Representing Software Engineering Models: The TAME Goal Oriented Approach

Markku Oivo and Victor R. Basili

Abstract— **This paper describes a methodology as well as a knowledge representation and reasoning framework for top down goal oriented characterization, modeling and execution of software engineering activities. A prototype system (ES-TAME) is described which demonstrates the underlying knowledge representation and reasoning principles. ES-TAME provides an object-oriented meta-model concept in order to provide effective support for tailorable and reusable software engineering models. It provides the basic mechanisms, functions and attributes for all the other models. It is based on inter-object relationships, dynamic viewpoints and selective inheritance in addition to traditional object-oriented mechanisms. Descriptive software engineering models (SEM's) include representations for basic software engineering activities like life cycle models, project models, resource models, design methods, quality models etc. They are controlled and made operational by active GQM models which are built by a systematic mechanism for defining and evaluating project and corporate goals and using measurement to provide feedback in real-time. A rule-based data-driven mechanism is defined for constructing and instantiating generic GQM templates into hierarchical GQM models. Support for the RT-SA/SD method is used as a case study of modeling the design phase of real-time software development.**

Index Terms— **ES-TAME system, goal/question/metrics paradigm, improvement paradigm, inheritance, inter-object relationships, knowledge-based techniques, object-oriented methods, reuse, rule-based techniques, software engineering, software modeling, TAME-project.**

I. Introduction

THERE is a great deal of software engineering research going on, i.e., people are building technologies, methods, models, etc. However, this research is mostly bottom-up, done in isolation. What is needed is a top down framework in which research can be focused, logically and physically integrated to produce quality software productively, and evaluated and tailored to the application environment.

TAME [4] is meant to serve as a framework for research and development activities by providing an integrating umbrella for various software engineering research projects, offering a focus and a laboratory environment for experimentation, and supporting the efficient transfer of technology into practice. It is an attempt at defining a measurement-based closed-loop process for software development and maintenance. TAME's specific goals are to provide a framework for 1) defining an integrated set of measurable software process and product models and goals relative to the project and the organization, 2) provide a quantitative basis for selecting the appropriate methods and tools and tailoring them to the needs of the project and the organization, 3) support the evaluation of the quality of the process and product relative to the specific project and organizational goals, and 4) provide an organizational structure to support building, analyzing, refining, and using experience models. The key components upon which TAME is based include an evolutionary improvement paradigm tailored for the software business, called the Quality Improvement Paradigm [2], a paradigm for establishing project and corporate goals and a mechanism for measuring against those goals, called the Goal/Question/Metric Paradigm [4], and an organizational approach for building software competencies and supplying them to projects, called the Experience Factory [5].

The *Quality Improvement Paradigm* (QIP) is defined by the following steps.

- Planning: an iterative process involving characterizing the current project and its environment, setting the quantifiable goals for successful project performance and improvement over past performance, and choosing the appropriate process model and supporting methods and tools for this project.
- Execution: a closed-loop project cycle that involves executing the processes, constructing the products, collecting and validating the prescribed data, and analyzing it in real-time to provide feedback for corrective action on the current project.
- Analysis and Packaging: a post mortem analysis of the data and information gathered to evaluate the current practices, determine problems, record findings, and make recommendations for future project improvements, and a packaging of the experience gained in the form of updated and refined models and other forms of structured knowledge gained from this and prior projects and the storing of the packages in an experience base so it is available for future projects.

The *Goal Question Metric Paradigm* (GQM) is a mechanism for defining and interpreting operational and measurable software goals. It combines models of an object of study, e.g., a process, product, or any other experience model and one or more focuses, e.g., models aimed at viewing the object of study for particular characteristics that can be analyzed

Manuscript received December 1, 1991; revised June 30, 1992. This work was supported in part by Air Force Grant AFOSR 90-0031, Technical Research Centre of Finland, Tekniikan Edistamisaatio Foundation, and Tauno Tonningin Saatio foundation. Recommended by M. Jarke and A. Borgida.

M. Oivo is with the Technical Research Centre of Finland, Computer Technology Laboratory, Oulu, Finland.

V. Basili is with the Department of Computer Science and the Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742

IEEE Log Number 9202818.

from a point of view, e.g., the perspective of the person needing the information, which orients the type of focus and when the interpretation/information is made available for any purpose, e.g., characterization, evaluation, prediction, motivation, improvement, which specifies the type of analysis necessary to generate a GQM model relative to a particular environment.

The *Experience Factory* is a logical and/or physical organization that supports project developments by analyzing and synthesizing all kinds of experience, acting as a repository for such experience, and supplying that experience to various projects on demand. It packages experience by building informal, formal or schematized, and productized models and measures of various software processes, products, and other forms of knowledge via people, documents, and automated support.

The Experience Factory requires an experience base that supports accumulating experiences (learning) via recording and analysis of experience, off-line generalizing and tailoring of experience, and formalizing of experience, storing experience models in a variety of modeling notations that are tailorable, extendible, understandable, flexible and accessible, and accessing and modifying packages of experience to meet the needs of the current project (reuse). An effective experience base must contain an accessible and integrated set of analyzed, synthesized, and packaged experience models that captures the local experiences.

A. Requirements Overview

To formalize the QIP, each of the various steps needs to be better defined and integrated. The experience base acts as the mechanism of information and integration. These next items implicitly define the genuine requirements for the experience base:

- We need to build and store models of various software engineering experiences that characterize the project and the organizational environment, e.g., products, processes, resources.
- We need to integrate these models based upon the various relationships between them, e.g., what resource model is appropriate for a particular class of products.
- The model definitions need to be able to evolve, be modified or refined based upon learning, e.g., we need to be able to modify a resource model by adding new project data, refine a process model by recognizing a different set of activities that need to be performed based upon a specific project characteristic.
- The model definitions need to be instantiated with specific project characteristics, e.g., we need to instantiate the parameters of a resource model based upon actual project values, and map process activities into a process model according to the actual life cycle model.
- Models need to be classified and subclassified based upon type so that the appropriate types of models can be combined in a GQM, e.g., that product evaluation qualities such as coupling or cohesion are applied to products defined in the appropriate notation such as RT-SA/SD.
- Some models may need to be applied to available data, so the experience base must permit access to a data base containing the current project and historical data, e.g., an evaluated GQM model.
- We need to initialize and evolve various versions of the experience base for different organizations.

Fundamental to the TAME concept is the ability to formally define software engineering models so that they can be integrated for evaluation, reconfigured based upon particular project needs, and stored for future use. This requires a more formal definition of the components of the QIP, including the GQM and the definition of an experience base that contains useful models and supports the configuration of models as needed.

Knowledge-based techniques have shown promise in modeling various aspects of software engineering. Some projects have taken the ambitious goal of providing knowledge-based support across the software life cycle, e.g., the knowledge-based software assistant or apprentice approach [7], [12], [18], [25], and knowledge-based support for team-work [13]. Other projects have concentrated on specific software development activities such as requirements specification and design [20], [23], [24], and knowledge-based modeling and simulation [22]. Knowledge-based techniques are also important from the project management viewpoint [26]. In this paper we describe a methodology and a knowledge representation and reasoning framework for the experience base [5]. In addition to being a comprehensive framework for modeling various aspects of software engineering, it is also a basis for automating the use of quantitative methods. However, in this paper we will concentrate on the modeling issues. In this section we have described the fundamental requirements of TAME and the experience base. In Section II we will present a meta-model concept which implements the basic requirements and supports tailorable and reusable models. It provides a foundation for software engineering models (SEM's) and GQM models. The knowledge representation mechanisms for SEM's are discussed in Section III. The modeling techniques are based on an enhanced set of inter-object relationships, dynamic viewpoints, and selective inheritance. Finally, Section IV presents a goal oriented top-down method and a rule-based construction tool for building active GQM object hierarchies which are used to control and make the mostly passive knowledge of SEM's operational.

