FEDERATED CASE ENVIRONMENT

S. Papahristos W. A. Gray
Department Of Computing Mathematics
University Of Wales College Of Cardiff
P.O BOX 916, Cardiff CF2 4YN, U.K

ABSTRACT

Although systems development methodologies and their supporting CASE tools were designed to address productivity and quality issues of systems development, it is now widely accepted that they have not totally succeeded in achieving their stated goals. They have been criticised for their lack of flexibility and potential inefficiencies. One approach to overcome the weaknesses of existing methodologies is to create an open methodology environment in which different methodologies are combined, in a federated architecture [12], to contribute to a system's development. This implies that the supporting CASE tools of the combined methodologies must also be allowed to co-exist in a way that enables reusability of information collected through one CASE tool by another. In [25] we described how a global Data Dictionary System can be made to act as the co-existence mechanism for different methodologies and identified the concepts which must be included in the data dictionary's data model in order to play such role. In this paper the development of a prototype Federated CASE Environment(FCE) is described. It has an open methodology architecture in that information collected through one CASE tool can be used by other CASE tools. This is achieved by exporting to a global data dictionary information required in other CASE tools and importing information from the global data dictionary if a CASE tool is to be used externally. This allows users developing a system in one methodology to have access to techniques in another methodology. The prototype FCE's global data dictionary is based on the ISO's Information Resource Dictionary System(IRDS). The previously identified data model concepts have been realised in this data dictionary which has been extended to support the physical concepts needed to describe a graphical diagram. These physical concepts represent the structure of a diagram in a representation that is independent of any graphical display system. This allows this information to be transferred to different display systems. The efficacy of the method has been demonstrated in a prototype environment by linking CASE tools supporting the SSADM methodology(AUTOMATE PLUS) with CASE tools supporting the YOURDON methodology(ADT). This environment is capable of incorporating other CASE tools and methodologies.

1. INTRODUCTION

There is now a strong consensus that productivity and quality in the development of complex systems can only be ensured by applying an engineeringlike form of discipline in its production. Such a discipline is called Software Engineering [27]. Within the context of software engineering one of the main facilities which can be used to improve the process of systems development is a systems development methodology [13], [22].

A systems development methodology supports the life-cycle activities by providing [28]:

(1) a method that specifies the activities performed, control functions exercised and documentation to be produced,
(2) a model or conceptual framework that is the basis for creating a particular target system instance,
(3) a system development language that is to be used to represent a particular target system,
(4) tools manual and/or computer aided that are used in creating a target system instance.

An integrated set of software tools automating the application of a methodology is known as a CASE tool [26]. A systems development methodology may have more than one CASE tool supporting it.

Systems development methodologies have a long pedigree. In the 1960s techniques developed prior to the computer era were modified and used to design computer based systems. System flowcharts, flow diagrams, IPC and MAP [4] were the first computer oriented techniques. Since then substantial effort has been expended on developing better and more advanced methodologies. However no single methodology vendor has the complete solution to the varying industry requirements for systems development methodologies, nor is this likely to be the case. Thus one step which many developers have taken is to combine techniques from different methodologies, and their supporting CASE tools, in order to take advantage of the strengths and overcome the weaknesses of existing methodologies [29].

This paper describes how a data dictionary system can form the basis for an open methodology environment in which techniques from different approaches are used in the development process. A prototype environment called Federated CASE Environment (FCE) has been defined, and partially implemented, which will allow this approach to be evaluated.

1.1 OPEN METHODOLOGY ENVIRONMENT

In spite of the recent advances made in systems development methodologies there has been some uneasiness about methodology choice and use, and in some cases their practical validity. This is due to a number of limitations that characterise most of the current methodologies. The most serious of these are:

(1) explicitly prescriptive,
(2) problem specific,
(3) inadequate support for all phases of the life-cycle.

Most of the available methodologies are explicitly prescriptive of what should be done, consisting of a recommended collection of phases, procedures, rules and techniques to be applied in a given order. However in most cases it is difficult to fit a project into such a fixed, rigid framework. Further by forcing the developer to use a particular methodology it can constrain his/her ability to select the most appropriate problem representation and can prevent the use of knowledge and experience gained from previous similar projects.

