Chapter 8

Schema Evolution and View Management in Metrics Databases using Recursive Graphs

8.1 Introduction

A central theme in this dissertation is to show the value of collecting, storing, and using software as well project management metrics in improving the quality of software as a product. A key lesson learned from any successful project is to gain valuable experience for any future projects that might produce similar or related products. Thus, metrics data collected during the execution of a project has value beyond the project for which it was collected, and thus the life of a metrics database may be well beyond the initial project it was created to help manage.

A key issue that arises for long-lived databases, especially those that are put to subsequent use in related applications, is one of *evolution.* One kind of evolution is the addition of new data. This is rather easy to handle, and some issue such as explicitly managing the temporal component may arise. Issues in, and techniques for, handling temporal data have been discussed in the previous chapter. A different, and perhaps more involved, is the issue of change in the structure of the database itself.

In this chapter, we discuss the issues in the evolution of metrics databases. Specifically, we examine how the database schema as well as the database evolve with time and the changing needs of various software projects. In Section 8.2 we present the problem of schema evolution of metrics databases. In section 8.3 we provide a brief overview of the theory of recursive graphs, or R-graphs, [BUC79,KUN80,KUN90], and in Section 8.4 show how the problem of database schema evolution can be formulated in terms of evolution of R-graphs. The theory of R-graphs has been well established, and its properties with respect to structural changes has been well

studied. In section 8.5 we discuss the issues in metrics database evolution, and how they are handled by the R-graph formalism. Based on our findings given in this thesis, we propose a software metrics database system in Section 8.6.

8.2 Schema Evolution in Metrics Databases

Consider an entity-relationship (E-R) based database schema as shown in Figure 8.1.

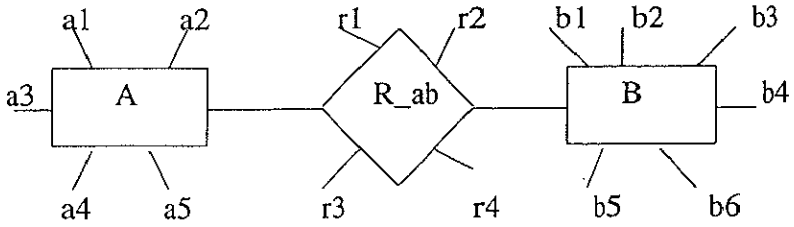

Figure 8.1. Initial Database Schema

This schema has two entities, namely *A* and *B*, and a relationship *R_ab* between them. The entities *A* and *B*, and the relationship *R_ab*, have five, six, and four attributes respectively. Assume that this schema represents the initial database. We now consider the following types of schema modifications:

- collection of new information about entities and relationships, leading to the addition of attributes to existing entities and relationships in the schema;
- change of domains of existing attributes, e.g. expansion of a valid range of values,
- collection of information about new relationships, leading to establishment of new relationships between entities, and
- collection of information about new entities, leading to creation of new entities.

8.2.1 Attribute Addition and Deletion

As databases evolve, new information may be collected about existing entities, as the pre-existing information may not be sufficient for newer applications. Figure 8.2 shows the new

schema after new attributes, *a6* and *r5*, have been added to the entity *A* and the relationship *R_ab* respectively. Also, the attribute *b6* has been deleted from entity *B*, as this is perhaps no longer needed.

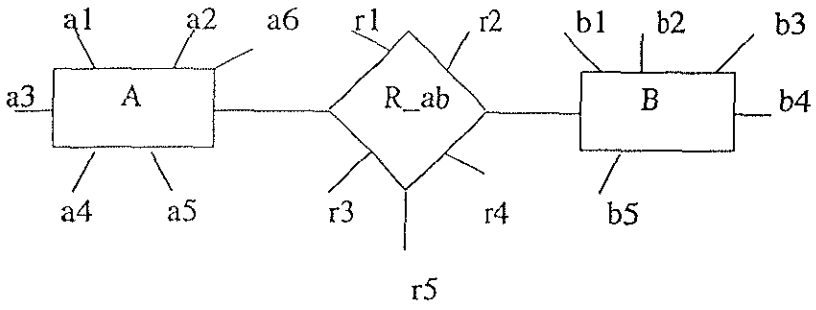

Figure 8.2: Schema After Attribute Addition/Deletion

An example of attribute addition, as given in Chapter 2, is the addition of the cost-of-manpower attribute as part of the cost metric. That is, we may want to collect data denoting the cost of manpower for each activity in addition to software costs. The original cost metric has the following tuples, as illustrated in Figure 8.3.

<Data_Date, System_Name, Activity_Type, BWCS, BWCP, AWCP>

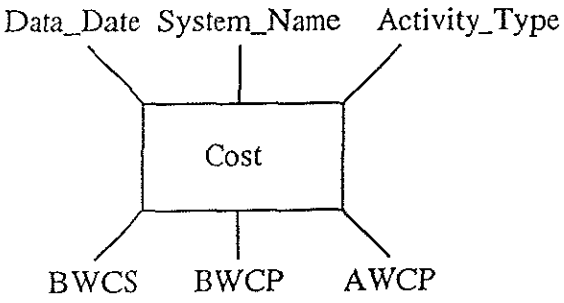

Figure 8.3: Original Schema for Cost Metric Entity

After we add the new cost-of-manpower attribute we have a new cost metric as follows:

<Data_Date, System_Name, Activity_Type, BWCS, BWCP, AWCP, Manpower>

This is illustrated in Figure 8.4.

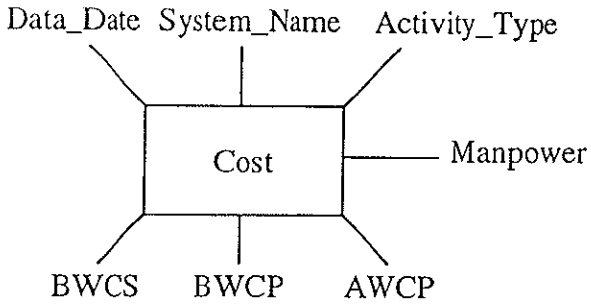

Figure 8.4: Adding the Manpower Attribute to the Cost Metric Entity

8.2.2 Creating New Relationships

If **SCH** is the set of all schemas, this schema evolution can be modeled as an operation **op_1** on schemas, which takes one as input and produces another schema as output, or

new_schema ← op_1(old_schema, attributes_to-Add, attributes_to_delete)

As databases evolve, new relationships may emerge amongst existing entities, which then need to be modeled for meeting the needs of new applications.

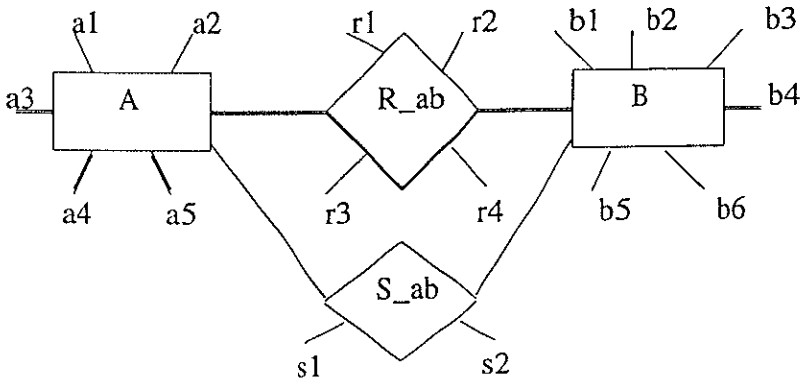

Figure 8.5: Schema After Adding Relationship

Figure 8.5 shows the schema that emerges when a new relationship, namely *S_ab*, between the entitities *A* and *B* has been identified and modeled as part of the database schema. This schema evolution could be modeled as an operation, **op_2**, which works on schemas, as follows:

new_schema_ ← op_2(old_schema, relationships_to_add, relationships_to_delete)

As an example, given the relationships between metrics illustrated in the table in Figure 2.4 of Chapter 2, that the Fault Profiles Metric entity is related to the Reliability Metric entity by the "decreases" relationship; that is, high overall values for the Fault Profiles metric decreases reliability. This will be modeled originally as given in Figure 8.6. For simplicity, the attributes for the Fault Profiles and Reliability metrics are referred to in generic terms as f1, f2, ..., and r1, r2, ... respectively.

