Notice

This paper is the author's draft and has now been published officially as:

Steels, Luc (2012). Design Methods for Fluid Construction Grammar. In Luc Steels (Ed.), *Computational Issues in Fluid Construction Grammar*, 3–36. Berlin: Springer.

BibTeX:

```
@incollection{steels2012designmethods,
   Author = {Steels, Luc},
   Title = {Design Methods for Fluid Construction Grammar},
   Pages = {3--36},
   Editor = {Steels, Luc},
   Booktitle = {Computational Issues in {Fluid Construction Grammar}},
   Publisher = {Springer},
   Series = {Lecture Notes in Computer Science},
   Volume = {7249},
   Address = {Berlin},
   Year = {2012}}
```

Design Methods for Fluid Construction Grammar

Luc Steels[1,2]

[1] ICREA-Institut de Biologia Evolutiva (CSIC-UPF), Barcelona, Spain
[2] Sony Computer Science Laboratory Paris, France

Abstract. The paper sketches a methodology for designing and implementing complex lexicons and grammars using Fluid Construction Grammar (FCG). FCG emphasizes a functional viewpoint of language and decomposes grammatical systems based on their semantic domains and communicative functions. Rather than directly specifying all the components of a construction explicitly, which would lead to highly complex definitions, FCG uses abstractions in the form of templates that implement design patterns common across human languages.

1 Introduction

Fluid Construction Grammar (FCG) is a new formalization of many ideas that have been proposed in the recent literature on cognitive linguistics ([20, 21, 46]) and construction grammar ([8, 14, 15, 18, 26]). A *construction* is a regular pattern of usage in a language - such as a word, a combination of words, an idiom, or a syntactic pattern - which has a conventionalized meaning and function. For example, a resultative construction implies a particular syntactic pattern of the form: Subject Verb Direct-Object Predicate, as in "Mary licked her plate clean". It expresses that the referent of the Direct-Object ("her plate") gets into a particular state ("clean") based on the action described in the main verb ("licked") and carried out by the subject ("Mary").

A *construction grammar* catalogs the different constructions in a language, both their semantic (including pragmatic) aspects and their syntactic (including morphological and phonetic) aspects. Although construction grammars are usually described only in verbal terms, particularly when the grammar is intended for second language learning or teaching, it is entirely possible to formalize and operationalize construction grammar in order to model human natural language processing. Such an implementation has the advantage of making it clear what a construction entails, and it makes the use of construction grammars in computational applications possible.

Formalizations of construction grammar differ from generative rewrite grammars in two ways:

1. The definition of constructions takes the form of *bi-directional associations* relating aspects of meaning to aspects of form, so that the same construction

can be used unchanged in parsing as well as production *without* compromising efficiency. Production here entails more than randomly generating a possible sentence. It is the process whereby the meaning resulting from conceptualization is turned into the best possible sentence respecting as much as possible known conventions of the language.

2. The bi-directional associations potentially have to take into consideration aspects from all levels of language (pragmatics, semantics, syntax, morphology and phonetics), simply because human language is not modularly organized. For example, Hungarian (poly-personal) verbal agreement is based on semantic considerations, because the position of the subject with respect to the deictic center is taken into account, syntactic considerations, because it happens only when a certain case structure is present, morphological considerations, because the form of the verb determines which suffix is used, and phonetic considerations because there has to be vowel harmony between the main vowel in the verb stem and the suffix [3]. Lexicon and grammar can therefore be best organized vertically based on data structures that cut across different levels rather than horizontally in terms of modular autonomous layers where syntax is treated independently from semantics or phonetics.

A construction in formal construction grammar therefore defines not only a particular syntactic pattern but also the semantic structure implied by the pattern, and it may include additional pragmatic, morphological and phonetic aspects, as well as the extra meaning that the construction contributes to the meanings contributed by its constituents.

FCG is one of a growing number of computational construction grammars, which also includes Embodied Construction Grammar [2] and Sign-based Construction Grammar [25]. It has been developed specifically for building deep production and comprehension systems that can act as the core of grounded human-robot or robot-robot interactions, utilizing world models derived from perception and motor activity (see Figure 1) [35]. Deep language processing requires handling rich representations of grammatical structure and an integration of semantics right into the grammar (as opposed to delegating the problem of semantics to another component).

FCG is based on techniques widely used in current computational linguistics, in particular the representation of linguistic structures with feature structures [4, 7], and the use of unification for applying constructions to expand linguistic structures in language parsing and production, as pioneered in Functional Unification Grammar [17], and also used in Lexical Functional Grammar [9], and Head-driven Phrase structure Grammar ([28, 29]). At the same time, FCG introduces a number of innovations, such as a powerful structure-building operator called the J-operator. FCG is implemented on top of a Common LISP environment as most other computational grammars. It is fully operational and made available for free to the research community (http://www.fcg-net.org/). It has already been used in a variety of experiments in (artificial) language evolution and human-robot interaction [38, 39, 43].

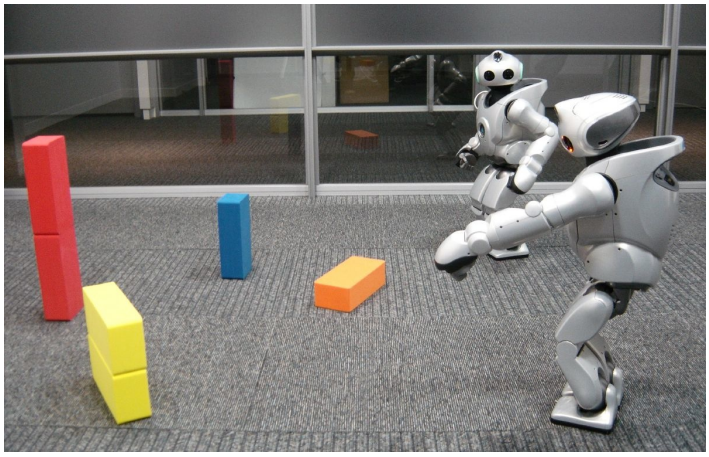

Fig. 1. FCG has been developed for experiments in human-robot and robot-robot interaction, which requires that not only lexical and syntactic issues be handled but also semantics and grounded meaning.

There are now various introductory texts and papers reporting worked out examples in FCG (see in particular [41] and [43]). The present paper focuses on the design methods that have emerged for coping with the complexity of real world grammars. Complexity is not meant here with respect to the size of the inventory of constructions (although that is also a critical point) but rather with respect to the depth with which the relevant linguistic phenomena are handled. Computer science has a lot of experience in building very complex systems and has proposed various design concepts such as the use of design patterns, computational abstractions, compilation from high level specifications, etc. These same concepts are potentially of great value in grammar design as well.

Typically, constructions in FCG are defined in terms of a layer of abstractions based on *templates*. Templates capture common design patterns relevant for human languages, such as functional structure, agreement, field topology, valence, linking, etc. [41] Different templates together build an operational construction so that a modular design remains possible, even though at the operational level this modularity is no longer explicitly present. Efficiency considerations are separated as much as possible from design considerations. Efficiency is gained by compiling templates, by maintaining dependencies between constructions that can be used for priming [49] and by chunking combinations of constructions [32]. The use of templates also plays an important role in modeling language learning, because they can act as primitive operators for expanding or changing constructions.

The remainder of this paper is structured along the three proposed levels of grammar design and implementation. The first section addresses the linguistic level. It introduces the main linguistic principles of construction grammar in general and the specific linguistic approach that is used in Fluid Construction Grammar in particular. These principles are familiar to linguists but are included to help computer scientists grasp the fundamentals of construction grammar. The next section turns to the design level, explicating the notion of a design pattern and the templates used to implement them. The third section turns to the operational level, introducing some of the main representational mechanisms and processing steps available in FCG. The operational level is defined here only very briefly. It is discussed more formally in a later chapter of this book [10] and the reader is referred to the introductory papers in [41] and the manual and other on-line resources on the FCG distribution website: http://www.fcg-net.org/.

2 Linguistic Level

This section briefly introduces the general approach to language that is advocated in construction grammars, and some of the more specific choices that have been adopted for Fluid Construction Grammar. One of the most principled points is that Fluid Construction Grammar advocates a functionalist rather than formalist perspective on language.

2.1 The Functionalist versus Formalist Perspective on Language

The debate between functionalists and formalists has been raging in linguistics for a long time, although the debate is often more rhetorical than real [27]. Nevertheless, there is a profound difference in point of view, with significant consequences on how one approaches language processing and language learning. From a functionalist point of view, language is primarily seen as a tool for communication. The speaker is influencing the cognitive activities that go on in the mind of the hearer, so that the hearer will pay attention to certain aspects of the world, perform certain actions, store information, start a thinking process, etc. A specific sentence is a way to evoke some of the tools provided by the language and provide specific settings for their usage.

From a formalist perspective, language is not primarily studied as a communicative tool. The focus of analysis is on the perceivable properties of sentences and their structure. It would be like describing a hammer as consisting of a cylinder and a block, attached in a particular way to each other, whereas a functionalist perspective would describe a hammer as consisting of a handle and a head. The handle is for holding the hammer and the head for hitting the object. The handle and the head typically take the form of a cylinder and a block, but the functionalist perspective emphasizes the importance of their function, particularly because the same function can often be achieved by objects with many different kinds of shapes. For example, the two cylinders could be of equal size, the handle could either stick through the head or be attached to it in some other

way, the head could take the form of a block instead of a cylinder, etc. More radically, an entirely different object such as a shoe can become a hammer with the sole functioning as head.

In linguistics, those adopting a formalist viewpoint emphasize the syntactic structures of a language, usually by defining a procedure for generating all possible structures judged to be grammatical in a language, whereas those adopting a functionalist viewpoint focus on identifying how forms achieve syntactic and semantic functions. Thus, the distinction between nouns and verbs is not viewed as purely structural, i.e. in terms of which syntactic structures they can be part of, but in terms of possible functions that nouns and verbs can have in communicative activities: Nouns can be used as nominals to introduce classes of objects for forming referring expressions whereas verbs typically introduce classes of events to describe a state of affairs.

