Faculteit der Economische Wetenschappen en Econometrie

Serie Research Memoranda

MEMOSPEL - A Generic MEta MOdel SPEcification Language for A SELC Repository

G.T. Vinig
S.J. Fischer
R.J. Groenendal

Research Memorandum 1992-25
juli 1992

vrije Universiteit *amsterdam*

MEMOSPEL - A Generic MEta MOdel SPEcification Language for A SELC Repository

BY

G.T. Vinig, S.J. Fischer, R.J. Groenendal[1]

Vrije Universiteit Amsterdam
Faculty of Economics
Department of Information Systems Studies
CASE Research Lab

TEL: +31-(0)20-5464606 FAX: +31-(0)20-6462645 EMAIL: TVINIG@SARA.NL.BITNET

Abstract

In this article we present a meta model specification language - MeMoSpeL, and its corresponding diagramming technique that we have developed based on the Object-Property-Role-Relationship (OPRR) technique. We demonstrate the use of MeMoSpeL in defining a meta model that supports Data Flow Diagram (DFD), Structured Chart (SC), Entity Relationship (E-R) and project management techniques. The MeMoSpeL specifications are used to generate SQL schema and to implement the repository in a RDBMS. We developed MeMospeL and OPRR+ as part of our Software Engineering Life Cycle (SELC) approach to CASE based systems development.

Introduction

The Software Engineering Life Cycle (SELC) approach, is an alternative to the Waterfall approach to software development. SELC is based on the use of CASE technology which brings automation support to software development [VINI91].

The assumption on which the SELC approach is based on is, the need to examine and redesign the development process, taking into consideration the available information technology - CASE, not only automates existing development process.

---

[1]Computer Sciences department

This is based on the notion of Business Process Redesign, or Re-engineering (BPR) as described by Hammer [HAMM90] that argues for "*Use computers to redesign not just automate existing business processes*" and Devenport & Short [DEVE90] that claim that "*Information technology should be viewed as more than an automating or mechanizing force; it can fundamentally reshape the way business is done*". We have implemented the BPR principles for the information systems development process when developing the SELC approach.

We have introduced a repository based development process using some new phases. The following table describes the phases and their deliverables:

Phase	**Deliverable**
Business Process Redesign (BPR)	Improved or New business process to be automated
Conceptual Modeling	Business process model, Business system model
Executable Model	Business system working model
System Engineering	Production Application
Reverse Engineering	Business system model

Table 1: SELC phases and deliverables

In addition we have introduced two cross life cycle activities: Configuration Management and Project Management which are also repository based. The following diagram describes the SELC approach.

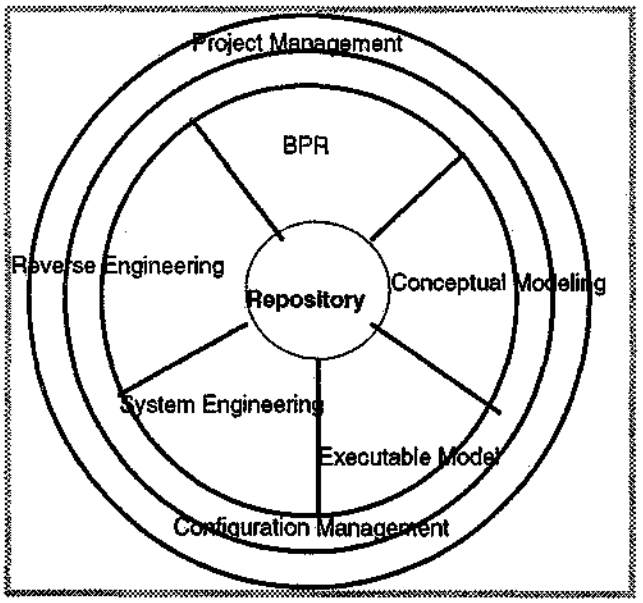

Figure 1 - The SELC approach

A repository is a core component for CASE based development in general, and in the SELC approach. In order to implement a repository (SELC or other), we need to define its meta model which describes the schema, semantics and structure.

In the next section we discuss in short, the meta models and the need for a meta model specification language.

Meta models

A CASE repository consists of three layers:

1. A meta-meta model layer
2. meta model layer
3. System under development (SUD) layer

Meta-meta model defines the meta model layer that in fact is the definition of the modelling technique used. It defines objects, properties, relationships and possible constructs of the meta model layer.

Meta model layer is an instance of the meta-meta model and defines the modelling techniques used. It defines the elements of the SUD model like process, store and module.

The third layer - model of the SUD is a view of the reality in the form of a E-R diagram, DFD diagram or other modellingtechnique used.

The idea of META (language, data, model, system) is a natural way to describe generic characteristics of a system. The notions that are used here are as described by [GORE88], from which we concentrate on the meta model.

- Meta-language is a language to define language
- meta-data is data for defining data
- **meta-model is a model for defining models**
- meta-system is a system for defining systems

A meta model for a CASE repository should allow the definition of various modelling techniques and tools used in the process of information systems development. In order to allow such a definition, a meta model must be able to afford syntax and semantics to be used in definition of the meta model. A meta model for CASE repository should have the following characteristics:

- Contain a schema (i.e. E-R model)
- Support rigorous specification of the schema (i.e. language)
- Include abstraction mechanisms of aggregation, generalization and classification
- Ability to represent system design objects

- Generic - allow the definition of multiple methods, techniques and tools

Meta system approach for information systems development is not new. [DEME82, GORE88]. The emphasis on a CASE repository as a central and essential component of a CASE environment, has focused attention on the way of modelling and representing repository meta data.