Since existing methodologies have been developed with the intention of building systems of a specific type, they force a user of the methodology to an early decision on the type, structure and scope of the information system being developed. Although such decisions may result in a quicker and more standardised systems development, it makes the system under development relatively inflexible, and restricts exploration of alternatives.

Particular methodologies provide limited facilities for the complete life-cycle in that most of them provide effective support, to the point of overkill, for one or a limited number of stages, of the systems development life-cycle, while treating the other stages inadequately. For example structured methodologies such as SSADM [23] and YOURDON [30] focus on the analysis and design phases while others such as JACKSON [17] are targeted at the design and implementation phases.

These limitations of systems development methodologies could be overcome by developing an open methodology environment where several methodologies, and their supporting CASE tools, are allowed to co-exist with their techniques being used in a project development. Such a development environment can:

(1) Result in a highly flexible methodology being used which is capable of blending with other organisational procedures.
(2) Facilitate the development of procedural pattern in response to a system's contextual changes.
(3) Allow choice for the system developer in selecting the most appropriate problem representation or the preferred development methods.
(4) Result in a complete and consistent methodology, supporting all phases of the life-cycle.

Our key notion in creating an open methodology environment is that it should allow users of one methodology to access techniques from other methodologies while at the same time preserving the autonomy of a methodology. Autonomy will allow a developer to select each methodology independently of the others and use a single methodology if that is required.

In order to achieve these objectives the methodologies must co-exist in an environment which allows them to share a common conceptual framework and:

(1) is flexible enough to accommodate the idiosyncrasies of each methodology,
(2) facilitates reusability of information in different CASE tools,
(3) permits local autonomy of a CASE tool,
(4) allows incorporation of new CASE tools and methodologies,
(5) allows customizability to local preferences,
(6) it is easy to use and learn.

1.2 OPEN METHODOLOGY ENVIRONMENT IMPLEMENTATION

Three ways of implementing such an environment have been considered:

1. Super Methodology

Recognising the advantages of combining different methodologies many researchers have advocated the need of a "super-methodology" [6] supporting a variety of approaches to different aspects of systems development. However it is questionable if such an approach to methodology integration is achievable since it involves a number of compromises:

(1) It is not easily extended to incorporate new methodologies that will appear in the future or changes to existing ones.
(2) It is costly in terms of the development time and resources that would be needed to ensure the creation of such a methodology.
(3) It may be difficult to learn and use.
(4) There is a loss of autonomy since each methodology will probably have to conform to the rules of the super-methodology resulting in the loss of its individual existence.
(5) It is all embracing.
(6) Existing CASE products will not be able to support it, so new products would need to be developed.

2. Interfacing

A different approach which could solve some of the above problems would be to establish communication between methodologies, and their CASE tools, by custom-built filters which translate between methodologies. This approach is based on interfacing the methodologies. However it is harder to achieve reuse of information within different tools in this approach [11] due to the number of interfaces required.

3. Federated Environment

Systems development methodologies are part of an organisational information resource policy governing the control and sharing of the information resources. This means that we must examine the implementation of an open methodology environment in the context of the information resource management (IRM) [10] of an organisation. A primary control tool of an IRM environment is its Data Dictionary(DD) [1], [20]. The Data Dictionary is instrumental in the planning, administration and operation of an organisation's information processing activities. A more suitable approach therefore for achieving an open methodology environment could be to interface a common Data Dictionary to a variety of CASE tools while letting them retain their own local data dictionaries. Under this approach the common Data Dictionary would be made to act as the co-existence mechanism for the different methodologies, while the local data dictionaries would maintain the autonomy of an individual CASE tool, allowing their techniques and information to be shared between methodologies.

2. SPECIFICATION OF A COMMON DATA DICTIONARY

The ISO has drafted a proposal on a family of standards for an Information Resource Dictionary System (IRDS) [14]. Figure 1 illustrates the conceptual structure of the ISO IRDS. There are three data levels in the ISO's IRDS architecture; Fundamental level, IRD Definition level, and IRD level.

