object which cannot be described exactly, but which contain in addition to restrictions given certain unrepresented features that characterise an object as an instance of this concept.

(3)
```
(defconcept work-contract
    :is (:and contract
              (:filled-by is-governed-by contract-law)
              (:filled-by authoriser labour-inspectorate)
              (:exactly 1 has-employer)
              (:exactly 1 has-employee)
              (:at-most 2 property)
              (:at-most 1 has-start-date)
              (:at-most 1 has-job-description)))
```

Being defined as a type of **contract** means that in addition to the restrictions contained in the definitions of **work-contract** itself, all instances of type **work-contract** inherit as part of their definition the restrictions contained in the description of the concept **contract**. By means of such inheritance very complex definitions may be built up using relatively simple descriptions, at the same time capturing crucial relationships between the objects of the domain. In addition, certain other restrictions have be added.

Firstly, the role filler of **is-governed-by** has now been fixed to an instance of a law, namely **contract-law**. **authoriser** (the institution which authorises a contract) is filled for all instances of **work-contract** by the instance **labour-inspectorate**. **has-employee** and **has-employer** are sub-types of the relation **contract-party**. A **work-contract** may have at most two fillers for the **property** role, which has as sub-types **has-start-date** and **has-job-description**.

Notice that the definition does not however specify that the two property roles *must* be present. That is left to the concept **legal-work-contract** where the **:at-most** restriction has given way to **:exactly**:

(4)
```
(defconcept legal-work-contract
    :is (:and work-contract
              (:exactly  2 property)
              (:exactly  1 has-start-date)
              (:exactly  1 has-job-description)))
```

In an individual of type **contract-with-missing-property**, a subtype of **illegal-contract**, one (or both) of these properties is missing, i.e.

(5)
```
(defconcept contract-with-missing-property
  :is (:and work-contract
            (:at-most 1 property)))
```

The relation **property** is characterised as being **closed-world**.[2] This means that if a given instance lacks either a **has-start-date** or **has-job-description** it is correctly classified as an instance of **contract-with-missing-property**. There are then two subtypes of **contract-with-missing-property**:

(6)
```
(defconcept contract-without-start-date
   :is (:and illegal-contract
             (:at-most 0 has-start-date))
   :implies (:filled-by  missing-property contract-start-date))
```

(7)
```
(defconcept contract-without-job-description
   :is (:and illegal-contract
             (:at-most 0 has-job-description))
   :implies (:filled-by missing-property employee-occupation))
```

The **:implies** relation allows us to specify a role of the concept which is not taken into account in classification, but which is a condition implied by membership of the concept. This information is, as we will see later, useful for the text planner. **contract-start-date** and **employee-occupation** are examples of generic instances.

While in our current domain we have only illegal contracts of subtype **contract-with-missing-property** the domain model could easily be expanded to contain contracts with other types of error.

As already stated, LOOM may be used not only to model the domain, but also dynamically to classify the objects of the domain. So, for example, creating an instance of **work-contract** by the following input:

(8)
```
(tellm (:about vertrag1
          work-contract
          (has-employee matiasek)
          (has-job-description shelf-filler)))
```

leads to its correct classification as an instance of the concepts **illegal-contract, contract-with-missing-property** and **contract-without-start-date** since it has no specified **has-start-date** role.

[2]Globally, the domain model does not work under the closed world assumption.

4.1.2 Representing Communicative Knowledge

In addition to using concept definitions to express information about the objects of the domain, we can also use them to model the communicative information about the domain we wish to express. This can, as suggested by Hovy & al. (1992), enable us to represent information about the types of text we are aiming to generate as well as the various subcomponents of such texts. So, for example, we may define the concept **text** as follows:

(9)
```
(defconcept text
  :is-primitive
   (:and
    (:exactly 1 speaker)
    (:at-least 1 hearer)
    (:at-most 1 introduction)
    (:exactly 1 main-body)
    (:at-most 1 conclusion)
    (:at-most 1 topic)
    (:the  main-body message)
    (:same-as (:compose main-body speaker) speaker)
    (:same-as (:compose main-body hearer) hearer))
```

text does not belong to any supertype, but, like **contract** is directly subordinate to the superconcept **thing**. In addition all instances of **text** have a **speaker**, at least one **hearer**, and **introduction**, **main-body** and **conclusion**. The **introduction** contains thematic information for the text. Furthermore, it is *about* something, i.e. has a **topic**. Its **main-body** is filled by an instance of **message**:

(10)
```
(defconcept message
   :is (:or dc-plan dc-act dc-sp-act))
```

whose **speaker** and **hearer** roles are filled by the same instances as those of **text**. Here **dc-plan** is a *domain communication plan*, a procedural entity to be expanded by the plan mechanism; **dc-act** a *domain communicative act*, which represents a generalisation over the intended meaning of a group of speech acts in the domain; and **dc-sp-act** is a *domain communication speech act*, a speech act relevant to the domain. These three concepts are related to each other in that the fillers of the **sub-action** relation of a **dc-plan** are of type **dc-act**, while each **dc-sp-act** is a sub-type of **dc-act**.

A subtype of **dc-plan** is **correct-contract** which contains roles corresponding to each of its parameters:

(11)
```
(defconcept correct-contract
    :is (:and dc-plan
              (:exactly 1 instigator)
              (:exactly 1 corrector)
              (:exactly 1 has-contract)))
```

In the current state of the model the description of dc-plans is very idiosyncratic. It is however possible, by means of classification, to generalise over the types of dc-plans, introducing, for example, a dc-plan **correct-object**, should this be required by the domain.

There are currently two subtypes of **speech-act**, **request**, in which **main-body** filled by an instance of **action** and **inform** in which **main-body** filled by an instance of **proposition**.

There are two subtypes of the concept **action**, **simple-action** and **constrained-action**. A **constrained-action** is an action which should only be performed if the situation given in the **has-constraint** role, (an instance of **constraint**) holds.