Modelling technique - OPRR+ and MeMoSpeL

The most used modelling technique for database design and therefore for repository design and meta model development, is the Entity-Relationship model [CHEN76]. Many research environments [KOTT84], proposed standards [IRDS] and commercial environments [IBM's AD/Cycle] use the entity relationship model to describe their meta model. Entity relationship model provides simple yet expressive power and semantics. There is no accepted standard for ER specification language. All of them are graphical using the same basic notion: a rectangle represents an entity and a line or a diamond between entities represents relationship. There were some efforts to propose textual ER specification language, but none gained acceptance.

Since the introduction of the entity-relationship model, few extensions were proposed. The different variants can be classified by the number of entities participating in a relationship and if attributes are allowed for the relationship. There are ER techniques that support only binary relationship, while other support n-array relationship.

We find that Object-Property-Role-Relationship (OPRR) modelling technique [WELE89] more adequate for specifying a generic meta model. We expanded OPRR with the notion of CLASS, uniqueness and optionality so that constraints in the developed meta model can be better expressed. We call our diagramming version OPRR+, and the specification language **Me**ta-**Mo**del-**Spe**cification-**L**anguage or MeMoSpeL. OPRR+ diagram can be specified by MeMoSpeL and MeMoSpeL specifications can be written as an OPRR+ diagram. This solves a major problem with ER model which uses only a graphical specification for an ER model.

It is difficult to verify or test the graphical model for consistency. In the following section we describe OPRR+ diagramming notation and some examples for OPRR+ diagram with the corresponding MeMoSpeL specification language.

We have defined the following elements of OPRR+ diagram, the **bold** are extensions to OPRR:

- **class**
- object

- relationship
- role, **mandatory/optional**
- property, **mandatory/optional**
- **unique** property, **mandatory/optional**
- **validation check**

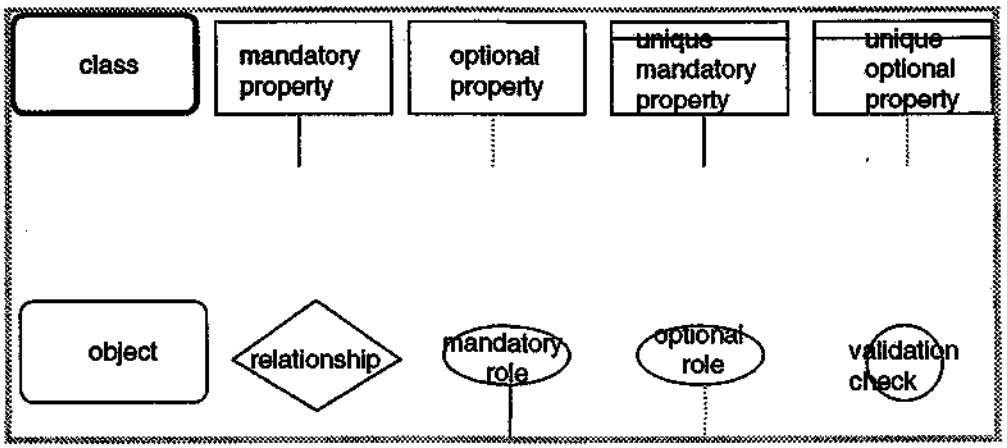

Figure 2. Diagramming notation for OPRR+

Class - We extended OPRR with the notion of class to make it more convenient and efficient to deal with meta models. A class can have one or more object types, which has a set of properties associated with it. The class and its properties are defined once and shared by all object types that belong to that class. The use of class provides for improved consistency control. Without the use of class property that is common to multiple object types has to be defined for each one on them. If the property is changed, it should be changed in every object type in order to have a consistent model. Using the class we need to define the change only once, which is at the class property level. This makes consistency control more convenient than changing a property across many objects in the model.

Mandatory and optional property - A property of a class, object, relationship and role can be mandatory but must have a value or optional where no value is defined for that property. Use of mandatory and optional properties makes the model easier to read and understand.

Unique and Non-unique property - A unique property is (uniquely) identifying an instance of the object (or relationship).

Validation - Validation is enabling us to define multiple modelling techniques using OPRR. When defining a relationship-type we also define roles. Those roles could be fulfilled by objects, and validation can define which objects could fulfill that role. Validation for type checking can be used so that the consistency of the model is checked. For a given relationship-type we can define multiple validations, which can be very useful for defining meta model of a given modelling technique. Defining meta model for Data Flow Diagram (DFD) technique illustrates the use of validation:

One of the rules in DFD is that a data flow can connect process-process, process-store or process-external. Data flow between store-store or store-external is not valid. The relationship-type *data-flow* is assigned multiple validations which makes it easy to model the *data-flow* rules of the DFD technique. The validation we define are: <OBJECT:process, OBJECT:process>, <OBJECT:process, OBJECT:store>, <OBJECT:process, OBJECT:external>, <OBJECT:store, OBJECT:process>, <OBJECT:external, OBJECT:process>. By using these validations we can define a meta model for DFD that contain the DFD modellingrules. When using validation with class we reduce the complexity of the model by reducing the number of validations we have to define. If we have a binary relationship-type (relationship-type between two objects), where 8 object-types can fulfill the role in any combination, we have to define 36 validations. By using a class that contains those 8 object types we need only one validation: <CLASS:name, CLASS:name>.

