The Role of Conceptual Modelling Abstractions in Compiler Development

CRISTINA SERNADAS*† AND R. CARAPUÇA‡
* *FCUL, Rua da Escola Politécnica 58, 1294 Lisboa, Portugal*
‡ *INESC, Rua Alves Redol 9, 1000 Lisboa, Portugal*

The role of conceptual modelling abstractions for defining attribute grammars in a structured way is discussed. The information structure of the conceptual modelling approach provides the guidelines for obtaining the context-free grammar and the attributes. The alteration structure of the conceptual modelling approach leads to the definition of the semantic rules and semantic conditions. A methodology is proposed and illustrated by an attribute grammar for a conceptual schema language.

Received October 1985, revised March 1986

1. INTRODUCTION

One of the main steps in compiler development is the definition of the static semantics, that is to say the semantics of context dependencies. The static semantics must be defined independently of the adopted implementation environment, so that the same language can be implemented in different environments.

The most frequent way of defining the static semantics[1] is either through attribute grammars[2] or through two-level grammars.[3] Attribute grammars have been used extensively in the literature to define the static semantics of programming languages (for an example of an attribute grammar for the ADA programming language see Ref. 4).

In Ref. 5 an attribute grammar was developed for a conceptual schema language which was implemented with the YACC/LEX/C tools available in the UNIX system.[6] As far as the authors know this was the first attempt to bring the attribute grammar techniques to the environment of conceptual schema languages. The experience revealed the necessity of introducing a methodology for defining attribute grammars for conceptual schema languages in a structured way. Perhaps the problem is not so relevant concerning attribute grammars for programming languages since the respective context dependencies are better known.

The objective of this paper is to introduce a methodology for defining attribute grammars for conceptual schema languages taking advantage of conceptual modelling abstraction tools. The basic idea is to develop the conceptual schema (the data dictionary) of the modelling approach whose conceptual schema language is to be defined. The conceptual schema can be defined using any modelling approach including that same one. The information and the alteration structures of the modelling approach allow the identification of the different components of the attribute grammar. Namely, the information structure guides the definition of the context-free grammar as well as the choice of the attributes. The environment attribute corresponds to the state concept in conceptual modelling approaches. The alteration structure guides the definition of the semantic conditions and the semantic rules.

† To whom correspondence should be addressed.

The methodology can be used for the definition of any attribute grammar, with a small modification. Namely, in this situation it is necessary to get the context-free grammar and to develop the conceptual schema of the context dependencies.

The methodology is illustrated by an attribute grammar for the Infolog conceptual schema language. Since attribute grammars are not very well known in the field of conceptual modelling, it seems that the methodology can be very helpful for aiding in the definition of attribute grammars for conceptual schema languages.

In Section 2 the definition of attribute grammar is introduced and a small example is presented. In Section 3 the information and the alteration structures of the Infolog modelling approach are briefly introduced as well as the Infolog data dictionary. In Section 4 the methodology is presented and illustrated by an attribute grammar for the Infolog conceptual schema language.

2. ATTRIBUTE GRAMMARS

Attribute grammars were first introduced in Ref. 7. Several aspects of attribute grammars are discussed in Refs 2, 8–11. Illustrations of attribute grammars for programming languages are presented in Refs. 4 and 12. In Ref. 5 an attribute grammar for a conceptual schema language is presented and some ideas about its implementation using the YACC/LEX/C tools available in the UNIX system are discussed. Finally, a bibliography of attribute grammars can be found in Ref. 13.

2.1. Definition

An attribute grammar is a 5-tuple

$$(G, A, VAL, SR, SC)$$

where G is a reduced context-free grammar, A is a set of attributes, VAL is a set of domains of attribute values, SR is a set of semantic rules and SC is a set of semantic conditions.

The attribute set is the union of the sets of attributes for each symbol in the G grammar. The set of semantic rules is a set of equations allowing the evaluation of the

attribute instances. Thus a semantic rule is an equation of the following form:

$$(a0, p, Xk0) = f((a1, p, Xk1), \ldots, (ar, p, Xkr))$$

where p is the production $X0 \rightarrow X1, \ldots, Xn$, ai $(i = 0, \ldots, r)$ is an attribute of Xki, Xki is one of the grammar symbols involved in production p and f is a function whose range is the domain of attribute $a0$.

The semantic conditions are Boolean expressions used to constrain the set of sentences in the language, by allowing only the sentences of the G grammar than satisfy them.

An attribute has many instances according to the production rules where instances of the corresponding symbol appear. The instances of an attribute can be evaluated by different semantic rules and must satisfy conditions depending on the associated production rules.

The attributes can be classified either as synthesised or as inherited. A synthesised attribute is evaluated when its owner symbol is on the left side of a production. On the other hand, an inherited attribute is evaluated when its owner symbol is on the right side of a production. Generally, the start symbol only appears on the left side of productions, having therefore only synthesised attributes.

The main advantage of attribute grammars is that they are completely independent of any implementation issues. For instance, no symbol table structure has to be defined when specifying an attribute grammar.

2.2. An example

Herein, a small example of an attribute grammar is given. Consider that in a programming language the declaration of the variables of the same type is defined in BNF as follows:

⟨*declaration*⟩::= *VAR* ⟨*list-of-idents*⟩:⟨*type-var*⟩

⟨*list-of-idents*⟩::= ⟨*ident*⟩

| ⟨*list-of-idents*⟩1 ⟨*ident*⟩

⟨*type-var*⟩::= *CHAR* | *INT*

Assume also that there is a context dependence stating that the variables of the same type must have different identifications.

