Supporting OIS design through semantic queries

B. Pernici
Politecnico di Milano

Office information system design is a complex activity which needs conceptual modeling to represent office data and activities in a formal way. The problem of retrieving information from conceptual schemas of office systems is addressed, to support the designer in the different phases of office information system design. A semantic query language is proposed, based on a semantic representation of the TODOS Conceptual Model for office design.

1. INTRODUCTION

The complexity of information system (IS) design and in particular of office information system (OIS) design requires the use of design tools for supporting the different design activities.

Two types of tools have been proposed in the literature: conceptual models and computer based design tools. *Conceptual models* allow to represent data and procedures in routine office activities /Bra 84a, Bra 84b, Nut 81/. Such models are often associated with a methodology for collecting and modeling requirements. The use of *automated design tools* has been advocated in /Sch 82/ for IS design, and in /Kon 82/ for OIS design, due to the amount of data to be collected and the effort to produce project documentation, the focus being on producing correct and easy to read reports on requirements. Information resource dictionaries /Nav 86/ have been proposed to collect project management data, while data dictionaries are used to collect meta-data about database schemas /Dat 85/. The structure of such dictionaries is very often based on the relational model of data.

The TODOS (Automatic Tools for Designing Office Systems) project, a three year project in the ESPRIT Programme of the Commission of European Communities, has the purpose of providing a design support environment for the design of OIS. TODOS automatic tools include a requirement collection tool, a conceptual modeling tool, an office rapid prototyping tool, and a tool for selecting an architecture for the office system being developed. The TODOS approach to OIS design is presented in /Per 86/. The OIS group at Politecnico di Milano has been working in particular on the development of the conceptual modeling tool C-TODOS /Per 87/.

C-TODOS is based on the TODOS Conceptual Model (TCM) and on a Specification Database (SDB), storing the formal specification of office elements being defined. While the first phase of C-TODOS development focused on the memorization of specification in the SDB, recent work showed that a sophisticated query module is needed, to allow the designer to interrogate the base of collected requirements and specifications, to support further analysis and modelization of the system being developed, and to support the other design phases of prototyping and architecture selection.

The purpose of this paper is to propose a method for allowing complex queries on an office conceptual schema. The basis of this method is a semantic representation of conceptual models. The powerful query language on semantic schemas being developed to this purpose and tailored specifically to design queries is presented.

In Sect. 2., the TODOS project is presented, with a short overview on the TODOS Conceptual Model. The semantic representation of TCM and the semantic query language (TOD-QueL: TODOS Query Language) are described in Sect. 3. In Sect. 4., the use of such a query language for supporting office design is discussed. We also discuss how the use of semantic queries could be generalized to build design support tools in database design, software engineering, object oriented system design, and knowledge base design.

2. AUTOMATIC TOOLS FOR DESIGNING OFFICE SYSTEMS

2.1 Overview

The TODOS project aims at studying and realizing an integrated set of automatic tools to support OIS design. TODOS tools support office designers in producing correct and complete functional specifications of office systems, in defining user interfaces, and in selecting office system architectures.

The definition of functional specifications is supported by the *conceptual modeling tool C-TODOS*. Office conceptual schemas, defined using the TODOS Conceptual Model (TCM) are stored in a specification database

(SDB). C-TODOS allows the designer to enter office element specifications, automatically checking their correctness and consistency with elements already defined in the SDB.

The functional specifications stored in the SDB are used as an input by the two other design support tools: the office rapid prototyping tool and the architecture selection tool.

The *rapid prototyping tool* semiautomatically builds the user interface of the office system, on the basis of the elements defined in the functional specifications. Future users of the office system are thus able both to verify using the prototype whether the office system being developed meets their requirements and to suggest modifications to the interface layouts.

The *architecture design tool* receives as its input the office conceptual schema and other non-functional requirements defined during the requirement collection and analysis phase, such as office layout, performance and cost constraints. Based on these elements, an architecture for the office system, including both software and hardware components and their connections is proposed, satisfying the given constraints. The office schema is therefore a central element of TODOS design methodology.

During the first phase of the TODOS project, it was assumed that functional specifications where passed altogether onto the following phases of rapid prototyping and architecture selection. On going research is now concentrating on how the SDB, storing all functional specifications, can be interrogated, allowing complex queries to extract specifications and their characteristics selectively. In the following of this paper, we briefly introduce the characteristics of TCM first, then we discuss how to interrogate office schemas with a semantic query language.