Mandatory and optional role - This extension makes it possible to model situations where a role of a relationship type can not be fulfilled. In modellingStructured Chart design technique we use the relationship-type *invokes* with the roles *invoker*, *invoked*, *passed* and *returned* to model call between modules where a variable is being passed. Using the validation <OBJECT:module, OBJECT:module, OBJECT:variable, OBJECT:variable> we define that a module can call other modules and pass (or return) a variable. It can also be that there are no variables passed in the call. By defining that the roles *passed* and *returned* are optional, that rule can then be defined in OPRR based meta model for structured chart.

Uniqueness constraint - We define uniqueness constraints only for mandatory roles of a relationship-type. Uniqueness constrains means that only given objects can fulfill a role only once. Using the example of Structured Chart mentioned above, if we define a uniqueness constraint over the roles of invoker and invoked. We are actually defining that a given module can call other given modules only once.

Cardinality constraints - Cardinality constraints adds expressiveness power to a OPRR diagram. Using cardinality constraints we define the number of objects with which a given object can be associated in a given relationship.

Object type can have zero or more properties. A property can be mandatory or optional. A mandatory property can be unique or non-unique.

Relationship type - Definition of a relationship-type is dependent on the number of its validations. A relationship-type with one validation is associated via a role with two or more performers (object or a class). These performers together define the validation. Relationship-type has a property set defined for it. Each relationship-type must have at least two mandatory roles.

In the following section we discuss examples of OPRR+ diagrams and their corresponding MeMoSpeL specification. The diagrams are based on the notion given in figure 1. The specifications are given in the form of MeMoSpeL[2] statements. We use statements to define object-type, class and relationship-type. The General form of MeMoSpeL statements are given in the following table:

Object-type definition:

OBJECT:name = [property DATA-TYPE mandatoryloptional uniquelnot-unique
property2, DATA-TYPE, mandatoryloptional, uniquelnot-unique
...];

Class definition:

OBJECT:O1 = [];
OBJECT:O2 = [];
CLASS:name = [{O1, O2},
property, DATA-TYPE, mandatoryloptional, uniquelnot-unique
...];

Relationship-type definition:

RELATIONSHIP:name = [[role_name, madatoryloptional, cardinality],
(*uniqueness constraints*) [<role_name, role_name>],
(*validation*) [<CLASS:name, OBJECT:name, OBJECT:name>],
[relationship_property, DATA-TYPE,
mandatoryloptional, uniquelnot-unique],
[role_name, role_property, mandatoryloptional]];

Table 2. MeMoSpeL statements

The following diagram demonstrates the OPRR+ diagramming technique we use. The *class AUTO* has two *object-types - SEDAN and STATION*. The class AUTO has a *non-unique, mandatory property MAKE* of type CHAR and an *optional non-unique property METALIC_COLOR* of type CHAR. It has a *unique mandatory property SERIAL#* of type CHAR and *optional unique property CAR_PHONE#* of

[2]The syntax of MeMoSpeL is given in appendix A

type INTEGER. The objects SEDAN and STATION have also the properties of the class AUTO. (In addition to specific properties that each object might have).

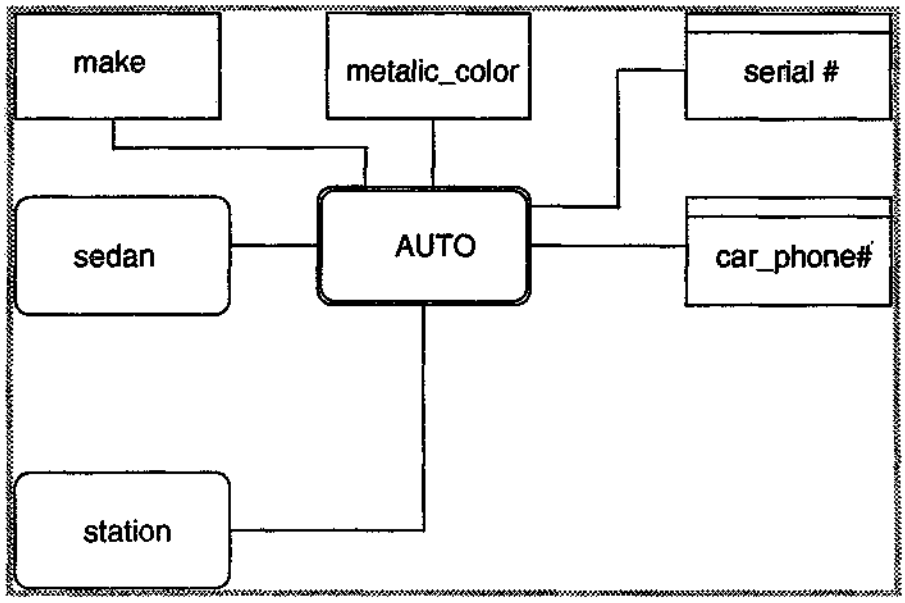

Figure 3. OPRR+ class diagram

The corresponding MeMoSpeL specification of the class diagram is:

```
OBJECT:SEDAN = [];
OBJECT:STATION = [];

CLASS:AUTO = [    {SEDAN, STATION}
                  [ MAKE CHAR mandatory  non-unique,
                  METALIC_COLOR CHAR optional non-unique,
                  SERIAL# INTEGER mandatory unique,
                   AUTO_PHONE#  INTEGER  optional  ,unique  ]];
```

In class definition we define both, objects belonging to that class and the class properties itself.