An attribute grammar for expressing the static semantics has the following attributes:

Attribute	**Symbol**
type	*declaration*
type	*type-var*
list-val-idents	*list-of-idents*
val-ident	*ident*

The semantic rules are the following:

type (⟨*declaration*⟩) = *type* (⟨*type-var*⟩)

type (⟨*type-var*⟩) = *CHAR*

in the scope of the production

⟨*type-var*⟩::= *CHAR*

and

type (⟨*type-var*⟩) = *INT*

in the scope of the production

⟨*type-var*⟩::= *INT*

list-val-idents (⟨*list-of-idents*⟩) = *val-ident* (⟨*ident*⟩)

in the scope of the production rule

⟨*list-of-idents*⟩::= ⟨*ident*⟩ and

list-val-idents (⟨*list-of-idents*⟩) =

append (*list-val-idents* (⟨*list-of-ident*⟩1), *val-ident* (⟨*ident*⟩))

in the scope of the production rule

⟨*list-of-idents*⟩::= ⟨*list-of-idents*⟩1 ⟨*ident*⟩

assuming that append is the operation append defined on the data type list.[14]

The last semantic rule gives the value of the attribute *list-val-idents*. According to the context dependency informally stated above, the new value can only be appended to the list defined provided there is no other variable with the same identification as the new one. This means that the following semantic condition must be introduced:

~ *EXISTS* (*list-val-idents* (⟨*list-of-idents*⟩1)) *val-ident* (⟨*ident'*⟩):
val-ident (⟨*ident'*⟩) = *val-ident* (⟨*ident*⟩)

This semantic condition must be verified whenever introducing a new element in the list. Note that the semantic condition above involves one instance of the attribute *val-ident*.

3. THE INFOLOG CONCEPTUAL MODELLING APPROACH

Conceptual modelling approaches include the specification of the information (data), the alteration (events) and the evolution (processes) structures of a system. Herein, a brief description of the information and the alteration structures of the Infolog modelling approach is presented.[15–18] The information structure includes abstractions for dealing with specialisation, generalisation and aggregation. The event structure includes the specification of events and rules stating the conditions that must be verified before the occurrence of an event, and the effects of an event.

3.1. Information structure

The basic Infolog information abstraction is the *archetype*. An archetype is a collection of archetype occurrences that share the same attributes. An *attribute* is a mathematical function that can either be a designator or a property. The co-domain of a designator is also an archetype and the co-domain of a property is a data type.

There are several kinds of archetypes: surrogate, relation, characteristic, specialisation, generalisation and aggregation. In this paper, no discussion is included concerning the generalisation abstraction.

The different kinds of archetypes can be distinguishable from each other through a definition involving compulsory properties and/or designators.

A *surrogate* is an archetype for which there is a key mechanism allowing the unique identification of the

different occurrences independently of the other archetypes. The key mechanism is composed of the following elements: the key data type, the key map and the key property. The key map is a function that by giving a value of the key data type allows the unique identification of the corresponding occurrence of the surrogate. The key property is a property whose co-domain is the key data type. For example, in a conference support system *PAPER* can be considered a surrogate with the following definition:

NUMBER: *PAPER* → *INTEGER*

PAP: *INTEGER* → *PAPER*

Given an occurrence of *PAPER*, the key property *NUMBER* refers to an integer value which identifies the occurrence of *PAPER* itself through the key map *PAP*.

A *relation* is an archetype that depends for existence on some other archetypes, called the arguments of the relation (the number of argument archetypes is the arity of the relation). The definition of a relation includes argument designators for each of the arguments of the relation and a reverse designator, such that by giving a tuple of occurrences of the argument archetypes returns a unique occurrence of the relation providing that it exists. For example, *AUTHORSHIP* can be considered as a binary relation between *PAPER* and *AUTHOR* with the following definition:

PAPE: *AUTHORSHIP* → *PAPER*

AUT: *AUTHORSHIP* → *AUTHOR*

PAPAUT: *PAPER* * *AUTHOR* → *AUTHORSHIP*

Given an occurrence of *AUTHORSHIP*, the above argument designators *PAPE* and *AUT* refer, respectively, to the corresponding occurrences of *PAPER* and *AUTHOR*, whereas the reverse designator *PAPAUT* refers to an occurrence of *AUTHORSHIP* given a tuple of occurrences of *PAPER* and *AUTHOR*.

A *characteristic* archetype also depends for existence on some other archetypes. The definition of a characteristic involves argument designators for each of the argument archetypes, and a partial key mechanism that allows the distinction between the different occurrences of a characteristic that can be connected to a tuple of argument occurrences. The partial key mechanism is also composed of three elements: the partial key data type, the partial key map and the partial key property. For example, *ACCEPTED-PAPER* can be considered as a unary characteristic of *SESSION* with the following definition:

SESS: *ACCEPTED-PAPER* → *SESSION*

NUMP: *ACCEPTED-PAPER* → *INTEGER*

ACCP:
SESSION * *INTEGER* → *ACCEPTED-PAPER*

Given an occurrence of *ACCEPTED-PAPER*, the argument designator *SESS* refers to the corresponding occurrence of *SESSION*. The partial key map *ACCP* refers to an occurrence of *ACCEPTED-PAPER* given a tuple composed by an occurrence of *SESSION* and an integer. The partial key property *NUMP* refers to the integer value above given the respective occurrence of *ACCEPTED-PAPER*.

