A grammar-based approach for populating HCI design spaces

Margherita Antona, Demosthenes Akoumianakis and Constantine Stephanidis
Institute of Computer Science, Foundation for Research and Technology-Hellas, Science and Technology Park of Crete, GR-71110, Heraklion, Crete, Greece
e-mail: cs@ics.forth.gr

1 Introduction

Human-Computer Interaction (HCI) design is anticipated to become in the future a much more complex activity than today, and to require appropriate supporting tools (Stephanidis and Akoumianakis 2000), capable, amongst other things, to facilitate designers in the population and exploration of the design space, i.e., the (sets of) design alternatives, or combinations of interface artefacts, possibly integrating elements from different toolkits. Furthermore, design tools will be required to facilitate designers in shaping such design spaces in a principled and coherent way on the basis of the user’s characteristics (i.e., abilities, preferences, skills, requirements), and the context of use.

This paper proposes a grammar-based approach to the population of HCI design spaces (i.e., the generation of alternative design specifications) on the basis of task models. Such an approach builds, on the one hand, on the frequently occurring parallelism between user interfaces and sign systems (Institute for Personalized Information Environment 1995), and on a view of user interface design as a process of mapping concepts in a function domain to symbols in a presentation domain, and vice versa (Stephanidis and Akoumianakis 1999). On the other hand, it builds on concepts and techniques recently elaborated in the field of Natural Language Processing (NLP).
Head-Driven Phrase Structure Grammar (HPSG, Pollard and Sag 1987, 1994), is a recent language theory that has emerged as one of the best suited for NLP purposes. HPSG views linguistic expressions (such as words, phrases, and sentences) as signs in the structuralist sense, relating a (phonological) form to a meaning in an utterance context (Pollard and Sag 1994).

Linguistic information in HPSG is modeled through typed feature structures (TFS, e.g., Carpenter 1992, Carpenter and Penn 1998). In TFS, elements in the modeled domain are declared as types organized in an inheritance hierarchy. Types may have subtypes (more specific instances of types), which inherit and further specify their properties. Each TFS is composed of a type and a collection of (possibly empty) attribute-value pairs. Appropriateness of attributes for a type must be declared. Values of attributes are also typed and they may be atomic or feature structures. Feature structures may share values, i.e., the values of two attributes in a TFS may be declared as identical. TFS are inherently partial with respect to the information they provide, and may be ordered according to how specific is the encoded information. Such an ordering is called subsumption. The most important operation on TFS is unification, which is defined in terms of subsumption: the result of unification amounts to the most general feature structure subsumed by the input TFS.

HPSG is a lexicalist theory, since it encodes the largest part of linguistic information in the lexicon. Inheritance hierarchies allow the cross-classification of words into word-classes according to their properties, and each word inherits the characteristics of all classes to which it belongs. Some words, the so-called heads, are considered as determining the syntactic structures of the phrases in which they occur, since they impose constraints on other phrase components. These constraints are defined as the valence of a sign, i.e., a list of specifications of possible complements the head can take. Phrases (i.e., phrasal signs) are seen in HPSG as projections of their heads, constructed on the basis of the valence information encoded in head lexical entries. Phrasal signs are represented as mother feature structures that include the specification of their daughters, i.e., of the signs composing the phrase. Phrase structure rules, i.e., rules that recognize, or generate, phrasal signs, are called schemata, since they do not depend on the specification of syntactic categories. A phrasal sign is saturated when all complements required by its head are included in the feature representing the sign daughters. Schemata are constrained through general principles that apply to the information contained in the mother and in the complement daughters. Two particularly important principles are: (i) the Head Feature Principle, which states that a mother and its head daughter always share the same syntactic category (the phrasal projection of a verb is of category verb, etc.); (ii) the Subcategorisation Principle, which states that the valence of the head daughter...
of a phrase is equivalent to the valence of the mother with the addition of the identified complements (daughters fill the slots foreseen in the valence of the head). Other, less general, principles may concern: (i) construct specific phenomena, e.g., specific types of phrases such as coordinate phrases, etc; (ii) language specific phenomena. HPSG grammars are completely declarative, and reversible, i.e. they can be used for both parsing and generation purposes (Neumann and van Noord 1993).