The **Fundamental level** provides the context for the definition of the IRD standard. It consists of the types of data, instances of which are to be recorded on the IRD definition level.

The **IRD Definition level** provides an extensible definition of the types of information which, may be recorded at the IRD level. For example this level would contain information that "record type" and "program type" are concepts, the instances of which are recorded on the IRD level.

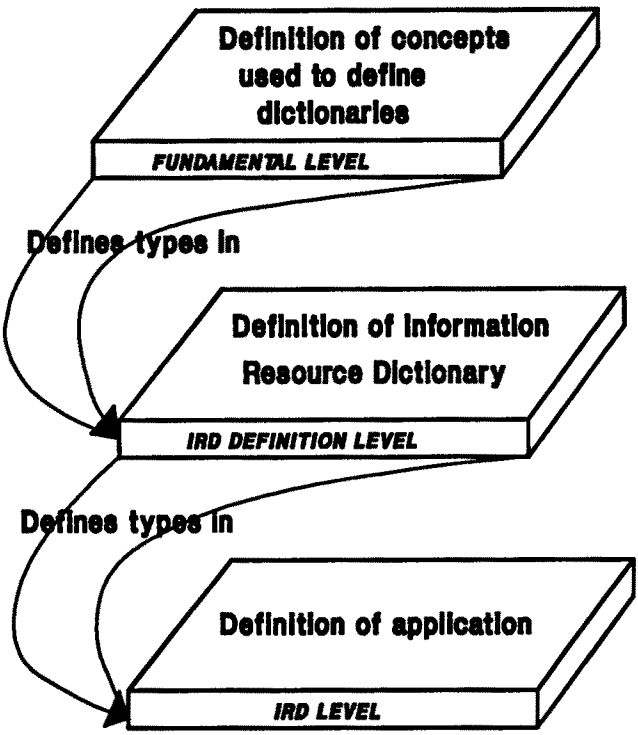

Figure 1 : IRDS Architecture

The **IRD level** is the level on which the content of an Information Resource Dictionary is recorded. For example this level could contain the information that EMPLOYEE and DEPARTMENT are two instances of the concept "record type".

The underlying data structures for the IRD Definition level and Fundamental level are completely prescribed in the ISO IRDS family of standards. The concepts defined for the Fundamental level are those supported in the ISO SQL schema DDL [16]. The concepts defined for the IRD definition level include object type, association class, association type, exclusive constraint, uniqueness constraint, attribute type, as well as several relationships between these concepts [14].

2.1 DATA STRUCTURE FOR THE IRD LEVEL

To achieve the objectives of an open methodology environment the methodologies must share a common conceptual framework, with each methodology implementing part of this conceptual framework.

In order to identify the concepts and structure of this common conceptual framework an analysis of a number of methodologies was undertaken. The identified concepts were classified into five groups; external, conceptual, logical, architectural and physical, corresponding to the five levels of abstraction for information systems as identified in [24]. Currently this analysis has been completed for the first three levels as reported by us in [25].

3. FEDERATED CASE ENVIRONMENT

The activities involved in the systems development process can be viewed as model building activities. Starting with a conceptual model of the application area, which results from the analysis activities, the final objective is to end up with an implementation model. Systems development methodologies use a variety of techniques producing, diagrams, forms or text, for model building.

System models require frequent modifications as requirements change. Manually drawn diagrams are problematic in that the work involved in producing, maintaining, communicating and documenting them, makes it practically impossible to take advantage of using diagrams, and this has limited the use of diagram based methodologies, in systems development, until recently. The spread of low-cost workstations in the mid 1980s have seen the introduction of software systems that automate and enhance the manual methods of the 1970s and 1980s, thus making it more practical and economical to use diagrams. This technology known as Computer Aided Software Engineering (CASE) [8],[26], allows systems developers to document and model an information system from its initial user requirements through design and implementation, and lets them apply tests for consistency, completeness and conformance to standards.

CASE tools are supported by a Data Dictionary in which a user's various work products are stored. Each CASE tool available in the market today has its own, usually non standard, Data Dictionary, thus making it difficult to integrate them. In an open methodology environment, connectivity between different CASE tools is a critical requirement. This can only be effectively achieved if the tools are made to operate from a common shared Data Dictionary. This Data Dictionary must be

accessible to tools via a well defined interface which allows the transfer of information from the tool to the Data Dictionary and vice versa. This is the aim of the Federated CASE Environment(FCE).