Meta-meta model

We have defined a generic meta-meta model that provides for the definition of a flexible meta model for a SELC repository, such that the meta model can contain multiple techniques represented by OPRR+ diagram and MeMoSpeL specification. By defining the a meta-meta model we provide the SELC repository implementator with the flexibility to accommodate multiple techniques by specifying their meta model in OPRR+ / MeMoSpel.

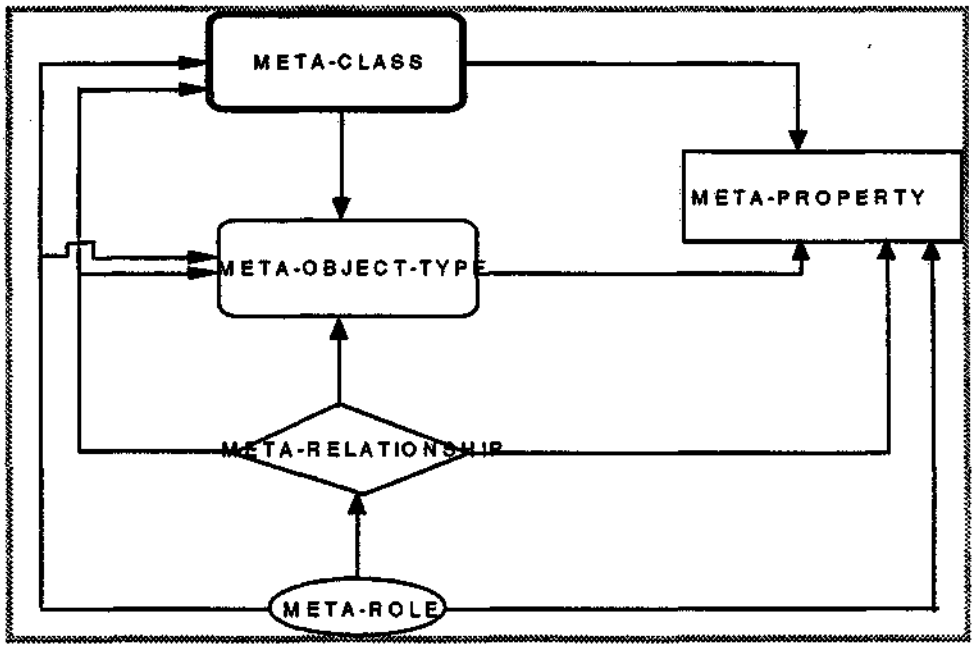

Figure 4. OPRR diagram of a SELC meta-meta model

The meta-meta model we present is actually a model of the modellingtechnique used to define the meta model. Meta models defined for different methods are instances of the meta meta model. In our case the meta meta model defines the OPRR modelling technique with the extensions we have added. A meta model based on this meta meta model can contain *classes, objects, properties, relationships* and *roles*. These elements are based on the meta meta models' *meta-class, meta-object, meta-property, meta-relationship* and *meta-role*.

Each element in the meta-meat model has the property name. In addition to the property name, *meta-property* has the property mandatory or optional (moro) and unique or non-unique (uorn). The *role* element has in addition to property name , the property moro.

In a meta model there are relationships between the elements. A class can have one or more objects associated with it. This relationship is an instance of the relationship *contains* in the meta-meta model.

Defining meta model - Implementation example

The meta-meta model gives the flexibility to define meta models based on multiple methods and techniques. The meta model provides the flexibility to model any information system under development (SUD). We present the use of OPRR+ and MeMoSpeL specifications for defining meta models of representative techniques that gained general acceptance and are widely spread among information systems developers.

The example we use is a meta model for Data Flow Diagram (DFD) which is a common process modelling technique originated by Yourdon & DeMacro [YOUR89] and is used by the Structured Analysis method. Some extensions to the DFD were suggested since its introduction, the most known are those of Gane & Sarson. We use the original notation here as presented by Yourdon/DeMacro. Data flow diagram contains the following four elements:

- Process (i.e. VALIDATE-ORDER)
- Data Store (i.e. INVENTORY)
- External (i.e CLIENT)
- Data flow (i.e. ORDER)