Text-types too are included in the text hierarchy. An example is **official-letter**, which represents the conjunction of the concepts **document** and **text**, inheriting from the former the relations **authoriser** (the body responsible for validating the document), and **is-governed-by** (a law). In addition the **speaker** of the text is equated with the letter **sender** and the **hearer** with the letter **recipient**. The idea here is to capture the fact that letters are made up a a number of rigid components, essentially those which dictate its form, and a variable content (**main-body**). Already at this level then the letter **conclusion** is filled, and the **introduction** restricted to being of type **letter-introduction**, which leads to the phrase 'in Bezug auf', with the letter **topic** as argument.

```
(defconcept official-letter
 :is (:and document
           text
           (:exactly 1 sender)
           (:same-as sender speaker)
           (:exactly 1 recipient)
           (:same-as recipient hearer)
           (:the introduction letter-introduction)
           (:filled-by conclusion "Mit freundlichen Gruessen")))
```

correct-contract-letter (letter to correct a contract) is a subtype of text in which the **topic** is an instance of an **illegal-contract** allowing information about the contract to be carried over to the letter, and the **main-body** is an instance of **correct-contract**, a dc-plan, which leads to the application of the production described in (*16*). This enables us to realise the **main-body** in a number of different ways, as we will see in a later section.

```
(defconcept correct-contract-letter
  :is (:and
       official-letter
       (:same-as (:compose topic authoriser) authoriser)
       (:the topic illegal-contract)
       (:the main-body correct-contract)
       (:same-as (:compose main-body instigator) speaker)
       (:same-as (:compose main-body corrector) hearer)
       (:same-as (:compose main-body contract) topic)
       (:the recipient company)
       (:same-as (:compose topic has-employer) recipient)))
```

The keyword **:compose** builds a role chain, so that, for example, the filler of the **topic** (a contract instance) has an **authoriser** set to be the same as the **authoriser** of the letter.

5 Text Planning

The task of the text planner is, given some kind of message specification from the user, to construct a text appropriate to this input. How this is carried out depends then partly on the form of the message specification. The types of message specification looked at will be the subject of section 5.1.

The text planner is also constrained by the results of the analysis of intentional and rhetorical structures of the corpus, revealing the possibility of variation at various levels which must be reflected in the planner. This variation may consist in alternative modes of expression, or the inclusion/exclusion of information. The following types of variation have been observed:

1. variation at the level of text — A text is designed to communicate an intention (intentions) of the speaker. The same intention may be achieved by a number of different types of discourse, each consisting of a sequence of *communicative acts*. So, for example, the intention to correct a contract consists of the two acts to explain the error and to request a correction of the error.

2. variation at the level of *communicative act* — Each communicative act has an intention which contributes to the intention of the discourse. The same communicative act intention may be achieved by a number of different speech acts. So, for example, an indirect and a direct speech act with the same goal would be instances of the same communicative act.

3. optional inclusion of definitions — Each speech act refers to instances of concepts which may simply be lexicalised, or defined in various ways. There are two situations

where this may occur:

(a) on any first occasion an instance is referred to.

(b) in certain specific contexts, e.g. when a user requires certain information about an instance in order to carry out an instruction.

4. inclusion of referring expression — Instances of concepts in particular text types must be referred to by rigid patterns, e.g. in legal texts a law is always referred to by its number and the date passed.

5. inclusion of metacomments, Zuckerman (1991) — This refers to expressions not part of the subject matter of the text, but which are included to organise the text in a more comprehensive manner. Metacomments include discourse cue words such as 'however', 'next' as well as use of certain phrases, e.g. 'die folgenden Punkte sind zu beachten'.

6. variation at the level of propositions — A process may be lexicalised as a verb or as a nominalisation, a discourse relation as a conjunction or a verb. Such decisions are influenced to a large degree by the thematic structure of a text and cause variation in the placement of sentential boundaries.

Variations of the type described in 1. and 2. can be dealt with straightforwardly by planning. The choice to include definitions of type 3.(a) depends on the information contained in the user model, but is a choice to be made at the time of lexicalisation so will not be dealt with here. The inclusion of type 3.(b) definitions can however be realised by planning, since the inclusion is dependent on dc-plans, as will be shown.

The inclusion of referring expression described in 4. is dependent on text type. Since this is currently fixed, the inclusion of such descriptions will be dealt with during lexicalisation, making use of the information contained in the domain model.

The inclusion of metacomments (5.) can to some extent be handled by planning, e.g. the insertion of discourse cues could be triggered by the expansion of certain (types of) dc-plans. But the introduction of phrasal metacomments may also be triggered by patterns in the overall structure of the text which only become apparent at the end of the planning stage. This means that one task of the planner is to build up a text structure so that metacomment insertion can be carried out.

Finally, variation at the propositional level as described in 6. is restricted by the thematic structure of the text and must be performed after the planning process is complete. The types of manipulation it entails, however, places restrictions on the output of planning, which is the subject of section 5.3.

In section 5.2 I will show how the patterns of variation which can be dealt with by planning can be can be realised making use of LOOM's production rule facility.

5.1 Message Specification

I have investigated two possible methods of message specification providing input to the text planner. They are described in the following two sections.

5.1.1 Specifying a dc-plan

The first method of providing input to the text planner requires the user to specify the type of discourse plan she wishes to generate. The user is given a menu of possible dc-plans to choose from, and having made her selection must then provide the role fillers of that particular type. For example, in the case of an instance of type **correct-contract** described in the previous section the user would be required to provide the **illegal-contract** instance which is to be corrected, as well as specifying the **speaker** and **hearer** of the text to be generated.

5.1.2 KB Triggers

Text planning can be also initiated automatically by the application of production rules. Here it is the creation of a new instance in the domain model which acts as a trigger for the application of a production rule which initiates the text planning process. This is best illustrated by means of an example.

The example concerns the generation of instances of **correct-contract-letter**. The generation of such letters occurs in the situation in which the labour inspectorate has received a work contract from a company which does not meet the legal specifications, but which can be altered in order to comply with them. We have then as input an instance of **illegal-contract** *(5)*, e.g.