3.1 FCE ARCHITECTURE

Figure 2 shows the major components of the FCE together with their data and control flow relationships.

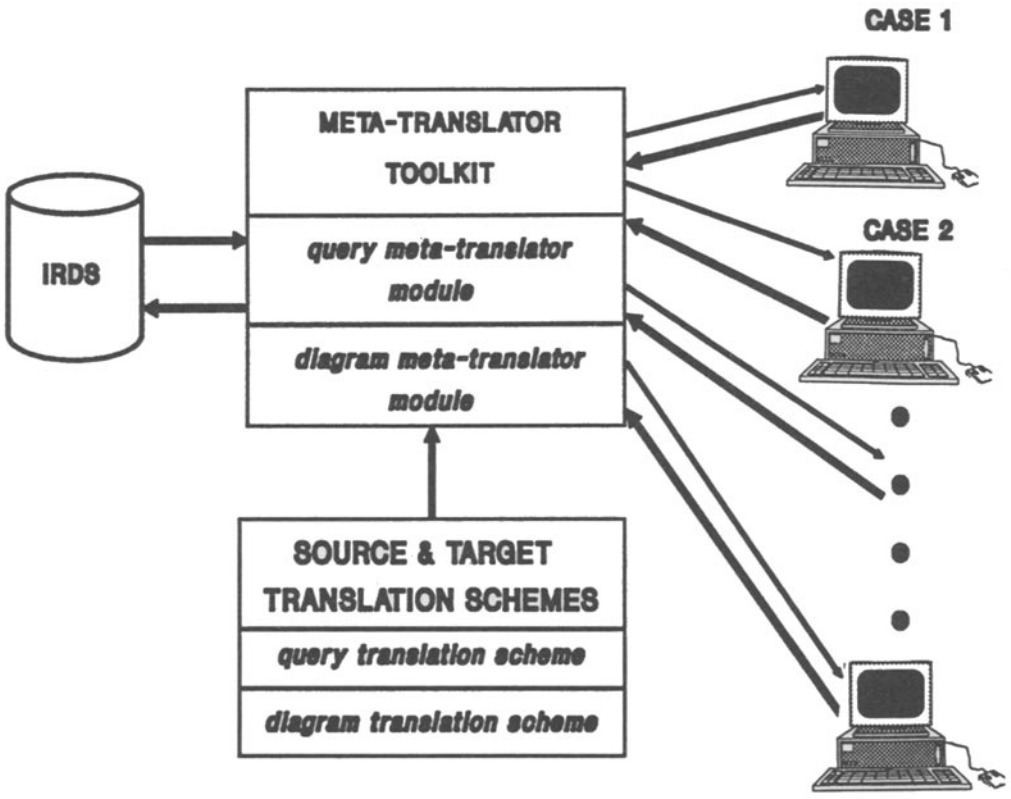

FIGURE 2 : FCE Architecture

The heart of the system is a meta-translator toolkit which performs translations of queries and diagrams between CASE tools. This meta-translator is structured into two module; Query Meta-translation Module(QMM) and Diagram Meta-translation

Module(DMM). Each of these modules has access to a number of translation schemes in the translation scheme library.

An approach to implementation of the QMM is presented in [15]. In the following sections we describe the implementation of DMM.

3.1.1 DMM SYSTEM OVERVIEW

The DMM has dual functionality:

(1) it allows the automatic production of diagrams, by different CASE tools, from data in the global DD.

(2) it allows the storage of information about the target system, entered via diagrams, in the global DD.

Currently only the first of these is implemented in the prototype, as the second can be built by similar process. The prototype DMM was implemented using C [18] and INGRES [5] DBMS, and runs on a VAX 8200 series. The major components of the system are shown in figure 3. The Diagram Constructor component produces a specification of the contents of a diagram instance. These specifications are transformed by the Diagram Description Processor component into a form appropriate for output to a specific CASE tool.