A *specialisation* archetype groups in a new archetype occurrences of another archetype (the argument of the specialisation) for which it is needed to have additional information. The definition of a specialisation includes a discriminant property in the argument of the specialisation, a value in the co-domain of that property and a discriminant condition that must be verified whenever an occurrence of the argument archetype is an occurrence of the specialisation. Moreover, the definition includes an argument designator and a reverse designator as in the case of the unary relation. For example, *AUTHOR* can be considered as a specialisation of *PERSON* with the following definition:

PERS: *AUTHOR* → *PERSON*

TYPEP: *PERSON* → {*A*, *R*, *O*)

TYPEP (*p*) = *A*

AUTH: *PERSON* → *AUTHOR*

Given an occurrence of *AUTHOR*, the argument designator *PERS* refers to an occurrence of *PERSON*. The discriminant property *TYPEP* relates each occurrence of *PERSON* with a specific value of the enumerated set {*A*, *R*, *O*}. The discriminant condition *TYPEP* (*p*) = *A* imposes which occurrences of *PERSON* are to be occurrences of *AUTHOR*. The reverse designator *AUTH* refers to an occurrence of *AUTHOR* given an occurrence of *PERSON*, provided it satisfies the discriminant condition instanced for the relevant *PERSON* occurrence.

An *aggregation* archetype includes several definitions of archetypes. For example, *ACCEPTED-PAPER* can be considered as an aggregation of a unary characteristic of *SESSION* (as discussed above) and as a specialisation of *PAPER*. Thus, in this case, the aggregation includes two definitions of archetypes.

3.2. Alteration structure

The basic abstraction at this level is the *event type*. An event type is a collection of events (event type occurrences) whose information is structured in details and references. Both are mathematical functions. However, the co-domain of a detail is a data type, while the co-domain of a reference is an archetype.

There are several event types in the Infolog modelling approach: insertion and deletion of occurrences in archetypes and modification of properties and designators. An event is seen as an atomic state transition (leading the system from one state to another state) whose occurrence is indivisible. Thus the events must have a certain granularity. For example, an insertion occurrence allows the insertion of only one occurrence of an archetype. Only *insertion event types* are dealt with in this paper. An insertion event type is denoted by = } *A*, where *A* is the archetype affected by the event.

There are two main aspects related to insertion events. The conditions that allow the occurrence of insertion events (for example the target occurrence must not be present immediately before the insertion) and the effects of the insertion. The conditions are referred to as *enabling rules* and the effects are named *change rules*.

The following is an example of an enabling rule:

{{ ~ *EXISTS* (*TARGET* (= } *a*))}} = } *a*

where *EXISTS* is a Boolean property that must be defined for every archetype stating whether a given occurrence is or is not present, *TARGET* is a compulsory reference of every insertion event type, allowing the identification of the archetype occurrence that must be inserted when the corresponding event-type occurrence is to occur, and $=\}a$ is an occurrence of the insertion even type $=\}A$. This means that an insertion event can only occur if the target occurrence does not exist in the state immediately before the occurrence of the event.

The following is an example of a change rule:

$\{\} =\}a \{EXISTS\ (TARGET\ (=\}a))\}$

where the symbol $\{\}$ denotes the empty condition. Since the event occurs only when the enabling rule above is true, there is no need to state which condition is satisfied before the occurrence of the event. The meaning of the change rule is the following: after the occurrence of the event, the target occurrence must exist, no matter what happened in the state before the occurrence of the event.

It is worthwhile to say that the set of occurrences of an archetype that exist in a given moment is called its *valuation*. For example, the occurrence of an insertion-event type alters the valuation of the target archetype by including one more occurrence. The valuation before the occurrence of an event is designated by *pre-valuation*. The valuation after the occurrence of an event is called *pos-valuation*.

3.3. The Infolog data dictionary

As an illustration of the Infolog modelling approach, consider that one wants to develop the conceptual schema of the Infolog specifications (that is to say the data dictionary of the Infolog modelling approach). The discussion below includes the information and the alteration descriptions of the conceptual schema of Infolog specifications using the Infolog modelling approach.

Information structure

The first concept is specification. Consider the archetype *SPECIFICATION* as the collection of all Infolog specifications. The different specifications can be distinguished through a name. Thus the archetype *SPECIFICATION* must be considered as a surrogate whose key property is the name of the specification (*spec-name*).

Each specification can have several archetypes as a part of its information structure. Moreover, the archetypes must be uniquely identifiable in each specification. Thus the archetype *ARCHETYPE*, the collection of all archetypes, must be a characteristic of *SPECIFICATION*. The argument designator must be defined as follows:

spec-arc: $ARCHETYPE \rightarrow SPECIFICATION$

The partial key property *arcname* (the name of the archetype) allows the unique identification of the different occurrences of *ARCHETYPE* for a given occurrence of *SPECIFICATION*.

For similar reasons, consider the archetype *DATA-TYPE* as the collection of all data types that can be defined within a given specification. The archetype *DATA-TYPE* must also be a characteristic of the archetype *SPECIFICATION*. thus it is possible to identify uniquely all the data types belonging to the same specification.

Each archetype has two main aspects: a compulsory definition and the collection of optional attributes. Thus the archetype *COMP-DEF*, whose occurrences are the compulsory definitions of the archetypes, must be a relation of the archetype *ARCHETYPE*. On the other hand, an archetype can have several attributes. Then, the archetype *ATTRIBUTE*, the collection of all attributes, must be a characteristic of the archetype *ARCHETYPE*.

Assume that the argument designator of *COMP-DEF* is defined as follows:

arcname: $COMP\text{-}DEF \rightarrow ARCHETYPE$

This designator identifies to whose archetype a given compulsory definition belongs.