(12)
```
(tellm (:about vertrag2
              work-contract
              (has-employee matiasek)
              (has-start-date 1.1.94)))
```

Since this description lacks a **has-job-description** property it is correctly classified as an instance of an **contract-with-missing-property** (itself a subtype of **illegal-contract**) and, more specifically, a **contract-without-job-description**.

The creation of the instance **vertrag2** fires the production **generate-correct-contract-letter**:

(13)
```
(defproduction P1-generate-correct-contract-letter
   :when (:detects (illegal-contract ?contract))
   :do (make-ccletter ?contract))
```

Here the **:when** clause serves here as a trigger for the production. The rule states that when an instance of the concept **illegal-contract** is detected the Lisp form **make-ccletter** should apply.

(14)
```
(defun make-ccletter (?contract)
   (let* ((?letter
           (create 'letter 'correct-contract-letter :add-suffix-p t))
          (?title (get-value ?contract 'is-governed-by))
          (?authoriser (get-value ?contract 'authoriser))
          (?intro (create 'introduction nil :add-suffix-p t))
          (?recipient (get-value ?contract 'has-employer)))
     (tell (:about ?intro (has-range ?contract)))
     (tellm (:about ?letter (topic ?contract)
                    (introduction ?intro)
                    (sender letter-sender)
                    (authoriser ?authoriser)
                    (title ?title)
                    (recipient ?recipient)))))
```

make-ccletter creates an instance of **correct-contract-letter** in which the fillers of the **recipient**, **title** and **authoriser** roles are set equal to the **employer**, **is-governed-by** and **authoriser** role fillers of the contract to be corrected. A further instance is also created to fill the **introduction** role of the letter, classified as being of type **letter-introduction**. The **sender** is a generic instance to be later specified.

This instance of **correct-contract-letter** itself fires a second production which creates the frame of the letter. Here the argument of **:do** is the Lisp function:

(15)
```
(defun  generate-letter (?letter)
   :title "generate cc-letter"
   :response ((with-open-file
                (*standard-output* "cc-letter" :direction :output)
                (make-title ?letter)
                (make-intro ?letter)
                (make-main-body ?letter)
                (make-ending ?letter)
                (make-sender ?letter))))
```

The functions **make-title**, **make-intro**, **make-ending**, and simply look up and generate the required information from the letter. **make-sender** sends a request to the user to sup-

ply this item. **make-main-body** leads to the creation of an instance to fill the**main-body** of the letter, which is classified by the domain as being of type **correct-contract**, and leads to a further round of productions. How the dc-plan **correct-contract** is expanded will be described section 5.2.

5.2 Planning via Productions

Planning may be directly executed in LOOM by the use of the productions. The productions, as described in section 5.1.2 are triggered by the detection of instances in the DM. This time, however, the productions contain not Lisp functions but calls to LOOM *actions*.

Actions in LOOM represent procedural behaviour which is performed on request. Actions are implemented by a set of *methods* that implement the responses necessary to produce the desired behaviour. An action represents the generic solution to a task request, while methods provide context-specific options for accomplishing the task. Attaching a set of methods to a single action allows variation can be achieved.

The operations contained in a method may lead to the creation of new instances, firing further productions, so that the dc-plan which constitutes the input to the planner can be expanded in a step-by-step fashion. In the following sections, I will describe this process in more detail.

5.2.1 Variation at Text Level

A dc-plan consists of a number of consecutive communicative acts. An example of a discourse plan cited earlier and found in the official correspondence corpus is **correct-contract**. The production rule which fires the expansion of **correct-contract** is the following:

(16)
```
(defproduction P2-generate-correct-contract
    :when (:detects (correct-contract ?correct-contract)
                    (instigator ?correct-contract ?instigator)
                    (corrector ?correct-contract ?corrector)
                    (has-contract ?correct-contract ?contract))
    :perform (generate-correct-contract ?instigator
                                        ?corrector
                                        ?correct-contract
                                        ?contract))
```

The **:perform** argument is a call to an action **generate-correct-contract** defined as follows:

(17)
```
(defaction generate-correct-contract (?instigator
                                      ?corrector
                                      ?correct-contract
                                      ?contract))
```

LOOM actions have associated filters which offer a strategy for choosing among the methods for the action. In this case, since no specific filter is supplied, the default **(:most-specific :last-one)** applies eliminating any method whose situation pattern is specialised by some other candidate's method, and selecting the most recently defined of the remaining methods.

One of the methods, identified by its unique **:title** argument called by this action applies when the relevant instance of **contract** has some missing property:

(18)
```
(defmethod generate-correct-contract (?instigator
                                      ?corrector
                                      ?correct-contract
                                      ?contract)
    :title "contract has missing property"
    :situation (:and (contract-with-missing-property ?contract)
                     (missing-property ?contract ?missing-property))
    :response ((make-inform-about-missing-property
                          'inform-about-missing-property
                          ?instigator
                          ?corrector
                          ?missing-property
                          ?correct-contract)
               (make-request-to-send-missing-property
                           'request-missing-property
                           ?instigator
                           ?corrector
                           ?missing-property
                           ?correct-contract)))
```

The **:situation** argument provides the context-specific situation when the method should apply, i.e. in the case that the contract is of type **contract-with-missing-property**. The arguments of **:response** are to be evaluated if the conditions for the method are met. In this case the funtions contained in **:response** serve two purposes:

a) to create instances of the relevant type, triggering a new round of productions.

b) to attach the instances to concepts of the domain model so that the relations between these instances are captured. It is for this reason that the **?correct-contract** parameter is specified.

For example

(19)
```
(defun make-inform-about-missing-property (?dc-act
                                           ?instigator
                                           ?corrector
                                           ?missing-property
                                           ?dc-plan)
        (setq ?inform
         (create ?dc-act 'inform-about-missing-property :add-suffix-p t))
        (attach-instance ?inform ?dc-plan)
        (tellm (:about ?i (object ?missing-property)
                          (speaker ?instigator)
                          (hearer ?corrector))))
```

creates a new instance of type **inform-about-missing-property**, which is attached by the function **attach-instance** to the instance of the **dc-plan** which fired the production. The role fillers of the new instance are carried over from the instance of **dc-plan** as indicated.