2.2. TODOS Conceptual Model: a semantic office model

Abstraction mechanisms

Semantic models have been used in many areas of computer science, e.g., for knowledge representation in artificial intelligence and in database and information system modeling /Bro 84/.

TCM is a semantic model for office systems, utilizing the concepts of entity type, generalization, aggregation, association. Moreover, it is possible to establish semantic links ('references') between entities.

Each entity type is defined as a specialization of another type defined in the schema, that can be either one of the predefined types defined in the model or another type already defined in that particular office schema. *Specializations* of types inherit all properties of their supertypes. Moreover, each type can have associated its own properties, using the constructs of *aggregation* (list of properties), *association* (repeating groups of properties) and *semantic links* (properties defined as references from a property to another entity type).

An example of definition of an office element is graphically represented in Fig. 1., which illustrates the static entity *letter* and its specialization *reply-letter*.

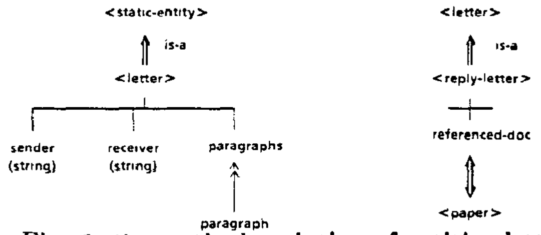

Fig. 1. Semantic description of entities *letter* and *reply-letter*

The entity *letter* is a subtype of entity type *document* and is defined as an aggregation of properties 'sender', 'receiver', and 'paragraphs', which in turn consists of an association of items with property name 'paragraph'.

Entity *reply-letter* is defined as a subtype of *letter* and inherits all properties defined for entity *letter*. In addition, it has its own particular property 'referenced-doc', which is a reference to another entity, *paper*.

Elements of the model

Office element types belong to two basic types: static entities and dynamic entities.

Static entities allow to model information in the office, such as documents, messages, information bearing objects, and agents performing office activities. The entities *letter* and *reply-letter* defined in Fig. 1. are examples of static elements. *Documents* are office elements of the office system which are important in the office for storing information. The data they contain are usually indivisible, that is documents should be archived in the same form they are presented and distributed to the office workers. *Messages* contain information coming to the office system from the outside world (e.g., another office, another organization) or distributed to the outside world. *Objects* contain information needed to perform office activities and not directly contained in documents.

Dynamic entities model information flows mainly making use of the concepts of event and action. *Actions* on static elements are specified in TCM in dynamic-transitions, where for each *event* type all its consequences are specified in terms of actions performed on static elements. These actions can cause new events, thus having event - action - event cycles. Predefined events are: arrival of a message, modification or removal of a static entity, temporal event, completion of the content of a static entity. A predicate is associated to each event definition which specifies the occurrence condition for the event.

3. SEMANTIC QUERY LANGUAGE

In this section we present the TODOS Query Language (TODQUEL), which allows to interrogate TCM schemas.

TODQUEL is based on a semantic representation of TCM and provides constructs for navigation in a semantic schema. TODQUEL constructs are not specific to TCM and can be used to query semantic representations of other (semantic) models, working on a semantic representation of these models.

3.1. Semantic representation of a model

The semantic representation of a model is built using the following semantic modeling constructs: subtyping, aggregation, association, and semantic links.

The semantic representation of the model has the purpose of being a guidance schema for user queries, and not that of defining the schema, therefore redundancy is used in describing concepts to enable to express complex queries in an easy and understandable way.

In Fig. 2. a semantic representation of TCM is given. It contains a description of the TCM concept *entity*, of the TCM modeling constructs used to define entities (*structure, aggregation-structure, association-structure, reference-structure*), and of some (for sake of brevity) of the predefined elements in the model (*document, dynamic-transition, action*).

3.2. Semantic query constructs

The semantic description of a model can be used as a basis for asking queries on schemas defined using that model. In the following, we present a semantic query language which allows to navigate in semantic networks. Using TODQUEL, it is possible to retrieve the name and/or the definition of all entities in the office schema satisfying a condition. TODQUEL is based on set theory and predicate calculus, and provides navigation constructs for the basic constructs of the semantic network: aggregation, association, reference, and subtyping. The grammar for TODQUEL is presented in App. 1. In the following, we consider each construct separately.