Generally speaking, taking a functional stance for dealing with tools has a number of clear advantages. The better you understand the function of each component of a tool the more you can use it properly. A functional perspective also helps to select the right variant of a tool for a particular context, for recognizing a tool even if it does not have a classical shape, and for improving it. Of course, often people use tools without understanding them fully and just imitate how they have seen others using the tool. This stage is often the first step towards acquiring full mastery which unavoidably requires a functionalist perspective.

The same advantages can be seen when language is treated as a tool. A functional analysis leads to a better understanding of why language is the way it is and to parsing and production systems that are more flexible and robust [44]. It suggests also an alternative to purely statistical language learning techniques, one which uses *functional inference* instead of inductive inference. Concretely, the learner figures out the meaning and the function of unknown words or constructions by reconstructing which unknown forms were introduced to achieve them [34].

2.2 Semantic Functions and Cognitive Operations

Once we adopt a functionalist view on language, it becomes clear quite quickly that every element in a sentence has both a meaning and a semantic function. The *meaning* or semantic content of a word is the concept it introduces, and the *semantic function* is what is to be done with this building block during interpretation. For example, the word “slow" in “the slow train" evokes the concept ‘slow’, which concerns the relative speed of moving objects. Its semantic function, at least in this phrase, is to restrict the set of possible referents of the noun, specifically to restrict the set of trains in the context to those that have a slow speed.

The same meaning can have a variety of semantic functions, as illustrated in the following sentences:

1. The *slow* train.

2. They *slow* down.
3. The *slow* go first.
4. The train was *slow*.
5. They ran *slowly*.

All these sentences make use of the concept of 'slow', but they differ in what they require the hearer to do with this concept: (in 1.) to use it to further restrict the class of possible referents, (in 2.) to circumscribe the movement of the subject of the sentence, (in 3.) to identify a class of objects based on their speed, (in 4.) to assert a property of the subject, and (in 5.) to provide an additional attribute of the movement introduced by the verb.

The notion of semantic function is often only vaguely defined. When we want to formalize and operationalize grammar we need a more precise definition. FCG assumes the perspective of procedural semantics, which proposes that the interpretation of a sentence requires executing a set of cognitive operations over a perceptually grounded world model and discourse model [50, 51]. Each operation yields a particular result (for example it delineates a set of objects in the context) which can then again be used by other operations. The operations include set operations, selection of elements out of sets, filtering operations, the computation of perceptual features of specific entities, geometric transformations of positions of objects to achieve perspective reversal, etc.

Here are some concrete examples of cognitive operations:

1. `get-context`, which gets the objects in the present context and stores them in a discourse model.
2. `filter-set-class`, which takes a set of objects in the discourse model and filters out those which belong to a certain identified class.
3. `count-elements-set`, which takes a set of objects in the discourse model and counts how many elements it contains.
4. `select-unique-element`, which selects the unique element out of a singleton set.
5. `set-union`, which forms the union of two sets.
6. `describe-event`, which asserts that a particular event takes place in the current context

Our group has developed a meaning representation system called Incremental Recruitment Language (IRL) that can be used for grounded procedural semantics [33]. Each cognitive operation in IRL takes a number of arguments, one of which (usually called the target argument) is considered to be the result of the cognitive operation. Cognitive operations can be combined together in a network with one operation providing or using results produced by another one, as shown for example in Figure 2 from [31], which represents the procedural semantics of the German phrase "der vordere Block" (the front block). When such a network is executed, each cognitive operation computes specific values for its arguments as fitting with the present world model and discourse context. These values propagate in the network until no more computation can be done. IRL is not further discussed in the present book because its details are not directly

relevant to grammatical processing. See reference [30] for an introduction and more examples.

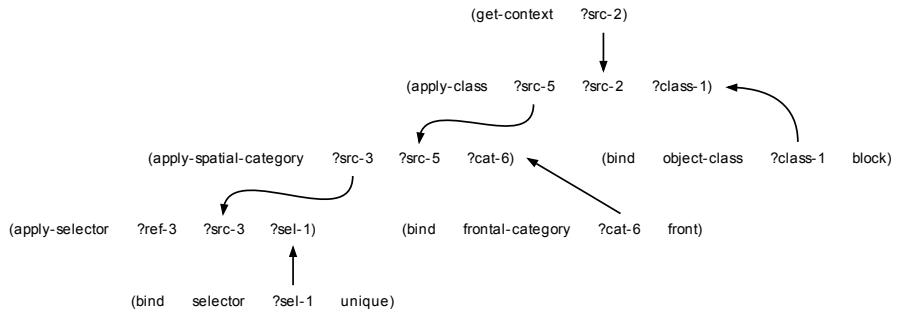

Fig. 2. Example of a network of cognitive operations as needed for interpreting the German sentence "der vordere Block" (the front block). Each cognitive operation has a number of arguments (indicated with a question mark in front of them) which are to be filled by specific values. The arrows indicate that there are data flow relations between the different cognitive operations.

We can now be more precise about what a semantic function is. It specifies the *role* of a conceptual building block, such as 'slow', in a particular cognitive operation. For example, the semantic function of "slow" in "the slow train" is called a *qualifier* because the concept 'slow' is used in this case as the qualifier of a `filter-set-qualifier` operation. This operation filters a set of objects in the discourse model with respect to whether they satisfy a given attribute. The target outcome of this filtering operation is to further restrict a set of possible referents.

With each semantic function corresponds also a *syntactic function* which is associated with a *lexical category*, also known as a part of speech or a word class. For example, the semantic function of "slowly" is to modify the movement concept introduced by the verb "ran". This syntactic function is usually called adverbial. The adverbial function is signalled by the affix "-ly", which turns "slow", by default an adjective, into an adverb.

The same lexical category is associated with many potential syntactic functions and the same syntactic function with many potential semantic functions. For example, an adjective may be used as a qualifier (as in "the slow train") but also as a predicate (as in "the train is slow"). The same word "light" may be used in an adjectival function as a qualifier (as in "the light block") or in an adverbial function as a modifier of another color concept (as in "the light green block").

In parsing, the choice of which syntactic or semantic function is actually the case is based on the syntactic and semantic context. In production, the choice is based on what cognitive operations the speaker wants the hearer to perform. For example, if the speaker wants the hearer to perform a `filter-set-qualifier`

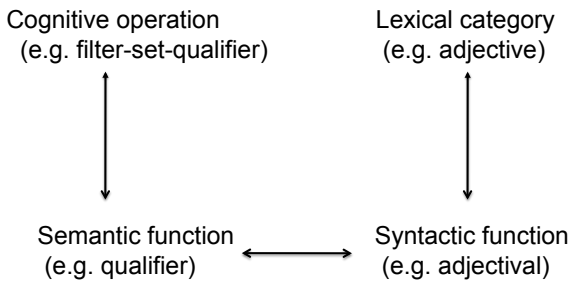

Fig. 3. From a functional point of view, lexical category are associated with possible syntactic functions, which are associated with possible semantic functions. A semantic functions specifies the role of a concept in a cognitive operation.

operation using 'slow', this will require a word that can act in an adjectival function. Many human languages (including English) are quite flexible with respect to which lexical category can satisfy which syntactic function because a word which by default belongs to one category can often be coerced into another category, and this coercion can then become conventionally accepted in the language. For example, in "The slow go first", the adjective "slow" has been coerced into a nominal function so that it can be used semantically to identify a class of objects.

The associations between lexical categories, syntactic functions and semantic functions are bi-directional (see figure 3). Information about the lexical category of a word is used during parsing to hypothesize a possible syntactic function, which is then used to hypothesize a possible semantic function. During production, the mappings are used in the reverse order. A particular semantic function is potentially actualized with a particular syntactic function, which is then expressed using words with specific lexical categories. The table in Figure 4 contains some more examples of such associations, all involving the word "slow" as illustrated in the sentences given earlier.

2.3 Phrasal Structures

One of the key characteristics of human languages is their hierarchical or compositional nature. A sentence does not only consist of individual words which each have their own syntactic and semantic functions, but of phrases that group several words or phrases together so that they can function as units in their own right. Phrases belong to a particular syntactic type (for example nominal phrase) and have the potential to take on syntactic or semantic functions within larger phrases, just like individual words. The meaning of a phrase consists of a network of cognitive operations whose arguments are interlinked. These operations and

lex-cat	syntactic fnct	semantic fnct	cognitive oper
1. adjective	adjectival	qualifier	filter-set-qualifier
2. verb	verbal	event	describe-event
3. noun	nominal	identifier	filter-set-identifier
4. adjective	predicate	predicate	describe-predicate
5. adverb	adverbial	modifier	apply-modifier

Fig. 4. Examples of associations between lexical categories (parts of speech), syntactic functions, and semantic functions. The relevant cognitive operation is given in the last column.

some of the bindings for their arguments are provided by the individual words, but the phrase may add additional cognitive operations and linkings of its own.

From a functionalist perspective, phrases are primarily defined in terms of the functions of their constituents, just as a hammer is primarily defined from a functional point of view in terms of the functions of its components. For example, a nominal phrase like “the train" consists of a determiner and a nominal. The semantic function of the nominal is to identify a set of objects in the present context, and the determiner then specifies which element or elements needs to be selected out of this set.