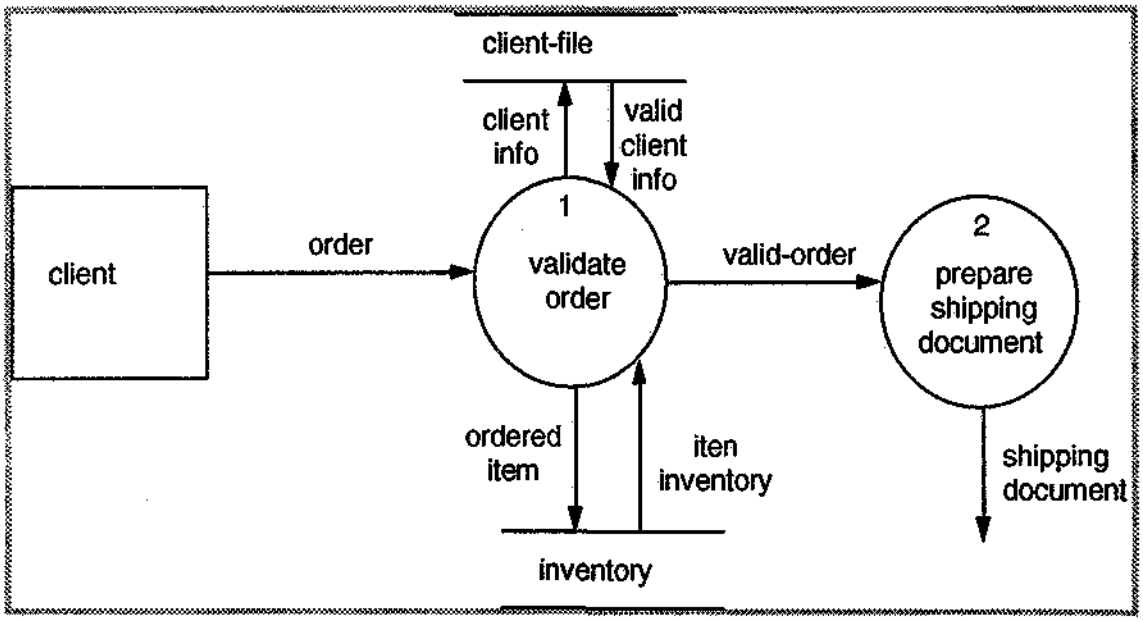

Figure 5. A simple data flow diagram

The following diagram is a meta model of a generic data flow diagram using OPRR+.

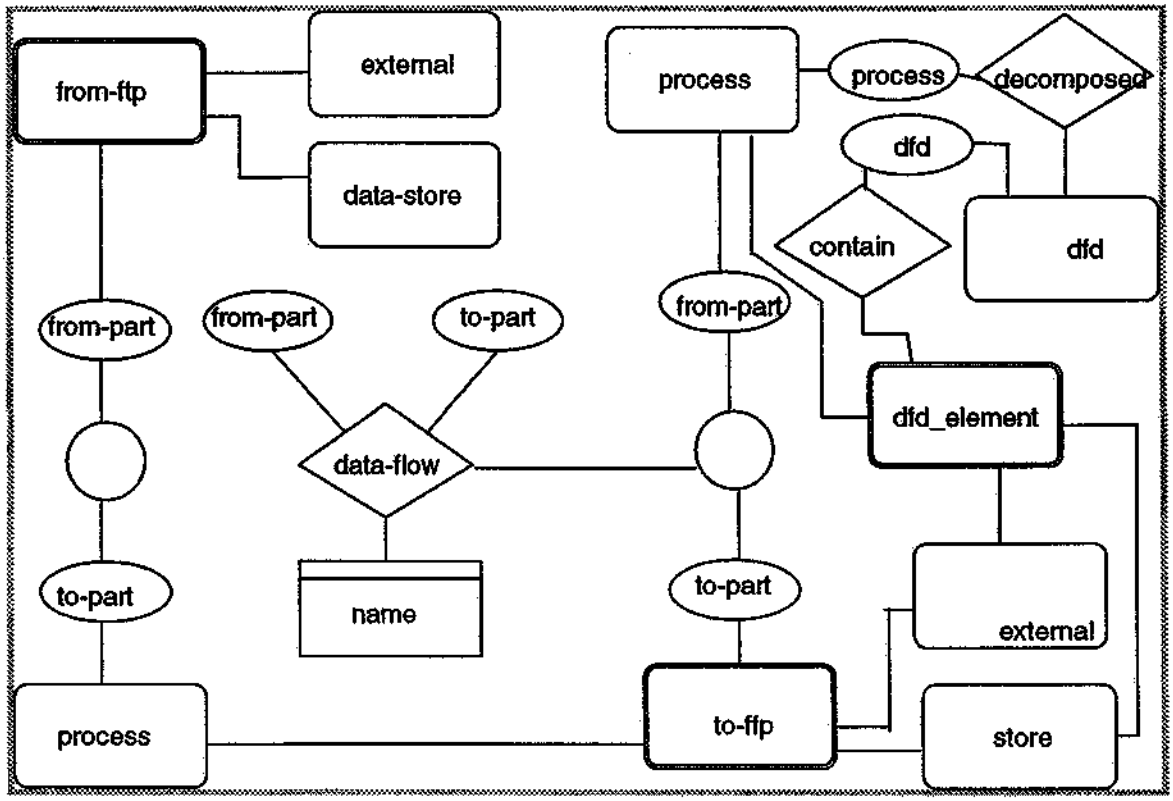

Figure 6. Meta model of data flow diagram

The corresponding MeMoSpeL specification for that meta model are:

```
OBJECT: dfd          = [ name        CHAR(16) mandatory unique    ];

OBJECT: process      = [ p_number    CHAR(8)  mandatory unique,
                         description CHAR(32) mandatory non-unique ];

OBJECT: external     = [ name        CHAR(16) mandatory unique    ];

OBJECT: store        = [ name        CHAR(16) mandatory unique    ];

CLASS: to_ffp        = [ { process, store, external }, [ ] ];

CLASS: from_ftp      = [ { external, store }, [ ] ];

CLASS: dfd_element = [ { process, external, store }, [] };

RELATIONSHIP: data_flow = [ [ from_part mandatory 0..,
                              to_part mandatory 0.. ],
                            [ ],
                             [ < OBJECT: process, CLASS: to_ffp >,
                            < CLASS: from_ftp, OBJECT: process > ],
                             [ name CHAR(24) mandatory non-unique ],
```

```
[ ] ];

RELATIONSHIP: dfd_contains   =  [ [ dfd   mandatory 1..,
                                  element mandatory 0.. ],
                                  [ ],
                                  [ < OBJECT: dfd, CLASS: dfd_element
                                        > ],
                                  [ ],
                                  [ ] ];

RELATIONSHIP: decomposed_into = [ [ process mandatory 0..1,
                                   dfd     mandatory 0..1 ],
                                  [ ],
                                  [ < OBJECT: process, OBJECT: dfd > ],
                                  [ ],
                                  [ ] ];
```

The meta model represents all the data flow diagram elements and the rules used in creating a data flow diagram. Process, data-store and external are represented by the objects *process, data-store, external* . The object *dfd* represents the data flow diagram itself. The data flow element of the DFD is represented by the relationship type *data_flow.* To model the DFD rules for data flow (process-store etc.) we use two classes: *to_ffp* (flow from process) and *from_ftp* (flow to process) and validation on the relationship *data_flow* which gives the possible flows within a DFD. The relationship *decomposed* represents the decomposition possibility within a DFD. The relationship *contains* and the class *dfd_element* represents that a DFD can contain the PROCESS, STORE and EXTERNAL elements.