TAME is a very large concept and too huge a task to be implemented in one step. We have implemented a domain specific version, called ES-TAME, to provide more comprehensive support for building embedded systems. It uses RT-SA/SD (Real-Time Structured Analysis and Design [33]) method as a case study of modeling the design phase of building software for embedded systems. This aspect of the system is related to systems like IDeA, which supports analysis and design phases in the context of a knowledge-based refinement paradigm [20], and ESPRIT project ASPIS, which provides support for the early phases of the software life cycle [24]. An extensive discussion of the knowledge representation issues of the RT-SA/SD method can be found in [23] which describes the basis of the modeling principles for the various RT-SA/SD design

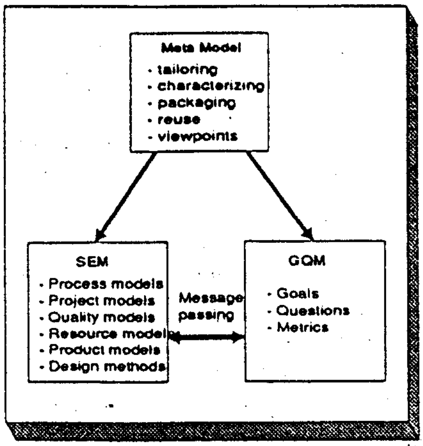

Fig. 1. The basic models.

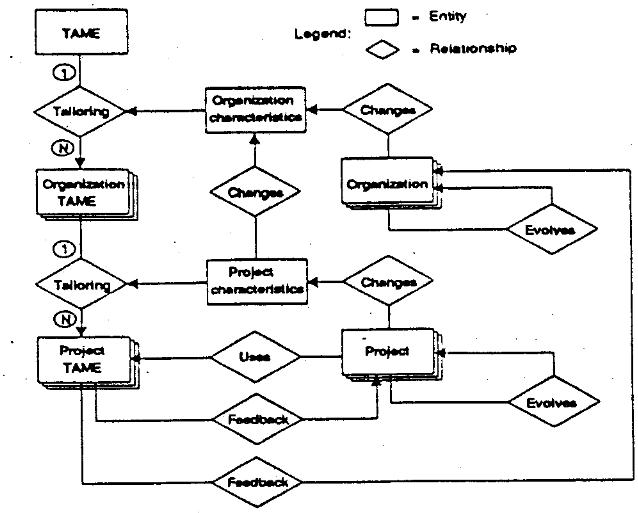

Fig. 2. TAME instances tailored for various needs.

elements and how to provide knowledge-based support for the requirements specification and design method.

II. Meeting the Requirements

Obviously, the previous requirements call for numerous models for representing all the relevant aspects and knowledge needed to build a viable software engineering environment. However, despite the large variety of requirements we can identify several principles, attributes and functionalities which are common to most of the models. Consequently, we introduce a meta-model concept for defining an overall knowledge representation and reasoning framework for all the models. It is an object-oriented model which specifies the basic mechanisms, functions and attributes for all the other models. The meta-model includes support for characterizing, planning, and packaging activities as well as user interface issues. It provides all the necessary functions and attributes for building and maintaining the actual tailorable models. It can also be used like a software process meta-model to describe and build customized software process models using pre-defined components [22], [35]. Essentially our meta-model is a virtual model which has to be refined and augmented to implement the TAME models. Furthermore, it provides a uniform mechanism to link the models to various additional tools like spreadsheets, project management tools, database management systems and metrics software, and combines their data under a rigorous object-oriented formalism.

We have classified our models into two categories: software engineering models (SEM's) and GQM models (Fig. 1). Both are generic models which are defined using the meta-model as a basis for their specification. SEM's include representations for the basic software engineering activities like life cycle models, project models, resource models, design methods, quality models, etc. They involve mostly descriptive knowledge which is known and available during the characterization and planning activities of a project life cycle. GQM's involve mainly procedural knowledge which is used to make the descriptive knowledge of SEM's operational. They manipulate and use the knowledge of SEM's in setting goals, answering questions and collecting data.

By making a clear distinction between the SEM's and GQM's we can create a highly modular system architecture and achieve far better support for representing knowledge in a reusable form. The descriptive knowledge of SEM's can be created and maintained without having to know how they are used and made operational by the more complicated GQM's. On the other hand, the constructing of GQM's is simpler because the user can concentrate on the essential features of GQM's without having to worry about the vast amount of knowledge involved in the SEM's.

The meta-model with the SEM and GQM models constitute a generic meta-tool environment which has to be tailored for each organization and project (Fig. 2). Note that Fig. 2 does not imply any static relationships. The tailoring diamonds stand for concurrent processes that relate the basic TAME environment to various corporations and in each corporation to various projects. All the entities in the figure are constantly evolving as we learn more about the changing environment and requirements. New features are introduced and existing ones are modified by evolving the objects and their relationships inside the TAME meta-tool.

Fig. 3 describes the overall architecture of ES-TAME. It depicts the usage of ES-TAME to support the design activities of software development. Other activities and their corresponding documents would be represented in a similar way. For example, testing would have its own user interface controlled by the viewpoint manager and test documents would be stored in the Model Base in an analogous way as the design documents. The main parts of ES-TAME include the Model Base, Model Management, User Interface Manager, Reuse Repository and Analyzing and Packaging Unit. This paper focuses on the most essential concepts of the Model Base, Model Management, and User Interface Manager. Furthermore, we demonstrate the modular Designer Interface with a support system for the RT-SA/SD method.

Model Base implements the main knowledge representation techniques and models in the system. It includes all the Software Engineering Models (SEM's) and design documents

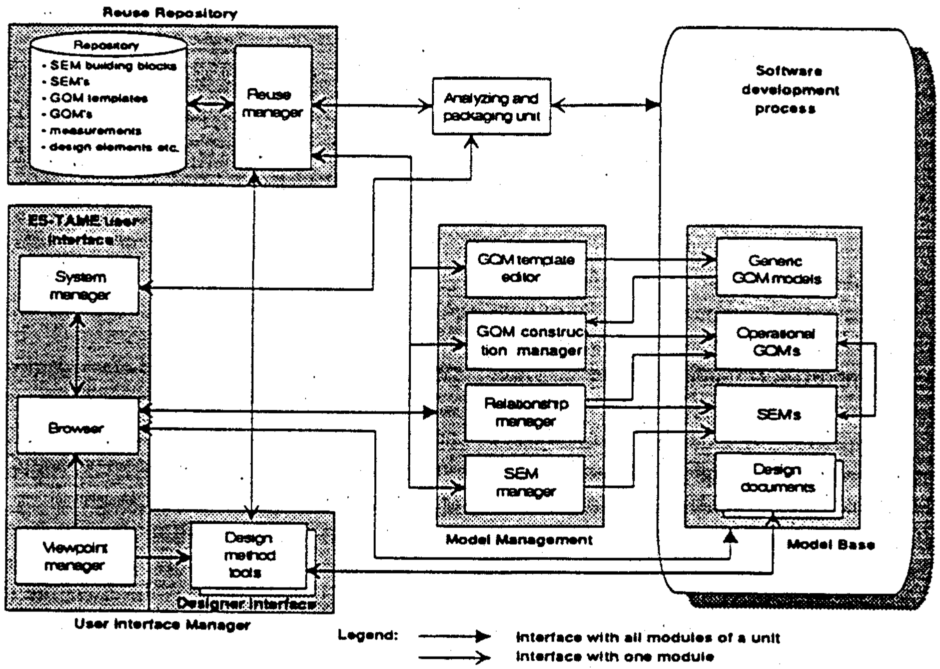

Fig. 3. ES-TAME architecture for design support.