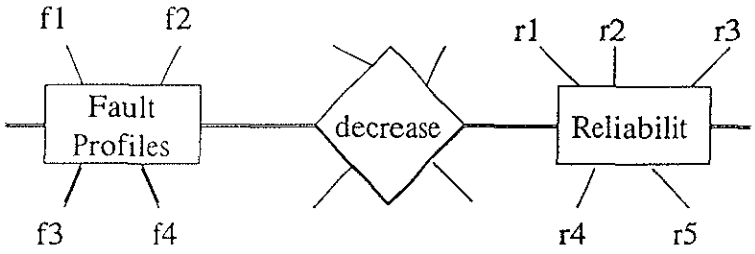

Figure 8.6: Original Relationship Between Fault Profiles and Reliability Entities

Suppose we now want to add another "time" relationship to the two entities; that is, as the overall values for the Fault Profiles metric drop over time, the reliability of the system correspondingly increases. The new schema is illustrated in Figure 8.7.

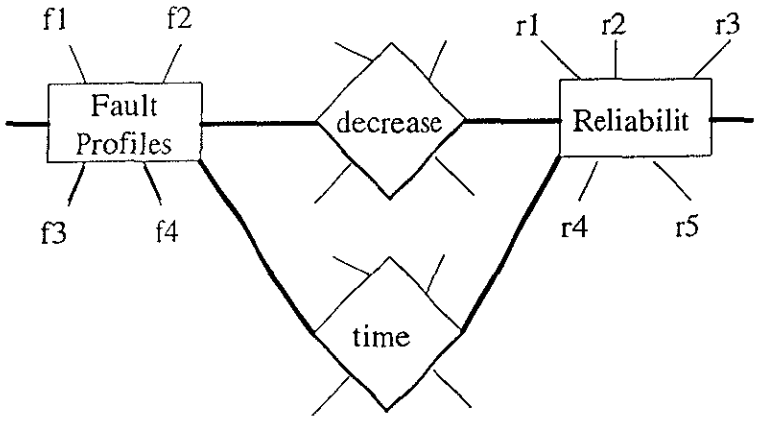

Figure 8.7: New "Time" Relationship Between Fault Profiles and Reliability Entities

8.2.3 Creating New Entities

Sometimes, as a metrics database evolves, it may be deemed necessary to collect information about entirely new kinds of entities, which were not even being considered earlier. Often this happens in response to (a) the needs of some new project, or (b) a more refined model of the reality whereby the need arises to model more entities. Conversely, sometimes it might be considered that some entities either do not exist anymore, or are simply not of interest, and hence information about them may not be collected any more. Such decisions can cause entities to be added to/ deleted from the metrics database schema, as shown in Figure 8.8.

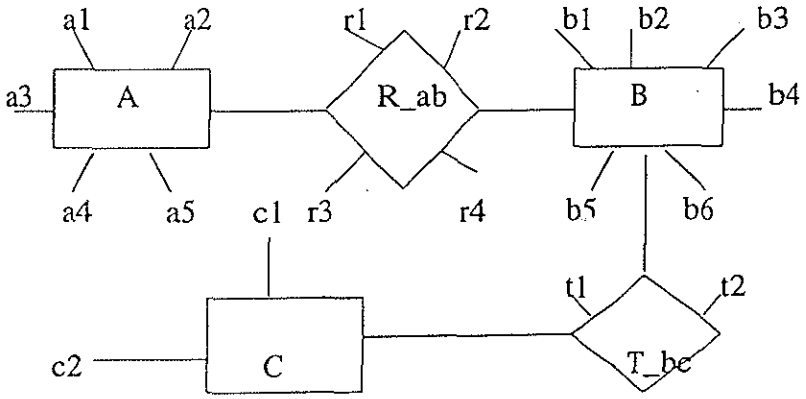

Figure 8.8: Schema After Addition of a New Entity

As shown in Figure 8.8, the original schema illustrated in Figure 8.1 has been modified to add the new entity C. A new entity is almost always added along with at least one new relationship, usually one that connects the new entity to some existing entity. The reason for this is that the existence of a new entity, and the subsequent need to model it, is realized in context of some existing portion of the database schema. This naturally gives rise to such a relationship. Once again, this schema restructuring can be modeled as an operation, **op_3** which operates on schemas to produce new ones, as follows:

new_schema ← op_3(old_schema, entities_to_add, relationships_to_add)

8.3 Recursive Graphs (R-graphs): A Brief Introduction

Recursive graphs, or R-graphs, is a formalism first introduced by Kunii et al [KUN90,BUC79] to develop the mathematical basis for developing a methodology and tool for interactive system design using a visual paradigm [KUN80]. The recursive graph formalism (RGF) consists of recursive graphs (R-graphs) and recursive graph operators (R-operators) to manipulate them.

8.3.1 R-Graphs

Traditionally, the most popular way of representing structures is by a graph, both because of its formalism as well as visual appeal. A graph G is a triple which consists of nodes N, arcs A, and an arc function af, specifying the ordered node pairs to which arcs are incident. Then,

$G = (N, A, af)$, where $af: A \rightarrow N \times N$.

Usually, systems are represented by nodes, and their relationships by arcs. To lay a mathematical foundation which can incorporate a system designer's intuitive recursive (hierarchical) view of system design, the standard graph theory formalism has been extended to R-graphs [Kunii & Harada 1980]. First, an arc function af is extended to map an arc to a pair of node subsets such that $af: A \rightarrow 2^{exp(N)} \times 2^{exp(N)}$. This extension is useful for a designer to relate groups of nodes by an arc. Next, a subnode function ($sn: N \rightarrow 2^{exp(N)}$), a port function ($pt: N \rightarrow 2^{exp(N)}$), and a subarc function ($sa: A \rightarrow 2^{exp(A)}$) are introduced to incorporate inclusion relationships among system, interface relationships among systems, and inclusion relationships among associations, respectively. These extensions increase the structure representation capabilities of graphs, i.e. a node semantic function ($ns: N \rightarrow NR$) and an arc semantic function

(as: A → AR) are introduced to link nodes and arcs to node records NR and arc records AR, respectively. Formally, an R-graph is represented by the following tuple:

$R = (N, A, af, sn, pt, sa, ns, as)$

Here:

N : set of nodes in R-graph

A: set of arcs in R-graph

af: $A \rightarrow 2^N \times 2^N$. It is arc function specifying the ordered node pairs to which arcs are incident. This function allows users to relate groups of nodes by an arc.

sn: $N \rightarrow 2^N$ (sub-node function), *pt:* $N \rightarrow 2^N$ (port function), and *sa :* $A \rightarrow 2^A$ (port function) are extension functions to incorporate system inclusion, system interface and association inclusion relationships, respectively. These extensions can enhance the abstraction capabilities of the R-graph.

ns: $N \rightarrow NR$ is a node semantic function, that links nodes to nodes records *NR*, in a relational schema.

as: $A \rightarrow AR$ is a link semantic function, that links nodes to nodes records *NA*, in a relational schema.

The above definition of R-graph is augmented with a set of operators (JOIN, ZIN, ZON, EXN, SEL, DEL, SURV) to complete the Recursive-Graph Formalism (RGF). These operators are discussed below.

The most prominent features of RGF include its capability to allow hierarchical design evolution of complex systems and design automation.