The data flow diagramming rules modeled are:

- A data flow must have at least a name and must have source and destination. In a decomposed DFD one of the two can be in an other level, but there are always two DFD elements connected by a data flow.

- A data-store or an external cannot be directly connected to other data store or external. They must be connected via a process.

- A process can be decomposed into a lower level data flow diagram

In working out our SELC approach we have used MeMoSpeL to define meta model of a Structured Chart [CONS79] and Entity-Relationship [CHEN76] techniques, in addition to DFD.

The SELC approach defines Project Management as a cross life cycle activity, which like conceptual modellingtechniques (E-R, DFD) is based on the repository. In the following example we use OPRR+ and MeMoSpeL to define a meta model for project management tool[3].

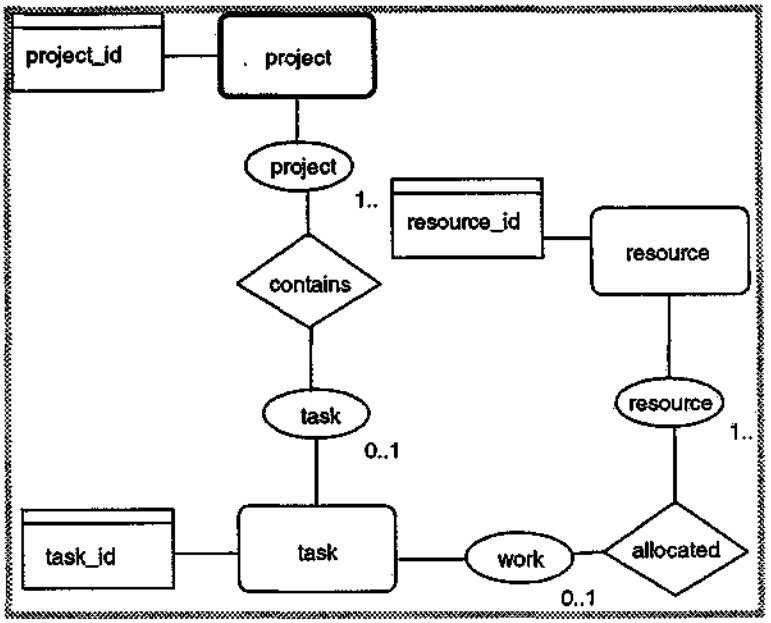

Figure 7. Meta model for project management

The corresponding MeMoSpeL specifications for project management:

OBJECT: resource = [r_id INTEGER mandatory unique,
r_name CHAR(15) mandatory non-unique,
unit_cost INTEGER mandatory non-unique];

OBJECT: task = [task_id INTEGER mandatory unique,
task_name CHAR(15) mandatory non-unique,
pre INTEGER mandatory unique,
priority CHAR(8) mandatory non-unique
duration INTEGER optional non-unique];

CLASS: project = [{resource, task},
[p_id INTEGER mandatory unique,
p_name CHAR(15) mandatory non-unique,
mng_name CHAR(15) mandatory non-unique]];

[3]We have used Microsoft Project as an example

```
RELATIONSHIP: contains = [[project  mandatory 1..,
                          task  mandatory 1..],
                          [],
                          [<CLASS:project, OBJECT:task>],
                          [],
                          [] ];

RELATIONSHIP: allocated = [[resource  mandatory 1..,
                          task  mandatory 1..],
                          [],
                          [<OBJECT:resource, OBJECT:task>],
                          [],
                          [] ];
```

We have shown the use of OPRR+ diagramming techniques and its corresponding specification language for the definition of meta model for some commonly used techniques for conceptual modellingof information systems, as well as for defining meta model for project management. Using textual specification (in addition to the semantics represented by the diagramming technique) makes it possible to automatically convert the specification into technology dependent implementation (i.e. RDBMS).

Implementing A SELC Repository using MeMoSpeL

We have used MeMoSpel to implement a SELC repository as a RDBMS application. Using specification language, it is possible to automatically convert MeMoSpeL specification into SQL schema that can be run against the RDBMS to implement the repository.

We have tested the implementation on ORACLE in a UNIX (IBM AIX) environment and on SQLBase - a RDBMS that runs in a PC environment under Windows.