Introducing two or more communicative acts together in a single production has two advantages. Firstly, it enables parameters to be shared by the two speech acts, so that we may generalise over them. Secondly, it allows the introduction of certain discourse-level rhetorical relations, where appropriate, which are reflected in the use of certain discourse conjuncts. So, for example, we could introduce a **sequence** relation between two acts which would lead to the generation of *then*.

A second method **generate-correct-contract** is further restricted to apply only in *spoken* contexts. In this case the **:response** consists of the single instance of **dc-act**, of type **inform-about-missing-property**. The request is, in this case, inferable from the remaining context.[3]

The corpus of application forms are examples of *instructional text*, i.e. they are designed for the purpose of eliciting a response from the reader, (usually filling in part of the form). The plans which give rise to the text generally consist then in giving the reader instructions, and contextual information deemed necessary by the system for carrying out these instructions. We can represent these as instances of **be-comp-about-plan**, a subtype of **dc-plan** whose purpose is to make to reader competent about plans to fill in sections of the form. The production needed to initiate text planning here is:

[3]Hence *Sie haben das Datum nicht angegeben* can be analysed as a request to supply the appropriate date.

(20)
```
(defproduction P3-generate-be-competent-about-plan
    :when (:detects (:and (be-comp-about-plan ?be-comp-about)
                          (has-plan ?be-comp-about ?plan)
                          (has-instructor ?instructor)
                          (has-instructee ?instructee)))
    :perform (generate-be-competent-about-plan ?instructor
                                               ?instructee
                                               ?plan
                                               ?be-comp-about))
```

An example of a method applicable in the environment of this production is given in section 5.2.3.

5.2.2 Variation at the Level of Communicative Act

In natural language the same communicative act may be formulated in a number of different ways. So, for example, when informing the user that a property is missing from a contract we can either straightforwardly state that this is the case, or appeal to the legal requirements of contracts. Both possibilities occur in the corpus:

(21) (a) *In Bezug auf den Teilzeitvertrag, der mit dem/der Arbeitnehmer/in ..... am ..... abgeschlossen worden ist, teilt man mit, daß das obgenannte Gesetz eine genaue Angabe der Tätigkeit vorsieht.*

(b) *In Bezug auf den Teilzeitvertrag, der mit dem/der Arbeitnehmer/in ..... am ..... abgeschlossen worden ist, weist man darauf hin, daß das Aufnahmedatum, bzw. der Beginn des Arbeitsverhältnisses nicht angegeben worden ist.*

Both versions serve the same function, i.e. informing the user of the missing property, so that they may be defined as subtypes of the same **dc-act**. This is achieved by using the same production to trigger the propositions leading to both expansions, with two associated methods:

(22)
```
(defmethod generate-inform-about-missing-property (?speaker
                                                   ?hearer
                                                   ?missing-property
                                                   ?inform-about)
    :title "inform the user that the user didn't give ?property"
    :situation (:predcall #'in-user-model-p ?missing-property)
    :response ((generate-general-inform (?speaker
                                         ?hearer
                                         ?missing-property
                                         ?inform-about))))
```

(23)
```
(defmethod generate-inform-about-missing-property (?speaker
                                                   ?hearer
                                                   ?missing-property
                                                   ?inform-about)
    :title "inform the user that the law requires ?property"
    :response ((generate-legal-inform ?speaker
                                      ?hearer
                                      ?missing-property
                                      ?inform-about)))
```

The **:predcall** represents a call to the function **in-user-model-p** which checks the user model to see if the user knows that a **legal-work-contract** possesses the (missing) property concerned. This information is contained in a separate domain model which contains only that subset of concepts and instances known to the user. If the user knows the information, the function **generate-general-inform** is called. This function classifies the instance of **inform-about-missing-property** as a type of **general-inform**, a type which eventually leads to the generation of *21(a)* above). Otherwise, by default, **generate-legal-inform** is called, adding the type **legal-inform** to the instance, and leading to the generation of *21(b)*.

5.2.3 The Inclusion of Context-Specific Definitions

An example of a context-specific definition is contained in the following text:

(24) Zwecks Familiengeldzahlung setzt sich die Familiengemeinschaft folgendermaßen zusammen:

- *Antragsteller*
- *Ehepartner des Antragstellers, sofern nicht gesetzlich und tatsächlich getrennt.*
- *Kinder und Gleichgestellte, minderjährige und nicht verheiratet.*
- *Arbeitsunfähige Kinder und Gleichgestellte jeden Alters, nicht verheiratet.*

This type of compositional definition occurs very frequently in the corpus. Other examples include definitions of the concepts **taxable-income**, **unemployable-family-members**, **sibling-family-members**, which may be found in the appendix.

Initially it seems in *24* as if we are simply dealing here with a generic instance of the concept **nuclear-family** (lexicalised as *Familiengemeinschaft*), an instance which has been defined by listing the subconcepts of **nuclear-family**, as given in the DM, i.e. it is defined *compositionally*. The use of the phrase *zwecks Familiengeldzahlung*, however, implies that the given definition is in some way dependent on the context, i.e. giving the user instructions about how to apply for family benefit.

There are two possibilities of dealing with this situation. The first is to set up a new concept **nuclear-family-wrt-family-benefit** which is a subtype of the concept **nuclear-family** (and hence distinguished from it). We could then call *24* a *lexicalisation* of an instance of this type. The problem with this approach is two-fold:

1. Lexicalisation occurs after sentence boundaries have been fixed. Introducing such complex structures at this stage may have repercussions for the sentential divisions of the text which we would at this late stage be unable to alter.
2. It could be that the user is already aware of the composition of **nuclear-family** as relevant to the plan under consideration. The definition is in this case not obligatory.