The domains of the functions *af, sn, pt, sa, ns,* and *as* are extended to incorporate the value "undefined" or "under-defined", and the value "over-defined" or "redundant".

Given an R-graph, repeated applications of *sn* and *pt* to the nodes and of *sa* to the arcs produces a hierarchy of the nodes and that of the arcs, respectively. Given a node or an arc of any hierarchy, the nodes or arcs produced earlier are called its *ancestor nodes* or *arcs*. It is reasonable to assume that in any system design, a subsystem cannot (directly or indirectly) be a part of itself. Thus the R-graph created will be a directed acyclic graph (DAG).

8.3.2 R-operators

All elementary operations on a node or an arc of a design schema are performed by recursive graph operators (R-operators). In total there are eight R-operators that have been defined in [BUC79]. The operations are integration operations, four are reduction operations and one is for deletion of components of a graph. The integration operations are:

(1) r1 ← JOIN(r2, r3, SN, SA),

where r2 and r3 are schemas to be joined, r1 is the resulting schema, and SN and SA are lists of node pairs (including port pairs) and arc pairs, respectively, to be merged.

(2) r1 ← ZIN(r2, r3, SN),

where r3 is the schema to be inserted into schema r2 in the "zooming-in" process. Here the zooming-in is with respect to a node.

(3) r1 ← ZIA(r2, r3, SA),

where the zooming-in is with respect to an arc.

The reduction operations are as follows:

(4) r1 ← ZON(r2, SN),

where r2 is the schema which has to be "zoomed-out" to a single node. This zoom-out to a node operation deletes the sub-graph that has been zoomed-out to a single node.

(5) r1 ← ZOA(r2, SA),

where the zooming-out is with respect to an arc.

(6) r1 ← EXN(r2, SN),

where the EXN operator extracts the portion of the subgraph which is abstracted as the node SN.

(7) r1 ← EXA(r2, SA),

where the EXA operator works similar to EXN, expect it works on edges.

The deletion operator is:

(8) r1 ← DEL(r1, SN, SA),

where SN and SA are the respective node- and arc-list of nodes and arcs to be deleted from the original R-graph.

8.3.3 Experience With R-graphs

The R-graph formalism has been used in a number of application domains, e.g. design of hospital information systems [KUN80] and design of petrochemical plants [BUC79], and valuable experience has been gained from it. Based on this, a design tool, called SID (system for interactive design), has been built [KUN80] to enable designers to develop complex systems in an easy-to-use and flexible, visually-oriented tool.

As mentioned above the problem of metrics database evolution can be modeled in terms of operations on R-graphs. This enables the analysis of the evolution problem in a rigorous formalism, as well providing an opportunity to potentially use the SID tool for the task of evolution management itself.

8.4 Modeling Metrics Database Schema as R-graphs

In this section we show how a metrics database schema can be modeled in terms of the formalism. Rather than introduce the complete formalism first, we proceed in an incremental manner by illustrating concepts with examples, and then formalizing them. Let us say that the mapping of metrics database schemas into R-graphs is accomplished by some mapping M, where

M: E-R schema → R-graph

8.4.1 Mapping of Simple Schemas

Consider the schema in Figure 8.1. Since the R-graph formalism focuses on the structural (or topological) properties of graphs, we omit entity and relationship attributes for the present. For simple schemas such as this, the proposed modeling maps entities into nodes and relationships into arcs. Figures 8.9 and 8.10 show the mapping.

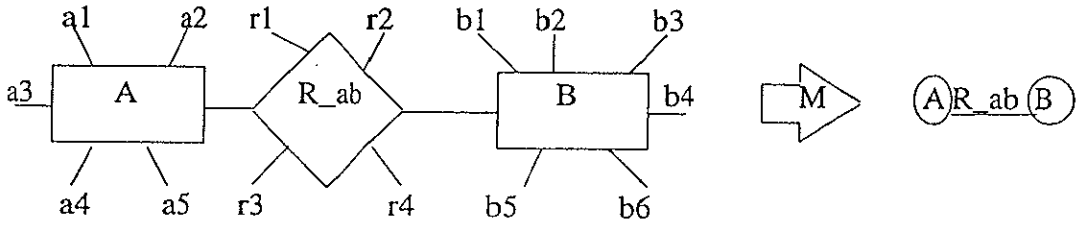

Figure 8.9: A Simple Schema

Figure 8.10: Mapping to R-Graph

Formally,

$N = \{M(e) \mid e \in E\}$,

$A = \{M(r) \mid r \in R\}$.

For example, Figure 8.6 denoting the original relationship between the Fault Profiles and Reliability entities can be modeled as illustrated in Figure 8.11, below.

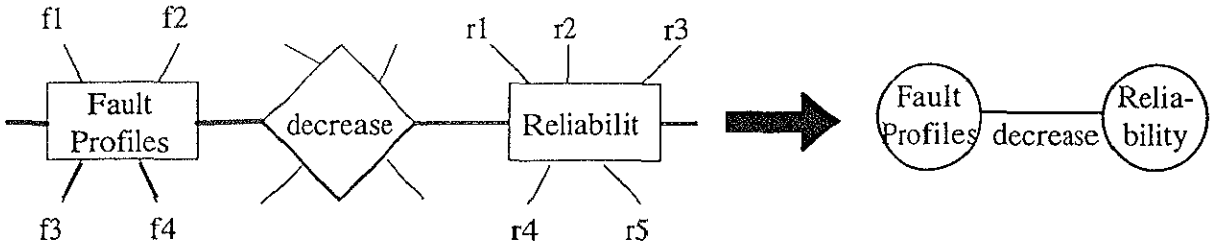

Figure 8.11: R-graph of Fault Profiles and Reliability

8.4.2 Mapping of Schema with Views

As shown above, mapping of simple schemas is quite straightforward. Once schemas contain views though, the mapping becomes more complex, since it must now handle multi-level schemas where entities at one level are (sub-)schemas at a lower level. However, it is here that the full power of the R-graph formalism becomes evident. Views [AST76] were introduced as a concept to capture an entire sub-schema as a single entity, perhaps to be used in a different schema. It is useful mechanism for many purposes such as handling database security and integrity, and managing database evolution [ELM94].

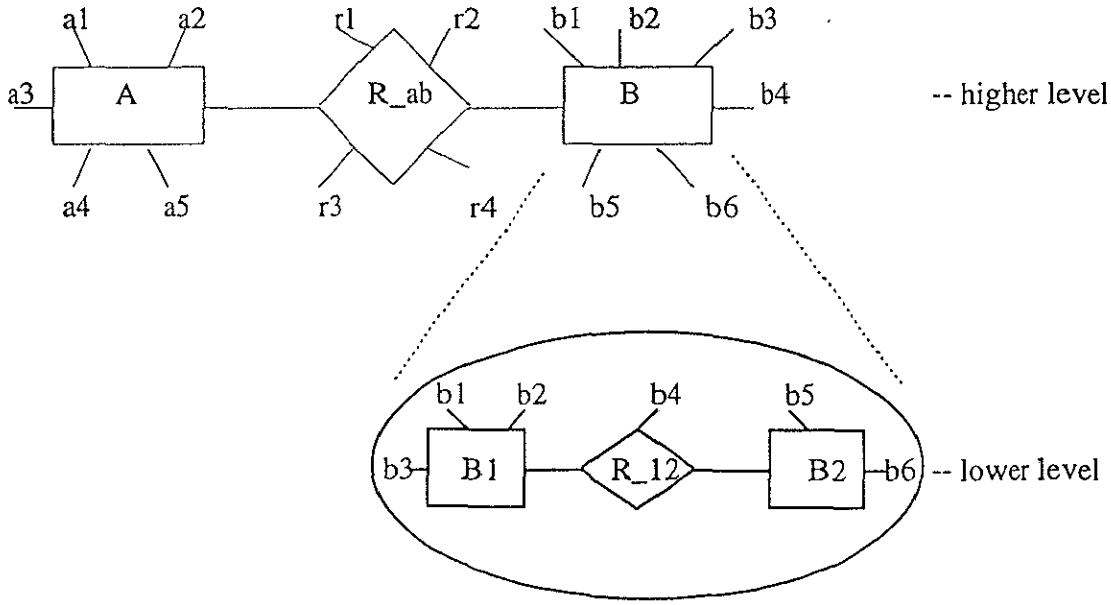

Figure 8.12: A Hierarchical Schema Involving a View

Figure 8.12 shows an example of a 2-level schema in which an entity at the higher level is expanded into an entire (sub-)schema at the lower level. The entity B on the higher level is made up of an E-R sub-schema at the lower level, consisting of entities B1 and B2, and relationship R_12. Note that the attribute set of B is the union of the attribute sets of B1, B2, and R_12. In relational terminology, an entity such as B is called a view. Figure 8.7 shows the corresponding R-graphs at the two levels.