The compulsory definition of an archetype depends on the category of archetype. An example is the surrogate case. Thus the different compulsory definitions must be specialisations of the archetype *COMP-DEF*. Assume that the discriminant property of the archetype *COMP-DEF* is defined as follows:

arccat: $COMP\text{-}DEF \rightarrow \{SUR,\ REL,\ CHA,\ SPE,\ GEN,\ AGG\}$

Each of the values corresponds to a possible category of archetype. The archetype *SURROGATE*, the collection of all surrogate archetypes, is a specialisation of the archetype *COMP-DEF* with the following discriminant condition:

$arccat\ (COMP\text{-}DEF) = SUR$

Similar observations could be stated for the other compulsory definitions. The argument designator of the specialisation can be defined as follows:

arcname: $SURROGATE \rightarrow COMP\text{-}DEF$

Note that with the modelisation above it is not possible to have archetypes with the same name in different kinds. On the other hand, the grouping of archetypes of the same kind in new archetypes, e.g. *SURROGATE*, is important because the different kinds have different definitions.

Whenever an archetype is a surrogate the key mechanism must be defined. Thus the archetype *KEY-DEF*, the collection of all key definitions, must be a relation of the archetype *SURROGATE*. The argument designator of the relation can be defined as follows:

arcname: $KEY\text{-}DEF \rightarrow SURROGATE$

As stated above, the key mechanism includes the key map, the key property and the key data type. They are represented in the data dictionary in the following way:

K-map: $KEY\text{-}DEF \rightarrow string$

K-prop: $KEY\text{-}DEF \rightarrow PROPERTY$

K-data-type: $KEY\text{-}DEF \rightarrow DATA\text{-}TYPE$

The key property and the key data type are defined as designators since they are supposed to be considered, respectively, as common properties or data types. For example, for a certain archetype no other property can have the same name as the key property.

In a diagram, each archetype can be represented by a rectangle with a letter in the lower right-hand corner

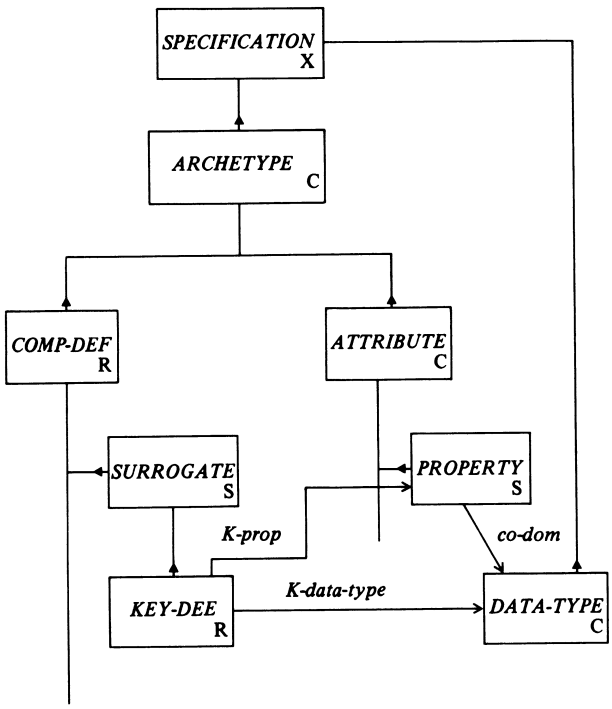


Figure 1. The Infolog partial data dictionary

indicating its kind. For example, X indicates that the corresponding archetype is a surrogate. The arrows denote dependencies between archetypes.

The final diagram for the Infolog partial data dictionary is presented in Fig. 1.

An attribute can be either a property or a designator. Thus the archetypes *PROPERTY* and *DESIGNATOR*, respectively the collection of all properties and designators in a given specification, must be specialisations of the archetype *ATTRIBUTE*. The archetype *DESIGNATOR* is not included in Fig. 1 for the sake of simplicity. The co-domains of the properties and the designators can be defined as designators in the following way:

Co-dom: *PROPERTY* → *DATA-TYPE*

Co-dom-des: *DESIGNATOR* → *ARCHETYPE*

Alteration structure

Consider that one wants to introduce a new surrogate in a given specification. A sequence of events of the following types must occur:

$=\}$*ARCHETYPE*, $=\}$*COMP-DEF*, $=\}$*SURROGATE*, $=\}$*KEY-DEF*

This sequence leads to the insertions of occurrences, respectively, in the following archetypes: *ARCHETYPE*, *COMP-DEF*, *SURROGATE* and *KEY-DEF*. For the sake of simplicity the optional properties and designators are not included. The definition of the even types is presented below.

$=\}$*ARCHETYPE*

References and details

spec-arc: $=\}$ *ARCHETYPE* → *SPECIFICATION*

target-arc: $=\}$ *ARCHETYPE* → *ARCHETYPE*

arcname: $=\}$ *ARCHETYPE* → *string*

The reference *spec-arc* defines the context (particular occurrence of *SPECIFICATION*) where the new occurrence is to be inserted. The reference *target-arc* refers to the integration of the new occurrence in *ARCHETYPE*. Finally, the detail arcname allows the unique identification of the new occurrence for the argument occurrence of *SPECIFICATION*.

Enabling rules

$\{\{\sim$ *EXISTS* (*target-arc* $\{=\}$ *archetype*))$\}\}$ $=\}$ *archetype*

$\{\{$*EXISTS* (*spec-arc* $(=\}$ *archetype*))$\}\}$ $=\}$ *archetype*

where $=\}$*archetype* is any occurrence of type $=\}$*archetype*.