The constraints that allow a constituent (word or phrase) to play a particular role within a phrase depend on the language. They typically include the following:

1. *Syntactic Categorizations*: Constituents of a phrase typically have to belong to certain lexical categories (if they are individual words) or phrase types (if they are phrases themselves), or they need to have at least the potential to take on certain syntactic functions, possibly after coercion. For example, a constituent can normally only be the determiner of a nominal phrase if it is an article or numeral.
2. *Semantic Categorizations*: The referent of a constituent typically has to be of a certain semantic type. For example, the meaning of a nominal has to produce a set, and this set is often restricted based on the kind of nominal phrase it occurs in. For example, a plural definite article implies that the set of objects from which the determiner selects one or more elements has to be a countable set.
3. *Ordering*: Often the constituents of a phrase are constrained in terms of the order in which they appear inside the phrase. For example, a compositional color description such as "light blue" requires that the constituent introducing the non-hue category acting as modifier appears before the hue category acting as the qualifier.
4. *Agreement*: Often there are syntactic and semantic agreement relations between the constituents of a phrase. For example, the constituents functioning

as the determiner, adjectival, and nominal of the same nominal phrase have to agree with respect to number and gender in French.

The primary function of a phrasal construction is to define all these characteristics and to specify also properties of the phrase as a whole, such as which features percolate up from constituents to the phrase (for example, if the noun in a nominal phrase is feminine then the phrase as a whole will be feminine in French), what additional syntactic and semantic properties hold for the phrase, and how the meaning of the phrase is composed by combining the meanings of the parts [40].

2.4 The Grammar Square

Lexical items and phrase structure form the backbone of a sentence because they signal the main conceptual building blocks that the speaker wants to convey and how these building blocks are to be used when interpreting a sentence. It turns out that human languages often convey various additional meanings (usually called *grammatical meanings*) by modulating and extending the basic skeletal structure or by adding morphological or phonetic variations on the words already used. This is similar to the way in which a (classical) figurative painting introduces a basic structure for the scene, such as the different figures against the background, and then superimposes additional aspects. For example, the painter may use a particular color palette to give the scene an emotional quality, or add contrast in terms of lighter and shaded areas to highlight some of the elements in the scene, or depict strong facial expressions and postures to add drama.

Grammatical meanings expressed in many human languages can be classified in terms of the different semantic and functional domains they are concerned with. Here are some examples:

- *Argument-structure*: Most languages have ways in which the roles of participants in events can be made clearer by specifying who does what to whom, overlaid on the basic phrase structure, for example, by using cases and morphological affixes or by exploiting the sequential ordering of constituents in the sentence and using specific prepositions.
- *Tense-Aspect*: Many languages have ways to be more precise about the relation of an event with respect to the time of discourse (present/past/future) or to highlight the internal structure of events (ongoing, terminated, repetitive, etc.).
- *Modality*: Many languages have ways to express the epistemic attitudes of the speaker with respect to the information that is provided in the sentence. For example, they may specify whether the state of affairs described by the main verb is true or only hypothesized, what evidence was available and how this was acquired, whether the source is reliable, etc.
- *Determination*: Many languages have ways to be more precise about how the referent of a nominal phrase can be accessed, usually in terms of different articles such as “the", “some", or “every", or in terms of marking morphologically the distinction between mass or count nouns or definite and indefinite referents.

- *Social status*: Many languages express the social attitude of the speaker or the relation between speaker and hearer by adding morphological markers or particles, or by choosing other lexical items. A very simple example is the use of “tu" versus “vous" for the second person pronoun in French.
- *Information structure*: Many languages are able to provide information related to discourse, such as what object is assumed to be known from earlier conversation, what information addresses a question posed in the dialog, what the main highlighted topic is, etc. For example, German marks a constituent as being the focal topic of a sentence by moving it into first sentence position (fronting) or by stressing it.

Not all grammatical meanings are explicitly expressible in all languages, and for each language there are significant differences in the semantic distinctions they express and how they do it. What is common, however, is that grammatical meanings are mapped to syntactic forms through the intermediary of semantic and syntactic categorizations that are language-specific (Figure 5).

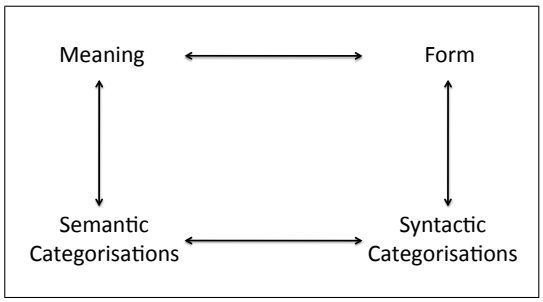

Fig. 5. The grammar square depicts the different associations between meaning and form. Meaning is directly related to a specific form in the case of lexical constructions. But in the case of grammatical meanings it goes through the intermediary of semantic and syntactic categorizations.

This multi-layered character of the mapping from meaning to form is characteristic for grammar, in opposition to the lexicon, which immediately maps aspects of meaning to aspects of form. Grammatical mappings are always bi-directional. In production, they are traversed from meaning to form and in production from form to meaning. The reason why languages use indirect mappings for grammar, rather than going directly from meaning to form, is that this allows the definition of more abstract relations between meaning and form and because other factors may play a role in the mapping so that more information can be packed into the same material.

Here is a concrete example of this multi-layered mapping for the domain of argument structure (illustrated in more detail in [47]). Specific participant roles such as the pusher of a push-event are not directly mapped onto syntactic markers, because that would require a large set of markers, specific to each

verb. Instead, participant roles are semantically categorized in terms of abstract semantic roles such as agent, patient, beneficiary, instrument, location, etc., and these semantic roles are first mapped onto syntactic cases such as nominative, accusative, dative, genitive (or in language without a case system onto grammatical relations such as subject, direct object, indirect object, oblique object), before they are mapped to surface markers such as morphological affixes or constituent orderings. At each step, context, and hence additional meaning, can play a role. For example, the semantic role of agent can be mapped to subject (or nominative) in active sentences only. When the speaker wants to highlight the patient, he can make it the subject, as in “the block was pushed to the edge of the table".

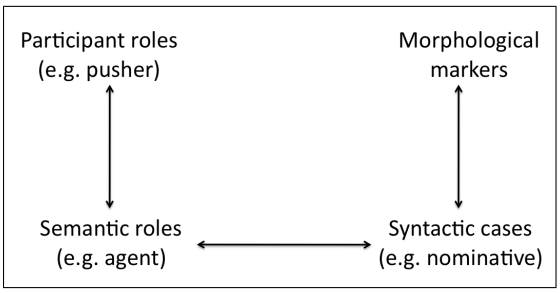

Fig. 6. Instantiation of the grammar square for argument structure. Participant roles in events are not directly mapped to specific markers but through the intermediary of abstract semantic categorizations in the form of semantic roles, such as agent or instrument, and abstract syntactic categorizations in the form of grammatical relations or cases, such as subject or nominative.

Semantic and syntactic categorizations cannot always be so neatly distinguished. For example, in the case of tense, many languages use a more direct mapping from semantic categorizations of the time moment of an event (present - past - future) to surface realizations based on auxiliaries or morphological markers (as in: come - came - will come). Nevertheless, in analyzing the grammar of a language it is extremely useful to keep the distinction in mind between the four layers of the grammar square and to study what contextual factors influence the mapping from one layer to another.

2.5 Analysis Steps

The various insights into the nature and functioning of human language briefly discussed in the previous subsections, translate into a series of steps for analyzing a fragment of language.

1. The first step is to delineate the primary semantic domain of interest and the communicative functions that will be investigated. For example, a study might focus on the domain of spatial language, which uses spatial relations and

perspective to identify objects in a scene, as in “the ball left of the box" [22]. Another study might focus on the description of events, using sentences such as “the ball rolled off the table", which would imply a classification of event types and aspects of events that are to be made explicit [45]. Next, the kind of world model that will be derivable from perceptual and motor processing and the information that needs to be stored in the discourse model has to be determined as well as the concepts that are available in this semantic domain, such as spatial categories, event types, or image schemas for objects. Finally, the cognitive operations that will make use of these building blocks need to be identified as well as the networks that are needed to achieve the communicative functions in the chosen domain (as shown in Figure 2).

2. The next step is to survey which meanings can be directly covered by words in the lexicon. Typically, words will introduce certain concepts, and each word will have a particular potential for expressing semantic functions. The potential comes from the lexical categories to which the word belongs, which allow it to have certain possible syntactic functions.

3. The third step focuses on phrases, identifying what kind of phrases are available in the language fragment under investigation, how each phrase combines the networks contributed by its constituents, and what additional meanings are to be added. This step also investigates what syntactic and semantic constraints have to be present, for example, what ordering constraints have to be imposed among the constituents, whether there are agreement relations between constituents, which features should percolate from constituents to the phrase as a whole, and so on.

4. In addition to the primary semantic domains which determine the basic skeleton and vocabulary of a language fragment, there may be secondary semantic domains that are overlaid on top of the primary domains, expressing grammatical meanings such as modality, tense, aspect, determination, or information structure. For each of these domains, the analyst needs to pin down what has to be represented in the world model or the discourse model as well as how this information can be derived through sensory-motor processing or inference. The domains will also introduce additional cognitive operations and subnetworks. Secondary domains usually do not introduce new syntactic functions, but they operate through syntactic and semantic features that translate into morphological markers, grammatical function words, modulations of constituent ordering or phonetic markings (intonation, stress). The grammar square mappings from meanings to semantic categorizations, syntactic categorizations, and finally to markers must be worked out and in particular the contextual constraints that determine each of the steps in the mapping.

3 The Design Level

In the previous section, the basic linguistic outlook used in Fluid Construction Grammar was briefly presented, and a set of methodological steps was introduced that help to structure the investigation of some fragment of a language.

However, it remains an enormous challenge to turn this kind of analysis into a formal system that is effective in parsing and producing utterances, particularly because language is to a large extent non-modular and multi-functional. The same element (for example, the same word) can have multiple functions and play a role in expressing different semantic domains, and constraints from many different levels (pragmatic, semantic, syntactic, morphological and phonetic) often interact strongly. This multi-functional nature of linguistic components introduces enormous challenges for grammar design, both to master the complexity of defining constructions that capture all constraints, but also to avoid combinatorial explosions in the search space built up during parsing or production.