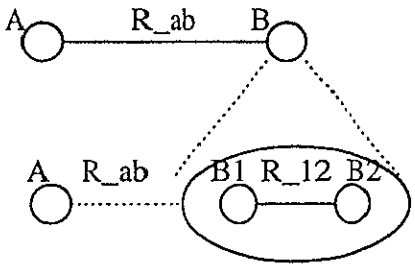

Figure 8.13: R-graph for Schema in Figure 8.6

8.5 Extending R-Graph to Support Object-Oriented Abstraction and Evolution

As mentioned above, the RGF can be elegantly extended to two important areas of software life-cycle:

i) to handle software database evolution over time, as discussed in the previous sections, and

ii) to develop an object-oriented based framework for supporting views of quality and risk management.

In this section we discuss the object-oriented extension in detail.

8.5.1 Object-Oriented View Abstractions using R-Graph

An R-graph can be mapped into an object hierarchy where its structure base is mappable to IS_PART_OF aggregation abstraction. Its semantic base is mappable to class definition. The merging process in an R-graph is equivalent to combining component objects in aggregation

abstraction to generate composite objects. Aggregation is generally recursive, and is analogous to the recursive structure of an R-graph.

Starting from an E-R model (Fig. 4.1) of a software metrics data, we can abstract various views directly from the model. Such views can be grouped or categorized into various classes, defined within the context of object-oriented paradigm. In other words, for the management of software development process, we build a quality view hierarchy for a software product by using the E-R model associated with the software metrics, shown in Figure 4.1. Since, we do not want to restrict ourselves to any specific underlying data schema, we can build a view hierarchy directly from the E-R model in form of an R-graph. For this purpose we introduce the following two semantic operators for view generation which can be used to map an E-R model to an object-oriented R-graph model.

Q: $N' \leftarrow C \mid K$

S: $N' \leftarrow N \mid P$

Here C represents the schema based on E-R or relational model. N represents subset of nodes in R-graph. N' set represents the newly generated node(s) in the R-graph to support the new view based on the predicates K or P. In other words, the semantic operator Q provides an abstraction by analyzing entities and their relations in the E-R model, according to the predicate K. For R-graph structure, this operator generates new nodes. For the new RGF, we take the operator Q to the operator SI, proposed in the previous chapter. In other words:

$Q \leftarrow\text{------}\rightarrow SI$.

Accordingly, K describes the qualification clause associated with operator SI, as discussed in the previous chapter.,

The semantic operator *S*, on the other hand, aggregates the semantics associated with the nodes in an R-graph. For this purpose, the predicate *P* analyzes the node semantic function *ns* associated with the existing nodes in the R-graph. The objective is to generate an R-graph that predicts the quality of software product at various levels of abstraction, during the evolution process of software life-cycle.

In order to express predicates *K* and *P* to carry the notion of quality and risk and to allow the generation of complex views within R-graph, we need to develop a formal framework which allows both temporal and spatial (across various metrics data) semantic to be expressed at any given level of abstraction in R-graph. The objective is to provide a theoretical mechanism for the user to express complex view/semantics, irrespective of any constraints which may be exhibited by the underlying data model. For this purpose we use the SMPN, proposed in the previous chapter, for specifying predicate *P*. We also formalize the concept of views present in the metrics data. The R-graph based framework allows the conceptualization of metrics data using both bottom-up as well as top-down object-oriented approaches for data abstraction. In the bottom-up approach, a user can build complex views using the semantic operators *Q* and *S* while in the top-down approach, a user can integrate/group views having identical semantics.

8.5.2 Spatio-Temporal Modeling of Metrics Data for Predicates *K* and *P* and View Formulation

In the example, given in Section 7.5.1.3, we have assumed a relational schema for the software data metrics. However, it an be noticed, that operator *Q* is applicable to any data model, including E-R, used for the software metrics data.

The result of operator Q can be stored in the form of a node in an R-graph. In other words, the result, such as shown in Table 7.4, provides a view that is represented by a node in an R-graph, with a node semantic function (ns) corresponding to the predicate K.

8.5.2.1 Semantic Operator S for R-Graph and A Petri-Net Based Formalism for Predicate P and Node Semantic Function

Modeling of a view or singularity condition requires occurrence of multiple temporally related *sub-events* (views). The overall process of expressing a predicate requires a priori specification of multiple temporal sub-events. It can be noticed that a simple temporal event can be expressed formally as a logical composition of various low level predicates (both spatial such as the type K discussed above and temporal) that analyzes the metrics data over a given time interval or over the specified part of the life cycle of software development process. Subsequently, more complex abstractions (views) can be defined recursively in terms of existing views through an arbitrary specification of temporal relations. This gives rise to a recursive structure of view abstraction in form of an R-graph. Clearly, the leave nodes of this graph represents the semantics based on operator Q.

As mentioned above, for the new RGF, we propose to use SMPN to represent the predicate P for the operator S. In other words, from Section 7.5.2.3,

$$P \leftarrow N_{SMPN} = \{T, P, A, D, M\}$$

Accordingly, the result of operator S can be stored in the form of a node in an R-graph. In other words, the SMPN associated with the predicate P, provides a view that is represented by such a node. The node semantic function (*ns*) in this case corresponds to the temporal specification given by the SMPN.

8.5.2.2 View Generation, Node Semantic and Arc Semantic Functions in R-Graph

It is important to mention that both predicates K and P for the semantic operators Q and S, respectively, can be recursively applied to build higher level views. This is possible due to embedded nature of the predicate K within the places of an SMPN and the recursive formulation of SMPN to build complex Petri Net structures from the simple ones. A Petri-net can be embedded in another Petri-net since, associated with each SMPN is a time interval. These intervals can be aggregated to build large intervals.

As mentioned above, each graphical primitive (or predicate K) and each SMPN (or P) yields a view which can be mapped to a node in a R-graph in form of its semantic function (ns). Such views can be dynamically built or updated as new data metrics or attributes are introduced in the E-R model of the software metrics data, as discussed earlier in this chapter. In other words, data evolution can be managed by specifying predicates K and P *a priori* for the operators Q and S, and constantly monitoring of emerging views and singularity conditions.

It can be noticed that as nodes of an R-graph are linked to construct a recursive hierarchy, the links among these nodes represent object-oriented abstractions and associations. In our formalism, two abstractions, namely; generalization and aggregration, provide the basis for arc semantic (as) function of the R-graph.

8.5.2.3 Example of an R-Graph for Software Metrics Data

Using a detailed example, we now elaborate the formalism presented above. Suppose, we would like to assess the quality of a software system, named MDBMS, based on its metrics data. We assume that the E-R model of this data is already available. Without loss of generality, in this example, we consider the notion of quality based on four metrics, namely, cost, schedule, requirement traceability and testing. We formulate our views about the quality as follows:

- View V1: We specify the rule for the view that the software system MDBMS will be of high quality if the evolution of the above mentioned metrics satisfy the following predicates:

  All the requirements of MDBMS become traceable within a time period of d1 program months followed by the testing phase which should have been completed within a time period of d2 program months. Also, the gap between these two phases is from 1 to at most 2 program months. The scheduled completion date for the testing phase must note slip beyond the program month M1. Also, the actual cost must not have exceeded the planned cost for the entire duration of these two activities.