In operation the system has two distinct phases; Diagram definition phase and Diagram construction phase. A general understanding of these phases can be gained by examining what would be required to automatically construct the entity-relationship diagram of figure 4 using the CASE tool AUTOMATE PLUS [19].

In the diagram definition phase the graphical object types used in building an entity-relationship diagram in AUTOMATE are defined. These include rectangles representing entities, arrows representing one-to-many relationships and two types of text representing the names of entities and relationships respectively. Also a tool dependent displayable form of the diagram type is specified. In the diagram construction phase the specific instances of rectangles, arrows and text to be shown on the final figure are created. This is a high level description of the diagram instance which is finally mapped into its equivalent (tool-dependent) output representation

In order to make the communication between the Data Dictionary and the CASE tool possible a facility is required which allows the natural declaration of the information which is to be transmitted between the Data Dictionary and the tool. Since the tool is part of the development environment, whose definitions are recorded at the IRD

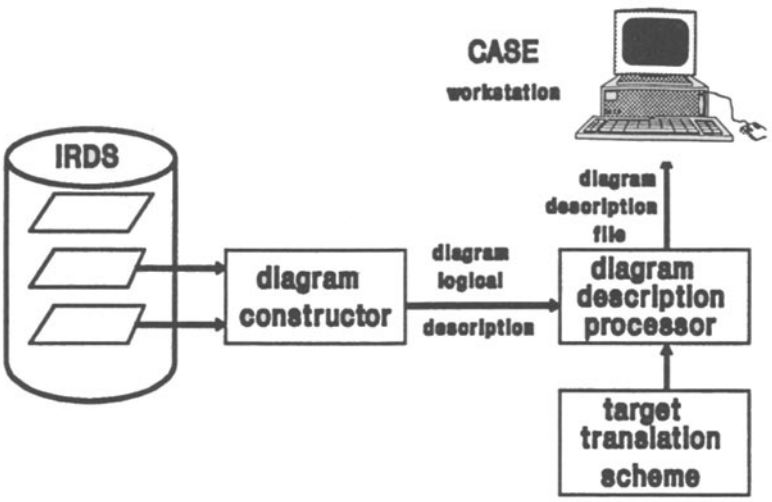

FIGURE 3. : System Architecture

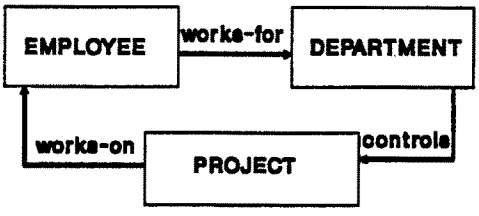

FIGURE 4. : Entity-Relationship Diagram Example

Definition level, we must extend the IRD Definition level structure to allow the definition of a diagram information model for the types of diagrams supported by the tools. A diagram information model consists of three parts:

(1) Morphology
(2) Topology
(3) Semantics

The **morphology** part identifies the set of geometric objects provided by the tool in order to construct a diagram instance. For example an SSADM data flow diagram is constructed from boxes, arrows, circles, text etc.

The **topology** part identifies the topology constraints that dictate the general form of a diagram produced through the tool. For example an arrow(representing a data flow in a data flow diagram) should be linked to two boxes(representing processes), which shows the origin and destination of the data flow.

The **semantics** part identifies the mapping between the geometric objects and the concepts of the Data Dictionary. For example a box in a data flow diagram corresponds to a process in the Data Dictionary.

3.1.1.1 EXTENDING THE IRDS DEFINITION LEVEL STRUCTURE

In order to create a diagram information model a metamodel is required which provides facilities to define the morphological, topological and semantic characteristics of the types of diagrams associated with a tool. This can be achieved by combining existing IRDS features with modest extensions to the IRDS definition level structure defined in the ISO standard.

Figure 5 illustrates diagrammatically, using an entity-relationship notation [3], the main additions to the IRD Definition level structure. Conceptually a specification diagram can be viewed as an abstract object which is defined along three orthogonal dimensions:

Graphical objects: Defines the set of geometrical shapes which are used in order to construct a diagram instance. For example some commonly used graphical objects for data flow diagrams are shown in figure 6. Each graphical object description contains a list of **graphical properties**, which specify how the graphical object is to be displayed. The graphical objects are also arranged in a **taxonomic hierarchy** in which graphical properties can be shared through inheritance.