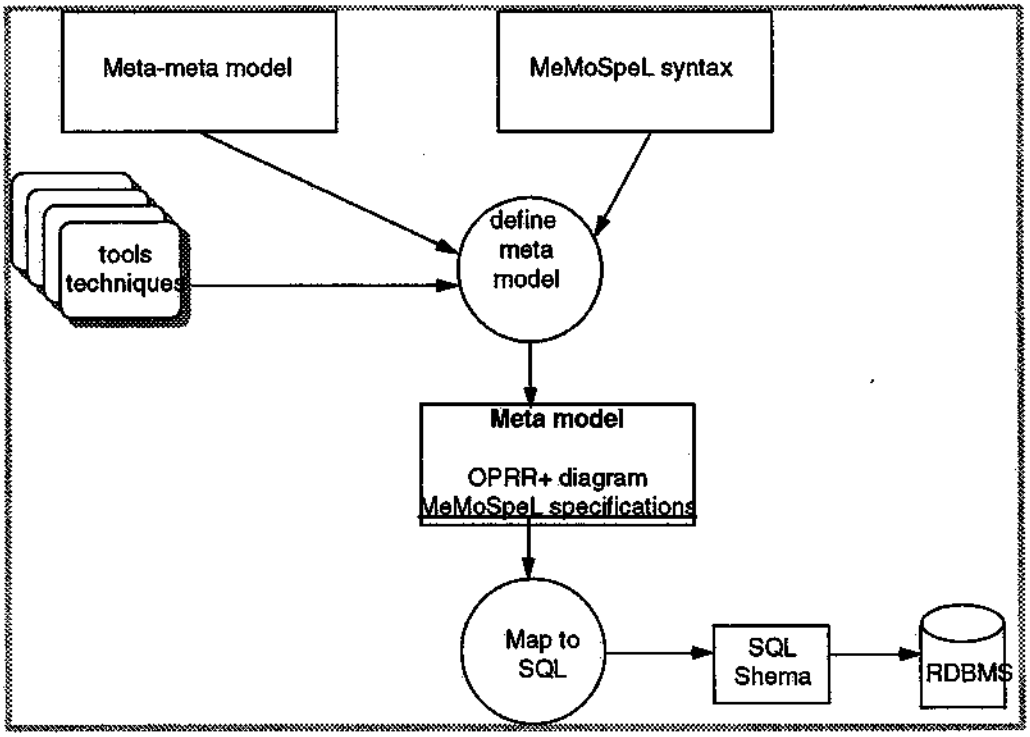

Figure 8. SELC repository implementation

The mapping from MeMoSpeL to SQL is done using the following mechanism:

- Classes are represented in term of their objects, where every object inherits the properties of the class in which it is a member. The following example demonstrates this.

OBJECT: sedan= [no_of_doors INTEGER mandatory non-unique];

OBJECT:station = [no_of_seats INTEGER optional non-unique];

CLASS:car = [make CHAR(12) mandatory unique];

Representing the class in term of its member objects where objects inherit the property of the class results in

OBJECT: sedan= [no_of_doors INTEGER mandatory non-unique,
make CHAR(12) mandatory unique];

OBJECT:station = [no_of_seats INTEGER optional non-unique,
make CHAR(12) mandatory unique];

and in definition of a relationship-type where the class car participates

RELATIONSHIP:rel = [[role1 mandatory 0..,

```
                                role2 madatory 1..],
                                [],
                                [<CLASS:car, OBJECT:x>]
                                [],
                                [] ];
```

we get

```
RELATIONSHIP:rel = [[ role1 mandatory 0..,
                      role2 mandatory 1..],
                      [],
                      [<OBJECT:sedan, OBJECT:x>,
                       <ONJECT:station, OBJECT:x>  ]
                      [],
                      [] ];
```

- Meta model tables are created in the RDBMS.
- Meta model tables are populated with the meta data .
- Tables are created for object-type and relationship-type. The table for object-type is populated with the object-type properties. Two tables are created for relationship-type. One contains relationship properties, role properties and the second table in which the validation of the relationship type is kept.

This way of implementing the meta model makes it easier to use any RDBMS that support SQL for implementing our generic repository.

Conclusions

We have presented in this article a meta model specification language - MeMoSpeL, and its corresponding diagramming technique - OPRR+, for definition of meta models. It was used to define and implement a SELC repository. We have demonstrated its use by defining meta models of some commonly used modelling techniques - DFD, E-R, SC and project management.

The use of specification language facilitates automatic conversion of the specification (into SQL schema), and enables consistency checking and verification of the model, a task which is difficult to do on a diagramming based specification. The importance of MeMoSpeL is to enable CASE developers and CASE users to define an environment that supports methodology integration by specifying the meta models of the techniques they choose to implement. In future, we plan to provide automatic conversion from a OPRR+ diagram to MeMoSpeL specifications.

APPENDIX A - BNF definition of MeMoSpeL syntax

<memospel-specification> ::= (memospel-statement ;)*

<memospel_statement> ::= <object-type-definition>
<class-definition> |
<relationship-type-definition>

<object-type-definition> ::= OBJECT : <identifier> = [
<property-list>]

<identifier> ::= <alpha> (<alpha-num-plus>)*

<alpha> ::= a | b . . . | z | A | B . . . | Z

<alpha-num-plus> ::= <alpha>|<digit> |_ |. |) |(

<property-list> ::= (<propery-definition>)*

<property-definition> ::= <identifier> <data-type> <moro>
<uorn>

<data-type> ::= INTEGER | SMALLINT | FLOAT |
CHAR (<size>)

<size> ::= <notzero> (<digit>)*

<not-zero> ::= 1|2|3|4|5|6|7|8|9

<digit> ::= <not-zero>|0

<moro> ::= mandatory | optional

<uorn> ::= unique | non-unique

<class-definition> ::= CLASS:<identifier>=[{<class-
member>}],
[<property-list>]]

<class-member> ::= (identifier>)*

<relationship-type-definition> ::= RELATIONSHIP : <identifier> = [[
<role-list>], [<uniqueness-constraint-
list>]
[<validation-list>]
[<property-list>]
[<role-property-list>]]

<role-list> ::= (<identifier><moro><cardinality>)*

<uniqueness-constraint-list> ::= (<uniqueness-constraint>)*

<uniqueness-constraint> ::= <(<identifier>)*>

<validation-list> ::= <validation>*

<validation> ::= <(<object>)*>

<object> ::= OBJECT:<identifier>

<class> ::= CLASS:<identifier>

<role-property-list> ::= <identifier><identifier>
<data-type><moro>

<cardinality> ::= 0..1 | 1 | 0..| 1..