(see Section IV) as well as GQM models (see Section V). SEM's and GQM's interact in terms of both relationship links between models and by GQM's using and making the descriptive knowledge of SEM's operational. Because the Model Base includes elements that are developed and modified during the software development, it is considered as a part of the development process that includes additional elements and activities.

SEM's and GQM's are created and managed by a set of tools in the Model Management unit (GQM Template Editor, GQM Construction Manager, and SEM Manager). The relationships between the various SEM and GQM models are established and maintained by a Relationship Manager.

The user interface consists of two main units. The first unit, the ES-TAME User Interface, provides the main functions which are relatively independent of the design methods. It includes three modules. The Browser offers graphical tools to view and manipulate the various relationship hierarchies. The System Manager controls the analysis of the software development process and the packaging of the results into the experience base. Viewpoint Manager provides several different perspectives to the system using the Browser and the Designer Interface as tools for viewing the system. The second main unit is the Designer Interface. It is a plug-in module that can be changed to other design method tools without much effect on the rest of the system.

The Reuse Repository consists of a Reuse Manager that stores and retrieves SEM's and GQM's in the Repository. The Analyzing and Packaging Unit measures, collects, and packs data from the software development process. The knowledge representation principles of these systems are essentially analogous to the principles presented in this paper. A discussion of the reuse management and measurement issues related to TAME can be found in [3], [4], [6].

We have built an ES-TAME prototype system to demonstrate the ideas of this paper. The run-time environment can be a standard 386 PC with a minimum of 6 MB of RAM. The development tools include Kappa™ expert system development environment [17], ToolBook™, Excel™, and C, all running under Windows 3.0.

III. Software Engineering Models

The SEM's provide the essential means for characterizing the current project and its environment as well as representing the knowledge involved in them. Their underlying object-oriented structure supports tailorability and reusability. SEM's consist of mainly passive objects that serve as a basis for project execution and are governed by the active objects of GQM's.

A. Modeling Mechanisms

In order to have a better understanding of the underlying modeling principles of ES-TAME, we will first study the major features needed to model the SEM's. The model building is based on *object-oriented modeling, inter-object relationships*, and *a dynamic viewpoint* mechanism with a highly *selective inheritance*. Object-oriented modeling is the basis of most of the technical topics. Since the basic object-oriented techniques are well documented in the literature [8], [11], [21], [30], [34] they are not described explicitly in this paper. Inter-object relationships are used to construct models consisting of various types of objects and define the relationships between them. Dynamic viewpoints with selective inheritance are used to view the models from various perspectives and to control their inheritance via the relationships.

1. Inter-Object Relationships: In addition to the basic Is-A hierarchy found in object-oriented systems, the meta-model

```
FOR EACH attribute inherited from the Old-Parent in the Object
    IF attribute has a local value in the Object
    OR attribute is selected by the user to be inherited THEN
        Make attribute local in Object and maintain the local values
    ELSE
        Remove attribute from Object
Change IS-A link of Object to the New-Parent
```

provides a set of predefined relationships for building various model hierarchies and networks. By offering a limited collection of relationships we can maintain consistent models and provide automated support for managing the models. The basic inheritance hierarchies or lattices (Smalltalk-80 [11], Eiffel [21], KEE [17], C++ [31], etc.) are not enough for modeling SEM's and GQM's. On the other hand, using attributes[1] to store relationships without a rigorous set of rules can easily lead to a spaghetti-like relationship network which is very difficult to maintain in a large modeling application. With a well-defined set of relationships we can build models which are flexible and yet manageable.

The relationships offered by ES-TAME are Is-A/Children, Instance-Of/Instances, Part-Of/Has-Parts, Compatible-Objects, Dynamic-Attribute, and a Counterpart relationship. The principle of having all the relationships in pairs is important because of the emphasis of using ES-TAME to build reusable objects. Each object can be taken out of its original hierarchy and subsequently be stored into the reuse repository for future use. It must retain knowledge not only of its descendants in the hierarchy but also of its possible ancestors, parts if it is a composite object, to which context it belongs and information on how its relationships can be used in new applications. It is a reusable object with relationships as connectors that can plug into other objects both upwards and downwards in any of the relationship hierarchies.

The relationships are created and managed internally by the Relationship Manager module in the Model Management unit (Fig. 3). The graphical user interface to the relationship is provided by the Browser which is controlled by the Viewpoint Manager.

The *Is-A/Children* and *Instance-Of/Instances* relationships are similar to the standard class/subclass and class/instance relationship offered by most object-oriented and frame-based systems [8], [10], [11], [21]. They are the only relationships that employ the conventional inheritance in ES-TAME. However, we do not provide traditional multiple inheritance. Instead we provide dynamic linking of the Is-A relationships which considerably enhances the capabilities of the traditional inheritance. Each object can have a potential Is-A relationship to several super classes but only one of them is active at any point in time. All the attributes of the active super class are inherited, whereas inheritance via the other Is-A relationships is highly selective and must be explicitly defined. This is the foundation of the dynamic viewpoints described in Section 3.1.2. The children relationship is used to catalog all the subclasses or instances of a given class.

[1] We will use attribute as a collective synonym for instance variables of objects and slots of frames.

Despite the rather novel approach to inheritance we still consider our system a class based system as opposed to an object-based system such as Self [32]. Self has the notion of prototype metaphor instead of classes and variables. It searches values for slots using parent pointers instead of inheriting according to a class hierarchy. In ES-TAME each class and instance always has an active parent class and inherits all the attributes of that parent. If we did not have the dynamic linking mechanism, our strategy could be considered a single inheritance approach similar to what is used in Smalltalk, KEE, and Eiffel. However, the dynamic linking mechanism of the Is-A relationships provides multiple viewpoints to object models. Furthermore, it facilitates context sensitive behavior for objects by changing relationships on the fly and inheriting new attributes and functionalities from the new parent. The old relationships can be restored without any loss of information due to the dynamic relationship manipulation.

The fact that we do not currently use multiple inheritance does not mean that we would argue that it is useless in the context of software modeling and construction. On the contrary, it is easy to identify numerous cases where objects are conceptually related to more than one parent and multiple inheritance is useful. The multiple viewpoints and selective inheritance offer many of the benefits of multiple inheritance. We can avoid the well-known name collision and repeated inheritance problems involved in multiple inheritance ([8], [30], [34]) because we have only one parent link or viewpoint active at any point of time. The optimal strategy for ES-TAME would be to use mainly our current mechanisms and carefully use multiple inheritance in selected cases.

The dynamic manipulation of the Is-A links is done at the meta-model level in order to assure the propagation of the viewpoint to all the pertinent elements. During a link change, all the application level local values of an object, i.e., instance values that are not inherited from the old parent, must be maintained in the object in order to be accessible also under the new parent. All the attributes selected by the user to be inherited and ported under the new parent must also be maintained. We can recover these values if the old parent becomes the current parent again. Even if the old parent is deleted or is not accessible anymore, we can maintain the attributes which were initially inherited from the deleted parent. This is useful for reusing objects in new systems where the parent may not be included in the new systems. Attributes without a local value and those that are not explicitly defined to be maintained by the user can be removed in the object level because if the IS-A link is changed to point back to the old parent the attributes are automatically inherited again from the old parent.