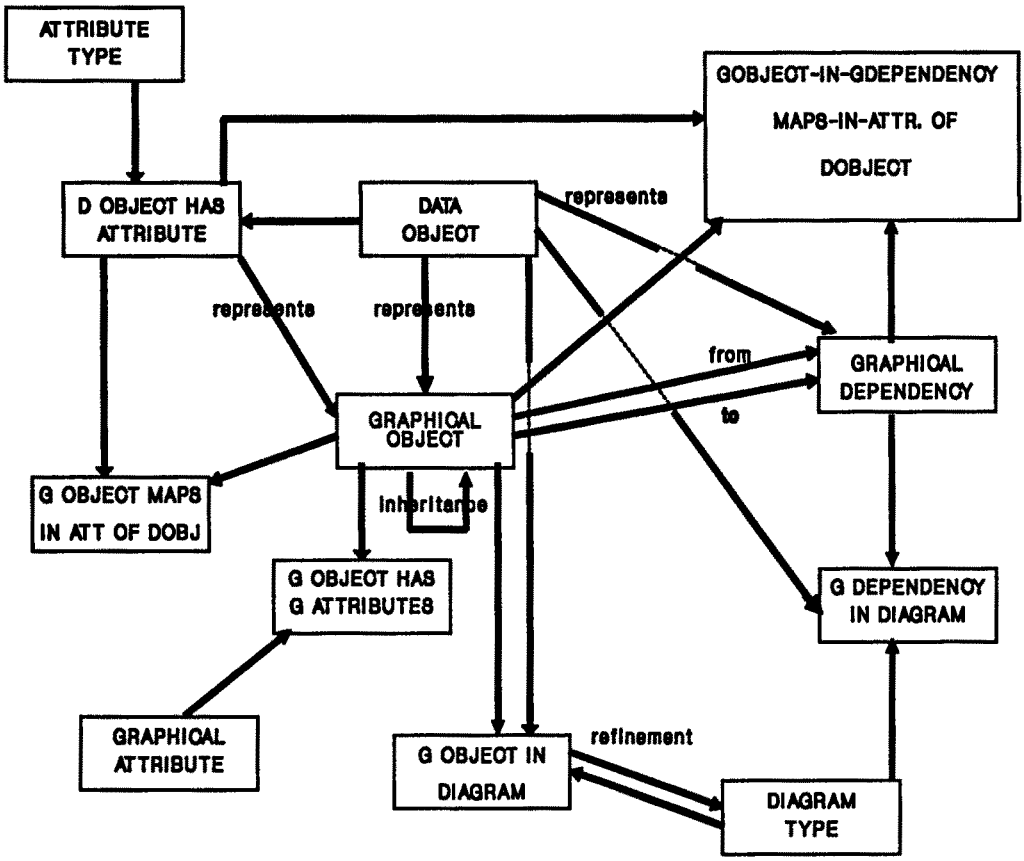

FIGURE 5 : IRDS DEFINITION LEVEL EXTENSION

Graphical dependencies: Graphical dependencies support the maintenance of important graphical constraints, such as connectivity and containment, between graphical objects.

Semantics: Graphical objects indicate semantics. For example a box in a data flow diagram represents a PROCESS. Each graphical object is associated with the existence of a data object or a data relationship or a data attribute of a particular type. Another indication of semantics comes from the graphical dependencies between graphical objects. For example the graphical dependency defined between a box and an arrow in a data flow diagram is represented by the data relationship

"FROM" associating the Data Dictionary concepts "PROCESS" and "DATA FLOW" (see figure 7).