The other solution which we shall adopt here is to introduce the definition by means of a method relevant in the context of the production given in *20*. This method fires in the context of a plan to make the user competent to enter family benefit and results in the generation of a compositional definition of the object **nuclear-family**, if the user model indicates that the user is not aware of this.

The method requires the introduction of two new concepts into the DM;

- **compositional-concept** whose sub-types include all those concepts defined by listing their subtypes, e.g **nuclear-family**, **taxable-income**, etc.

- **enter-compositional-object**, a subtype of **action** and **enter-action** which is restricted so that the **:range** role of **enter** is filled by an instance of **compositional-concept**. **enter-compositional-object** itself has subtypes **enter-family-member**, **enter-taxable-income**, etc.

An instance of the dc-plan **be-comp-about**, whose **has-plan** role is filled by an instance of **enter-family-members** fires the production given in *20* and leads to the application of the following method:

(25)
```
(defmethod generate-be-competent-about-plan (?instructor
                                             ?instructee
                                             ?plan
                                             ?dc-plan)
    :title "request to enter compositional object"
    :situation (:detects (:and (enter-compositional-object ?plan)
                               (has-subaction ?plan ?enter-action)
                               (s2 ?enter-action ?compositional-object)))
     :response ((make-inform-about-compositional-concept
                 'inform-about-compositional-concept
                 ?instructor
                 ?instructee
                 ?compositional-concept
                 ?dc-plan)
                (make-request-action 'request-action
                                     ?instructor
                                     ?instructee
                                     ?enter-action
                                     ?dc-plan)))
```

make-inform-about-compositional-concept is a **dc-act**, expanded by a production, whose methods lead to various propositional patterns of giving compositional definitions.

A second example of a context-bound definition is:

(26) Bei Antrag auf Familiengeld ist zu beachten, daß dieses ab 1. Juli jeden Jahres bis zum 30. Juni des darauffolgenden Jahres ausbezahlt wird.

Here the plan involved is to apply for family benefit, defined is the extent of the family benefit year. This can be handled in a parallel fashion to the above example by the introduction of two new concepts, **extent-object** and **apply-for-extent-object**, a subtype of **be-comp-about**.

5.3 Output of the Text Planner

The output of the text planner is a sequence of instances of **dc-sp-act**. The instances supply two types of information. Firstly, by means of the creation of instances of the appropriate type in each round of productions we create a *text structure* for the **dc-plan**. Secondly, the descriptions of the instances themselves are realised in a form appropriate for the remaining tasks of text planning. In this section we will have a look at these components in more detail.

The creation of instances of **dc-act** and **sp-act** lead to two different kinds of modifications. The role filler of each **sub-action** relation in the **dc-plan** is an instance of **dc-act**. The creation of a **dc-act** instance then, leads to its attachment via a **sub-action** relation to the **dc-plan** which triggered it. An instance of **dc-sp-act** on the other hand classifies as a type of **dc-act**; methods introducing speech acts do not lead to the creation of a new instance, but to the instance of **dc-act** responsible for their introduction being ascribed to a new type.

For example, an instance of **correct-contract** (a **dc-plan**) leads to the creation of two instances, an instance of **inform-about-missing-property** (a **dc-act** and an instance of **request-missing-property** (a **dc-act**, both later classified as instances of **dc-sp-act**. Part of the output is an instance of **inform**, say **inform-1** whose description includes:

(27)
```
(tell (:about inform-1
              general-inform
              inform-about-missing-property))
```

which itself fills the **sub-action** role of **cc-1** an instance of **correct-contract**.

The remaining part of the instance description contains its relations and their role fillers, which may, following modiification by the remaining components of the text planner, be mapped into an output suitable for the tactical generator. The chosen input for the tactical generator is a set of ESPL assertions.[4]. In ESPL, the text has already been ordered and carved up into sentence-sized chunks, the only tasks which remain are lexicalisation and (syntactic) generation. Since there remain a number of tasks to be performed on our text planner output we need some kind of intermediate representation for the instances, for which I have chosen the *abstract situations*, taken over by Wanner & Bateman (1990) from Mel'cuk's theory of lexical functions.

Abstract situations are characterised by **key terms**, and their participants. The key terms are usually realised on the syntactic level as nominals. As an example, the situation "teaching" may be represented as:

[4]For a description of ESPL see Vander Linden & al (1994)

S_0 ("teaching") = "teaching"
S_1 ("teaching") = "teacher"
S_2 ("teaching") = "pupil"

where S_i is the **i**th participant of the situation.

The chosen key terms represent very basic semantic notions, usually corresponding to processes, which may be lexically realised according to context in quite different ways, (so, "teaching" -> instruct, teach, learn). Wanner & Bateman (1990) use this flexibility to handle collocations, making use of the factor of salience; my intention is to build on their approach to deal with the issue of clausal and sentential boundaries by making use of the text structure obtained in text planning and the observed thematic structure.

Using abstract situations, our instance **inform-1** now contains the following information:

(28)
```
(tell (:about inform-1
              general-inform
              inform-about-missing-property
              (s1 candela)
              (s2 mayer)
              (s3 state-1)))
```

where

(29)
```
(tell (:about state-1
              not-state
              (s1 mayer)
              (s2 contract-start-date)))
```

The instances may also contain a subset of the RST relations observed in the analysis, namely those types of relations which according to Maier & Hovy (1993) deal with the content of a text rather than its organisation. An example is the RST relation **condition** which is relevant to the description of instances of **constrained-action**. Here an instance generated by the text planner described here takes the form:

(30)
```
(tell (:about condition-1
              condition
              (s1 action-1)
              (s2 constraint-1)))
```

where **action-1** is an instance of **simple-action** and **constraint-1** of **constraint**.

6 Conclusion

The text planning component of a natural language generation system is responsible for the transformation of a message input into a form suitable for manipulation by the tactical generator. This involves the following tasks:

- the precise specification of the form message input and output should take.
- the construction of a knowledge base for representing both information about the subject matter of the domain and domain communication knowledge.
- the construction of a text planner which utilises the information in the knowledge base to make the transformation from input to output.

