ELSEVIER

Decision Support Systems 17 (1996) 313–330

Decision Support Systems

A meta-model for formulating knowledge-based models of software development

Peiwei Mi *, Walt Scacchi

Information and Operations Management Department, University of Southern California, Los Angeles, CA 90089-1421, USA

Abstract

In this paper, we introduce a knowledge-based meta-model which serves as a *unified resource model* for integrating characteristics of major types of objects appearing in software development models (SDMs). The URM consists of *resource classes* and a *web of relations* that link different types of resources found in different kinds of models of software development. The URM includes specialized models for software systems, documents, agents, tools, and development processes. The URM has served as the basis for integrating and interoperating a number of process-centered CASE environments. The major benefit of the URM is twofold: First, it forms a higher level of abstraction supporting SDM formulation that subsumes many typical models of software development objects. Hence, it enables a higher level of reusability for existing support mechanisms of these models. Second, it provides a basis to support complex reasoning mechanisms that address issues across different types of software objects. To explore these features, we describe the URM both formally and with a detailed example, followed by a characterization of the process of SDM composition, and then by a characterization of the life cycle of activities involved in an overall model formulation process.

Keywords: Meta-modeling model composition; Software process modeling; Knowledge-based modelling; Knowledge-based models of software development

1. Introduction

Our contribution in this paper is to describe a knowledge-based *meta-model* representation formalism for building composite models of software development efforts. Our meta-model is based on a resource-centered ontology. It accomodates the formulation of large and complex models based on the composition of five principal classes of models appearing in our problem domain. This is the domain of *large-scale software development*. We have constructed and deployed a knowledge-based support system which utilizes this representational formalism in ways that support the articulation of models throughout the formulation life cycle we describe. In turn, our support system has been used to formulate, analyze, refine, and redesign various models of large-scale software development for our research sponsors. Some of these models have addressed software development projects where hundreds of software developers were employed (e.g., AT&T Bell Laboratories in Naiperville, IL, and Northrop-Grumman B-2 Division in Pico Rivera, CA) to build multi-million line software systems. In settings such as these, the (re)formulation of current/new models of large-scale software development was often done to identify new ways to systemically analyze the consistency, completeness, cost, resource dependen-

* Corresponding author. Tel.: 213 740 4782. Fax: 213 740 8494. E-mail: {pmi,scacchi}@gilligan.usc.edu

PII S0167-9236(96)00007-3

cies and conflicts of local software development efforts in order to make their organization and management more tractable and predictable. We have also found that with little effort, our approach can be adapted and applied to other decision-making and business process domains cite [1].

In the remainder of this paper, we describe our approach to the model formulation life cycle, meta-modeling, model composition. In the next section, we provide the background for our approach. This includes a short characterization of our focal problem domain, as well as related approaches and studies. It also describes our view of the model formulation life cycle, which positions meta-modeling as its driving activity. We next describe our meta-modeling representation and how it serves as a foundation for formulating models which are used to organize, guide, and manage large software development projects. We then follow with an illustrative example which highlights modeling details from the five component models that are part of a large model now used in a number of industrial firms. We also use this example to explain some of the critical details involved in model composition. Finally, we conclude and add some additional remarks regarding our current efforts.

2. Background

In the domain of large-scale software development, many new product, process, and decision support models have been proposed [2–11]. These models are used to facilitate different aspects of software development when employed within support systems we call "process-centered software development support environments" (PSDSEs). These models also facilitate the archiving and reuse of product, process, and decision-making artifacts [11,12] through sharing the object classes among software developers, as well as query and reasoning about relations between modeled objects and their attributes [7,9]. For example, Ramesh and Dhar [9] advocate the need for a PSDSE that records and organizes for retrieval the kinds of decisions made and rationales employed when building large software systems. Since different models and PSDSEs usually deal with only one particular aspect of software development, it is hard to compose and integrate them for use within a comprehensive or project-wide PSDSE. Part of the difficulty here arises from the adoption of different modeling approaches which emphasize representation of different classes of objects and their interrelationships. Another difficulty emerges when trying to address the formulation of large development models that must accomodate or be composed from independently developed component models. Finally, it is also difficult to put these various classes of objects and models together so that they can be interpreted or executed in one or more independently developed PSDSEs.

In order to better understand these issues, let us review some comparable research efforts, particularly as related to structural modeling, process modeling, meta-modeling, model construction and integration, and model management systems.