FCG uses several techniques from computer science for handling complexity. The first one is to make a distinction between a design level and an operational level, in the same way computer programming makes a distinction between a high level programming language and a low level machine-oriented language. In the case of FCG, the operational level is based on the detailed definition of operational constructions that are applied through a process of matching and merging. At the design level, these constructions are defined more abstractly using a series of templates. Each template deals with a particular aspect of a construction and helps to implement a certain design pattern.

3.1 Design Patterns

The notion of a *design pattern* was originally introduced by architects [1]. A design pattern captures a particular solution to a design problem that can be reused, after adaptation to local circumstances. For example, a dome structure can be used for spanning large spaces, but many different types of domes are built depending on the materials used or the aesthetic qualities the architect is after. Design patterns are also very common in software engineering where they refer to reusable approaches for tackling a class of software design problems [12]. They are also an important concept in biology, where they refer to a particular class of physiological and metabolic solutions for recurrent problems like maintaining body temperature, extracting oxygen, building a basic body plan [5]

In human languages, we also find common design patterns across language families. The most obvious one is to build *phrases* by grouping words (or phrases) in order to express compositional meanings. The nature and complexity of phrases, and which mechanisms are used to indicate which constituents can be grouped together, differs significantly from one language to another. Nevertheless, there are a lot of common mechanisms.

Another design pattern commonly used in human languages is that of *agreement systems*. Agreement means that some syntactic or semantic features of one constituent are shared with other constituents. Agreement is used to indicate that there is some sort of linking between constituents, for example, because they are components of the same phrase or because they share the same referent. Thus, the subject and the verb of a sentence agree with respect to number and person in English. In German, articles and nouns which belong to the same

nominal phrase agree for number, gender, and case. In Spanish, pronouns agree with respect to gender and number with the referring expression that introduces their referent. The features entering in agreement relations are typically derived based on semantic categorizations (for example the distinction between singular and plural number) or on the syntactic context (for example the case distinction between nominative and dative).

Features are either associated with words themselves, or they are explicitly marked through morphology or phonetic variations of the word stem. Within phrases, they often percolate up from one constituent (usually the so called head of the phrase) to the phrase as a whole. For example, definiteness percolates from the determiner to the determiner-nominal phrase. Individual languages differ in terms of when they use agreement relations, which features are supposed to agree with each other, and which features percolate. But again, we find some common fundamental mechanisms. For example, features usually come in feature bundles , and often a particular word or phrase has multiple alternative feature bundles so that special processing techniques are needed to avoid combinatorial explosions [48]. A concrete example for Polish agreement is discussed in a later chapter of this book [16].

Another example of a design pattern is *field topology*. Field topology is a way to introduce more flexibility and sophistication in the use of word order within a phrase. In most languages, it is sufficient to express ordering constraints among the constituents of a phrase with the `meets`-relation, which is valid between two constituents X and Y if X immediately precedes Y. But sometimes this kind of representation is too weak because the ordering relations are exploited to express many different grammatical meanings. This is the case, for example, in German sentences, where almost any constituent can be put in front of the sentence in order to emphasize that it is the topic, for example the answer to a recently asked question.

Field topology associates with each phrase a set of fields. A field can capture one or more constituents, depending on a number of interacting constraints, and the final ordering of the sentence is then derived by sequentially retrieving the fillers of each field. For example, in German, five fields are typically hypothesized. There is the so called *Vorfeld*, or front field, which may capture constituents that express the topic of the sentence. The finite verb always comes into the second field (the *linke Klammer*, or left bracket), followed by three fields for other constituents: the *Mittelfeld*, or middle field, the *rechte Klammer*, or right bracket and the *Nachfeld*, or end field. The *rechte Klammer* contains the non-finite verb. Most implementations of German constituent order use a field topology approach by postulating these five fields. A worked out example in FCG is discussed in [24].

Although field topology has been primarily used for German (and Dutch) sentence constituent ordering, there are many other ordering phenomena that can potentially be handled using this design pattern. For example, the ordering of multiple adjectives in French follows a particular sequence, with some adjectives appearing before and some after the noun [19]. The ordering is mostly based

on the semantic categories to which these adjectives refer. For example, size adjectives appear before the noun whereas color adjectives come after, as in “un grand ballon rouge" (literally: "a big ball red", to mean "a big red ball") We can handle this phenomenon as well by postulating a set of fields for each type of adjective. During production, these fields capture the adjectives that satisfy the constraints of the field, and the nominal phrase is then constructed by collecting the fillers of these fields sequentially, which are non-empty.

Agreement and field topology are design patterns oriented towards the syntactic structuring and morphological markings of sentences. There are other design patterns that are oriented towards semantic aspects. For example, the networks contributed by individual words need to be linked together in phrases, and networks from different phrases need to be linked together into networks defining the meaning of larger phrases [37]. A design pattern for handling this kind of issue is known as a linking pattern.

A *linking pattern* relies on the definition of the external arguments of subnetworks supplied by words or constituents. These arguments can be linked to the external arguments supplied by other subnetworks. For example, the cognitive operation `filter-set-class`, which filters a set of objects based on a class, has two external arguments: one for the class and the other for the filtered set. The cognitive operation `select-unique-element` selecting the unique element out of a singleton has two external arguments as well: one for the set from which an element has to be selected and the other for the chosen element. When these two cognitive operations are combined in a single network (as in the phrase “the mouse", where “mouse" introduces the class identifier), the new network has only two external arguments: One for the selected element and another for the original source set as used by the filter operation. Internally, the non-external arguments are linked together in the combined network so that the set derived by `filter-set-class` on the basis of the class ‘mouse’ is the set from which the element is selected by `select-unique-element`.

3.2 Constructions

A *construction* is the basic computational unit at the operational level in FCG. It contains a *semantic pole* that captures aspects of pragmatics, meaning, semantic structure and semantic categorization, and *a syntactic pole* that captures aspects of syntactic structure, syntactic categorization, as well as phonetic and morphological marking. Constructions typically have a set of units for the different constituents. Information is represented in terms of features associated with each of the units.

The data structure built during the parsing and production of a particular sentence is called a *transient structure*. It has the same division into units and features as a construction, as well as a semantic and a syntactic pole. Initially the transient structure contains only one unit (usually called the *top-unit*), which contains everything that is known when processing starts. In parsing, it contains a feature `form` that contains a description of all the form characteristics of the

utterance. In production, the top-unit contains a feature `meaning` that contains the complete meaning which the speaker wants to express.

A construction is viewed as a bi-directional association between meaning (the semantic pole) and form (the syntactic pole). In production the semantic pole of the construction is matched against the semantic pole of the transient structure in order to check whether the construction is applicable, and then the information contained in the syntactic pole of the construction is added by merging it with the syntactic pole of the transient structure. In parsing, the syntactic pole of the construction is matched against the transient structure to see whether the construction is applicable, and if this is the case then information from the semantic pole is merged with the transient structure so as to progressively reconstruct the semantic structure and meaning of the sentence. One construction thus prepares the ground for the application of the next construction, so that we get a chain of construction applications (see Figure 7). Usually there is more than one construction that can apply at any point in time, and we therefore get a search space in which different search paths have to be tried out to find the best possible solution.

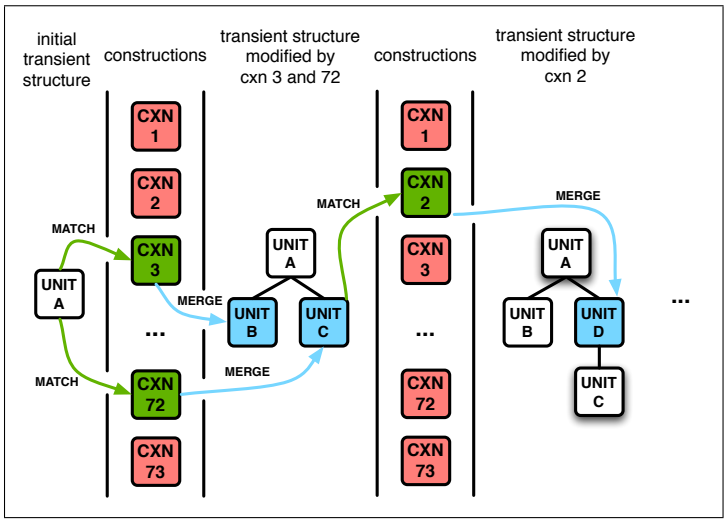

Fig. 7. Constructions are applied in a chain starting from an initial transient structure. Application either fails in mid-stream or continues until a complete sentence can be produced (in language production) or the meaning could be completely reconstructed.

Constructions are complicated because of their bi-directional nature. It is not enough to say, for example, that a determiner-nominal phrase consists of a determiner and a nominal, we also need to say when such a determiner-nominal phrase should be used (in production) and how the meaning of the determiner and nominal is to be combined (in parsing). We also need to specify the agree-

ment relations, percolations, and linkings of subnetworks. An additional factor that makes constructions complicated is that both syntactic *and* semantic issues need to be considered at the same time. This leads to much greater efficiency compared to a separation of the grammar into different layers or a decomposition of linguistic decisions into small steps. A construction should take as many constraints as possible into account before building more structure, so that search gets maximally avoided. On the other hand, it makes it much harder to write grammars.