This paper has presented an approach to the accomplishment of these tasks. Based on the analysis of a corpus of texts a domain model has been constructed in the Knowledge Representation Language LOOM. Two methods of specifying input to the generator have also been presented.

The text planner itself is responsible both for content selection and text organisation. An AI planning mechanism, by allowing the system to capture patterns of variation, is obviously useful for this. I have discussed in the text what types of variation need to be dealt with, and, using the production rule facility of LOOM, how some of these can be implemented.

However, as stated in the introduction, several tasks of text planning are best to be performed by a different kind of mechanism. These tasks include decisions on the placement of sentence boundarises; determining thematic structure; lexicalisation decisions such as whether to represent processes as nominals or verbs and how to lexicalise RST relations. For this reason I have here specified an output representation which can be further manipulated before providing the input to the tactical generator. The treatment of these remaining issues is the subject of my current work.

References

Brill, D. (1993) *Loom Reference Manual Version 2.0*, University of Southern California

Brachman R.J. and J.G. Schmolze (1985) 'An overview of the KL-ONE knowledge representation system', *Cognitive Science,* **9(2)**, 171-216

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

Hovy, E., J. Lavid, E. Maier, V. Mittal and C. Paris (1992) 'Employing Knowledge Resources in a New Text Planner Architecture', in R. Dale, E. Hovy, D. Rösner and O. Stock (eds.) *Aspects of Automated Natural Language Generation*, Springer Verlag, Berlin

Kittredge R., T. Korelsky and O. Rambow (1991) 'On the need for domain communication knowledge', in *Computational Intelligence* **7**, 305-314

MacGregor, R. (1988) 'A Deductive Pattern Matcher' in *Proceedings of the AAAI-88*, Menlo Park, CA

McKeown, K.R. (1985) *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*, Cambridge University Press, Cambridge

Maier, E. and E. Hovy (1993) 'Organising discourse structure relations using metafunctions', in H. Horacek and M. Zock (eds.) *New Concepts in Natural Language Generation*, Pinter Publisher, London

Mann, W.C. and S.A. Thompson (1987) 'Rhetorical Structure Theory: A Theory of Text Organization', in L. Polanyi (ed.) *The Structure of Discourse*, Ablex Publishing Corporation, Norwood, N.J.

Moore, J.D. and C.L. Paris (1992) 'Explcoiting User Feedback to Compensate for the Unreliability of User Models', in *User Modeling and User-Adapted Interaction* **2**, 287-330

Vander Linden, K., C.L. Paris, R. Power and T. Hartley (1994) 'Specification of the Extended Sentence Planning Language',

Wanner, L. and J.A. Bateman 'A collocational based approach to salience-sensitive lexical selection', in *Proceedings of the 5th International Workshop on Natural Language Generation*, 31-38

Wiebe J. (1993) 'Issues in Linguistic Segmentation, in O. Rambow (ed.) *Intentionality and Structure in Discourse Relations*, ACL

Zuckerman, I. (1991) 'Using meta-comments to generate fluent text in a technical domain', in *Computational Intelligence* **7**, 276-295

A Text1

A.1 Rst Structure

A.1.1 Rst Clauses

R1 Es sind alle Mitglieder der Familiengemeinschaft des Arbeitnehmers zu melden, für die Familiengeld beantragt wird.

ZUSAMMENSETZUNG DER FAMILIENGEMEINSCHAFT

R2 Zwecks Familiengeldzahlung setzt sich die Familiengemeinschaft folgendermaßen zusammen:

R3 Antragsteller

R4 Ehepartner des Antragstellers, sofern nicht gesetzlich und tatsächlich getrennt.

R5 Kinder und Gleichgestellte, minderjährige und nicht verheiratet.

R6 Arbeitsunfähige Kinder und Gleichgestellte jeden Alters, nicht verheiratet.

R7 Teil der Familiengemeinschaft können auch sein:

R8 Geschwister, Enkel und Neffen des Antragstellers,

R9 sofern minderjährig – oder arbeitsunfähig und volljährig

R10 sofern diese:

a) Vollwaisen sind.

b) kein Anrecht auf Hinterbliebenrente haben.

R11 In diesem Fall

R12 muß zum Bezug des Familiengeldes die Genehmigung des NISF auf eigenem Formbl. ANF 42 eingeholt

R13 und dem Arbeitgeber vorgelegt werden.

R14 Den gesetzlichen und gesetzlich anerkannten Kindern sind gleichgestellt:

R15 Adoptivkinder, an Kindes statt Angenommene, gesetzlich anerkannte oder auf Gerichtsbeschluß anerkannte natürliche Kinder, Kinder aus einer vorhergehenden Ehe des Ehepartners, im Sinne des Gesetzes in Pflege gegebene Kinder.

R16 Volljährige Arbeitsunfähige sind jene, die aufgrund eines körperlichen oder geistigen Mangels vollständig und dauerhaft außerstande sind, einkömmliche Arbeitstätigkeiten zu verrichten,

R17 bei Minderjährigen jene, die andauernd daran gehindert sind, die ihrem Alter entsprechenden Tätigkeiten auszuüben.

R18 Für Familiengemeinschaften mit arbeitsunfähigen Mitgliedern wird die Einkommensgrenze, die für Bemessung und Anrecht auf Familiengeld vorgesehen ist, angehoben.

R19 Die Arbeitsunfähigkeit muß mittels folgende Unterlagen in Beilage nachgewiesen sein:

R20 BESCHEINIGUNG über die Anerkennung der vollstaendigen Invalidität für Volljährige und der Zuerkennung der Begleitzulage für Minderjährige, ausgestellt von der Sanitätsbehörde.

R21 KOPIE DER BESCHEINIGUNG über eine INAIL-Rente oder einer Arbeitsunfähigkeitsrente zu Lasten des NISF.

R22 Bei Fehlen dieser Unterlagen

R23 muß eine Genehmigung auf eigenem Formbl. ANF 42 des NISF beantragt

R24 und diese sodann dem Arbeitgeber ausgehändigt werden.