Aggregation

We have chosen the dotted notation to address aggregations of properties. In the dotted notation, used for instance in functional query languages, such as /Shi 81/, and in relational query languages, such as SQL /Dat 85/, it is possible to refer to a property of a type as follows:

type.property-name. ....property-name

Association

Members of an association can be addressed in three different ways in queries:

- retrieval of ´one of´ (any) the member properties

For instance: "retrieve entities with a property with property name equal to 'sender'"

retrieve t.name ($\forall$ t ϵ entity | t.property-list one of p.property-name = 'sender')

- requiring that ´all´ member properties present the same characteristics

For instance: "retrieve those entities whose properties are all not of type 'image', i.e., non graphical documents

retrieve t.name ($\forall$ t ϵ entity | t.property-list all p.property-type $\neq$ 'image')

- retrieval of *single members* of an association. For the purpose of retrieving single members of associations, we assume to have an implicit internal order of the members: So, we can address the members of 'property-list' as follows:

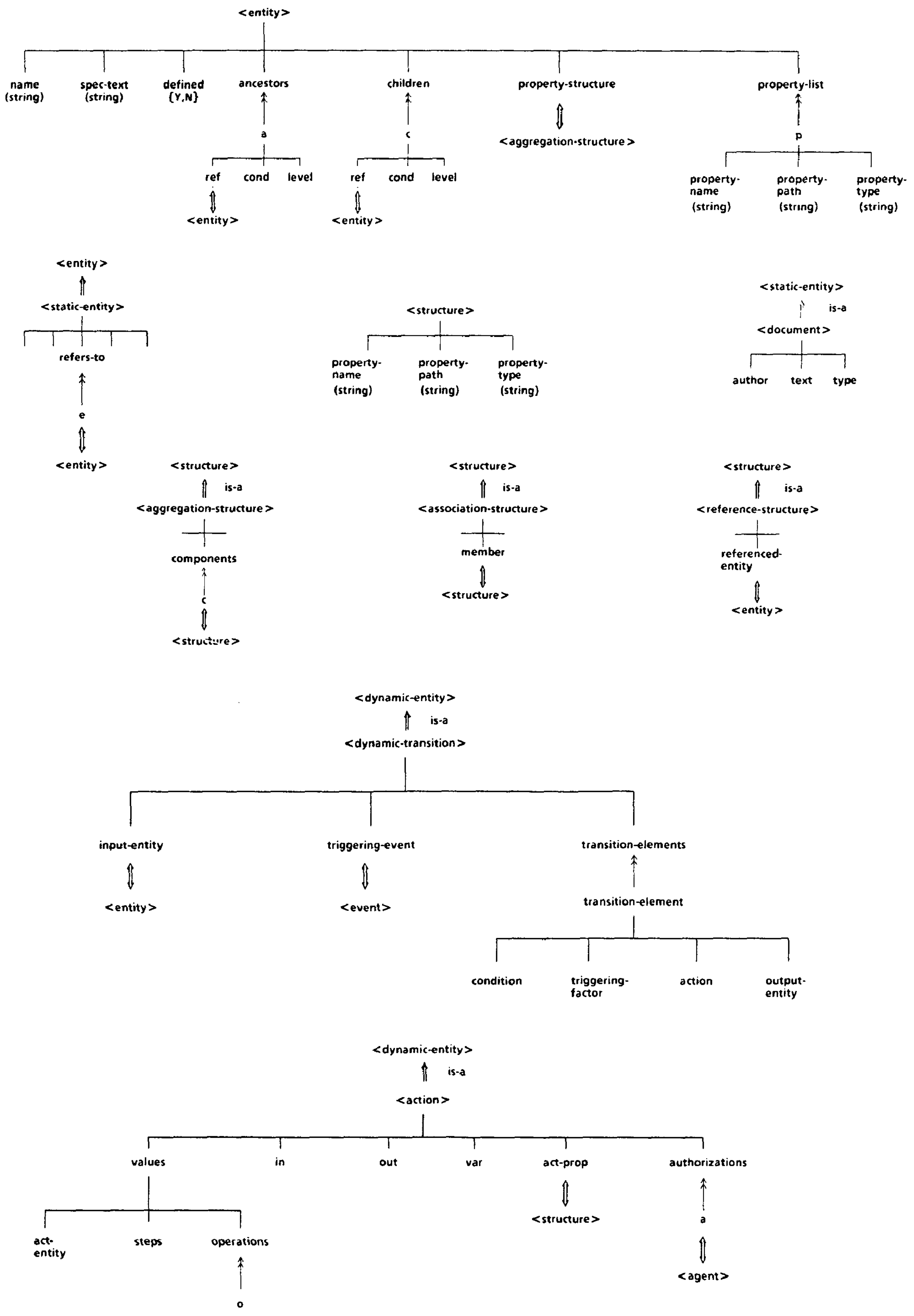

Fig. 2. Semantics of some general element types in TCM

t.property-list ith p

References

In the retrieval of elements, queries may use semantic links between entities.

For instance, we might ask "the name of the entities referred to from a static entity":

retrieve t.name (∀ t ϵ entity, ∃ s ϵ static-entity | s.refers-to one of e = t)

In some other cases, it may be necessary to control semantic links traversals mentioning the type and number of semantic links to be traversed, using the following construct:

through n references to *type-name*

Generalization abstraction

Properties inherited from supertypes are not distinguishable from properties specifically defined for a certain type. Although property inheritance is a useful concept for type definition, it may be heavy to consider all inherited properties when querying about entity properties. In many cases, the designer wants to know which properties have been specified specifically for a particular entity.

We provide the 'hide' query mechanism to control the result of queries in relation to their supertypes:

(hide supertypes from nth level up)

For instance:

retrieve p (∀ p ϵ string, ∃ t ϵ entity |
t.name = 'E' ∧ p = t (hide supertypes from 1st level up).property-list one of p.property-name

retrieves all properties specifically defined for a certain entity type 'E' (all supertypes are hidden in the query result).

On the other hand, it may be useful to know only which are the inherited properties. We obtain this with the following construct:

hide supertypes from 0th level down

Containment property

Since TCM entities are defined in general as complex structures, it is useful to provide the capability of querying structures about their similarity. For instance, it is of interest to select all structures including a given sub-structure. This condition can be expressed with a special keyword *contains* that allows to match pairs of structures.