To make the analysis and implementation of grammars nevertheless doable, FCG divides constructions up into different *construction sets*. This helps also to streamline the processing of constructions because all constructions in one set can be considered before the next set in the sequence is tried. Here are some example construction sets that are typically found for most applications:

1. *Lexical constructions* introduce lexical items, i.e. word stems. They specify the meaning, external arguments, phonetic, syntactic and semantic categories, and form of a word.
2. *Morphological constructions* introduce morphemes, i.e. prefixes and suffixes that are attached to word stems. They specify with what word the morpheme can be combined, the form the combintion takes, syntactic and semantic categorizations, the form of the morpheme, and possibly its phonetic features.
3. *Functional constructions* are concerned with defining associations between lexical categories, syntactic functions, and semantic functions. They define potential values. The actual values are decided based on the syntactic and semantic context.
4. *Phrasal constructions* are concerned with capturing constraints on phrases: What constituents there are, what the semantic constraints on constituents are, which semantic and syntactic function they should have, what kind of agreement, ordering, and percolation phenomena must be taken into account.

There are usually additional construction sets that deal with the expression of grammatical meaning. For example, it could be that a language expresses argument structure using semantic roles and cases, and this would then lead to the inclusion of *argument-structure constructions* [47]. Or it could be that a language has an elaborate system of aspect which would require constructions that handle the semantic and syntactic features related to aspect so that morphological constructions can mark them with external forms [13].

3.3 Templates

A design pattern does not directly translate into a particular construction because one construction will integrate aspects of many different design patterns, and a single design pattern has an impact on many different constructions. For example, to implement agreement requires that lexical constructions, morphological constructions, and phrasal constructions introduce syntactic and semantic features and that agreement relations are defined within the context of the relevant phrasal constructions. It is nevertheless possible to capture some of the

basic aspects of design patterns into abstractions that hide a lot of the implementation details. FCG does this using templates.

A *template* determines some of the aspects of a construction. It has a number of slots which act as parameters for how exactly the template should be instantiated. When a template is applied to a construction it extends the construction as a side effect (Figure 8). Each construction has a unique name so that templates can retrieve the construction they want to have an impact on. Moreover, the units in a construction are associated with variable-names so that they can be used by different templates to add more information.

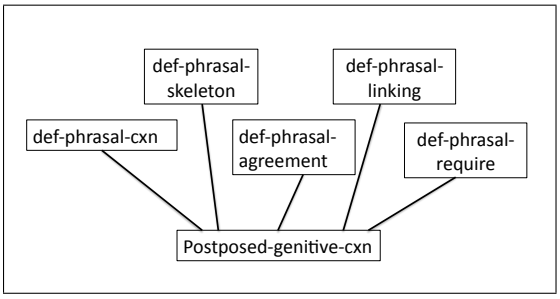

Fig. 8. Different templates progressively add more structure to a construction. Each template adds information relevant to a particular design pattern. For example, the `def-phrasal-agreement` template adds mechanisms to implement the relevant agreement relations to the `postposed-genitive-cxn`.

The general syntax for using a template takes the following form:

(*template-name* *construction-name* *optional-parameters*
:slot-name-1 *value-name-1*
. . .
:slot-name-n *value-name-n*)

The set of possible templates is open, and a grammar designer implements the specific templates required for the language phenomena that he or she is interested in and from then on uses these templates. There are libraries of templates made available with each FCG release. The more specific templates are, the easier it is to focus on the specific linguistic aspects of the language being studied because computational issues are hidden as much as possible. But the less they will be relevant for other languages.

The values of slots can either be symbols, lists of symbols, or expressions using the same special operators as used at the operational level. FCG uses logic variables as commonly used in logic programming languages. They are denoted by putting a question mark in front of the name of the variable. Variables get bound as a side effect of the matching and merging process, either to constants or to other variables. (Details of FCG-variables, special operators, and the basic unification operations that use them are discussed in a follow up chapter [10].)

Here are a few examples of templates. There are first of all some 'shell templates' that create a shell for a construction with a given name. The template also puts the construction in a particular construction set. Shell templates are of the form

`(def` *-construction-type construction-name*
`...` *invocation of other templates* `... )`

Typical names for shell templates are `def-lex-cxn`, `def-morph-cxn`, `def-fun-cxn`, `def-phrasal-cxn`, etc., to build lexical, morphological, functional or phrasal constructions respectively.

For example, let us initialize the definition of a lexical construction called `mine-cxn` for defining the word "mine", as it may appear in possessive constructions, such as "this house of mine". The process of building this construction starts with the creation of a shell using the `def-lex-cxn` template. Only the name of the construction has to be supplied:

```
(def-lex-cxn mine-cxn)
```

Next, there are typically templates that define the basic skeletal structure of each construction. For example, lexical constructions primarily associate meaning with a string. Thus there is a template called `def-lex-skeleton` with slots for `:meaning` and `:string`.

```
(def-lex-skeleton mine-cxn
  :meaning (== (context ?context)
               (dialog-participant ?indiv speaker ?context))
  :string "mine"))
```

The linking design pattern requires defining the external arguments of the subnetwork introduced by this lexical item, which is usually done with an extra slot called `:args` in the `def-lex-skeleton` template:

```
(def-lex-skeleton mine-cxn
  :meaning (== (context ?context)
               (dialog-participant ?indiv speaker ?context))
  :args (?indiv)
  :string "mine"))
```

Syntactic and semantic categorizations are associated with lexical items by another template, called `def-lex-cat`. It has a slot `:sem-cat` for the semantic categorizations and a slot `:syn-cat` for the syntactic categorizations:

```
(def-lex-cat mine-cxn
  :sem-cat (==1 (sem-function possessive))
  :syn-cat (==1 (lex-cat pronoun)
                (person 1st)
                (number singular)
                (case genitive)))
```

The semantic function of “mine" is that of possessive. From a syntactic side, it is a 1st person, singular pronoun in the genitive case. All of this is of course meant to be an example. FCG allows the grammar designer to use any kind of feature deemed necessary. Templates can also incorporate more information or less, depending on preferred implementation style. For example the syntactic and semantic categorizations could also be put in a single lexical template together with the meaning and string.

Here is another more elaborate example to illustrate the use of templates for building constructions (discussed at length in [40]). It defines a possessive phrasal construction, underlying a phrase such as “this house of mine". The possessive phrasal construction involves two constituents: a nominal phrase (“this house") and a possessive pronominal “mine". The construction itself is called `postposed-genitive-cxn`. It starts with the creation of a shell that makes this construction a member of the set of phrasal constructions:

```
(def-phrasal-cxn postposed-genitive-cxn)
```

Next, the `def-phrasal-skeleton` template is used to introduce units both for the phrase as a whole, with a slot called `:phrase`, and for the different constituents, with a slot called `:constituents`. The constituents are defined in terms of their semantic functions, syntactic functions, lexical categories, phrase types, or syntactic and semantic categorizations. Information is provided on the phrase-type of the parent and its possible syntactic and semantic functions:

```
(def-phrasal-skeleton postposed-genitive-cxn
  :phrase
  (?possessive-nominal-phrase
    :sem-function referring
    :phrase-type nominal-phrase)
  :constituents
  ((?nominal-unit
     :sem-function referring
     :phrase-type nominal-phrase)
   (?pronominal-unit
     :sem-function possessive
     :lex-cat pronoun
     :syn-cat (==1 (case genitive)))))
```

The variables that are used for the constituents (i.e. `?nominal-unit`, `?pronominal-unit`) and for the parent phrase (i.e. `?possessive-nominal-phrase`) can be used by other templates to address these units and add more information. Other variables used in feature values can also be used across templates.

Other templates implement other design patterns. For example, the postposed genitive construction requires that the number of the nominal unit percolates to the possessive nominal phrase as a whole. This is specified with a template called `def-phrasal-agreement`. For all constituents that share features and for the parent phrase in which features percolate up from the constituents, the def-phrasal-agreement lists the following:

```
(def-phrasal-agreement postposed-genitive-cxn
  (?possessive-nominal-phrase
    :syn-cat (==1 (number ?number)
                  (is-definite ?definiteness)))
  (?nominal-unit
    :syn-cat  (==1 (is-definite ?definiteness)
                   (number ?number))))
```

The variables `?possessive-nominal-phrase` and `?nominal-unit` are used to retrieve which units are involved, and the slot-values specify which syntactic and/or semantic categories have to agree. Use of the same variable name indicates that this indicates that an agreement relation is established. For example, `?number` of the possessive nominal phrase is shared with `?number` of the `?nominal-unit`. Thanks to the unification operation, bindings can flow in both directions: It is not only possible that the number value propagates up from the nominal-unit to the phrase but also that it propagates down from the nominal phrase to the nominal unit.

Phrasal constructions may add some meaning of their own, and they may add form constraints over and above the form constraints supplied by individual constituents, which is usually specified with another template called `def-phrasal-require`. It has a slot for constructional meaning, called `:cxn-meaning`, and a slot for constructional form, called `:cxn-form`. The `postposed-genitive-cxn` illustrates how the construction adds a possessive relation as part of the meaning, a grammatical function word, namely “of", and ordering relations between these components:

```
(def-phrasal-require postposed-genitive-cxn
  (?possessive-nominal-phrase
    :cxn-meaning (== (possessive ?referent-nominal
                                 ?referent-pronominal))
    :cxn-form (== (meets ?nominal-unit ?word-of)
                  (string ?word-of "of")
                  (meets ?word-of ?pronominal-unit))))
```

As a final example, we use a template called `def-phrasal-linking`, to establish the linking between the external arguments of the constituents and the parent phrase [36]. The template simply lists for each constituent which external arguments are involved and if the same variable-name is used they are assumed to be linked. The unification operation takes care of the binding of the relevant variables.

```
(def-phrasal-linking postposed-genitive-cxn
  (?possessive-nominal-phrase
    :args (?referent-nominal))
  (?nominal-unit
    :args (?referent-nominal))
  (?pronominal-unit
    :args (?referent-pronominal)))
```

4 The Operational Level

This paper advocates that the design of a lexicon and grammar for a particular language fragment should proceed in a top-down manner, starting from an analysis at the *linguistic level*, identifying the semantic domains and functions, and the representations and communications functions that need to be expressed in the language and how they are expressed. It then moves to the *design level* with the identification of design patterns and the templates that actualize them. For example, to implement agreement requires templates that add syntactic and semantic categorizations to units and templates that specify what agreement and percolation relations need to be established for a particular phrase.