- View V2: We specify the rule for the view that the software system MDBMS will be of moderate quality and has a slight risk of not being marketed in time if the evolution of the above mentioned metrics satisfy the following predicates:

  The total duration for which requirements remain untraceable exceeds d3 program months but is less than d4 program months. Also, the testing phase is considerably prolonged and its duration exceeds d5 program months, although it is still less than d6 the gap between two phases is more than d7 months but still less than d8 months. The scheduled completion date for testing phase slips beyond the program month M2. For the whole process, the actual cost exceeds the planned cost by 20%.

- View V3: We specify the rule for the view that the software system MDBMS will be of poor quality and has a high risk of not being marketed in time if the evolution of the above mentioned metrics satisfy the following predicates:

  Requirements are not completely traceable even beyond program month M3. Also, the testing phase starts, even though many requirements remain untraceable and there is a considerable

overlap between these two phases. The testing phase is also prolonged beyond target date M4. The overlap is at least d7 program months. During the whole process, the actual cost exceeds the planned cost by more than 40%.

We elaborate the process of recursively generating views V1, V2 and V3 from the metrics data and building an R-graph. The crux of this process is the use of semantic operators Q and S and various object-oriented abstractions, including generalization and aggregation. From the Requirement Traceability metric, we first generate a sub-view that provides the information whether or not all the requirements are traceable. In case they are traceable, the overall duration of this fact is computed. For this purpose, we use the semantic operator Q and its concatenability property. The resulting sub-view, labeled as SV1 in Figures 8.14, is specified using the graphical primitives associated with the operator Q, as discussed in Figure 7.7. According to the semantics of the predicate of operator Q, the duration attribute of the Requirement Traceability metrics is analyzed over all the intervals during which the system "MDBMS" has been under development. Using the concatenability property of the operator Q, all these intervals are aggregated to generate a super interval over which all the requirements are ultimately traceable. The sub-view SV1 can be represented in form of a node of an R-graph. This node is labeled as N1 in Figure 8.14. As mentioned earlier, the specification of SV1, which is based on operator Q, can serve as the node semantic function (*ns*) for the node N1. Using a similar approach, we can generate another sub-view, SV2, for the completion process of the testing phase of system MDBMS. For this purpose, we analyze attributes associated with the Testing metrics of the E-R schema and use the semantic operator Q, along with its concatenability property. Accordingly, associated with S1 is a duration which is equal to the duration of the super interval. The sub-view SV2, as depicted in Figure 8.14, is built using a graphical primitive and it corresponds to node N2 of the R-graph

shown in Figure 8.14. Similarly, we can build another sub-view, SV3, to analyze the cost associated with the system MDBMS. SV3 maintains the condition whether or not the actual cost has exceeded the planned cost. In case, the actual cost has not exceeded, it provides the overall duration of this phenomena. For this purpose, various intervals are identified in the cost metrics and are aggregated based on the concatenability property. The sub-views SV1, SV2, and SV3 can then be temporally related using a SMPN. The temporal relations in an SMPN follow an object-oriented abstraction known as aggregation (IS_PART_OF). This aggregation provides the arc semantic function (as) for the arcs from nodes N1, N2, and N3 to node N5. Different SMPNs are needed to construct the desired views, namely; V1, V2 and V3. The specifications of these SMPNs are given in as SV5, SV8, and SV12 as shown in Figures 8.14, 8.15, and 8.16, respectively.

For example, for V1, the SMPN that temporally relates SV1, SV2, and SV3, as depicted in Figure 8.14, provides a higher level sub-view, labeled as SV5. The places in the SMPN represent various sub-views and a delay place. The delay place corresponds to the gap between the two phases of software development process and in this case they represent requirement traceability and testing. The gap is specified according to the rule for building view V1. Also, the duration of each place (ti) is labeled consistently with the temporal conditions specified by the rule for generating view V1. The generation of this SMPN is based upon the predicate *P* of the semantic operator *S*. The sub-view SV5 can be represented by a node of the R-graph, which is labeled as N5 in Figure 8.14. Clearly, the SMPN embedded in this node serves as its node semantic function (*ns*).

In order to build, the final view, V1, we also need to generate a sub-view (SV4) for the test completion date. In this sub-view, the test completion date is tested against a target date which is the program month M1. Subsequently, SV4 is aggregated with SV5 using the aggregation abstraction IS_PART_OF, to build the final view V1. This abstraction provides a logical AND relation for the two sub-views SV4 and SV5. To manage these two new views (SV4 and V1), we generate two additional nodes, N4 and N6 for the R-graph, as shown in Figure 8.14. The node semantic function of N6, representing view V1, corresponds to the specification of the object-oriented abstraction used for its generation. In this case, the arcs from nodes N4 and N5 to N6 has arc semantic function (as) given by the IS_PART_OF association.

The other two views, V2 and V3 and their R-graphs are constructed in a similar manner. Figures 8.15 and 8.16 depict the detail of these graphs and the specification of their node semantic and arc semantic functions in terms of semantic operators, Q and S, and object-oriented abstractions.

Collectively, V1, V2 and V3 can be further grouped together, to provide a global quality view V. For V, the views V1, V2 and V3 can serve as different types of perceived quality views for the software product. The grouping of V1, V2, and V3 to generate V can be carried on the basis of another object-oriented abstraction, known as specialization (IS_A). Accordingly, V can be represented as a node (N14) of the R-graph, with the abstraction IS_A and its inclusion function serving as its node and arc semantic functions, respectively. This is depicted in Figure 8.17.

It can be noticed from this example, that the proposed R-graph based formalism provides a powerful methodology for quality and risk management for software development process. Although, for these examples only four metrics are used, one can build arbitrary views of quality

and risk by choosing any set of metrics. We can also observe that the proposed formalism can be applied to any data model, since the graph-based spatial primitives for operator Q, do not assume any specific data model. However, as we have mentioned earlier, use of an E-R model for the software metrics data greatly facilitate management of evolutionary aspects of the data.