IM AUSLAND ANSÄSSIGE FAMILIENMITGLIEDER

R25 Für Familienmitglieder eines italienischen Staatsbürgers oder Bürgers eines Staates (z.B. EG-Staaten), der italienische Staatsbürger nach dem Prinzip der Gegenseitigkeit behandelt bzw. mit dem ein internationales Abkommen bei Familiengeldern besteht, muß

R26 sofern die Familienmitglieder im Ausland ansässig sind,

R25 eine Genehmigung derjenigen NISF-Stelle eingeholt werden, die für den Arbeitsort des Antragstellers zuständig ist.

R27 KEINE MITGLIEDER DER FAMILIENGEMEINSCHAFT SIND:

R28 der gesetzlich getrennte Ehepartner;

R29 die natürlichen Kinder, Teil der Familiengemeinschaft des anderen, nicht mit dem Antragsteller zusammenlebenden Elternteils;

R30 Kinder, die dem anderen Ehepartner zugesprochen sind.

R31 natürliche Kinder des verheirateten Antragstellers, der nicht gesetzlich getrennt ist, und die nicht Teil der anerkannten Familiengemeinschaft sind.

R32 Ehepartner and Familienmitglieder eines EG-fremden Ausländers, die nicht im Inland ansässig sind, ausgenommen den Fall, daß der betreffende Staat die italienischen Staats bürger nach dem Prinzip der Gegenseitigkeit behandelt oder ein internationales Abkommen zum Familiengeld besteht;

R33 Ehepartner und Familienmitglieder eines italienischen Staatsbürgers oder Ausländers, die im Ausland ansässig sind und denen Familienleistungen zu Lasten des Auslandes zusteht, ausgenommen Schweiz, Liechtenstein und Staaten von ex Jugoslawien.

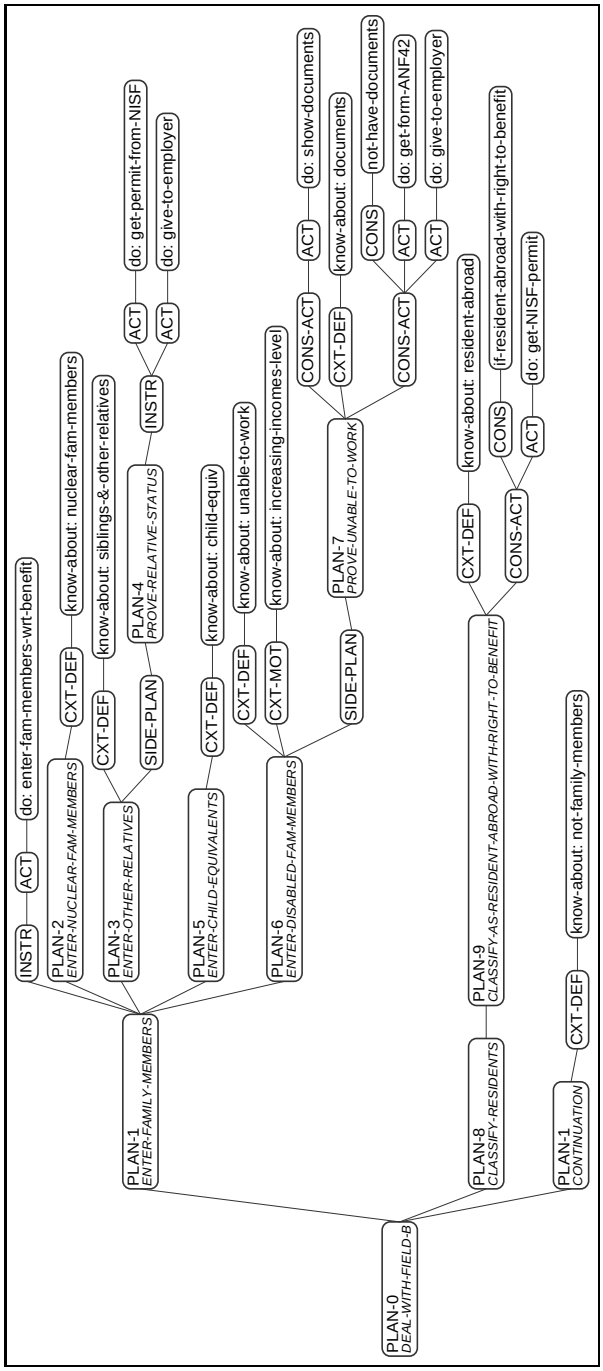
A.1.2 Relation Between Clauses

In the case of assymetric relations the nucleus occurs first.

elaboration	(*R1, R2–R33*)
abstract-instance	(*R2, R3*)
abstract-instance	(*R2, R4*)
abstract-instance	(*R2, R5*)
abstract-instance	(*R2, R6*)
whole-part	(*R2, R7–R10*)
abstract-instances	(*R7, R8–R10*)
condition	(*R8, R9*)
condition	(*R8–R9, R10*)
elaboration	(*R7–10, R11–R13*)*
condition	(*R12–R13, R11*)
sequence	(*R12, R13*)
background	(*R5, R14–R15*)
abstract-instances	(*R14, R15*)
background	(*R6, R16–R17*)
whole-part	(*R6, R16*)
whole-part	(*R6, R17*)
set-member	(*R19, R20*)
set-member	(*R19, R21*)
condition	(*R23–R24, R22*)
sequence	(*R23, R24*)
condition	(*R25, R26*)
abstract-instance	(*R27, R28*)
abstract-instance	(*R27, R29*)
abstract-instance	(*R27, R30*)
abstract-instance	(*R27, R31*)
abstract-instance	(*R27, R32*)
abstract-instance	(*R27, R33*)

**R11–R13* represents a process which must be performed when the applicant's nuclear family *does* contain the type of member described *R7–10*.