Geoffrion [13–15] is among those who can be noted for their pioneering of the computational rendering and use of structural modeling. His structures model targetted problem domains using *attributed directed graphs*, which in turn can be supported using entity-relation-attribute (ERA) based model management systems. However, it should be noted that ERA models and support systems can lack the representational flexibility and economy offered by *object-oriented models and support systems* [7,16,17], such as found in the use of inheritance mechanisms and object-oriented data management systems.

In the area of process modeling Jarke [18], Dhar [19] and their colleagues are among those that advocate the need *to model the processes associated with decision making and with the development of software-based information systems.* In particular, they advocate the use of knowledge-based process representation formalisms which enable process support systems to support complex reasoning, inheritance, and query-based computations. We concur with this direction for process modeling as a necessity, and we follow it in this paper and in related work elsewhere [7,20]. These knowledge-based formalisms provide for the representation and use of interrelated classes of attributed objects that are manipulated by rule-based inference systems.

Other work in the area of process modeling in support of the domain of software development has brought attention to the need and utility of *process*

meta-models and meta-modeling [7,20–23]. The purpose of meta-models is to provide a formal language or representational system for specifying an interrelated family (or genus [14]) of process models, or other models of software development objects. Our approach described in this paper employs a knowledge-based meta-modeling scheme.

Given that these modeling approaches are moving toward ever more powerful model representations, we next consider recent advances in model construction. Significant efforts in model construction, such as [24–26], emphasize construction of mathematical optimization models. In these efforts, attention is focused at providing supports systems that facilitate *the composition of models*, parameters, and algebraic formulas. Dolk and Kottemann [26] then advocate that model construction support systems must also address the need *to execute, interpret, or enact the model*, and how this may influence model formulation. We believe these concepts should therefore be applicable to other model formalizations, integration and formulation support, particularly those for software development models [12,20,27].

In our view, we find that the concepts that we have emphasized above can best be tied together through an understanding of the overall set of mechanisms and activities that must be addressed to formulate complex models of software development projects. Over the past eight years, we have been developing, using, and evolving a knowledge-based PSDSE called the Articulator [7]. The Articulator's knowledge representation is an implementation of the URM. In this way, the Articulator's meta-model serves as a *formalism* for the URM to represent a software development model and its supporting resource infrastructure [7,28].

Based on our experience in using and refining the Articulator in the formulation of a few dozen SDMs for our research sponsors, we have developed a characterization of the set of activities we perform in constructing and manipulating software development models. We refer to this set of activities as the model formulation process life cycle. The set of activities that entail this life cycle include the following:

- *meta-modeling*: constructing and refining a concept vocabulary and logic (an ontology) for representing classes of models and model instances in terms of object classes, attributes, relations, constraints, control flow, rules, and computational methods.
- *model definition*: eliciting and capturing of informal descriptions of software development objects and their conversion into formal development models or model instances.
- *analysis*: evaluating static and dynamic properties of a model, including its consistency, completeness, internal correctness, traceability, as well as other semantic checks.
- *simulation*: symbolically enacting modeled SDMs in order to determine the path and flow of intermediate state transitions in ways that can be made persistent, replayed, queried, dynamically analyzed, and reconfigured into multiple alternative scenarios.
- *visualization*: providing users with graphic views of SDM object classes and instances that can be viewed, navigationally traversed, interactively edited, and animated to convey modeled process statics and dynamics.
- *prototyping*: incrementally enacting partially specified SDM instances in order to evaluate presentation scenarios to end users, prior to performing tool and document model integration.
- *administration*: assigning and scheduling specified users, tools, and development data objects to modeled user (agent) roles, product milestones, and development schedule.
- *integration*: encapsulating selected software tools, repositories, and document objects that are to be invoked or manipulated when enacting an SDM.
- *environment generation*: automatically transforming a process model or instance into an executable SDM producing a PSDSE that selectively presents prototyped or integrated tools/objects to end-users for enactment.
- *enactment*: executing a generated SDM program and resulting environment by a process interpreter that guides or enforces specified users or agent roles to enact the SDM as planned and scheduled.
- *monitoring, recording, and auditing*: collecting and measuring SDM enactment data needed to improve subsequent process enactment iterations, as well as documenting what process steps actually occurred in what order.
- *replay*: graphically simulating the re-enactment of a previously executed SDM, in order to more

readily observe process state transitions or to intuitively detect possible process enactment anomalies.