The enabling rules above state, respectively, that the new occurrence (indicated by *target-arc*) must not belong to the pre-valuation of *ARCHETYPE* and that the referred-argument occurrence (indicated by *spec-arc*) must belong to the value of *SPECIFICATION*.

Change rules

$\{\}=\}$*archetype* (*arcname* (*target-arc* $(=\}$ *archetype*)) = *arcname* $(=\}$ *archetype*)$\}$

$\{\}=\}$*archetype* $\{$*arcname* (*target-arc* $((=\}$ *archetype*)) = *spec-arc* $(=\}$ *archetype*)$\}$

The first rule states that after the occurrence of the insertion event the value of the property *arcname* must be the value of the detail *arcname* of the occurrence (referred by *target-arc*) of the event. The second change rule states that after the occurrence of the event the value of the argument designator *arcname* must be the value of the reference *spec-arc* of the insertion event.

$=\}$*COMP-DEF*

References and details

arcname: $=\}$ COMP-DEF → *ARCHETYPE*

target-comp-def: $=\}$ *COMP-DEF* → *COMP-DEF*

arccat: $=\}$ *COMP-DEF* → {*SUR*, *REL*, *CHA*, *SPE*, *GEN*, *AGG*}

Change rules

$\{\}=\}$ *comp-def* $\{$*arcname* (*target-comp-def* $(=\}$ *comp-def*)) = *arcname* $(=\}$ *comp-def*)$\}$

$\{\}=\}$ *comp-def* $\{$*arcat* (*target-comp-def* $(=\}$ *comp-def*)) = *SUR*$\}$

$=\}$*SURROGATE*

References and details

ARCNAME: $=\}$ *surrogate* → *COMP-DEF*

target-sur: → *SURROGATE* → *SURROGATE*

Change rules

$\{\}=\}$*surrogate* (*arcname* (*target-sur* $(=\}$ *surrogate*)) = *arcname* $(=\}$ *surrogate*)$\}$

$=\}$*KEY-DEF*

References and details

arcname: =} *KEY-DEF* → *SURROGATE*

target-key-def: =} *KEY-DEF* → *KEY-DEF*

K-map: =} *KEY-DEF* → *string*

K-prop: =} *KEY-DEF* → *PROPERTY*

K-data-type: =} *KEY-DEF* → *DATA-TYPE*

Change rules

{ } =} *key-def* {*arcname* (*target-key-def* (=} *key-def*)) = *arcname* (=} *key-def*)}

{ } =} *key-def* {*K-map* (*target-key-def* (=} *key-def*)) = *K-map* (=} *key-def*)}

{ } =} *key-def* {*K-prop* (*target-key-def* (=} *key-def*)) = *K-prop* (=} *key-def*)}

{ } =}*key-def*{*K-data-type*(*target-key-def*(=}*key-def*)) = *K-data-type* (=} *key-def*)}

The details, the references and the change rules for the last events are not detailed herein, since they have similar interpretations to those of the details, references and change rules for the insertion event =}*ARCHETYPE*.

4. DEVELOPING AN ATTRIBUTE GRAMMAR FOR A CONCEPTUAL SCHEMA LANGUAGE

Herein a methodology is introduced for defining attribute grammars in a structured way. The methodology is exemplified for the Infolog conceptual schema language, but it could have been used for defining attribute grammars for any conceptual schema language.

4.1. The methodology

The methodology for introducing attribute grammars for conceptual schema languages assumes that the conceptual schema of the modelling approach is defined (that is to say its data dictionary). Two assumptions are made. In the first place, the data dictionary of the modelling approach for which a conceptual schema language is to be developed must be defined by using the Infolog abstractions. In the second place, the conceptual schema of the data dictionary must be *simple*. That is to say the conceptual schema has no *n*-ary relation and characteristic with $n > = 2$. The case when the conceptual schema is not simple is discussed below.

The methodology

Input. A simple conceptual schema of the data dictionary of a modelling approach defined with the Infolog abstractions

Output. An attribute grammar for a conceptual schema language for the referred modelling approach

Steps

(1) *Context-free grammar*

(1.1). All the archetypes correspond to syntax symbols in the context-free grammar.

(1.2). If an archetype *A* is a unary characteristic of another archetype *A* 1, the following production must be included:

⟨*a* 1⟩ ::= ⟨*list-of-a*⟩ ⟨*a*⟩

assuming that *a* 1 and *a* are the symbols that correspond respectively to the archetypes *A* 1 and *A*. Thus the symbol that represents the argument archetype of the unary characteristic must be the left-hand side of the production, and a list of symbols must be included in the right-hand side representing the characteristic archetype. Moreover, the following production must be introduced:

⟨*list-of-a*⟩ ::= ⟨*a*⟩
| ⟨*list-of-a*⟩ ⟨*a*⟩

(1.3). If an archetype *A* is a unary relation of another archetype *A* 1, the following production must be included:

⟨*a* 1⟩ ::= ⟨*a*⟩

where *a* 1 and *a* are the symbols that represent, respectively, the archetypes *A* 1 and *A*. Thus the symbol that represents the argument of the relation must be the left-hand side of a production rule where the symbol that represents the relation is on the left-hand side.

(1.4). If an archetype *A* has several specialisations *A* 1, ..., *An*, the following production rule must be considered:

⟨*a*⟩ ::= ⟨*a* 1⟩ | ... | ⟨*an*⟩

where *a*, *a* 1, ..., *an* are the symbols that represent, respectively, the archetypes *A*, *A* 1, ..., *An*. Thus, the symbol that corresponds to the argument archetype must be the left-hand side of several productions (as many as the specialisations) whose right-hand sides are the symbols that represent the specialisations.