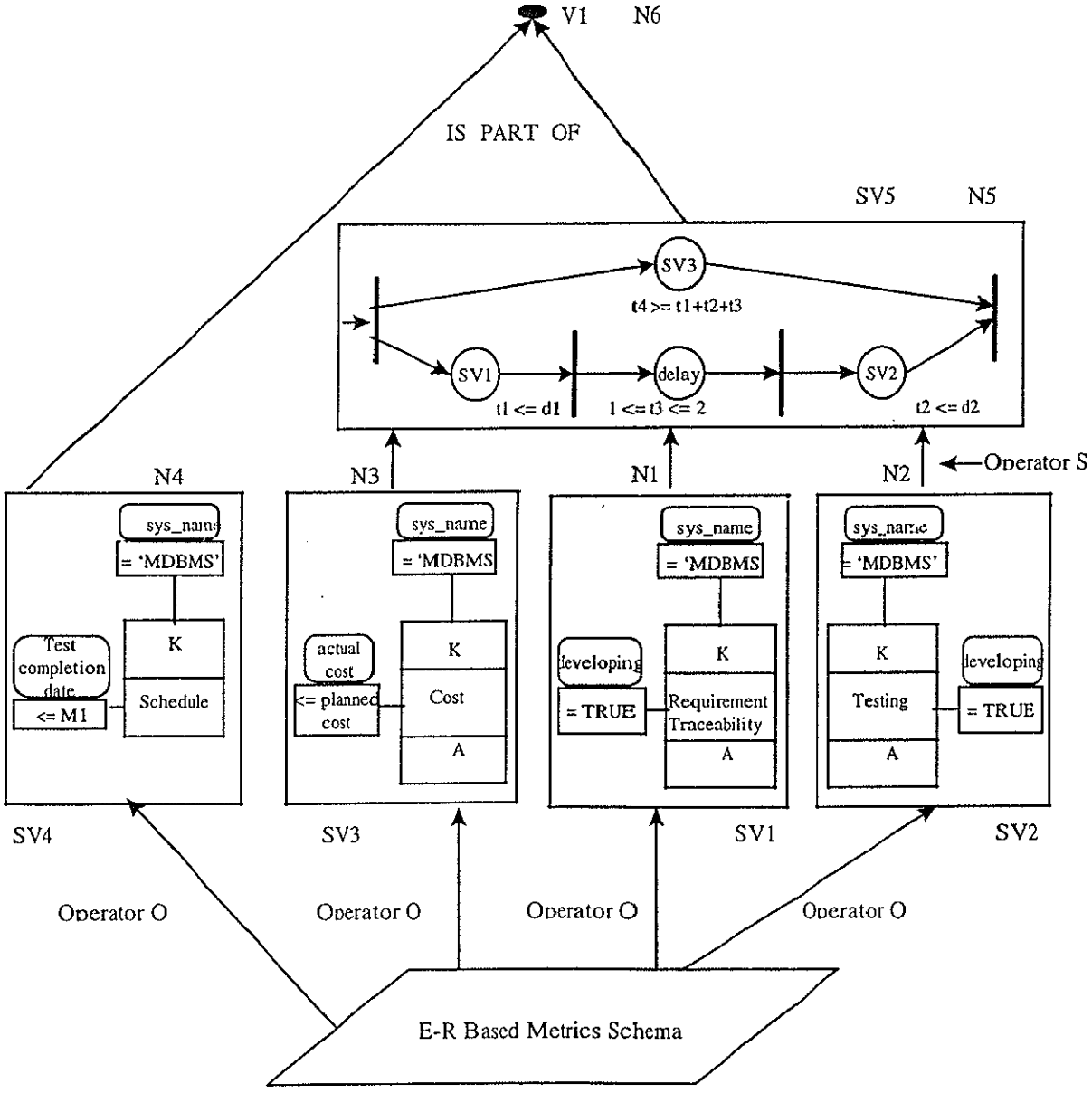

Figure 8.14: An R-graph to Build Quality View V1

(Detail of nodes is obtained using ZIN operator of the R-graph)

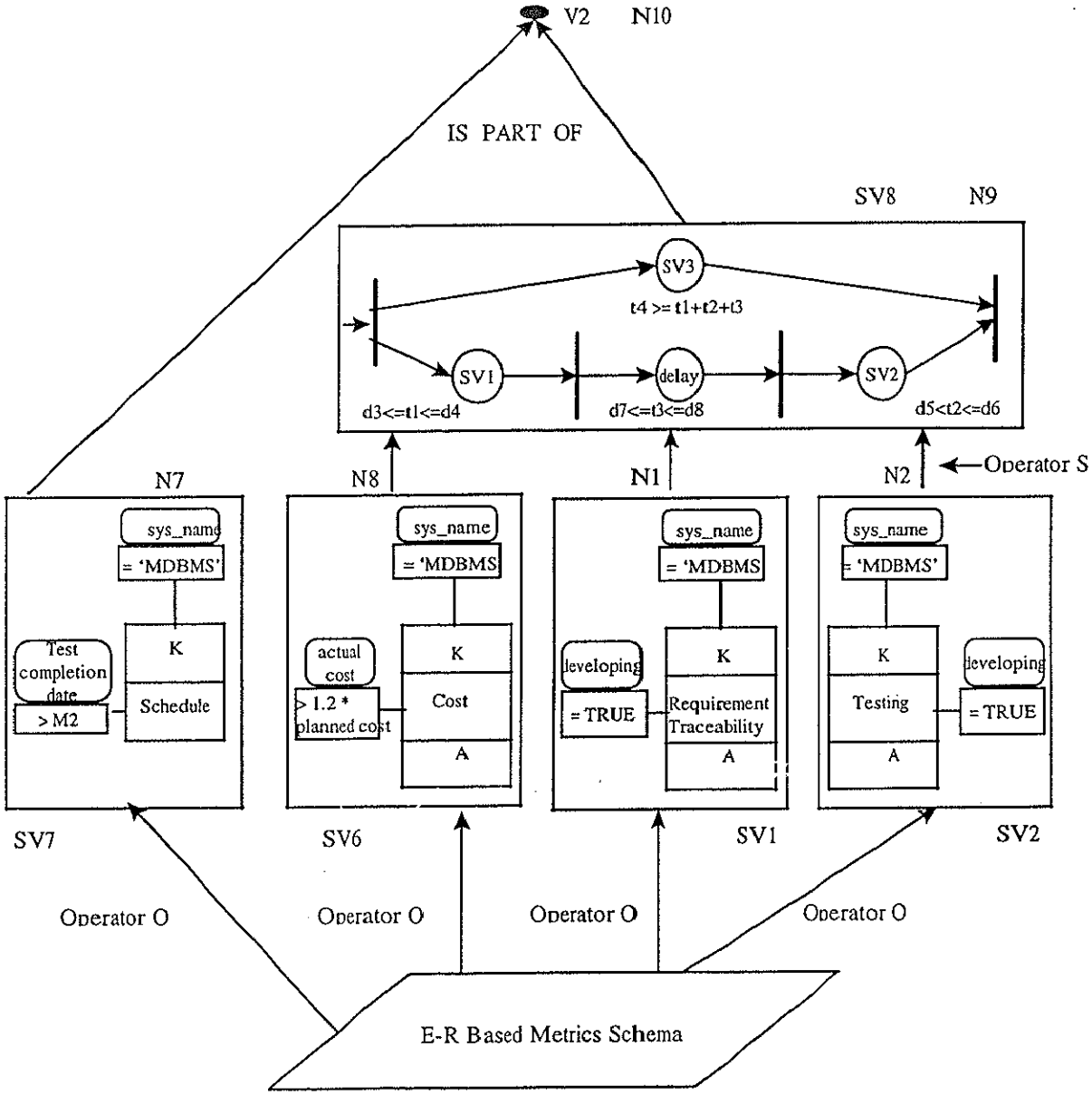

Figure 8.15: An R-graph to Build Quality View V2

(Detail of nodes is generated using ZIN operator of the R-graph)