A.2 Intentional Structure



B Text 2

B.1 Rst Structure

B.1.1 RST Clauses

FELD A

R1 Nachdem der Arbeitnehmer seine Personalien eingetragen hat,

R2 muß er melden,

R3 indem das entsprechende Kästchen angekreuzt wird,

R2 ob er Familiengeld beantragen oder eine Änderung der Familienlage mitteilen will.[5]

R4 Bei Antrag auf Familiengeld

R5 ist zu beachten, daß dieses ab 1. Juli jeden Jahres bis zum 30. Juni des darauffolgenden Jahres ausbezahlt wird.

R6 Sollte das Anrecht auf Familiengeld nach dem 1. Juli eintreten,

R7 ist das Anlaufdatum des Anrechts anzugeben.

R8 Falls das Anrecht vor dem 30. Juni verfällt

R9 (z.B bei befristetem Arbeitsverhältnis),

R10 ist das Verfallsdatum anzugeben;

R11 bei Meldung einer Änderung des bereits dem NISF bekanntgegebenen Familiensituation

R12 ist das Datum der Änderung anzugeben.

R13 In diesem Fall

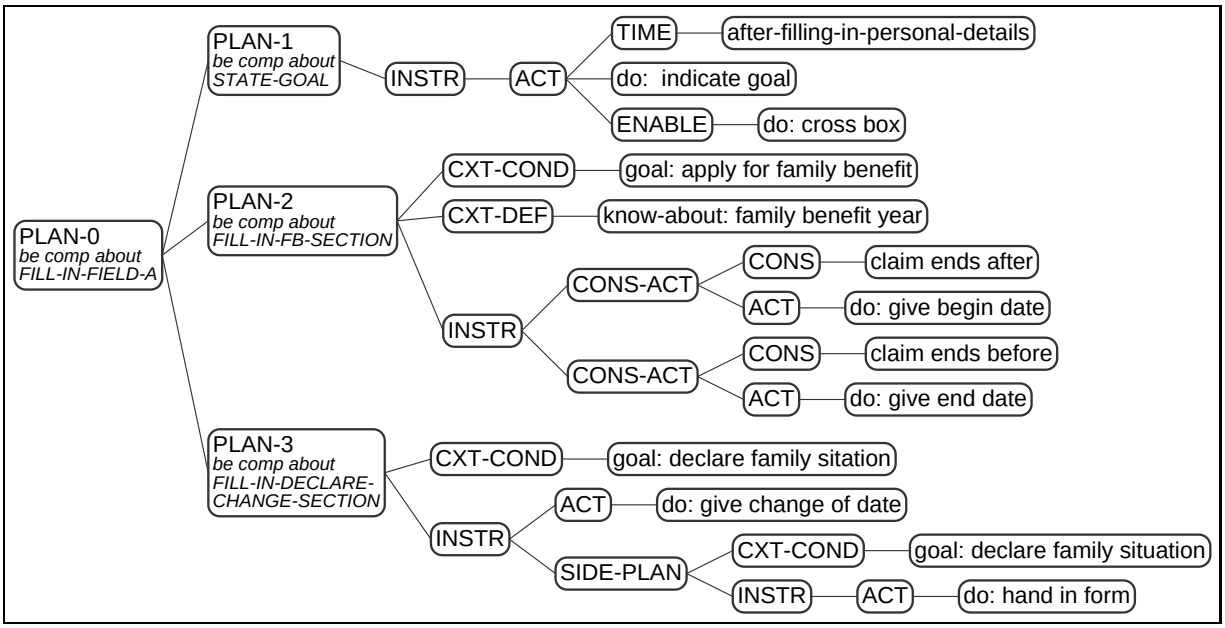
R14 ist das Formblatt innerhalb von 30 Tagen dem Arbeitgeber auszuhändigen.

[5] *or* and *and* are logical relations which do not usually play a role in the RST Structure.

B.1.2 RST Relations between Clauses

circumstance	(*R2–R3, R1*)
means	(R2, R3)
condition	(*R5, R4*)
condition	(*R7, R6*)
condition	(*R8–R9, R10*)
abstract-instance	(*R10, R9*)
condition	(*R10, R9*)
condition	(*R12–R13, R11*)
sequence	(*R12, R14*)
condition	(*R14, 13*)

B.2 Intentional Structure



C Text 3

C.1 Rst Structure

C.1.1 Rst Clauses

R1 Die Verantwortlichkeitserklärung ist vom Ehepartner des Antragstellers genauestens auszufüllen und

R2 zu unterschreiben.

R3 Nicht abzugeben ist die Erklärung von einem gesetzlich und tatsächlich getrennt lebenden Ehepartner des Antragstellers.

R4 Wird Antrag gestellt für Familiengemeinschaften mit Kindern von Geschiedenen, gesetzlich Getrennten oder natürlichen Kindern (des Antragstellers oder des Ehepartners) die vom anderem Elternteil rechtlich anerkannt worden sind oder mit Kindern eines Ehepartners aus einer vorhergehenden und geschiedenen Ehe,

R5 muß eine Genehmigung des NISF bei der für den Wohnort der Betreffenden zuständigen Amtsstelle eingeholt werden.

R6 Die Genehmigung ist auch erforderlich,

R7 wenn die Erklärung nicht vom anderen Ehepartner unterschrieben wird,

R8 entweder im Fall von Verhinderung

R9 oder wenn der Ehepartner verlassen worden ist.

R10 Der Antragsteller, der sich im Besitz der Genehmigung befindet, muß jedes Jahr,

R11 ausgenommen im Jahr, in dem die Genehmigung ausgestellt worden ist,

R10 im Feld H "Allfällige Mitteilungen" angeben, daß die Gründe, die zur Genehmigung geführt haben, weiter bestehen.

C.1.2 Relations between Clauses

sequence	(*R1*, *R2*)
contrast	(*R1–R2*, *R3*)
condition	(*R5*, *R4*)
condition	(*R6*, *R7–R9*)
abstract-instance	(*R7*, *R8*)
abstract-instance	(*R7*, *R9*)
contrast	(*R8*, *R9*)
exception	(*R10*, *R11*)

C.2 Intentional Structure