(1.5). If an archetype *A* is the argument of several unary characteristics *A* 1, ..., *An* and several unary relations *B* 1, ..., *Bm*, then the following production must be added:

⟨*a*⟩ ::= ⟨*list-of-a* 1⟩ ... ⟨*list-of-an*⟩ ⟨*b* 1⟩ ... ⟨*bm*⟩

where *a*, *a* 1, ..., *an*, *b* 1, ..., *bm* are symbols that represent, respectively, the archetypes *A*, *A* 1, ..., *An*, *B* 1, ..., *Bm*.

(1.6). The starting archetype (which is always a surrogate) leads to the introduction of a production rule where the corresponding symbol is the left-hand side and whose right-hand must include all the symbols that represent the archetypes that have the starting archetype as argument.

(2) *Attributes*

(2.1). The attributes of each syntax symbol in the attribute grammar must correspond to the properties and the designators of the archetype represented by the syntax symbol;

(2.2). The attribute environment of each syntax symbol represents the valuation of the corresponding archetype. The in-environment attribute represents the pre-valuation of the archetype (before the insertion of the new occurrence) and the out-environment attribute represents the pos-valuation of the archetype (immedi-

ately after the insertion of the new occurrence). The in-environment and the out-environment are designated below, respectively, by inenv and outenv followed by a mnemonic of the symbol to which they refer.

(3) *Semantic conditions*

The semantic conditions involving each syntax symbol must correspond to the enabling rules that restrict the occurrence of the insertion events related to the archetype that is associated with the syntax symbol.

(4) *Semantic rules*

The semantic rules involving each syntax symbol must represent the change rules that define the values of the properties and the designators of the new occurrence of the archetype associated with the syntax symbol.

4.2. Illustration of the methodology

The methodology is illustrated for the Infolog conceptual schema language. The Infolog data dictionary described in 3.3 above guides the definition of the attribute grammar for the conceptual schema language. The use of keywords is not part of the methodology itself. The objective of introducing keywords is for simplicity reasons. Whenever a syntactic category needs a name the syntactic category ident is introduced.

(1) *Context-free grammar*

⟨*specification*⟩::= **Specification**
⟨*ident*⟩ ⟨*list-archetypes*⟩ ⟨*list-data-types*⟩

For example, the syntax symbol specification corresponds to the archetype *SPECIFICATION*. For the sake of simplicity, the symbol ⟨*ident*⟩ is added to the specification without any reference to how it was obtained using the proposed methodology. This production rule conforms to rule 1.5 of the methodology, since both the archetypes *ARCHETYPE* and *DATA-TYPE* are unary characteristics of the archetype *SPECIFICATION*. The symbol ⟨*specification*⟩ appears at the left-hand side of this production, since according to rule 1.6,

⟨*list-archetypes*⟩::= ⟨*archetype*⟩
| ⟨*list-archetype*⟩ 1 ⟨*archetype*⟩
⟨*archetype*⟩::= ⟨*comp-def*⟩ ⟨*list-attributes*⟩

The symbol *comp-def* corresponds to the archetype *COMP-DEF*. On the other hand, the symbol *list-attributes* corresponds to the characteristic archetype *ATTRIBUTE*.

⟨*comp-def*⟩::= ⟨*surrogate*⟩
| ⟨*relation*⟩
| ⟨*characteristic*⟩
| ⟨*specialisation*⟩
....

The different options in the right-hand side of this production correspond to the different specialisations of the archetype *COMP-DEF* (e.g. the archetype *SURROGATE* whose syntax symbol is *surrogate*).

⟨*list-attributes*⟩::= ⟨*attribute*⟩
| ⟨*list-attributes*⟩ ⟨*attribute*⟩
⟨*attribute*⟩::= ⟨*property*⟩ | ⟨*designator*⟩
⟨*surrogate*⟩::= **Surrogate** ⟨*ident*⟩
⟨*key-def*⟩

The symbol *key-def* corresponds to the relation archetype *KEY-DEF*. The *key-def* symbol appears in the right hand side of the production rule, since the archetype *KEY-DEF* is a unary relation of the archetype *SURROGATE*.

⟨*key-def*⟩::= **Key**
Map ⟨*ident*⟩
Prop ⟨*ident*⟩: ⟨*ident*⟩

An example of a fragment for the conference support system is the following:

Specification *CSS*
Surrogate *paper*
Key
Map *pap* **Prop** *code*: *integer*
Title: *string*
Number-of-Words: *integer*

(2) *Attribute choice*

As was stated above, the attributes of a given symbol correspond to the properties and designators of the corresponding archetype. The symbol specification that corresponds to the archetype *SPECIFICATION* has the following attributes:

spec-name: *specification* → *string*

corresponding to the key property of *SPECIFICATION*.

Consider the symbol *archetype* that represents the archetype *ARCHETYPE*. The following are attributes of the symbol *archetype*:

spec-arc: *archetype* → *string*
arcname: *archetype* → *string*

The attribute *spec-arc* corresponds to the argument designator with the same identifier. It indicates the specification to which an archetype must be connected. The attribute *arcname* corresponds to the partial key property of the archetype *ARCHETYPE*.

The attributes of the symbol *comp-def* are the following:

arcname: *comp-def* → *string*
arccat: *comp-def* → {*SUR*, *REL*, *CHAR*, *PART*, *GEN*, *AGGR*}

The attribute *arccat* corresponds to the discriminant property of the same archetype (which allows the definition of the different specialisations).