BIBLIOGRAPHY

1. **ANSI88** ANSI, Information Resource Dictionary System. American National Standard X3.138 1988.

2. **BALZ83** Balzer R., Cheatham T.E., Green C., Software technology in the 90's: Using a new paradigm. Computer, November 1983, 39-45.

3. **BASI91** Basili V.R., Musa J.D. The Future Engineering of Software: A Management Perspective. IEEE Computer, Sept. 1991, 90-96.

4. **BERS91** Bernsoff E.H., Davis A.M. , Impact of Life Cycle Model on Software Configuration Management. Communication of the ACM, Vol. 34, No. 8, 1991, 104-118.

5. **BOEH82** Boehm B.W., A Spiral Model of Software Development and Enhancement. IEEE Computer, May 1988.

6. **BROO86** Brooks F.P., No Siver Bullet: Essence and Accidents of Software Engineering. Computer Vol. 20, No. 4 1987, 10-19.

7. **BROW87** Brown A., Integrated Project Support Environments. Information and Management ,15 1987, 125-134.

8. **CHEN76** Chen P., The Entity-Relationship Model: Towards a Unified View of Data. ACM Trans. Database Systems, Jan. 1976, 9-36.

9. **CHEN91** Chen M., Sibley E.H., Using a CASE Based Repository for Systems Integration. IEEE Proc. of the Hawaii Conf. on Sys., January 1991, 578-587.

10. **CONS76** Constantine L.L., Yourdon E., Structured Design. Prentice-Hall

11. **CORB91** Corbin D.S., Establishing The Software Development Environment Journal of Systems Management September 1991 28-33

12. **DEME82** Demetrovics J., Knuth E., Rado P., Specification Metasystems.

13. **DEVE90** Devenport H.T., Short J.E., The New Industrial Engineering: Information Technology and Business Process Redesign. Sloan Management Review, Summer 1990, P. 11-27.

14. **DOLK87** Dolk D.R., Kirsch R.A., A Relational Information Resource Dictionary System. Communication of the ACM, Vol. 30, No. 1 1987, 48-61.

15. **ECMA90** European Computer Manufacturer Associaltion, Portable Common Tool Environment (PCTE) EMCA, December 1990.

16. **EIA/IS91** Electronic Industries Association, CDIF - Framework for Modellingand Extensibility EIA, July 1991.

17. **GORE88** Gorenson G.P., Trembly J.P., McAllister A.J., The Metaview System for Many Specification Environments. IEEE Software, March 1988, 30-38.

18. HAMM90 Hammer M., Reengineering Work: Don't Automate, Obliterate. Harvard Business Review, July-August 1990. P. 104-112.

19. **HAZZ90** Hazzah A., IBM's Information Model Rosetta Stone for Developers? Software Magazine, July 1990 ,87-96.

20. **HIEN85** Hein K.P., Information system model and architecture generator. IBM Systems Journal, Vol. 24, No. 3 1985, 213-235.

21. **HITC88** Hitchcock P., A Database View of the PCTE and ASPECT in Software Engineering Environments. John Wiley & Sons, 1988, 37-49.

22. **HSU91** Hsu C., Bouziane M., Rattner L., Yee L., Information Resource Management System in Heterogeneous, Distributed Environments: A Metadata ... IEEE Transactions on Software Engineer, 17, 1991, 604-625.

23. **IBM90** IBM, Overview of Repository Manager Supplied E-R Model. IBM, March 1990.

24. **LAMS88** Lamsweerde A., Delcourt B., Delor E., Generic Life Cycle Support in the ALMA Environment IEEE Transactions on Software Eng., 14, 1988, 720-741.

25. **OLLE88** Olle T.W., ed, Computerized Assistance During the Information Systems Life Cycle. Elsevier Science Publishing, 1988.

26. **RINE91** Rine D.C., An Approach for Developing an Integrated ModellingEnvironment: A Case Investigation. Journal of Software Maintenance, Vol. 3, No. 2, 1991, 65-83.

27. **SAGA90** Sagawa J.M., Repository Manager Technology. IBM Systems Journal, Vol. 29 ,No. 2, 1990, 298-327.

28. **SHAW90** Shaw M., Prospect for an Engineering Discipline of Software. IEEE Software, November 1990, 1299-1315.

29. **SHOR91** Short K.W., Methodology Integration: Evolution of Information Engineering. Information and Software Technology, Vol. 33, No. 9, 1991, 720-732.

30. **VINI91** Vinig G.T., Achterberg J.S., CASE Technology - from Software Development Life Cycle (SDLC) to Software Engineering Life Cycle ((SELC). Research Memorendum 1991-66, Vrije Universiteit Amsterdam, 1991.

31. **WELK89** Welke R.J., Forte G., Meta Systems on Meta Models. CASE Outlook, 2, 1989, 35-45.

32. **YOUR89** Yourdon E., Modern Structured Analysis. Yourdon Press, 1989.