- *articulation*: diagnosing, repairing, and rescheduling actual or simulated SDM enactments that have unexpectedly broken down due to some unmet resource requirement, contention, availability, or other resource failure.
- *evolution*: incrementally and iteratively enhancing, restructuring, tuning, migrating, or reengineering an SDM or the preceding activities to more effectively meet emerging user requirements, and to capitalize on opportunitistic benefits associated with new tools and techniques.

While such a list might suggest that formulating a SDM through its life cycle proceeds in a linear or "waterfall-like" manner, this is merely a consequence of its narrative presentation.

The Articulator meta-model serves as a representation language that is capable of specifying a large family of software engineering processes. So far, we have specified more than twenty process models for different kinds of applications ranging from small-scale research-oriented models to large-scale commercial development models [7,20,27,29–32]. The Articulator meta-model has also served as the basis for integrating [1] the Articulator environment [7] with (i) the generic process engine in the AP5 environment at the Information Sciences Institute by Balzer and Narayanaswamy [33], (ii) the Matisse team programming environment from HP Laboratories [4], (iii) the commercially available SynerVision process enactment engine from HP [20], and (iv) other in-house environments being researched by some of our other corporate sponsors. Of these (i) and (ii) employ automation rules for process support, while (iii) utilizes a procedural process programming language, a broadcast message server, and encapsulated off-the-shelf CASE tools. As a number of other process-centered environments or enactment engines utilize one or more of these mechanisms, we therefore believe our experience has begun to demonstrate the plausibility and validity of the URM. However, the focus of our contribution in this paper is on presenting the structure, organization, and selected details of the URM meta-model, and how it supports the formulation of models via composition.

Therefore, in the remaining sections, we will show how these related concepts, particularly those emphasized, are brought to bear in describing (a) a meta-model appropriate for the domain of software development, and (b) the composition of models from component models.

3. A unified resource model of software objects

In this section, we introduce a unified resource model (URM) that represents a knowledge-based foundation for modeling and interrelating the objects associated with the development of large software systems. The model seeks to organize and codify the classes of objects, attributes, and values that denote the entities associated with software development processes, products, organizational roles, and development tools, together with the relationships that associate each to the other. In this way we can formulate object-oriented models that characterize which people (roles) perform what tasks (processes) that consume or produce specific outcomes (documents, systems) using CASE environments (tools). However, software development projects occur in different settings, where different people may act in one or more different roles, performing different tasks or task sequences, building different intermediate and final products, using different tools. Thus, any one model of software development can have many distinct instantiations, and different organizations can subscribe to one or more different models. As a result, we seek to avoid constructing a model of each such software development instance from scratch, and instead seek to employ a codified representational framework that organizes, structures, and simplifies model formulation, composition, and enactment. Thus, the URM we have developed serves to broadly define the concepts and associations that span a family (i.e., a related set) of software devel-

[1] The primary technical effort encountered during integration is the construction of a translator that maps a given process notation into or out of the Articulator's meta-model notation. Our experience has been that building a new translator takes a few person weeks of effort. More complex translators for producing natural language paraphrases of SDMs or enactment histories are also being developed elsewhere [24].

opment models, where each such model then can span a family of possible instantiations.

The URM unifies and integrates the principal characteristics of the types of objects which typify software development processes, systems, roles, documents, and tools and which represents them in terms of a higher level of object class called **resource**. This URM consists of two subclasses of resource: the *simple result* for individual resource types, the *aggregate resource set (AggRS)* for heterogeneous collections of resources of different types. It also consists of a *web of relations* that link different types of resources [7] [2]. Through the use of different software development object models, individual software applications can be built, and PSDSEs which support their construction can be realized in a more integrated manner. Furthermore, the URM has the ability to abstract and specify classes of relations across different kinds of software objects [3].

The URM for software development objects we describe consists of a generic resource model and five models of software resource classes. The *generic resource model* represents a hierarchy of generic resource classes and relations that describe the basic characteristics for all kinds of software objects. Each subclass of resource models is a specialization of the generic resource model and represents a single class of software resources. Currently, we have defined a number of such specialized models for resources such as software systems, documents, agents, tools, and processes. Depending on an application domain, more specialized models can be easily added or customized. Accordingly, this URM is not a final statement, but instead it is a coherent and foundational representation that can combine many disparate software object models in a way that can be extended, specialized, instantiated or otherwise evolved. In this section, we first discuss the generic resource model and then five specialized object models.