3 A design grammar for interface signs

The HPSG framework can be adopted for defining a design grammar which describes interface signs as the combination of a form (i.e., an interface object) with a content (i.e., a specification of an application function, or user tasks), and captures the principles which constrain the combination of such signs into phrasal signs (i.e., composite interface elements such as dialogues, or complete interfaces).

The grammar-based approach relies on the representation of interface objects, their syntactic properties and their semantics into a TFS inheritance hierarchy. Lexical interface signs, i.e., representation of toolkit objects, as the lexicon of natural languages in HPSG, can be cross-classified on the basis of their ability to represent different tasks, and to combine with each other, independently from the specific metaphor in which they are embedded. The semantics of interface signs can be modeled in the grammar as a hierarchy of user and system tasks, based on the notion of abstract task context (Akoumianakis and Stephanidis, 1997) which entails that the same task may be performed differently by different users, or the interface may exhibit differentiated behavior in the same task context depending on the current user. In the grammar, tasks can have subtasks, which are either tasks (i.e., they can have their own subtasks), or lists of tasks of the same type. Additionally, the proposed approach assumes that interface objects combine in head-driven fashion, i.e., some of them behave like heads which specify a valence partially determining the category and semantics of the other interface objects they can combine with. Abstractions over properties of interface objects are adopted for modeling interface sign syntax. A typical example of such an abstraction is the concept of container (Savidis et al., 1997), which captures the characteristics of objects like windows, books, html pages, etc. In the grammar, containers play the role of heads, i.e., they are assumed to have a (partially specified) valence, which becomes further instantiated according to the task assigned to a container. Categories such as container, menu, toolbar, etc., are used in the grammar in a fashion similar to that of syntactic categories in natural language grammars. The inheritance hierarchy also includes interface objects, classified according to their physical properties. Furthermore, the hierarchy subdivides interface signs into lexical and phrasal
signs, and assigns feature description to signs. In the lexicon, semantic, and, to a lesser degree, syntactic properties of signs are underspecified. Lexical signs are assigned a general task type, so that they are compatible with all the specific tasks subsumed by such a type.

A very small number of general phrase structure rules is sufficient for generating design alternatives. Schemata combining heads with daughters are subject to the Head Feature Principle and the Subcategorisation Principle. Additional principles could apply to specific types of constructs, e.g. enforcing metaphor coherence or object identity in some of their component parts. Another category of principles could relate to the context of use of an interface, such as user abilities, requirements and preferences, and factors related to the type of usage and access device. Context-related principles would be applied along with grammar internal principles in order to ‘filter out’ incoherent or contextually inappropriate design alternatives.

When given as input a structured user task, the grammar produces all (multiple-metaphor) interface signs that can convey such a task. The semantic underspecification of lexical entries allows the direct unification of the input with all interface signs whose semantics unify with the task at hand. A grammar of this type may also handle redundancy in user interface signs, i.e., the degree to which alternative objects for the same subtask(s) are included in an interface sign. In summary, generated design alternatives may vary with respect to: (i) semantically equivalent objects from the same toolkit; (ii) semantically equivalent objects from different toolkits; (iii) the degree of redundancy.

4 Conclusions

The outlined grammar-based approach to HCI design is characterized by: (i) a sign-based account of interface objects and their semantics; (ii) the organization of knowledge concerning tasks, interface object properties and combinatorial behavior into an inheritance hierarchy; (iii) the projection of complex interface constructs via general principles from rich lexical information; and (iv) the organization of such lexical information via a system of lexical types. The design grammar proposed in this paper is not meant to produce interface implementations, but, rather, interface specifications, and to be applied in design tools integrated into specification-based development environments (Stephanidis et al. 1999). In such environments, a design tool based on the proposed grammar would act as a source of design recommendations, in the form of specifications, which, once compiled, can be interpreted and applied by the run-time libraries of the user interface development system, thus allowing the instantiation of alternative interactive behaviors. The proposed design grammar is claimed to introduce a new approach to the population of design
spaces in multiple-metaphor environments, capable of supporting the designer in a task up to now largely carried out without system support, or at best, with artefact-oriented rather than user- and context-oriented support.

5 References