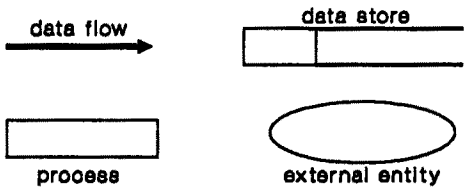

FIGURE 6. : Sample of diagram elements

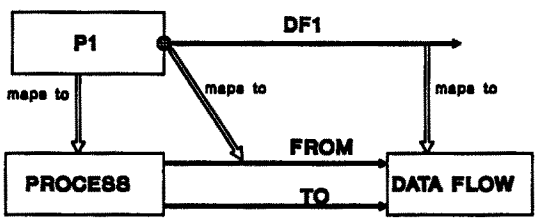

FIGURE 7 : Graphical dependency representation in Data Dictionary

All the information contained in the IRD level and the extended IRD Definition level structures are stored in a collection of relations managed by the INGRES [5] relational DBMS. Relations are defined for all the major classes of objects of the two levels.

3.1.2 AUTOMATIC DIAGRAM PRODUCTION

Having described how the types of diagrams associated with a particular CASE tool can be defined as part of the development environment, we can describe how DMM can assist in the automatic construction of a diagram instance to be presented to a CASE tool user. This is a two steps process:

step 1 : a logical description of the diagram instance in terms of graphical object instances, composing the diagram, and graphical dependencies is created. This is achieved by retrieving the corresponding data dictionary entries(for graphical objects and graphical dependencies) according to the knowledge contained in the extended IRD definition level structure. Also information about the way in which the graphical object instances are to be displayed is added by instantiating the graphical attributes (except for defaults). Due to the lack of an appropriate theory, diagram layout issues were not addressed systematically. However some strategies suggested in [2], [7], [9] and [21] have the potential to be incorporated in the system to achieve our goals.

step 2 : given the logical description of a diagram instance DMM produces a tool-dependent diagram description file. This is achieved in accordance with a diagram template which specifies the mapping between the diagram's logical description and its equivalent tool-dependent output representation. The diagram description file can then be used by the tool's diagram editor for loading.

DMM was tested using AUTOMATE PLUS [19] and YOURDON'S ADT [31]. Figure 8 shows the data dictionary entries describing an entity-relationship model. Figure 9 shows the corresponding entity-relationship diagram produced for AUTOMATE PLUS using DMM.

4. CONCLUSION

Many systems development methodologies have been proposed and used for developing systems. In the past few years systems developers have been emphasising the need to combine different methodologies and their supporting CASE tools. In this paper we have described an approach in which the Data Dictionary system can form the basis for an open methodology environment. The developed

E-R DIAGRAM

E-R REF	DNAME
er1	proj-dep

ENTITY

E-REF	ENAME	E-R REF
e1	employee	er1
e2	department	er1
e3	project	er1

RELATIONSHIP

R-REF	RNAME	E-R REF
r1	works for	er1
r2	work on	er1
r3	controls	er1

REL-ENT

R-REF	E-REF	CARDINAL
r1	e1	N
r1	e2	1
r2	e1	N
r2	e3	1
r3	e3	N
r3	e2	1

FIGURE 8 : DATA DICTIONARY ENTRIES

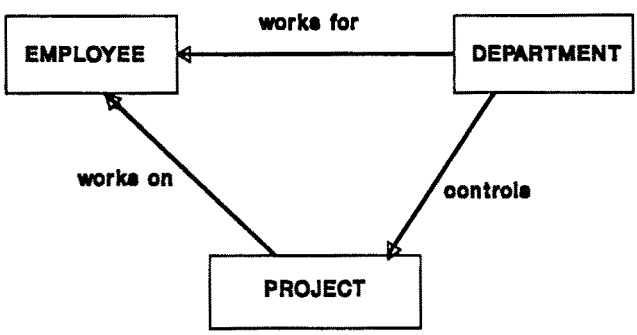

FIGURE 9. : Entity-Relationship Diagram produced for AUTOMATE

system, although still in prototype form shows the feasibility of such a development environment. Currently the system supports the Data Dictionary to CASE tools transfer of information for automatic diagram production. Future work will concentrate on implementing the CASE tools to Data Dictionary link in order to support interactive diagram production.