We now arrive at the *operational level*, which is the level at which constructions with all their details have to be defined. Although constructions can be defined by hand, it is much easier to do so using templates. Nevertheless it is import to understand the operational level also, partly to be able to follow in detail what changes a construction has made and why it does (or does not) trigger, and partly to be able to extend or adapt the set of available templates. The remainder of this section provides a brief introduction to the main structures and operations at the operational level. The reader is referred to a later chapter [10] for more details, and to the examples further discussed in this book or in other publications [41].

4.1 Representing Transient Structures

FCG uses feature structures for representing the information that is built up during parsing and production, thus following generally accepted practices in contemporary computational linguistics, The so called *transient structure* starts in parsing with all the information that can be extracted from the utterance (strings, ordering, possibly phonetic information) and progressively reconstructs semantic and syntactic structures and meanings. In production, the transient structure contains initially only the meaning that needs to be expressed. Different constructions cover parts of this meaning, progressively building up phrasal structures and constraining the form of the sentence until a concrete sentence can be derived.

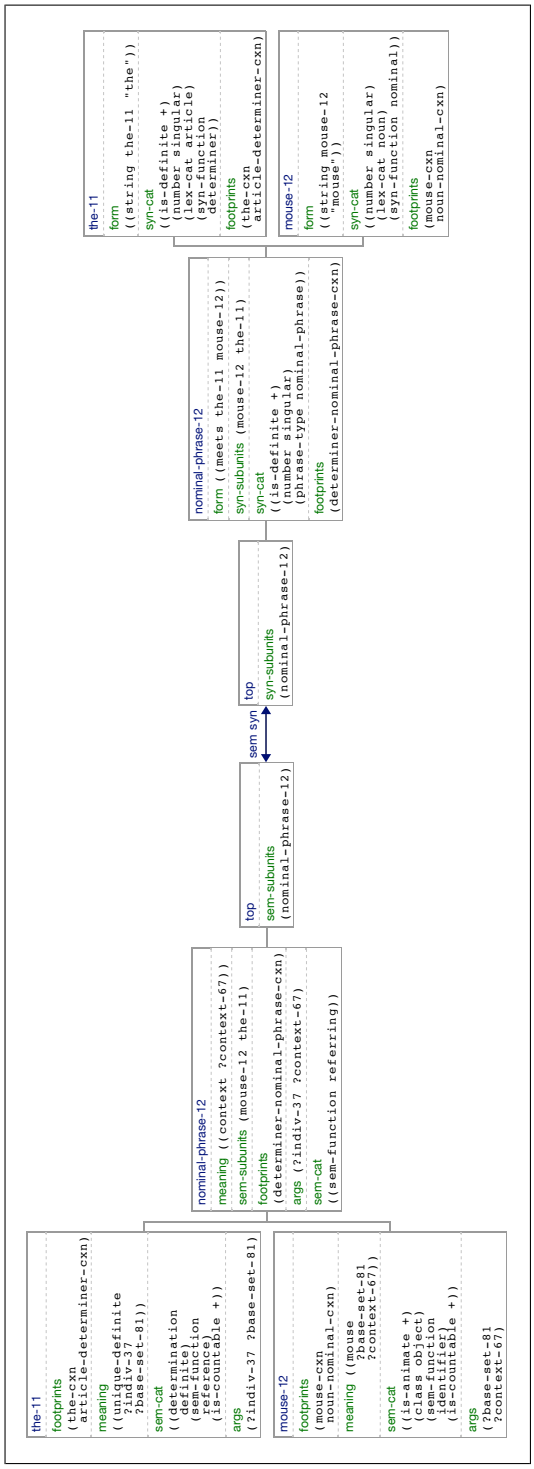

Fig. 9. Graphical display of a transient structure when parsing or producing "the mouse". Each box represents a unit with its name and feature values. All features of the semantic poles are displayed on the left side and all features of the syntactic poles on the right side. Both poles are shown in more detail in Figure 10.

The feature structures used in FCG compose the linguistic structure in terms of units with features and values. Units have names, and these names can be bound to variables for reference inside constructions. Consequently, hierarchical structure is represented explicitly by a feature called *subunits* filled by names of all subunits.

The transient structure is decomposed into a semantic pole and a syntactic pole to improve readability and efficiency. The graphical representation in Figure 9 provides an example of a simple determiner-nominal phrase in (taken from [42], which explains this example in detail). There is a list notation which reflects the internal LISP-based implementation of feature structures. Graphical representations are constructed automatically by the FCG-system and there is a browser for interactive and selective display (see [23]).

The same feature structure is shown in list notation in Figure 9. The unit names are in bold and the unit features in italics. In the semantic pole, there is a unit for `top`, which has one semantic subunit called `nominal-phrase-12`. `nominal-phrase-12` has two semantic subunits: `mouse-12` and `the-11`. The same unit-names are found on the syntactic pole with pending syntactic features. Indices like 12 or 11 are there to distinguish between instances of a symbol but do not carry meaning.

```
( (top
   (sem-subunits (nominal-phrase-12)))
  (nominal-phrase-12
   (sem-subunits (mouse-12 the-11))
   (meaning ((context ?context-67)))
   (args (?indiv-37 ?context-67))
   (sem-cat ((sem-function referring)))
   (footprints (determiner-nominal-phrase-cxn)))
  (the-11
   (meaning ((unique-definite ?indiv-37 ?base-set-81)))
   (args (?indiv-37 ?base-set-81))
   (sem-cat
    ((determination definite)
     (sem-function reference) (is-countable +)))
   (footprints (the-cxn article-determiner-cxn)))
  (mouse-12
   (meaning ((mouse ?base-set-81 ?context-67)))
   (args (?base-set-81 ?context-67))
   (footprints (mouse-cxn noun-nominal-cxn))
   (sem-cat
    ((is-animate +) (class object)
     (sem-function identifier) (is-countable +)))))
<-->
( (top
   (syn-subunits (nominal-phrase-12)))
  (nominal-phrase-12
```

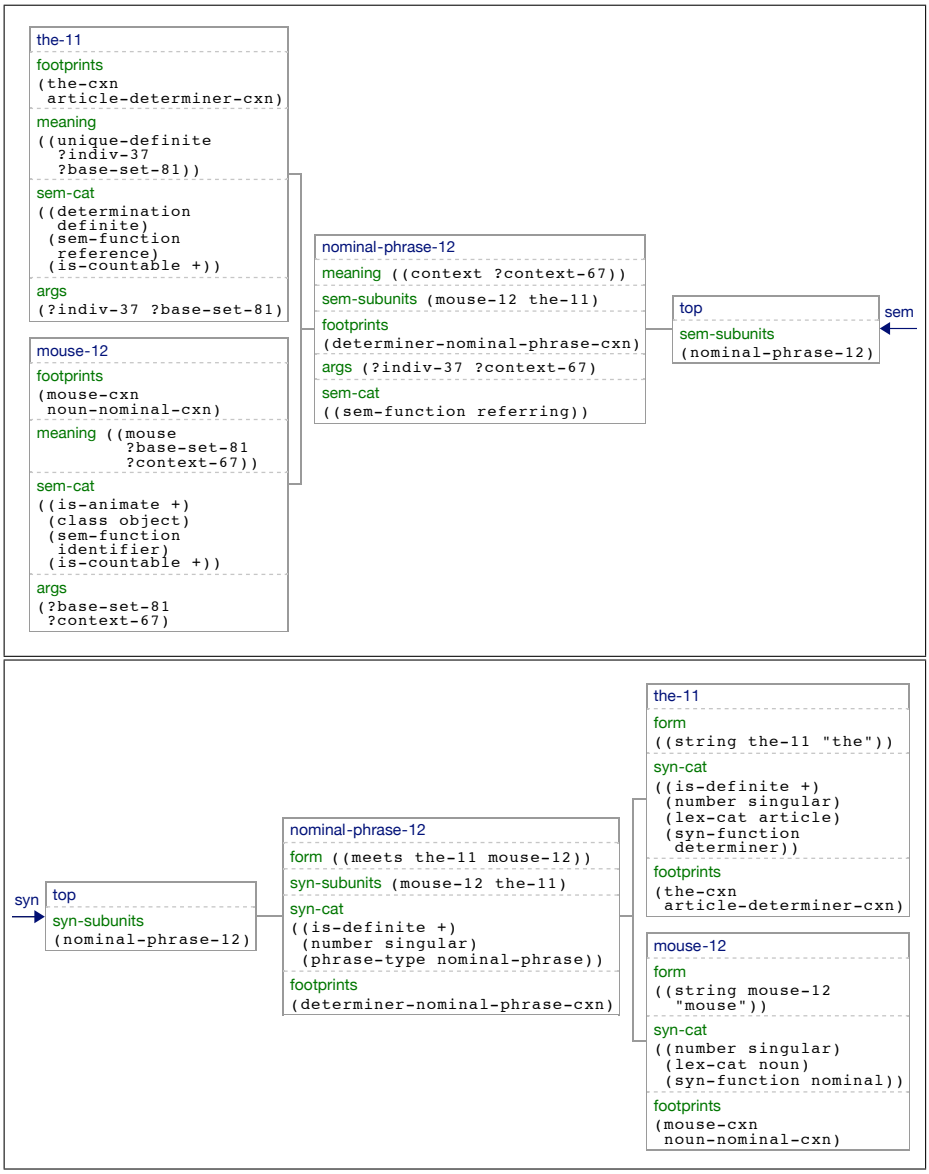

Fig. 10. Zooming in on the semantic (top) and syntactic (bottom) poles of the transient structure shown in Figure 9.

```
 (syn-subunits (mouse-12 the-11))
 (form ((meets the-11 mouse-12)))
 (syn-cat
  ((is-definite +) (number singular)
   (phrase-type nominal-phrase)))
 (footprints (determiner-nominal-phrase-cxn)))
(the-11
 (form ((string the-11 "the")))
 (syn-cat
  ((is-definite +) (number singular)
   (lex-cat article) (syn-function determiner)))
 (footprints (the-cxn article-determiner-cxn)))
(mouse-12
 (form ((string mouse-12 "mouse")))
 (syn-cat
  ((number singular) (lex-cat noun)
   (syn-function nominal)))
 (footprints (mouse-cxn noun-nominal-cxn))))
```