This mechanism avoids the problem of information maintenance involved in coersion in schema evolution for object-oriented databases [28]. In schema evolution the coersion mechanism discards information during type changes if their definition is not included in the new type. In the dynamic linking of the IS-A relationship ES-TAME maintains all the information even if the definition of the attribute is not included in the new parent. On the other hand, dynamic linking of the IS-A relationship is most often used as a run-time feature and the relationship can also be changed back to any of the previous parents, as opposed to the one way evolution of versions in object-oriented databases [28]. The algorithm at the top of the page describes the principle of the attribute manipulation of an object *Object* during dynamic changing of an Is-A link from *Old-Parent* to *New-Parent*.

The ***Part-Of / Has-Parts*** relationship pair is used to describe compound objects. A composite object is a collection of objects that can be managed as a single entity. Our composite object is roughly comparable to the related concepts of some other object-oriented languages and object-oriented database systems [19], [30]. However, it is important to notice that we do not require a composite object to be instantiated in a top-down fashion starting from the compound object and then instantiating the components [1], [30]. Due to the emphasis on reusable components and parallel design in large projects, we don't have any restrictions on the order in which compound objects are built and instantiated. For example, we may want to design a reusable door control unit that can be integrated, using a Part-Of relationship, into several different types of elevator control systems that use this type of door. Each component of a composite object can be independently defined in its own class hierarchy and used as a component in several compound objects (e.g., a door control can be Part-Of a simple elevator control system for low-rise buildings as well as a Part-Of a high speed elevator control system). This allows us to define objects in their most natural logical class hierarchies and use them in various compound objects without having to define the similar objects in different compound objects. Part-Of relationships can also be used for performing system level operations on compound objects and for broadcasting messages to all the components of a subsystem. For example, if a successful development team gets a raise in salary we can automatically propagate the change to every SEM object representing a member of the team via the Part-Of relationships and consequently automatically update the relevant cost estimation model. This can't be done using the Is-A hierarchy because team member objects and team objects are defined in different class hierarchies. Team members belong to teams (Part-Of relationship), they are not subclasses of teams (Is-A relationship). Furthermore, if we want to change an attribute in all the modules of an elevator control system we can automatically propagate the change to every object representing the module via the Part-Of relationship (e.g., DoorControl is a Part-Of the ElevatorControl).

The ***Compatible-Objects*** relationship is used to describe objects that can be used together, e.g., the function point method might be compatible with MIS projects but not with real-time projects. This information is used to ensure that the objects we include from the meta-model in the company and project level models are compatible with each other.

Compatible-Objects provide an important mechanism for reuse-oriented model building (see Section 3.3) and system design. By navigating in the compatibility network and picking from the list of compatible objects for each element, we can configure a system using the most appropriate objects from the reuse repository. This mechanism results in a procedure for building a hierarchical system design, starting with the root of the design model tree and successively adding nodes selected from the compatibility network.

The ***Dynamic-Attribute*** provides a way of associating an object's attribute with the attribute of another object [17]; e.g., if we have estimated the number of source lines (SLOC) in the product characterization and given it as an attribute to the product model, we can link the corresponding SLOC attributes of the resource estimation and defect slippage models to the product model's SLOC attribute. Thus we maintain the SLOC estimate in one place only and changing the estimate can be automatically updated in the other models. This would be impossible to implement with multiple inheritance because these models are conceptually totally different and belong to different class hierarchies. The dynamic attribute mechanism is similar to the inheritance of slot values in Self [32] in the sense that it can be changed on-the-fly to access information from a different object. However, the dynamic attribute is used only to access information in single slots and the main inheritance mechanism for classes and objects in ES-TAME is implemented via dynamic manipulation of the IS-A relationship.

The ***Counterpart relationships*** are provided for creating various domain specific relationships and links between objects. They are normally used to define relationships between objects which are used in the same context to build a larger scheme. The counterpart relationship has some similarities with Booch's association relationship which denotes a semantic connection among otherwise unrelated classes [8]. Counterpart relationships are also used for establishing links between SEM's and GQM's. Using counterpart relationships the user can create, edit and browse any kind of application specific hierarchies. Naturally, each object can also be viewed from all the standard viewpoints provided by ES-TAME. We could, for example, establish a counterpart relationship between data flow diagram models and design level coupling models. They are independent objects but they are both used in the same context in assessing the quality of the system design. These relationships are used to manage the interconnections and interactions between the related objects, including message passing, constraint reasoning and value propagation.

2. Dynamic Viewpoints and Selective Inheritance We introduce a mechanism for attaching a generic viewpoint mechanism to any of the models or model components and their relationships. It is provided by the Viewpoint Manager which controls the Browser and the Design Method Tools according to the choice of the user (Fig. 3). Normally each user has a default viewpoint to the system. For example, the system designer is mainly interested in the design models and their features, and views other models as different perspectives of systems, subsystems and objects. On the other hand, management is more interested in budgets, resources, cost, project schedule, etc. and can have models tailored according to the management perspective. The manager may impose a schedule for the whole project using the project model. The system designer may estimate cost and effort from the viewpoint of design models by taking a cost estimation viewpoint on the design models and using the tools of the cost estimation model on the design models.

Each model or component of a model is defined as an object. Each object is defined with attributes that are relevant to itself as a class or as an instance of a class. For example, a data flow diagram is defined with its relevant attributes in the context of structured analysis and design. However, as a part of the meta-model it inherits the capability of having several viewpoints. If the user wants to examine the quality aspects of a particular data flow diagram, he/she would change the viewpoint of that object to a particular quality model. As a result, the data flow diagram would be dynamically linked to that quality model and inherit its features and functionality. Note that this is different from multiple inheritance. Linking is dynamic and inheritance is applied only while the object is linked to the viewpoint. When changing the viewpoint again, only those attributes that are instantiated during the old viewpoint, i.e., those that have been modified or given local values, are ported into the new viewpoint.

One of the advantages of the dynamic viewpoint mechanism and selective inheritance is it limits the amount of information in each object. Because most of the objects can be viewed from a variety of predefined perspectives (quality models, cost estimation, testing, design, implementation, etc.), use of straightforward multiple inheritance or implementing the attributes and functions as part of the objects would yield excessive information and obscure the user's understanding of the object itself and its conceptual relationships to other objects. With dynamic viewpoints we can focus our attention on the features that are relevant to our current interest.

Our approach differs from the multiple interfaces defined in some object-oriented languages. For example, Snyder proposes two different interfaces to classes, one for public use and one for subclasses [29]. Others have proposed restricted subsets of operations for different users to facilitate multiple views to the same object [14]. Dynamic viewpoints and selective inheritance are primarily a means for changing run-time behavior, object attributes, or even class hierarchies. Changing a viewpoint adds new methods and attributes to an object and may remove old ones if they are no longer needed. This is a basic difference from controlling visibility in Trellis/Owl [27] or accessing an object in [14] and [29].

B. Principles of SEM's

The main purpose of the SEM's is to formalize various software engineering experiences and their relationships. The experience or knowledge associated with SEM's is recorded in various forms, including model level and object level descriptive knowledge and attributes, inter-class relationships, rules, procedures, spreadsheets, and diagrams. The recorded experience can be accessed from several viewpoints both by browsing the meta-model and by general purpose queries. Informal knowledge is accessed mainly by browsing whereas access to formalized knowledge is more automated. SEM's are internally created by the SEM Manager and they are maintained in the Model Base (Fig. 3). Their relationships to the GQM's are maintained by the Relationship Manager. The user can use the Browser and the Viewpoint Manager to create, modify and view the SEM hierarchies.