REFERENCES

1. F.W. Allen, M.E. Loomis and M.V. Mannino, "The integrated dictionary/directory system", Computing Surveys, vol. 14, No. 2, 1982.
2. R.I. Becker and S.R. Schach, "Drawing labelled directed binary graphs on a grid", Proc. ACM 20th Southeast Regional Conference, Knoxville, TN, 1982.
3. P.P. Chen, "The Entity-Relationship model - Towards a unified view of data", ACM Transactions on Database Systems, 1, 1976, pp. 1-36.
4. J.D. Couger, M.A. Colter and R.W. Knapp, "Advanced system development/feasibility techniques", Wiley, 1982.
5. C.J. Date, "A guide to INGRES", ADDISON-WESLEY, 1987.
6. E. Downs, P.Clare, and I. Coe "Structured Systems Analysis and Design Method", Prentice-Hall,1988.
7. P. Feldman, "A Diagramer for the automatic production of Entity type models", Proc. of the third British National Conference on Databases, Leeds, 1984.
8. C. Finkelstein, "An introduction to information engineering: from strategic planning to information systems", Addison Wesley, 1989.
9. E.R. Gasner, S.C. North and K.P. Vo, "DAG-A program that draws directed graphs", Software-Practice and Experience, Vol. 18, No.11, pp. 1047-1062, Nov. 1988.
10. A.H. Goldfine, "Database directions information resource management-strategies and tools", NBS report(500-92), 1982.
11. G. Grosh, "A recipe for cooking up a successful project", Computing, 11th May 1989
12. D. Heimbigner and D. McLeod, "A federated architecture for information management", ACM TOIS, Vol.3, No.3, pp. 253-278, 1985

13. S. Holloway, "Methodology handbook for information managers", Gower, 1989.

14 S. Holloway (ed.), "The future of Data Dictionaries", Proc. DATABASE 88, Conference, Milton Keynes ,U.K., 1988.

15. D.I. Howels, N.J. Fiddian and W.A. Gray, "A source-to-source meta-translation system for database query languages- Implementation in PROLOG", In: P.M.D. Gray and R.J. Lucas (eds), "PROLOG and Databases: Implementations and new directions", pp.22-38, Ellis Horwood, 1988.

16. ISO 9075, "Database Language SQL+Addendum 1", 1987.

17. M. Jackson, "Systems Development", Prentice-Hall, 1983.

18. B.W. Kernigham and D.M. Ritche, "The C programming language", Prentice Hall, 1978.

19. LBMS, AUTOMATE PLUS Manual, 1987.

20. B.W. Leong-Hong and B.K. Plagman, "Data Dictionary/Directory systems: administration, implementation and usage", Wiley, 1982.

21. R.J. Lipton, S.C. North and J.S. Samdberg, "A method for drawing graphics", Proc. Symp. Computational Geometry, Baltimore, MD, 1985, pp. 153-160.

22. R.M. Maddison, G.J. Baker, L. Bhabuta, G. Fitzgerald, K. Hindle, J.H.T. Song, N. Stokes and J.R.G. Wood, "Information systems methodologies", Wiley-Heyden, 1983.

23. NCC, SSADM Manual, Manchester, U.K., 1986.

24. A. Olive, "Analysis of conceptual and logical models", In: A. Olle, et al(eds), "Information systems design methodologies: A feature analysis", Elsevier Science, 1983.

25. S. Papahristos and W.A. Gray, "Data Dictionary support for integration of systems development methodologies", Proc. Second International Conference Software Engineering for Real Time Systems, 1989.

26. B. Sedacca, "CASE tools", Informatics, Vol. 2, No. 6, June 1990.

27. I. Sommerville, "Software Engineering", Third edition, Addison Wesley, 1989.

28. D. Teichroew, P. Macasovic, E.A. Hershey III, Y. Yamamoto, "Application of the Entity-Relationship Approach to Information Processing Systems

Modeling", In: P.P. Chen(ed.), "Entity-Relationship Approach to Systems Analysis and Design", North-Holland, 1980.

29. A.T. Wood-Harper, L. Antile, and D.E. Avison, "Information Systems Definition: The Multiview approach", Blackwell Scientific, 1985.

30. E. Yourdon, "Managing the Systems Life Cycle", Yourdon Press, 1982.

31. YOURDON, Analyst/Designer Toolkit User Guide, 1987.