Consider the symbol *surrogate* that corresponds to the

archetype *SURROGATE*. The following is an attribute of the symbol *surrogate*:

arcname: *surrogate* → *string*

corresponding to the argument designator that connects the specialisation *SURROGATE* with the argument archetype *COMP-DEF*.

The symbol *key-def* corresponding to the archetype *KEY-DEF* has the following attributes:

arcname: *key-def* → *string*

key-map: *key-def* → *string*

k-prop: *key-def* → *string*

k-data-type: *key-def* → *string*

The attribute *arcname* corresponds to the argument designator of the archetype *KEY-DEF* whose argument is the archetype *SURROGATE*. The other attributes correspond, respectively, to the property *K-map* and the designators *K-prop* and *K-data-type* of the archetype *KEY-DEF*.

All the syntax symbols introduced above have the attributes in-environment and out-environment, which correspond to the valuation of the different archetypes. A change of valuation resulting from the insertion of an occurrence in an archetype leads to a new value of the environment attribute.

(3) *Semantic conditions*

The following is an example of a semantic condition:

validarcname (*spec-arc* (⟨*archetype*⟩,

arcname (⟨*archetype*⟩), *inenv-arc* (⟨*archetype*⟩))

This semantic condition restricts the introduction of a new archetype to the validity of the condition. It corresponds to the following enabling rules:

{*EXISTS* (*spec-arc* (*target-arc* (=} *archetype*))}} =} *archetype*
{{~ *EXISTS* (*target-arc* (=} *archetype*))}} =} *archetype*

introduced above. Note that the semantic condition depends on the existence of the relevant specification, the name of the new archetype and the value of the in-environment attribute *inenv-arc* just as the enabling rules above. Note that if an archetype does not exist in the referred specification with the indicated name, there is no need to perform more validations on the archetypes *SURROGATE* and *KEY-DEF*. Concerning the attribute grammar, there are no semantic conditions for validating the attributes of the symbols *comp-def*, *surrogate* and *key-def*.

(4) *Semantic rules*

The semantic rules are defined below, taking into account the production rules, where the attributes of the symbol on the left-hand side of the production rule are evaluated.

⟨*specification*⟩::=

⟨*ident*⟩ ⟨*list-archetypes*⟩ ⟨*list-data-types*⟩

spec-name (⟨*specification*⟩) = *val* (⟨*ident*⟩)
spec-name (⟨*list-archetypes*⟩) = *spec-name* (⟨*specification*⟩)

Thus the attribute *spec-name* for *specification* is a synthesised attribute (since it is evaluated when the respective symbol is on the left-hand side of the production rule), while *spec-name* for *list-archetypes* is an inherited archetype (since it is evaluated when the respective symbol is on the right-hand side of the production rule). Note that *val* is an attribute of the symbol *ident*.

⟨*archetype*⟩::= ⟨*comp-def*⟩ ⟨*list-attributes*⟩

arcname (⟨*archetype*⟩) = *arcname* (⟨*comp-def*⟩)

outenv-arc (⟨*archetype*⟩) =

ins-arc (*spec-name* (⟨*archetype*⟩,

arcname (⟨*archetype*⟩), *inenv-arc* (⟨*archetype*⟩))

The first semantic rule corresponds to the following change rule:

{ } =} *archetype* {*arcname* (*target-comp-def* (=} *comp-def*)) = *arcname* (=} *comp-def*)}

Note that *arcname* (*target-comp-def* (=} *comp-def*)) refers to an occurrence of *ARCHETYPE*. The second rule corresponds to the change of valuation generated by the new insertion event. Note that the insertion function *ins-arc* must have as arguments the attributes of the symbol that represents all the properties and designators of the archetype represented by the symbol.

⟨*comp-def*⟩::= ⟨*surrogate*⟩

arcname (⟨*comp-def*⟩) = *arcname* (⟨*surrogate*⟩)

arccat (⟨*comp-def*⟩) = *SUR*

outenv-comp-def (⟨*comp-def*⟩) =

ins-comp-def (*arcname* (⟨*comp-def*⟩), *arccat* (⟨*comp-def*⟩),

inenv-comp-def (*comp-def*⟩)

The new compulsory definition must be inserted in connected with the archetype, the respective category and the previous valuation.

⟨*surrogate*⟩::= **Surrogate** ⟨*ident*⟩

⟨*key-def*⟩

arcname (⟨*surrogate*⟩) = *val* (⟨*ident*⟩)

arcname (⟨*key-def*⟩) = *arcname* (⟨*surrogate*⟩)

outenv-sur (⟨*surrogate*⟩) =

ins-sur (⟨*arcname* (⟨*surrogate*⟩, *inenv-sur* (⟨*surrogate*⟩))

Note that the value of the attribute *arcname* is effectively evaluated in the first rule, since its value depends on the value of the identifier on the right-hand side of the production rule. Moreover, the attribute *arcname* for *key-def* symbol is inherited.

⟨*key-def*⟩::= **Key**

Map ⟨*ident*⟩ 1

Prop ⟨*ident*⟩ 2: ⟨*ident*⟩ 3

k-map (⟨*key-def*⟩) = *val* (⟨*ident*⟩ 1)

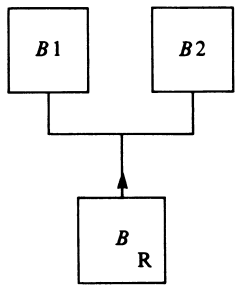
k-prop (⟨*key-def*⟩) = *val* (⟨*ident* 2⟩)

k-data-type (⟨*key-def*⟩) = *val* (⟨*ident* 3⟩)

outenv-key-def (⟨*key-def*⟩) = *ins-key-def* (⟨*arcname* (⟨*key-def*⟩), *k-map* (⟨*key-def*⟩), *k-prop* (⟨*key-def*⟩), *k-data-type* (⟨*key-def*⟩), *inenv-key-def* (⟨*key-def*⟩))

Note that the new key definition must include the definition of the designators *K-data-type* and *K*-prop and the definition of the *k-map* property.