Basically, the SEM's are built as class/subclass hierarchies using the Is-A relationship. Descriptive knowledge is stored in the attributes of the objects and can be shared among objects using inheritance or the Dynamic link relationship. Descriptive knowledge includes mainly textual, graphical and numerical characterization of the SEM objects. The Is-A classification hierarchy is extensively enhanced using the Part-Of, Compatible-Objects, and Counterpart relationships. These links often have no specific value in the generic classes. They may have constraints for attribute or link values. For example, a link might be allowed to be established only to subclasses or instances of certain classes. The undefined attribute values and links are defined in the lower levels of the object hierarchies, most often at instance level. Rules, procedures, spreadsheets, and diagrams are defined with methods that either fully implement the functionality or provide an interface to a tool that offers the service.

The meta-model defines the building blocks and their relationships for creating the actual models and environments for each project. For example, the waterfall model can be constructed using the Is-A and Part-Of relationships (Fig. 4). It is defined as a subclass of a generic life cycle models class with Part-Of relationships constrained to possible process activity classes (analysis, design, coding, test, maintenance, etc.) or their descendants which are defined as their own independent object models. The process activity objects can be used as building blocks for constructing different life cycle models. A tailored waterfall model is defined in three phases. First we define a customized waterfall model that is refined as a subclass or an instance of waterfall models. For example, we might specify the model as having separate phases for product design and detailed design instead of having only one design phase. As a second step, in the design activities, we might choose to represent the data structure, software architecture, and procedural design in terms of entity relationship diagrams, data flow diagrams, state transition diagrams, and structured English, respectively. As a third step the tailored process activities[2] are defined to be parts of the customized waterfall model. Thus the customized waterfall model is a compound object that is a subclass of waterfall models and its component objects are subclasses of the process activities. This same approach applies for most of the SEM models. The meta-model defines independent reusable building blocks and mechanisms for customization and interconnection. The actual environment is established by tailoring the classes and defining the relationships described in the previous section.

[2]These process activities can be reused as parts of other life cycle models, either as is or modified for the particular model.

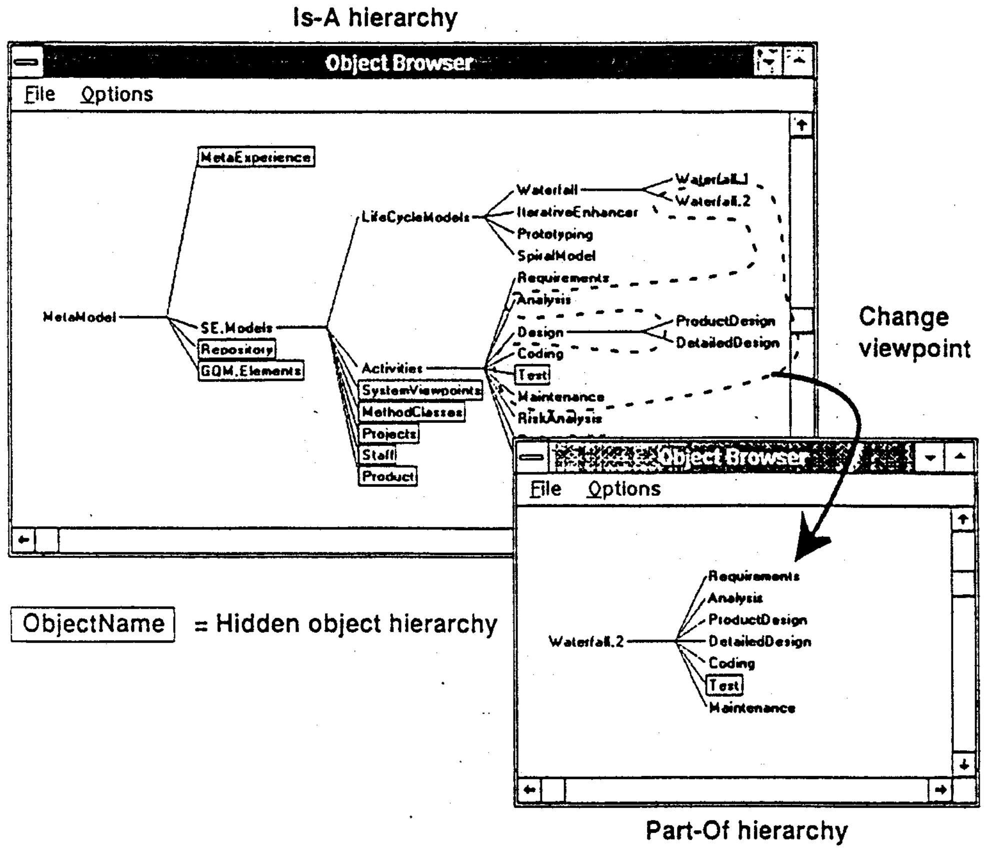

Fig. 4. Is-A and Part-Of relationships of the waterfall model.

C. Planning and Characterizing

This section provides an overview of how the meta-model supports the planning steps of the QIP. It does not, however, include the detailed goal construction techniques which are described in Section IV. The characterizing is based on refining and augmenting the generic SEM objects and components as well as building larger models and compound objects by combining the template objects with pertinent relationships.

The meta-model can be tailored for various organizations by refining and augmenting the objects, relationships and, more importantly, by using several viewpoints into the system and combining the model hierarchies according to the interest of the user. Selective inheritance can be used for picking up relevant attributes and functionalities from various object classes without the burden of inheriting too much information from several sources. Initial tailoring of the meta-model is performed during the project planning activities. The meta-model can be further modified at any point during the project.

ES-TAME encourages reuse of previously defined models and objects in the planning and characterizing phase as well as in building the actual application software. Using a compatibility relationship network it can suggest objects and object hierarchies from the experience base that can be used in building and tailoring the models for the current project. With a sufficiently large reuse repository this works like a chain reaction.

We normally start the planning from a previously built meta-model that is tailored either for the company or for the type of project that we are going to run. Thus, the starting point is a template model that has components with several compatibility relationships whose values are constrained to classes or class hierarchies which can be directly linked to this component. These compatible components are offered by the Reuse Manager (see Fig. 3). By retrieving a component from the repository we obtain a component that can suggest other components from the repository that are compatible with the current one. These in turn can suggest new components and so on. The procedure is like building a tree with nodes that can further suggest new nodes or sub-trees below themselves. The tree can be any of the relationship hierarchies supported by ES-TAME. We can, for example, start with a node and pick up from the list of potential Part-Of components building a Part-Of hierarchy. At any moment we could change our approach and start picking up from the list of potential subclasses of a class level component. This procedure can be repeated until we have exhausted the list of potential components from the various compatible components.

D. Modeling for the Design Phase of Project Execution

Execution in the context of the QIP is defined as a closed-loop process of executing the processes, constructing products, collecting and validating data, and giving feedback in real-time. This section describes the aspect of defining the SEM's to support these activities. We will use design activities of the project life cycle as an example of the modeling support for project execution. The process of making these models executable also involves the GQM models.

In order to be able to support the activities after the initial project planning phase, we have to support the methodology chosen by the user to model the system being built. Normally the design involves the decomposition of the system into subsystems and further into more detailed subsystems in a hierarchical manner. Our approach can be applied for functional decomposition as well as object-oriented decomposition. The main assumption is that the method supports some mechanism for decomposing the system into subsystems or class hierarchies. In functional decomposition, the design is internally represented with Part-Of relationships by ES-TAME.