The following query is an example of the use of the contains construct:

"Get the entity types containing the following structure:

"*: aggregation-of
referenced-doc :* "

The sub-structure definition is dependent on the model, and uses the schema definition language constructs.

The wild card '*' is used to indicate parts of the structure which are not of interest in the query (in this case, the first * indicates any property name, and the second * that any structure is accepted for property 'referenced-doc'.

4. SEMANTIC QUERIES FOR OIS DESIGN

Semantic queries are a powerful tool for the conceptual designer to verify the consistency and comleteness of the office schema. Moreover, semantic queries can be used to query the office schema during prototyping and architecture selection.

Queries for conceptual design

Queries on the conceptual schema can be performed at several points during a design session, as an aid to the designer to check completeness and consistency of the office schema being created: when elaborating new specifications, inserting specifications, at the end of a design session.

While defining elements of the office schema, the designer may be interested in knowing which element types have already been defined. We present here two examples of such queries:

(Q4.1) "retrieve all the subtypes of a given entity E"

retrieve sube.name ($\forall$ sube ϵ entity, $\exists$ t ϵ entity |
t.name = 'E' $\wedge$ sube = t.children one of c.ref)

(in Fig. 2., *entity* has a property named 'children', which is an association of elements c, with property *ref*, referring to elements of type *entity*)

(Q4.2) "all actions performed as a response to an event E"

retrieve a.name ($\forall$ a ϵ action, $\exists$ d ϵ dynamic-transition |
d.triggering-event.name = 'E' $\wedge$ a = d.transition-elements one of transition-element.action)

(dynamic elements in the schema are retrieved. The name of the triggering event is obtained through *dynamic-transition.triggering-event*. The *action* is part of the properties of each *transition-element* in the association *transition-elements*)

Queries for office rapid prototyping

Queries in this design phase should allow to aggregate elements defined in the office schema according to the different agent types defined in the office schema. For instance:

(Q4.3) "finding out all the document types which are used by a certain agent type AG1"

retrieve d.name ($\forall$ d ϵ document, $\exists$ a ϵ action |
a.authorizations one of a.name = 'AG1' $\wedge$ (a.in one of i = d $\vee$ a.var one of v = d))

(properties of *action* - *authorizations, in, var* - are used to retrieve the required documents; in, out, and var are properties of actions specifying which are the input, output, and internal information sources of a given activity type)

Queries for architecture selection

In this phase, it is interesting to aggregate data according to different points of view to analyze specifications for choosing an architecture. For instance, to retrieve multimedia documents with properties of 'image' type, or to examine whether multimedia documents are only retrieved or also modified by office activities.