The conceptual schema



is equivalent to the following conceptual schema

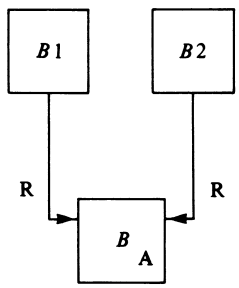


Figure 2.

Consider now that the conceptual schema of the dictionary is not simple. That is to say it has relations and/or characteristics with arity $\geqslant 1$. The first objective is to *transform* the conceptual schema into a simple conceptual schema according to the following proposition: 'An *n*-ary ($n \geqslant 2$) relation (characteristic) is equivalent (in terms of semantic power) to an aggregation of *n* relation (characteristic) archetypes each one with arity one (each corresponding to one of the arguments).' As an example, see the transformation in Fig. 2 for a binary relation.

Note that the following production rule must be introduced for an aggregation *A* of the archetype definitions *A* 1, . . . , *An*:

⟨*a* 1⟩::= ⟨*a*⟩

. . . .

⟨*an*⟩::= ⟨*a*⟩

where *a*, *a* 1, . . . , *an* are the symbols that represent, respectively, the archetypes *A*, *A* 1, . . . , *An*.

5. CONCLUSIONS

The basic objective of the paper was to introduce a methodology for developing attribute grammars for conceptual schema languages in a structured way. The methodology assumes that the conceptual schema of the data dictionary of the modelling approach, whose conceptual schema language is under development, is defined using the Infolog modelling approach. The information structure of the conceptual schema provides guidelines for defining both the context-free grammar and the attributes for each syntax symbol. The alteration structure of the conceptual schema allows the definition of the semantic conditions and the semantic rules. The methodology is illustrated for a conceptual schema language for the Infolog modelling approach itself.

It is worthwhile to report that the above methodology was used in the definition of a meta-knowledge representation language (Knowlog),[19] as well as in the definition of the Infolog conceptual schema language.

Acknowledgements

The authors would like to thank Professor A. Sernadas for many useful discussions. The work was partially supported by Junta Nacional de Investigação Científica e Tecnológica under research contract no. 417.82.57.

REFERENCES

1. F. Pagan, *The Formal Specification of Programming Languages*. Prentice-Hall, Englewood Cliffs, N.J. (1981).
2. P. Lewis, D. Rosenkrantz and R. Stearns, Attributed translations. *Journal of Computer and System Sciences* **9**, 279–307 (1974).
3. J. Cleaveland and R. Uzgalis, *Grammars for Programming Languages*. Elsevier, Amsterdam (1977).
4. S. Drossopoulou *et al.* An attribute grammar for ADA *Proceedings of the Sigplan 82 Symp. on Compiler Construction. ACM Sigplan Notices* **17** (6), 334–349 (1982).
5. C. Sernadas and G. Gaspar, The Infolog language facility. *Infolog* **RR 14** (1984).
6. K. Christian, *The Unix Operating System*. John Wiley, Chichester (1983).
7. D. Knuth, Semantics of context-free languages. *Mathematical Systems Theory* **2** (2), 127–145 (1968).
8. H. Alblas, A characterization of attribute evaluation in passes. *Acta Informatica* **16**, 427–464 (1981).
9. J. Engelfriet and G. File, The formal power of one-visit attribute grammars. *Acta Informatica* **16**, 275–302 (1981).
10. U. Kastens, Ordered attribute grammars. *Acta Informatica* **13**, 229–256. (1980).
11. G. Adorni, A. Boccalatte and M. Manzo, Top-down semantic analysis. *The Computer Journal* **27**(3), 233–237 (1984).
12. D. Watt, An extended attribute grammar for pascal. *ACM Sigplan Notices* **14** (2), 60–74 (1979).
13. K. Raida, Bibliography on attribute grammars. *ACM Sigplan Notices* **14** (8), 35–44 (1980).
14. J. Guttag, E. Horowitz and D. Musser, The design of data type specifications. In *Current Trends in Programming methodology*: *Data Structuring*, edited by R. Yeh. Prentice-Hall, Englewood Cliffs, N.J. (1978).
15. A. Sernadas and C. Sernadas, Infolog: an integrated model of data and processes, IFIP WG 8.1 Meeting, York, July 8–9. *Infolog* **RR 07** (1983).
16. A. Sernadas, C. Sernadas, J. Fiadeiro and J. Granado, Information systems development with Infolog. *Infolog* **RR 25** (1985).
17. A. Sernadas and C. Sernadas, Capturing knowledge about the organisation dynamics. In *Knowledge Representation for Decision Support Systems*, edited L. Methlie and R. Sprague. North-Holland, Amsterdam (1985).
18. C. Sernadas and A. Sernadas, Conceptual modelling abstraction mechanisms as parameterized theories in institutions. In *Database Semantics*, edited R. Meersman and T. Steel. North-Holland, Amsterdam (1985).
19. R. Carapuça, A. Sernadas and C. Sernadas, Knowlog: tools for the modular construction of knowledgebases. *Infolog* **RR 53** (1986).