For our first prototype of ES-TAME we have chosen RT-SA/SD (Real-Time Structured Analysis and Design [33]) as the case study for the system modeling and implementation oriented models. The modeling and support of the RT-SA/SD method is based on the work in Prospex project reported in detail in [23]. However, the viewpoint mechanism provides a significant improvement over the basic frame-based modeling in Prospex. Consequently, most of the principles in the following examples can be applied to other methods, often simply by replacing the name RT-SA/SD with the corresponding method name.

RT-SA/SD serves as a starting point for software developers to view various aspects of the product and process via multiple viewpoints of the ES-TAME models. The amount of information associated with each RT-SA/SD element in a real world ES-TAME would be overwhelming (both RT-SA/SD related information and more general information related to each sub-system in the RT-SA/SD models, including quality attributes, cost attributes, schedules, implementation, testing, etc.). Multiple viewpoints of the system help avoid cognitive overload of the user. For example, the user can choose to view the RT-SA/SD model from the point of view of testing and access information of the testing methods, test data, test results, etc., which are relevant to the particular RT-SA/SD model. Multiple viewpoints can be active at the same time providing features like checking the quality model and testing features of a specific RT-SA/SD model.

The entity relationship diagram in Fig. 5 shows the relationships of the various viewpoints of a subsystem in an imaginary elevator control system. It describes the viewpoints to a FancyDoor control system in an elevator control system and its relationship to the simplified product model. FancyDoor control subsystem has an Is-A (subclass) relationship to the RT-SA/SD diagram element which in turn has an Is-A relationship to the more general Method element. The Method element object has a property of being able to provide several viewpoints to itself. Each viewpoint (resource model, quality model, etc.) is dynamically linked to the Method element providing the user 1 to n different viewpoints into the Method element. The FancyDoor control inherits all the different viewpoints from the Method element via Is-A relationships and consequently has a capability of providing several viewpoints to itself.

The left side of the diagram illustrates how the FancyDoor element is related to the simplified product model of the

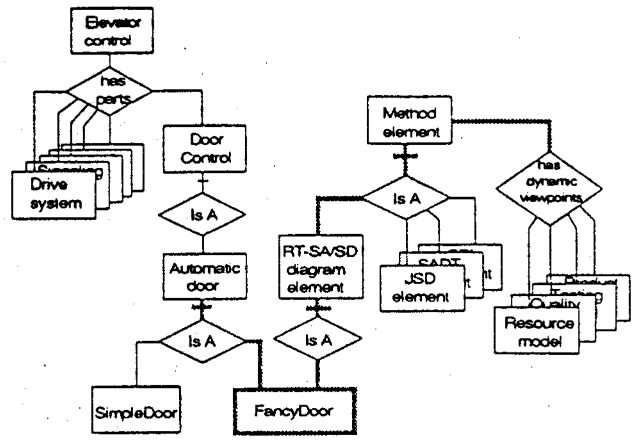

Fig. 5. The relationship between the product hierarchy and multiple viewpoints.

elevator control system. FancyDoor is conceptually a subclass of a more general class of Automatic doors which in turn is a subclass of a Door control class. The Elevator control has several parts, one of which is the Door control class.

Linking the different viewpoints into the generic method element provides an important independence of the design method. The mechanism for changing viewpoints is defined and implemented in the generic method element object and inherited by the elements in different methods. The first ES-TAME prototype can be enhanced by linking corresponding elements from other design methods (JSD, SADT, SDL, etc.) to the generic method element thus providing similar viewpoints for each method. The enhancement is implemented by creating an object-oriented model of each method (conceptually similar to the RT-SA/SD model). It will then inherit all the viewpoints, attributes, and functionalities of the generic method element that are further refined to meet the needs of each method. We have demonstrated this idea with a design level quality model example that was initially built for RT-SA/SD and was used for JSD with very few modifications.

IV. GQM's

GQM's are the primary means of making our models operational. They provide information to the analysis and packaging activities using data collection, metrics, analysis, and packaging procedures incorporated in the objects either as methods or as interface links to the appropriate tools. GQM's are the main execution and analysis "engine" of the system.

GQM models are an organized collection of active objects that can perform functions on their own without explicit activation by the user or other objects. SEM's, on the other hand, are a collection of passive objects that are used to formalize and package software engineering knowledge. They perform functions only when activated by the user or by GQM objects.

The construction of GQM's consists of two concurrent processes, which we will call P1 and P2. P1 involves the creation, tailoring, and reuse of GQM template objects to build a GQM model base which is used by the software development project (GQM Template Editor in Fig. 3). P2

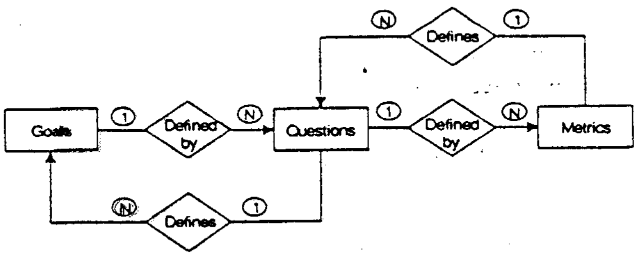

Fig. 6. Entity relationship diagram of goals, questions, and metrics.

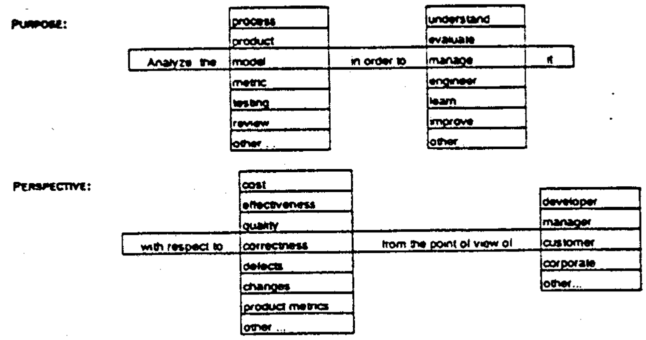

Fig. 7. Simplified GQM templates.

involves the rule-based construction and instantiation of the GQM model base into a collection of operational GQM's (GQM Construction Manager in Fig. 3). These processes are concurrent rather than sequential, supporting the iterative development of GQM models. The first process is actually a part of the characterization and planning phase of the QIP. It involves the construction of GQM object templates and the creation of a generic model using template objects as building blocks. The second process includes refining and augmenting the often incomplete objects and instantiating them into operational objects.

GQM models are basically compound objects consisting of goals, questions, and metrics that are normally modeled as structured objects. The top-down construction of a GQM model starts with a formulation of an overall top level goal object. It can be subsequently defined by lower level sub-goal objects. The goal objects at the lowest levels in the goal hierarchy are characterized by attaching question level attributes to the objects yielding more specific goal/question objects, needed for achieving the goals. Question objects inherit the goal definition from their ancestors in the hierarchy. Each goal can generate one or more questions. Each question in turn is defined by one or more metrics. Metrics can be either automated measurement, data collection and interpretation procedures, or interactive information gathering sessions with the user. They can also be combinations of these activities. Metrics inherit both goals objects and question objects. Each question can be used in the definition of several goals and each metric can be used to answer several questions (Fig. 6).

Each object is defined using a template driven editor (see Fig. 7). The templates have a predefined structure but the interpretation of the attributes can be different for various objects and object hierarchies. Moreover, attribute definitions and template values can be inherited via the GQM hierarchy. A free-form way of defining goals, questions and metrics is also provided by ES-TAME but automated support for them is limited.

GQM construction and instantiation is the final step of planning and characterizing before the project execution. The purpose of these activities is to perform the final refinement and augmenting of the GQM's in order to make them operational (the process P2 of GQM construction).