Feature structures in FCG do not fundamentally differ from those used in other feature-structure based formalisms. For example, part of the syntactic pole of the transient structure in Figure 10 would be represented in many other unification-based formalisms as follows:

$$\left[\begin{array}{ll} \textit{syn-cat} & \left[\begin{array}{ll} \text{phrase-type} & \text{nominal-phrase} \\ \text{is-definite} & + \\ \text{number} & \text{singular} \end{array}\right] \\ \textit{syn-subunits} & \left[\begin{array}{l} \left[\begin{array}{ll} \textit{form} & \left[\text{string} \quad \text{"the"}\right] \\ \textit{syn-cat} & \left[\begin{array}{ll} \text{is-definite} & + \\ \text{number} & \text{singular} \\ \text{lex-cat} & \text{article} \\ \text{syn-function} & \text{determiner} \end{array}\right] \end{array}\right] \\ \left[\begin{array}{ll} \textit{form} & \left[\text{string} \quad \text{"mouse"}\right] \\ \textit{syn-cat} & \left[\begin{array}{ll} \text{number} & \text{singular} \\ \text{lex-cat} & \text{noun} \\ \text{syn-function} & \text{nominal} \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

The main difference concerns the use of names for units and the use of logic variables for representing values of features that are unknown.

4.2 Representing Constructions

Constructions use the same representations as transient structures: They consist of units with features and values, which are matched against transient structures

and then merged so that information present in the construction, but not yet in the transient structure, gets added. Constructions are more abstract than transient structures. They leave out information so that the construction matches with a wide range of transient structures. They contain variables that can be bound to specific values contained in a transient structure. And they may specify partial values using a set of special operators, such as an includes operator (if only some of the elements have to be present), a uniquely includes operator (if a particular element can appear only once), an excludes operator (if an element should not occur), and so on.

Different templates build up different elements of a construction. For example, the lexical construction for “mine" was defined earlier using the following templates:

```
(def-lex-cxn mine-cxn
  (def-lex-skeleton mine-cxn
    :meaning (== (context ?context)
                 (dialog-participant ?indiv speaker ?context))
    :args (?indiv)
    :string "mine")
  (def-lex-cat mine-cxn
    :sem-cat (==1 (sem-function possessive))
    :syn-cat (==1 (lex-cat pronoun)
                  (person 1st)
                  (number singular)
                  (case genitive))))
```

The operational construction based on these templates looks as follows, with the semantic and syntactic pole separated by a double arrow <->:

```
(def-cxn mine-cxn
  ((?top-unit
    (tag ?meaning
         (meaning
            (== (context ?context)
                (dialog-participant ?indiv speaker ?context))))
    (footprints (==0 mine-cxn lex)))
   ((J ?word-mine ?top-unit)
    ?meaning
    (args (?indiv))
    (footprints (==1 mine-cxn lex))
    (sem-cat (==1 (sem-function possessive)))))
  <-->
  ((?top-unit
    (footprints (==0 mine-cxn lex))
    (tag ?form
         (form (== (string ?word-mine "mine")))))
   ((J ?word-mine ?top-unit)
```

```
?form
(footprints (==1 mine-cxn lex))
(syn-cat
  (==1 (lex-cat pronoun)
       (person 1st)
       (number singular)
       (case genitive))))))
```

In production, the construction is applied from the semantic pole to the syntactic pole. It looks out whether a particular meaning is present in the `?top-unit` (which is the initial top unit of a transient structure). If that meaning is found, the construction creates a new sub-unit (bound to the variable `?word-mine`) and hangs it from the top-unit on the semantic side. It also adds information about the external arguments of the word and its semantic categorizations. On the syntactic side, the construction creates a syntactic subunit and adds information about the word form (the string "mine") as well as syntactic categorizations concerning lexical-class, person, number and case.

In parsing, the construction is applied from the syntactic pole to the semantic pole. It looks out for the presence of a particular string (namely "mine") in the top-unit. If that is the case, the construction builds a new unit bound to `?word-mine` and hangs it from `?top-unit`. The string is moved from the form feature of the top-unit to the form feature of the new unit, and syntactic categorizations are added. On the semantic side, the construction creates a new semantic subunit and adds information about its meaning and its semantic categorization.

Parts of this operational construction are clearly based on the elements supplied by the templates: the `:meaning`, `:string` and `:args` come from the `def-lex-skeleton` template, and the `:syn-cat` and `:sem-cat` come from the `def-lex-cat` template. However, more is needed to make a construction fully operational. Choices have to be made as to whether information is put into the semantic pole or the syntactic pole, and if triggering the construction or additive in merging should be conditional. Other issues concern the question of how new units are built and how information is moved to them, and how the recursive application of constructions is regulated. This section briefly discusses some procedural annotations in operational constructions that have been designed for these purposes.

4.3 Procedural Annotations

Procedural annotations consist of extra information supplied with a construction to carry out structure building operations or to avoid that constructions keep applying indefinitely. Structure building requires two operations: a way to create new units and hang them somewhere from an existing unit in the hierarchy, and a way to associate information with the new unit, possibly by moving features or values that were located elsewhere.

The J-operator The J-operator is the main FCG primitive for building hierarchical structure [11]. It has three arguments: a *daughter-unit*, a *parent-unit*, and possibly a set of *pending-subunits*. These are either specified with concrete names or with variables that have been bound elsewhere in the matching or merging process. When the daughter-unit is an unbound variable at the time of merging, a new unit will be created for it. For example, in the `mine-cxn` above, the following expression evokes the J-operator.

```
(J ?word-mine ?top-unit)
```

It introduces a new daughter-unit bound to `?word-mine` and hangs it as a subunit from a parent-unit bound to `?top-unit`. There are no further pending units, otherwise they would be made subunits of the daughter-unit. The J-operator can associate additional information with the daughter-unit. In the example of the `mine-cxn` construction, the J-operator adds information about the lexical category, person, number, and case.

The TAG-operator The J-operator is made more versatile by introducing a way to *tag* parts of a feature structure so that they can be moved elsewhere in the transient structure. The tag-operator has two arguments: a variable, known as the *tag-variable*, and a set of features and values that are bound to the tag-variable. The normal matching process is still used to check whether the features and values match. If a tag-variable re-occurs inside a unit governed by a J-operator, then the structure is *moved* from its old position to its new position.

Here is an example. In production, the top-unit initially contains all the meanings that need to be covered , and lexical constructions take those parts that they can cover and encapsulate them in a new unit. This is done with the tag operator, which binds the meaning of the word “mine" and then moves into the `?word-mine` unit created by the J-operator, as illustrated in the semantic pole of the `mine-cxn` construction.

```
(tag ?meaning
     (meaning
       (== (context ?context)
           (dialog-participant ?indiv speaker ?context))))
```

The meaning covered by the word is tagged and then moved from the top-unit to the newly created unit that covers this meaning in production. The form introduced by the word is also tagged and then moved from the top-unit to a newly created unit in parsing.

Footprints One of the biggest issues in language processing is the management of the search space. This arises unavoidably in parsing because most word forms or syntactic constraints have multiple meanings and functions, and it is often not possible to make a definite choice until more of the sentence has been processed. It also arises in production because there is usually more than one way to express

a particular meaning, and it is not always possible to decide fully which choice is the most appropriate until other aspects of the sentence are worked out. Many techniques help to avoid search whenever possible, for example, choices can be left open as variables until enough information is available to choose their bindings, or the value of a particular syntactic feature (such as the lexical-category) can be a list of potential values from which one is then actually chosen to be the actual value.

Adding `footprints` to a transient structure is another technique for avoiding search and particularly the harmful recursive application of constructions. Footprints are represented as one of the features of a unit. They are left behind by constructions, so that other constructions (or the same construction) can see that this construction was involved in building a particular piece of structure and hence can refrain from application. By convention, the name of the footprint left behind by a construction is the name of the construction itself. A construction may also leave behind other footprints. For example, if the construction is a member of a family of constructions, each construction leaves behind a family footprint so that more general constructions of the same family will no longer trigger. Another example concerns the handling of defaults. Constructions that deal with overt cases leave behind footprints so that default construction dealing with an unmarked case does not need to trigger anymore (see examples in [3]).

In the example given earlier, the `mine-cxn` construction first checks whether it has not yet already applied on the `?top-unit`, so that recursive application is avoided. Once it has applied, it leaves behind a footprint that it was involved in building the new unit bound to `?word-mine` so that there can be no recursive application where the unit `?word-mine` becomes bound to `?top-unit`. Footprints are not only useful for controlling the application of constructions. They are also useful for a grammar designer who is inspecting transient structures in order to figure out which construction did what.

5 Conclusions

This paper introduced some of the design principles that are currently used in Fluid Construction Grammar. We distinguished three levels of analysis: a linguistic level, a design level and an operational level. The linguistic level starts from an analysis of which semantic domains and communicative functions are relevant for the language fragment being studied. It then investigates first the functional structure underlying sentences: which semantic functions are involved, how do they map to syntactic functions, and how are syntactic functions expressed in the language. Next it investigates the expression of grammatical meanings, such as tense and aspect, using the grammar square as guidance.