(Q4.4.) "retrieve actions in which there are editing operations in multimedia documents"

retrieve d.name, a.name ($\forall$ d ϵ document, $\forall$ a ϵ action |
d.property-list one of p.property-type = 'image' $\wedge$
a.values.act-entity = d $\wedge$ a.values.operations one of o = 'EDIT')

(property *operation* in *action.values* allows to retrieve action types which edit image documents)

5. CONCLUDING REMARKS

We presented a method and language constructs for querying a semantic model of the office with the specific goal of supporting design activities.

While the given semantic representation of a semantic conceptual model is tailored to the TODOS Conceptual Model, the given language constructs can be used for any semantic model. The ideas at the basis of semantic queries can be used in all areas where the design activity is complex and involves the definition of many different types and operations, such as, for instance, in the design of object oriented systems, of knowledge bases, of software systems. The solution proposed for the office conceptual design tool could be applied to such development environments, too. For instance, a formal definition of the software program concept could be given in terms of its components, its references and dependencies on other software objects, its variables, and so on. Then, it would be possible to ask queries about procedures utilizing some variables, in a certain way, with certain links to other procedures, and so on. The same technique could be applied to rule based systems, too. The rule concept could be defined in terms of accessed facts, triggered actions and conditions. Groups of rules with certain characteristics could then be retrieved, providing a powerful design tool to the designer, when defining new rules.

We are studying a user friendly interface, based on the semantic description of TCM, to provide an interactive menu based interface for asking queries about office schemas.

A first implementation of some types of semantic queries is being realized within the TODOS ESPRIT Project at Politecnico di Milano.

Acknowledgements

This work has been partially supported by the Commission of European Communities under ESPRIT Project N. 813 "Automatic Tools for Designing Office Systems" (TODOS).

We acknowledge the contribution to this work of all components of the Work Package 2 in TODOS: C. Rolland of University of Paris, J.R. Rames of Thomson Informatique Services, F. Barbic, M.G. Fugini, R. Maiocchi, and S. Pozzi of Politecnico di Milano.

M.G. Fugini and S. Ceri have provided useful remarks and suggestions on this work.

6. REFERENCES

/Alb 86/ Albano, A., Cardelli, L., and Orsini, R., 'Galileo: A strongly typed, interactive conceptual language', *ACM Trans. on Database Systems*, Vol. 10, N. 2., 1985.

/Ber 87/ Bertino, E. and Rabitti, F., 'Query processing based on complex object types', submitted for publication.

/Bor 82/ Borgida, A.T., Mylopoulos, J., and Wong, H.K.T., 'Methodological and computer aids for interactive information system development', in /Sch 82/.

/Bra 84a/ Bracchi, G. and Pernici, B. 'The design requirements of office systems', *ACM Trans. on Office Systems*, Vol. 2, N. 2, Apr. 1984.

/Bra 84b/ Bracchi, G. and Pernici, B., 'SOS: A conceptual model for office information systems", *Data Base*, Vol. 15, N. 2, Winter 1984.

/Bro 82/ Brodie, M.L. and Silva, E., 'Active and Passive Component Modeling: ACM/PCM', in Information Systems Design Methodologies: a Comparative Review (T.W. Olle, H.G. Sol, and A.A. Verrjin Stuart eds.), North Holland, 1982.

/Bro 84/ Brodie, M.L., Mylopoulos, J., and Schmidt, J.W., eds., On Conceptual Modeling, Springer-Verlag, New York, 1984.

/Dat 85/ Date, C.J., An introduction to database systems, Addison Wesley, 1985.

/Isr 84/ Israel, D.J. and Brachman, R.J., 'Some remarks on the semantics of representation languages', in /Bro 84/, pp. 119-143, 1984.

/Kon 82/ Konsynski, B.R., Bracker, L.C., and Bracker, W.E., 'A model for specification of office communications', *IEEE Trans. on Comm.*, Vol. COM-30, N. 1, Jan. 1982.