The formal semantics of the GQM's allow us to infer the underlying functionality of each attribute of a GQM model or its component. This feature is used extensively to assist in the process of constructing goals, questions and metrics. The user starts with a goal template (see Fig. 7), refining and augmenting its attributes according to the needs of the project. Each new piece of knowledge prompts the system to determine if it can automatically deduce the necessary elements for the definition of the goal or the subsequent questions and metrics. Thus, the process of iteratively defining goals, questions, and metrics can activate functions that are associated with the particular object. If a GQM model is not yet fully defined, a user input into the template can activate the automatic generation of questions for goals or metrics for questions. If a particular GQM model is fully specified when the attributes are filled in, the template can automatically activate the corresponding data collection procedures that interact with the user and the SEM's.

The construction of the GQM's is performed using a rule-driven GQM generator. It is a tool that uses forward chaining, data-driven rules to help in the process of creating GQM's. When the user creates a goal he/she uses an editor to fill in and instantiate a goal template. The semantics of the templates are defined by rules. When an attribute of a template is filled in, it can fire one or several rules. These rules can infer more information and fire additional rules in a forward-chaining manner. Fired rules can generate more information based on the given initial data, they can fill empty attributes of the template, suggest or generate questions based on the data, and so on.

The same rule-driven construction principle applies to creating questions. Normally the user has to be involved in defining the questions, although in some cases the questions can be created automatically based on the goal by the GQM generators rule-base. The construction includes choosing, filling in and instantiating question templates based on information from the goal definition.

Finally, the user chooses and defines the metrics and data collection procedures with the help of the GQM rules. This procedure uses the SEM's in the meta-model in an object-oriented way. For example, if the cost of a sub-system is not known and it is needed to answer a question of a GQM then a message is sent to the corresponding sub-system object in a SEM. The SEM object calculates the cost, possibly asking further questions of the user if sufficient information is not available. On the other hand, if the cost is already available in the SEM, either by previous calculations or as previously given by the user, then the method in the SEM object simply returns the value of the cost to the GQM. Furthermore, the mechanism for calculating the cost depends on the context of the object. It may be calculated by summing up the cost of sub-systems, based on recorded and user provided data, or estimated by a given formula (e.g., Cocomo). In all cases, the GQM model is the same and does not have to know anything about how the corresponding SEM gets the value. The differences are defined in the corresponding SEM's (product model, project model, cost model, etc.) and can be hidden from the objects who ask for the information.

GQM's and SEM's typically communicate using the metrics level objects of the GQM's. Goal and question level objects normally refer to lower level objects in the GQM hierarchy to obtain information. The links between the SEM's and GQM's are established during the construction and instantiation of the operational GQM's, either automatically or with user assistance. The rules and constraints for the relationships are defined in the GQM template objects by the person who is responsible for the ES-TAME system.

For example, consider a GQM that needs information on the experience of the manager in order to evaluate the development team, i.e., the GQM involves several questions and one of them is "What is the experience of the manager?" The GQM is initially constructed from a template object (using process P1 of GQM construction) which defines that its manager link (defined as a Counterpart relationship) must point to an Instance-Of managers class in the SEM's. When a GQM object needs the experience information for the first time, it doesn't know who the manager is. However, based on the manager link constraint, it knows that it must be an instances of the manager class. Consequently, it asks for the name by giving a list of instances of the manager class to the user. When the user selects the name of the manager in the menu, the system automatically initializes the Counterpart relationship between the GQM and the selected manager object in the SEM and all future references to the manager use this link. When the link is established, the GQM object sends a message to the manager instance asking for the experience of the manager. If the information is not available in the manager object, it activates characterization procedures which provide the user with a form editor for defining the necessary facts for the manager object. When the characterization is done, the manager object returns the experience data to the GQM object. Naturally, the manager instance saves this new information from the form editor during the characterization process and can immediately return the experience data, as well as any other characteristics defined in the characterization, without any user interaction during the next requests.

The communication between the GQM's and SEM's is analogous to the previous example when the information flow is reversed, i.e., when the GQM's are manipulating the information of the SEM's. For example, when a metrics method of a GQM has measured the error density it will send the results as a message to the corresponding quality model (SEM). The establishment of the link is also similar. The template objects provide the allowable quality models which can be linked to the particular GQM and the final establishment of the link is done either automatically or interactively during the construction and instantiation of the GQM's.

By having separate SEM's and GQM models we can have a clear interface between the general principles of creating GQM's and the project specific information defined and stored in the SEM's. All the complexities and implementation details can be hidden in the corresponding models.

The actual use of fully specified GQM's is performed by backward chaining rule-based reasoning. The goal part of a GQM is used as a high level goal[3] in the backward chaining process. The reasoning process establishes questions and metrics as backward chaining sub-goals. When metric level goals are established in the reasoning they activate the corresponding metrics procedures.

V. Conclusions

We have described a methodology, a knowledge representation, and a reasoning framework for the top down goal oriented characterization, modeling and execution of software engineering activities. This is done in the context of the Quality Improvement Paradigm (QIP), an evolutionary improvement paradigm tailored for the software business defined by three steps: 1) planning, 2) execution, and 3) analysis and packaging. The Experience Factory concept provides an environment for the organizational approach for building software competencies and supplying them to projects.

A prototype system (ES-TAME) is described which demonstrates the underlying knowledge representation and reasoning principles. Support for the RT-SA/SD method is used as a case study of modeling the design phase of building software for real-time systems. ES-TAME provides an object-oriented meta-model concept which supports tailorable and reusable software engineering models. It provides the essential mechanisms, functions and attributes for building other models. Modeling is based on inter-object relationships, dynamic viewpoints and selective inheritance in addition to the traditional object-oriented techniques. This extended object-oriented approach has proven to be effective in implementing the two types of highly modular and tailorable ES-TAME model categories: descriptive SEM's which consist of mainly passive objects and procedural GQM's which consist of active objects. By defining SEM's and GQM's as two clearly separate models, we can create a highly modular system and a far better support for representing knowledge in a reusable and easily maintainable form.

SEM models include representations for the basic software engineering activities. They involve mostly descriptive knowledge defined in the characterization and planning activities of a project life cycle. SEM's are used and made operational by the active GQM models that are defined by a systematic mechanism for defining and evaluating goals and using measurement to provide feedback in real-time. GQM's provide a paradigm for establishing project and corporate goals and a mechanism for measuring against those goals. A rule-based forward chaining mechanism provides a user-

[3]Notice the dual meaning of the word goal. It is used to refer both to the goal part of a GQM and to the goal of a backward chaining reasoning process. The context of the word should clarify the meaning.

friendly, incremental, and flexible way of constructing the GQM templates into GQM object hierarchies.

The current implementation of ES-TAME provides a framework for creating and maintaining tailorable SEM's and GQM's. It demonstrates the main knowledge representation and reasoning mechanisms of the Model Base, Model Management, and ES-TAME User Interface unit including the Viewpoint Manager (Fig. 3) and an interface to Design Method Tools. However, it does not include automatic support for the Analysis and Packaging Unit and these functions must be carried out manually. Furthermore, the Reuse Repository needs additional research to be useful in practical environments.

Potential directions for future research include comprehensive support for building and managing the reuse repository, using reverse engineering techniques for creating and maintaining the experience base for an organization, using case-based reasoning techniques for packaging information into the experience base and supporting GQM management with deep knowledge.

ACKNOWLEDGMENT

The authors wish to thank L. Briand, G. Caldiera, and R. France for their constructive and valuable comments. The authors also thank the editors of this issue and the reviewers for their constructive comments and suggestions that significantly helped us to improve this paper.