The design level starts from an analysis of the major design patterns that are used in the language fragment and then seeks to find out which templates could be used to implement them. A template emphasises linguistic content, hiding computational details as much as possible. Templates are translated automatically to the operational level by a compilation process so that we obtain the

'real' constructions that drive parsing and production processes. Constructions can be written by hand, but it is much more efficient to do so with templates, as later chapters with case studies show.

FCG is an attempt to capture many ideas that have been floating around in the construction grammar literature. But there are certainly still many ideas which are not yet incorporated. For example, inheritance plays an important role in many construction grammars but it is not a core component of FCG, and the same phenomena are captured in other ways. FCG uses many of the same techniques found in other unification-based grammars, but there are also profound differences. For example, FCG constructions are split into two poles which are used differently in parsing and production, whereas HPSG would put everything together in a single structure. Deeper comparisons with other attempts for formalizing construction grammar and the relation to other unification-based grammars are discussed in later chapters of this book (see particularly [6]). FCG is still a very new formalism and many issues remain to be explored. In some cases, solutions developed in other unification-based grammars can be nicely translated into FCG. In other cases, FCG suggests new venues that might be translatable to other formalisms, but this needs to be examined.

Acknowledgements

FCG has been under development for a decade with teams at the University of Brussels (VUB AI Lab) and the Sony Computer Science Laboratory in Paris. The primary source of funding has come from the Sony Computer Science Laboratory with additional funding provided by the EU-FP6 ECagents project and the EU-FP7 ALEAR project.

Bibliography

[1] Alexander, C.: The Timeless Way of Building. Oxford University Press, Oxford (1979)
[2] Bergen, B.K., Chang, N.: Embodied Construction Grammar. In: Östman, J.O., Fried, M. (eds.) Construction Grammars: Cognitive Grounding and Theoretical Extensions. John Benjamins, Amsterdam (2005)
[3] Beuls, K.: Construction sets and unmarked forms: A case study for Hungarian verbal agreement. In: Steels, L. (ed.) Design Patterns in Fluid Construction Grammar. John Benjamins, Amsterdam (2011)
[4] Carpenter, B.: The Logic of Typed Feature Structures. Cambridge University Press, Cambridge UK (1992)
[5] Carroll, S., Grenier, J., Weatherbee, S.: From DNA to Diversity. Molecular Genetics and the Evolution of Animal Design. Blackwell Science, Oxford (2001)
[6] Ciortuz, L., Saveluc, V.: Fluid Construction Grammar and feature constraints logics. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)

[7] Copestake, A.: Implementing Typed Feature Structure Grammars. CSLI Publications, Stanford (2002)
[8] Croft, W.: Radical Construction Grammar. Oxford University Press, Oxford (2001)
[9] Dalrymple, M., Kaplan, R., Maxwell, J., Zaenen, A.: Formal issues in Lexical-Functional Grammar. CSLI Publications, Stanford (1995)
[10] De Beule, J.: A formal deconstruction of Fluid Construction Grammar. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[11] De Beule, J., Steels, L.: Hierarchy in fluid construction grammar. In: Furbach, U. (ed.) Proceedings of KI-2005. Lecture Notes in AI 3698. pp. 1–15. Berlin (2005)
[12] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Pub. Co., Reading Ma (1995)
[13] Gerasymova, K.: Expressing grammatical meaning with morphology: A case study for Russian aspect. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[14] Goldberg, A.E.: A Construction Grammar Approach to Argument Structure. Chicago UP, Chicago (1995)
[15] Goldberg, A.E.: Constructions: a new theoretical approach to language. Trends in Cognitive Sciences 7(5), 219–224 (2003)
[16] Höfer, S.: Complex declension systems and morphology in Fluid Construction Grammar: A case study of Polish. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[17] Kay, M.: Parsing in functional unification grammar. In: Grosz, B., Spark-Jones, K., Webber, B. (eds.) Readings in Natural Language Processing. Morgan Kaufmann, San Francisco (1986)
[18] Kay, P., Fillmore, C.: Grammatical constructions and linguistic generalizations: the what's x doing y? Language 72, 1–33 (1996)
[19] Laenzlinger, C.: French adjective ordering: Perspectives on dp-internal movement types. Lingua 115/5, 645–689 (2000)
[20] Langacker, R.W.: Foundations of Cognitive Grammar. Volume 1. Stanford University Press, Stanford (1987)
[21] Langacker, R.W.: A dynamic usage-based model. In: Barlow, M., Kemmer, S. (eds.) Usage-Based Models of Language, pp. 1–63. Chicago University Press, Chicago (2002)
[22] Levinson, S.C.: Space in Language and Cognition. Language, Culture and Cognition 5, Cambridge University Press, Cambridge (2003)
[23] Loetzsch, M.: Tools for grammar engineering. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[24] Micelli, V.: Field topology and information structure: A case study for German constituent order. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[25] Michaelis, L.: Sign-based construction grammar. In: Heine, B., Narrog, H. (eds.) The Oxford Handbook of Linguistic Analysis. Oxford University Press, Oxford (2009)

[26] Michaelis, L., Lambrecht, K.: Toward a construction-based theory of language function: The case of nominal extraposition. Language 72, 215–247 (1996)
[27] Newmeyer, F. (ed.): Language Form and Language Function. MIT Press, Cambridge Ma (1998)
[28] Pollard, C., Sag, I.A.: Head-Driven Phrase Structure Grammar. Chicago University Press, Chicago (1994)
[29] Sag, I., Wasow, T., Bender, E.: Syntactic Theory. A Formal Introduction. CSLI Publications, Stanford (2003)
[30] Spranger, M., Pauw, S., Loetzsch, M., Steels, L.: Open-ended Procedural Semantics. In: Steels, L., Hild, M. (eds.) Language Grounding in Robots. Springer, New York (2012)
[31] Spranger, M., Loetzsch, M.: Syntactic indeterminacy and semantic ambiguity: A case study for German spatial phrases. In: Steels, L. (ed.) Design Patterns in Fluid Construction Grammar. John Benjamins, Amsterdam (2011)
[32] Stadler, K.: Chunking constructions. In: Steels, L. (ed.) Computational Issues in Fluid Construction Grammar. Springer Verlag, Berlin (2012)
[33] Steels, L.: Language as a complex adaptive system. In: Schoenauer, M. (ed.) Proceedings of PPSN VI. pp. 17–26. Lecture Notes in Computer Science, Springer-Verlag, Berlin (2000)
[34] Steels, L.: Constructivist development of grounded construction grammars. In: Scott, D., Daelemans, W., Walker, M. (eds.) Proceedings of ACL. pp. 9–16. ACL, Barcelona (2004)
[35] Steels, L.: Grounding Language through Evolutionary Language Games. In: Steels, L., Hild, M. (eds.) Language Grounding in Robots. Springer, New York (2012)
[36] Steels, L., De Beule, J., Neubauer, N.: Linking in Fluid Construction Grammars. In: Proceedings of BNAIC. pp. 11–18. Transactions of the Belgian Royal Society of Arts and Sciences, Brussels (2005)
[37] Steels, L., De Beule, J., Neubauer, N.: Bnaic. In: Transactions of the Belgian Royal Society for Science and Arts. p. October. Brussels (2005)
[38] Steels, L., Kaplan, F.: Spontaneous lexicon change. In: Proceedings of COLING-ACL 1998. pp. 1243–1250. Morgan Kaufmann, San Francisco, CA (August 1998)
[39] Steels, L.: The emergence of grammar in communicating autonomous robotic agents. In: Horn, W. (ed.) ECAI 2000: Proceedings of the 14th European Conference on Artificial Life. pp. 764–769. IOS Publishing, Amsterdam (August 2000)
[40] Steels, L.: A design pattern for phrasal constructions. In: Steels, L. (ed.) Design Patterns in Fluid Construction Grammar. John Benjamins, Amsterdam (2011)
[41] Steels, L. (ed.): Design Patterns in Fluid Construction Grammar. John Benjamins, Amsterdam (2011)
[42] Steels, L.: A first encounter with Fluid Construction Grammar. In: Steels, L. (ed.) Design Patterns in Fluid Construction Grammar. John Benjamins, Amsterdam (2011)

[43] Steels, L. (ed.): Experiments in Cultural Language Evolution. John Benjamins, Amsterdam (2012)
[44] Steels, L., van Trijp, R.: How to make construction grammars fluid and robust. In: Steels, L. (ed.) Design Patterns in Fluid Construction Grammar. John Benjamins, Amsterdam (2011)
[45] Talmy, L.: Toward a Cognitive Semantics, Typology and Process in Concept Structuring, vol. 2. MIT Press, Cambridge, Mass (2000)
[46] Tomasello, M.: Constructing a Language. A Usage Based Theory of Language Acquisition. Harvard University Press (2003)
[47] van Trijp, R.: A design pattern for argument structure constructions. In: Steels, L. (ed.) Design Patterns in Fluid Construction Grammar. John Benjamins, Amsterdam (2011)
[48] van Trijp, R.: Feature matrices and agreement: A case study for German case. In: Steels, L. (ed.) Design Patterns in Fluid Construction Grammar. John Benjamins, Amsterdam (2011)
[49] Wellens, P.: Organizing constructions in networks. In: Steels, L. (ed.) Design Patterns in Fluid Construction Grammar. John Benjamins, Amsterdam (2011)
[50] Winograd, T.: A procedural model of language understanding. In: Computation & intelligence, pp. 203–234. American Association for Artificial Intelligence, Menlo Park, CA, USA (1995), http://dl.acm.org/citation.cfm?id=216000.216012
[51] Woods, W.: Problems in procedural semantics. In: Pylyshyn, Z., Demopolous, W. (eds.) Meaning And Cognitive Structure. Ablex Publishing, New York (1986)