/Mai 87/ Maiocchi, R. and Pernici, B., 'Verification and refinement of office procedures', *IEEE Computer Society Symp. on Office Automation*, Gaithersburg, MD, Apr. 1987.

/Nav 86/ Navathe, S. and Kerschberg, L., 'Role of data dictionaries in information resource management', *Information and Management*, Vol. 10, pp. 21-46, 1986.

/Nie 87/ Nierstrasz, O.M., 'Hybrid - A language for programming with active objects', in /Tsi 87/, pp. 15-42, 1987.

/Nut 81/ Nutt, G.J. and Ricci, P.A., 'Quinault: an office modeling system', *Computer*, May 1981

/Per 86/ Pernici, B. and Vogel, W., 'An integrated approach to OIS development", *ESPRIT Technical Week '86*, Bruxelles, Sept. 1986.

/Per 87/ Pernici, B., Barbic, F., Fugini, M.G., Maiocchi, R., Rames, J.R., and Rolland, C., 'C-TODOS: An automatic tool for office system conceptual design', Politecnico di Milano, Electronics Dept., Rep. N. 87-15, 1987.

/Rot 85/ Roth, M.A., Korth, H.F., and Batory, D.S., 'SQL/NF: A query language for not 1NF relational databases', Department of Computer Sciences, Univ. of Texas at Austin, Austin, TX, TR-85-19, Sept. 1985.

/Shi 81/ Shipman, D.W., 'The functional data model and language DAPLEX", *ACM Trans. on Database Systems*, Vol. 6, N. 1, 1981.

/Sch 82/ Schneider, H.-J., Wasserman, A.I. (eds.), Automated Tools for Information Systems Design, North Holland, 1982.

/Tsi 87/ Tsichritzis, D. (ed.), Objects and Things, Centre Universitaire d'Informatique, Universite' de Geneve, Technical Report, Mar. 1987.

APPENDIX 1 - TODQUEL GRAMMAR

query $\longrightarrow$ **retrieve** select-part (type-variable-list '|' condition)

select-part $\longrightarrow$ type-part , type-list | type-part

type-part $\longrightarrow$ VARIABLE | VARIABLE **.name** | VARIABLE **.structure-spec**

type-list $\longrightarrow$ type-part , type-list | type-part

condition $\longrightarrow$ qualification $\wedge$ condition | (condition) | qualification | qualification $\vee$ condition | **not** condition

type-variable-list $\longrightarrow$ type-variable | type-variable , type-variable-list

type-variable $\longrightarrow$ $\exists$ VARIABLE ϵ type-name | $\forall$ VARIABLE ϵ type-name

qualification $\longrightarrow$ type-property-path predicate value | type-property-path predicate type-property-path
| type-property-path **contains** " struct-spec

predicate $\longrightarrow$ $=$ | $\neq$ | $\geq$ | ...

NUMBER $\longrightarrow$ *sequence of digits* | *

value $\longrightarrow$ STRING | NUMBER | ...

shorthand-list $\longrightarrow$ shorthand ; shorthand-list | shorthand

shorthand $\longrightarrow$ **through** NUMBER **references** | **through** NUMBER **references to** type-name |
hide supertypes from NUMBER **level up** | **hide supertypes from** NUMBER **level down** |
hide supertypes where qualification | **as** type-name |
include subtypes from NUMBER **level down** | association-member

type-property-path $\longrightarrow$ type-name property-path

property-path $\longrightarrow$ aggregation-member property-path | association-member property-path |
(shorthand-list) property-path | (shorthand-list) | property-name

aggregation-member $\longrightarrow$.

association-member $\longrightarrow$ **one of** | **all** | **1st** | **2nd** | **3rd** | NUMBER **th** | VARIABLE **th**

type-name $\longrightarrow$ STRING | * | VARIABLE

property-name $\longrightarrow$ STRING | type-property-path | *

struct-spec $\longrightarrow$ TSL-spec

TSL-spec $\longrightarrow$ property-name : TSL-construct

TSL-construct $\longrightarrow$ **aggregation-of** { TSL-property-list } | **association-of** TSL-spec |
reference-to type-name | domain

TSL-property-list $\longrightarrow$ TSL-spec | TSL-spec ; TSL-property-list

domain $\longrightarrow$ * | INTEGER | CHAR | ...