REFERENCES

[1] J. Banerjee *et al.*, "Data model issues for object-oriented applications," *ACM Trans. Office Information Systems*, Jan. 1987.
[2] V.R. Basili, "Quantitative evaluation of software engineering methodology," Keynote address, First Pan Pacific Computer Conf., Sept. 1985 (also available as *Tech. Rep., TR-1519*, Dept. Computer Science, Univ. Maryland, College Park, July 1985.
[3] V.R. Basili and H.D. Rombach, "TAME: Integrating measurement into software environments," *Computer Science Tech. Rep. Series*, (CS-TR-1764), Univ. Maryland, College Park, June 1987.
[4] ——, "The TAME project: Toward improvement-oriented software environments," *IEEE Trans. Software Eng.*, vol. SE-14, pp. 758–773, June 1988.
[5] V.R. Basili, "Software development: A paradigm for the future," in *Proc. Thirteenth Annual Int. Computer Software & Applications Conf.*, pp. 471–485, Sept. 1989.
[6] V.R. Basili and H.D. Rombach, "Support for comprehensive reuse," *IEE Software Engineering J.*, Sept. 1991.
[7] K. Bennet and D. White, "The knowledge-based software assistant, Overview," in *Proc. Second Annual Knowledge-Based Software Assistant Conf.*, pp. 13–24, Jan. 1988.
[8] G. Booch, *Object-Oriented Design with Applications.* Redwood City, CA: Benjamin/Cummings, 1991.
[9] C. Chambers *et al.*, "An efficient implementation of SELF, a dynamically-typed object-oriented language based on prototypes," in *Proc. ACM Conf. on OOPSLA*, pp. 49–70, Oct. 1989.
[10] R. Fikes and T. Kehler, "The role of frame-based representation in reasoning," *Commun. ACM*, vol. 28, pp. 904–920, Sept. 1985.
[11] A. Goldberg, *Smalltalk-80: The Language and its Implementation.* Reading, MA: Addison-Wesley, 1983.
[12] C. Green *et al.*, "Report on a knowledge-based software assistant," Kestrel Institute Technical Report, 1983.
[13] U. Hahn *et al.*, "Teamwork support in a knowledge-based information systems environment," *IEEE Trans. Software Eng.*, pp. 467–482, May 1991.
[14] B. Hailpern and H. Ossher, "Extending objects to support multiple interfaces and access control," *IEEE Trans. Software Eng.*, vol. 16, pp. 1247–1257, Nov. 1990.
[15] W.H. Harrison *et al.*, "Good news, bad news: Experience building a software development environment using the object-oriented paradigm," in *Proc. ACM Conf. OOPSLA*, pp. 85–94, Oct. 1989.
[16] IntelliCorp, KEE Software Development System User's Manual, 1986.
[17] IntelliCorp, 1986, Kappa System User's Manual.
[18] G.E. Kaiser, "Intelligent assistance for software development and maintenance," *IEEE Software*, vol. 5, pp. 40–49, May 1988.
[19] W. Kim *et al.*, "Composite object support in an object-oriented database system," in *Proc. ACM Conf. OOPSLA*, pp. 118–125, Oct. 1987.
[20] M. Lubars and M. Harandi, "The knowledge-based refinement paradigm and IDeA: Concepts, limitations, and future directions," in *Proc. Workshop on Automating Software Design, AAAI-88*, pp. 98–103, 1988.
[21] B. Meyer, *Object-Oriented Software Construction.* New York: Prentice Hall, 1988.
[22] P. Mi and W. Scacchi, "A knowledge-based environment for modeling and simulating software engineering processes," *IEEE Trans. Knowledge and Data Eng.*, vol. 2, , pp. 283–294, Sept. 1990.
[23] M. Oivo, "Knowledge-based support for embedded computer software analysis and design," Tech. Research Centre of Finland, VTT Publications 68, 1990.
[24] P. Puncello, "ASPIS: A knowledge-based CASE environment," *IEEE Software*, pp. 58–65, Mar. 1988.
[25] C. Rich and R. Waters, *The Programmer's Apprentice.* Reading, MA: Addison-Wesley, 1990.
[26] A. Sathi *et al.*,, "Representation of activity knowledge for project management," *IEEE Trans. Pattern Analysis and Machine Intell.*, vol. PAMI-7, pp. 531–552, Sept. 1985.
[27] C. Schaffert "An introduction to Trellis/Owl," in *Proc. ACM Conf. OOPSLA*, Sept. 1986.
[28] A.H. Skarra and S.B. Zdonik, "The management of changing types in an object-oriented database," in *Proc. ACM Conf. OOPSLA*, pp. 483–495, Sept. 1986.
[29] A. Snyder, "Encapsulation and inheritance in object-oriented programming languages," in *Proc. ACM Conf. OOPSLA*, pp. 38–45, Sept. 1986.
[30] M. Stefik and D. Bobrow, "Object-oriented programming: Themes and variations," *AI Magazine*, vol. 6, pp. 40–62, 1986.
[31] B. Stroustrup, *The C++ Programming Language.* Reading, MA: Addison Wesley, 1986.
[32] D. Ungar and R. Smith, "SELF: The power of simplicity," in *Proc. ACM Conf. OOPSLA*, pp. 227–241, Oct. 1987.
[33] P. Ward and S. Mellor, "Structured development for real-time systems," vol. 1, 1984.
[34] P. Wegner, "Concepts and paradigms of object-oriented programming," Expansion of Oct. 4 OOPSLA-89 Keynote Address, *OOPS Messenger*, vol. 1, pp. 7–87, Aug. 1990.
[35] J.C. Wileden, "This is IT: A meta-model of the software process," *ACM SIGSOFT Software Engineering Notes*, vol. 11, pp. 11–16, Aug. 1986.

Markku Oivo received the M.Sc. and Lic. of Tech. degrees from the University of Oulu, Finland, in 1982 and 1989, respectively.

From 1982 to 1986, he was with the Kone Corp. Research and Development Department, Helsinki, Finland, where he was involved in developing several real-time software applications, and in the research of industrial software engineering environments. In 1986, he joined the Computer Technology Laboratory (CTL), Technical Research Centre of Finland, and is currently head of the Knowledge Engineering Section. At CTL, he has been in charge of several projects developing knowledge-based systems and knowledge-based support for software engineering environments. In 1990 and 1991, he worked in the TAME project at the University of Maryland, College Park, where he was engaged in the research of knowledge-based support and modeling of software engineering activities. His main areas of interest include knowledge-based systems, object-oriented methods, and software engineering environments for embedded real-time systems.

Victor R. Basili (M'83–SM'84–F'90) is a Professor in the Institute for Advanced Computer Studies and the Department of Computer Science at the University of Maryland, College Park, MD. He was involved in the design and development of several software projects, including the SIMPL family of programming languages. He is currently measuring and evaluating software development in industral and government settings and has consulted with many agencies and organizations, including IBM, GE, and NASA. He is one of the founders and principals in the Software Engineering Laboratory, a joint venture between NASA Godard Space Flight Center, the University of Maryland, and Computer Sciences Corporation, established in 1976. He has been working on the development of quantitative approaches for software management, engineering, and quality assurance by developing models and metrics for the software development process and product. He has authored more than 90 papers. In 1982, he received the Outstanding Paper Award from the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING for his paper on the evaluation of methodologies. Dr Basili was Editor-in-Chief of the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING and was Program Chairman for several conferences, including the 6th International Conference on Software Engineering. He has served on the editorial board of the *Journal of Systems and Software*. He is a member of the Board of Governors of the IEEE Computer Society.