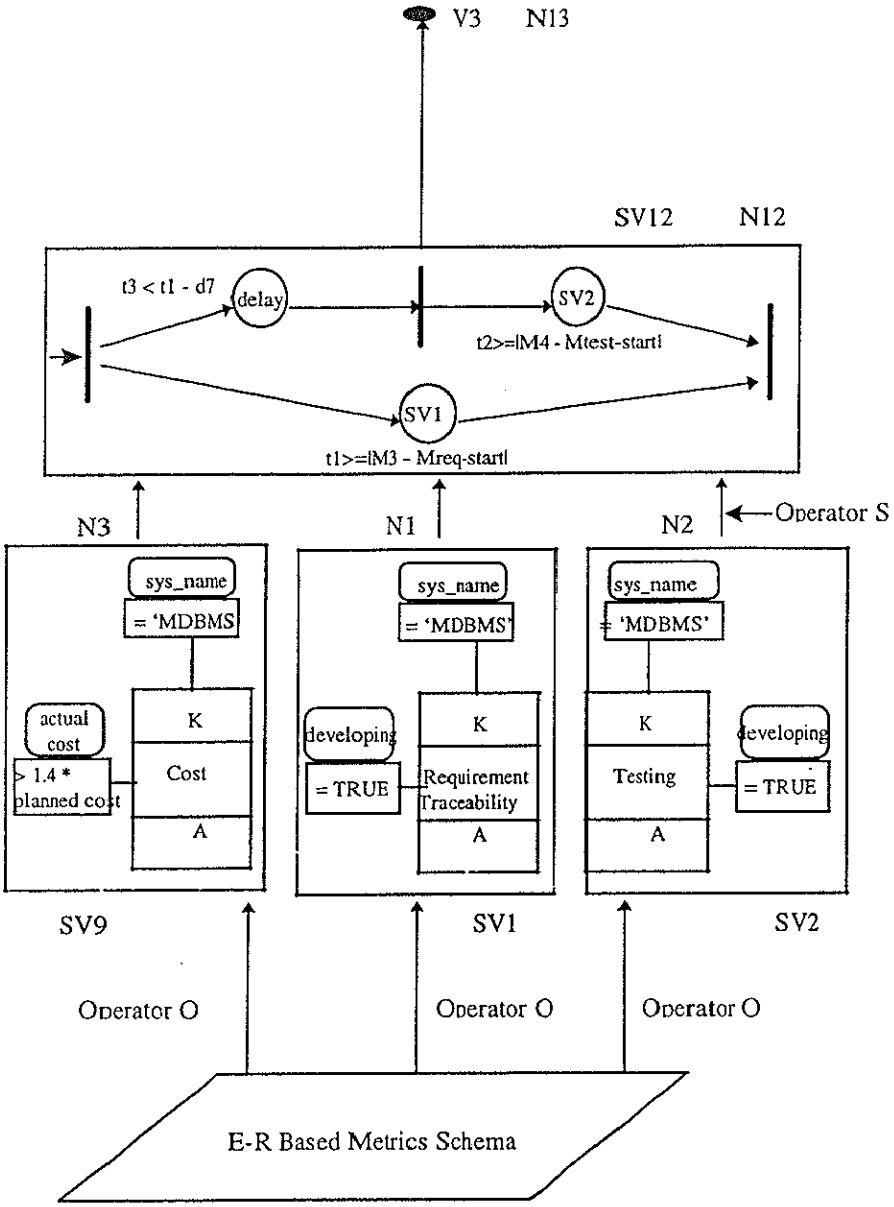

Figure 8.16: An R-graph to Build Quality view V3

(Detail of nodes is generated using ZIN function of the R-graph)

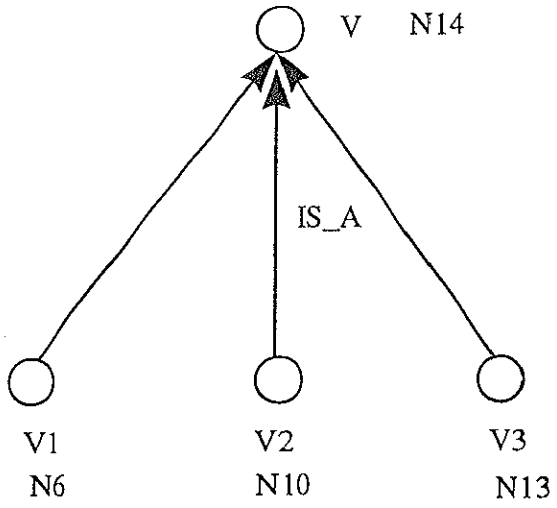

Figure 8.17: An R-graph to Support the Global Quality View V

8.6 An Architecture for Software Metrics Database Management System

We have compared software metrics queries using a number of prominent data models, namely, the relational data model, the object-oriented data model, and the graph data model. We observe that each model has its associated advantages and drawbacks with respect to different classes of metrics queries, and that no model is perfect. This can cause the designer of a metrics database a great amount of mental calisthenics to model his database in terms of the model being used. This can have the detrimental effect of detracting metrics analysts from the appropriate and timely use of database technology. In our opinion, to ameliorate this situation what is needed is the development of a data model suited to the needs of metrics queries, rather than trying to force-fit metrics queries to the extant data models. However, rather than re-inventing the wheel, the correct approach to this task is to take a combination of the best features of different data models, and add what is still needed, to develop a suitable data model for a software database management system. In the following, we first summarize the results of our findings and then present a list of requirements for a software data model.

8.6.1 Summary of Findings

Both practitioners and researchers in the area of software metrics often take a certain data model (say, the relational model) for granted and then try to phrase their queries to suit the peculiarities of the model. More often than not, they try to limit their queries to what is supported by the specific data model. Therefore, since the relational model in its pure form handles recursion poorly, i.e. only by embedding it in a host programming language, both practitioners and researchers attempt not to pose any queries that require recursion on the relational data model.

Our analysis of the suitability of some prominent data models for fulfilling the data management needs of software metrics queries can be summarized as follows:

1. For the most straightforward queries, the relational data model is adequate;
2. Any query that results in trend analysis or is schedule-related will invariably involve a temporal component; and currently we work around the deficiencies of the relational data model which updates data in place by adding a date field. A better solution would be to use temporal data models;
3. The relational data model handles recursion and transitive closure poorly. A good example discussed in this document are queries involving software modules. Since software modules may also contain other software modules, any query involving software modules would include some form of recursion, such as the following:
   a) How completely is software module "XYZ" tested?
   b) How many lines of code are there is software module "XYZ" and all its constituent modules?

4. Both the graph data model and object-oriented model support recursive queries such as the above; and in addition, handle regular queries well. However, they have steeper learning curves and are less commercially available.

5. The object-oriented data model appears to support most of the features required of MPMS, but query languages are complex, non-standard and difficult to learn;

6. Since each data model has its associated advantages and drawbacks with respect to its use in MPMS, the future appears bright for an extension of the relational model to support features such as recursion and temporal data. There is already some work undertaken by various researchers to incorporate extensions to the relational model.

8.6.2 Requirements for a Software Metrics Data Model

Based on our analysis, we propose the following set of requirements for a software metrics data model:

1. the model should be able to support complex relationships between data elements, e.g. compositions and hierarchies;

2. the model should support a high-level, preferably declarative, query language;

3. the model should have built-in support for temporal concepts;

4. the model should support recursive query processing and graph traversal operations;

5. the model should provide built-in support for metrics concepts like 'project', ' module', 'measurement', etc., as well as querying on metrics concepts such as 'critical path', 'project cost', etc.;

6. the new model must be a synthesis of the useful features of existing models, with additions where necessary. This 'evolutionary' approach, vs. a 'revolutionary' one where

everything is started from scratch, is more likely to be acceptable to the practicing community;

7. the new data model must be easily integrated with a host programming language, i.e. ideally the user should not have to change the mental model when moving from the application code to DBMS access; and

8. the model should lend itself to an efficient implementation, especially since to the metrics analyst a DBMS is but a tool, and an inefficient and poorly designed one is but a hindrance to his overall goal, which is metrics analysis.

8.6.3 The Proposed Approach

From the summary and the list of requirements above, we can observe that existing data models together can provide a number of features that are desired in an ideal software metrics data model. Specifically, one can draw upon the declarative querying features of SQL, the temporal component of temporal data models, recursion capability of GDL as demonstrated by the RGF, and flexibility and support for complex relationships from object models. However, two main requirements still remain. First is to allow a built-in support for metrics data and associated analytical methodologies, and the second is to provide a seamless integration between the host language needed to support database applications and metrics data models. The issue of efficient implementation is a crucial requirement. Figure 8.18 shows the architecture of the proposed approach, which we consider a 'synthesize-and-extend' approach.

The component labeled 'Metrics Class Library' is a library of objects which provide detailed analysis of metrics data for risk and quality assessment. A suite of analytical tools, such as the ones discussed in chapters 4, 5 and 6 of this thesis, form the kernel of this library. Using

an object-oriented approach allows embedding of data and computation in form of objects and can relieve users from knowing about the details of analysis, data format, etc.

The 'Temporal Class Library' maintains user's specified spatio-temporal semantics for supporting various levels of abstractions about quality and risk assessment of the software project. In particular, the semantics of SVI, SV2, ... etc., in Figures 8.14 - 8.16 are formulated and maintained in this library.

The component labeled 'Graph Class Library' is a library of objects which model high level operations on R-graph based objects, which are designed especially to support metrics analysis queries. Examples of such operations include breadth-first-traversal, depth-first-traversal, find-shortest-path, find-critical-path, find-all-preceding-nodes, find-all-following-nodes, etc. Having such graph analysis operations facilitate posing of above mentioned metrics queries. The proposed approach for implementing the new graph-based operations as a library enhances the existing capabilities of underlying DBMS rather than replacing the DBMS.

All the three libraries are integrated together to support complex queries. The degree of interaction among these libraries depends on the complexity of the query and the level of abstraction of information. For simple, queries each library can provide independent information. For example, for queries shown in Figure 7.11, objects from Temporal Class Library only are invoked. On the other hand, the generation of semantic view V2 of Figure 8.15 in response to a user's query requires interaction between both Temporal and Graph Class Libraries. Similarly, if the generation of a spatial view, say SV6, requires some analysis of Cost metrics data, then the appropriate analytical object from the Metrics Class Library is also invoked. We expect that a visual query interface can be utilized for the proposed system in order to allow users to formulate object-oriented semantic queries and access the proposed system. As mentioned in Sections

7.5.1.1 and 7.5.2.6, the proposed visual icons for spatial querying and SMPN provide an intuitive and natural graphical interface for expressing complex object-oriented semantic queries [PAU96a, PAU96c].

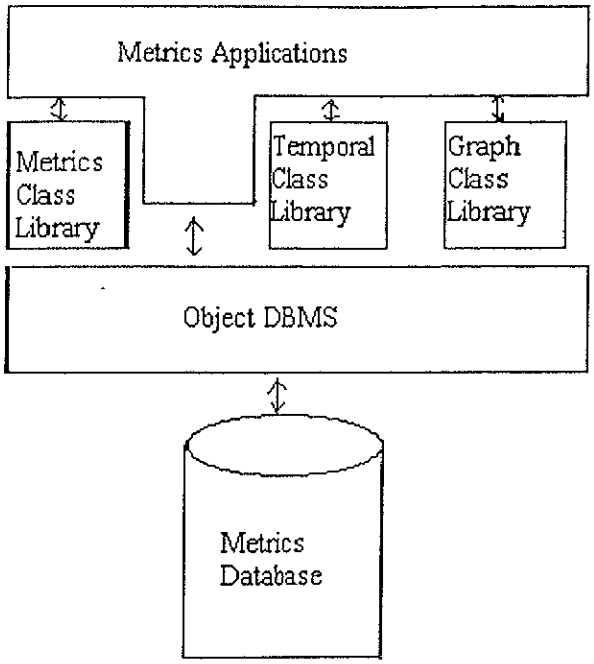

Figure 8.18: Proposed Approach to Software Metrics Data Management.

We propose the use of a commercial object-oriented DBMS, preferably one that uses an object data model standard such as ODMG93 [CAT94], as the base DBMS engine. The data definition and manipulation capabilities of the ODMG93 model's object query language (OQL) will automatically provide all of the object capabilities needed for the MPMS data model. Next, we propose to build a set of class libraries, as shown in Figure 8.18, that provide support for various kinds of other concepts needed. Thus, we have a set of classes for handling temporal objects and operations on them, and another for handling graph objects and their operations like recursion and traversals. Similarly, we have a set of classes that provide concepts specific to

metrics analyses. Each of these class libraries is built using the underlying base object model. *The base classes provided by ODMG93's object model, together with the class libraries for temporal, graph, and metrics concepts, make up the MPMS data model.* The metrics applications now have a whole range of concepts to use for modeling metrics data.

There are several advantages of the proposed approach. First, it takes advantage of commercial standards for object data models, which will be widely available and acceptable. Second, the concepts needed to implement the temporal and graph class libraries have been quite thoroughly investigated, and it is just a matter of implementing them in the base object model. It is quite possible that these might start becoming available as specialized class libraries from third party library developers. Third, the extensibility of the object model is a big benefit, especially if future needs arise that are unanticipated today, and lead to an extension of the model by adding new concepts. Finally, if the metrics applications are implemented in an object-oriented programming language, for which we advocate C++, there is a seamless integration between the application programming language and the MPMS data model query language.

Despite the many advantages of the approach touted above, there is one crucial element of Figure 8.18 that is not present today, and thus this system cannot be fully realized. This is the metrics class library. So far no research effort has focused on examining the specific data modeling needs of metrics databases, and *thus there does not exist any literature on the data model concepts needed to support the needs of metrics data modeling*. This is today an open research problem, and given its importance to achieving a full-fledged MPMS data model, we believe it should be addressed immediately by the data modeling research community.

8.6.4 Discussion

A metrics-based project management system integrates software metrics data, management queries based on the software metrics data, as well as their responses in a single location. Given the variations in projects, queries, metrics and physical databases to be used, the management system should be flexible from a user point of view, and also generic from a designer point of view. Two important steps in the design of a software metrics management system involve the determination of typical queries that may be posed by users, as well as the selection of the physical database for storing metrics and other project-related knowledge. In this paper we discussed the relative advantages and drawbacks of three selected physical database models, namely the relational data model, object-oriented data model, and graph data model, based on a variety of classes of metrics-based queries. We conclude that the choice of the appropriate data model varies according to the class of queries, and has strong implications on the ease and efficiency with which metrics-based queries may be posed and executed. Further, we observed that while none of the existing data models fulfills all the needs of software metrics, together they have a number of concepts that are useful. Thus, we believe there is no need to 'reinvent the wheel' unnecessarily and one must use relevant concepts where available. Based on our analysis, we proposed an approach to developing a data model for software metrics, which is based on the synthesis of a number of useful concepts from extant data models, with extensions where appropriate. Further, we advocate using a commercial standard object model, i.e. ODMG93, for the base DBMS, and a commercial standard object-oriented programming language, i.e. C++, for writing metrics database applications.

8.7 Conclusions

In this chapter we addressed the problem of metrics database evolution and have presented a framework for managing large software projects using a recursive graph (R-graph) model. The principal issues in database evolution are encountered in handling schema evolution. We modeled database schemas as R-graphs, and schema evolution as structural changes to an R-graph caused by R-operators. The R-graph formalism is well understood and has been used for modeling the top-down design process for domains as diverse as health informatics and petro-chemical plant design. Furthermore, a design tool called system for interactive design (SID) has been developed to facilitate the hierarchical design process using the R-graph formalism.

Kunii et al's research [KUN80,KUN90,BUC79] has shown that the R-graph formalism can be mapped into the binary association model. It is known that the binary association model is powerful enough to model the general relational model. Hence, by transitivity it follows that the R-graph model can be implemented on top of relational model. In the present work we have added one more link to this chain of argument. By introducing the mapping M, we have shown the hierarchical E-R model of schemas can be mapped into R-graphs. Thus, by transitivity we can argue that such schemas can be stored in relational databases. This makes their efficient implementation possible.

For temporal modeling and to express complex views and singularity conditions spanning various metrics, we have proposed a Petri net extension for the RGF. The extended RGF can allow users identifying risks and evaluating the quality of the software project at various levels of abstraction, It can allow representation of temporal views and singularity condition along with a graph-based representation of conventional database functionalities. In summary, the proposed

framework can provide a simple but comprehensive methodology for managing evolutionary aspects of software projects.

We conclude this chapter, be proposing an architecture for a software metrics database management system. The proposed architecture is built on the data analysis methodologies and data models proposed in this thesis.