[2] In earlier work, aggregated webs of related resources were called *computing packages* [25,37].

[3] In this regard, we allow for relations to be defined as object classes whose attributes can be single values or a set of values. This is one departure that distinguishes traditional object-oriented scheme (which generally lack relation class constructs) from knowledge-based schemes.

3.1. The generic resource model

The generic resource model describes a basic set of characteristics and uses for major kinds of software development resources typically manipulated in PSDSEs [34], which can be identified as follows:

3.1.1. Major kinds of software development resources

- *Software systems* which are products of software development. Normally, a software application system, written in a programming language, consists of a set of decomposible modules and non-decomposible functions. Software systems are produced by software developers through software development tasks that occur in different places at different times. Therefore, they have different development states indicating their status of development [27]. There also exists an invocation relation between software modules (e.g., what module calls what other module).
- *Software documents* (e.g., reports, deliverables, manuals, design descriptions, modification requests) which emerge as artifacts resulting from the execution of software development processes. These documents are composed aggregations of other development artifacts (e.g., design schemata) as products that are created, used, or modified by developers and their tools.
- *Software tools* which are executable programs invoked by developers to create/modify software systems and associated artifacts. PSDSEs can be viewed as collections of programs that developers progressively select for execution when performing different software development processes or activities.
- *Software developers* who are agents organized to perform software development. In a typical software development project, teams of developers work to produce a software system. There are many different working relationships among developers within a team, and individuals or work groups play different roles at different times.
- *Software development processes* which are a set of tasks linked by an execution plan. Software developers attempt to perform these tasks according to the process enactment plan while responding to local contingencies. These developers use software tools during the tasks, consume some software objects,

and produce other software artifacts, documents, and systems.

With these resources in mind, we can summarize the basic characteristics they share in common. These characteristics can then be abstracted into a generic resource model.

3.1.2. Common characteristic of software resources

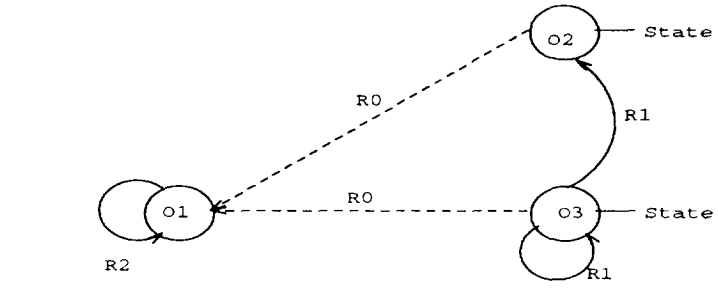
· *There is an object decomposition hierarchy for a software development resource.* For the purpose of modularity, we often view a software resource as a composition of other component resources. This reduces the complexity of the software object. For a given type of software resource, some are *decomposible*, while others that represent atomic units of computational processing are not. For example, a software system can be made up of decomposible modules and non-decomposible functions. A software development process is made up of decomposible and non-decomposible subtask steps.

· *There are associative links between components of decomposible software resources.* The component resources of a decomposible resource are not isolated. Relations (or links) between them identify processing orders or directions for message passing. For example, software modules may *invoke* each other. Software tasks can be *executed* sequentially, iteratively, conditionally, or concurrently.

· *Software resources have states that signify their processing status.* Although different software resources have different sets of state values, their state transitions can typically be described by a finite state machine. For example, a software module may undergo status changes such as created, reviewed, debugged, tested, baselined and released.

The generic resource model formally represents these characteristics through a set of object and relation classes. **Resource** is the root class of the generic resource model. It has two subclasses that describe two generic objects: **Simple-resource** represents a class for resources that are non-decomposible. **Aggregate-resouce-set (AggRS)** represents a package of member classes including **Simple-resource** and itself. It also represents an aggregate collection of **resource** classes with a set of relation classes across these resource classes. Fig. 1 shows the object hierarchy of the generic resource model.

There are three pairs of resource relation classes that represent the web of associative links between **resource**. Each of the relation classes is bi-directional, so that relation pairs denote the relation in each direction. The **is-a-subclass / subclass-of** characterizes the classic class-subclass relationship common to object-oriented models. The **has-components / component-of** relation pair forms the decomposition hierarchy for a software resource. **Has-neigbors / neighbors-of** form a processing link among a set of objects of a non-decomposible resource. Neighbor relations can be used to denote a transformational sequence of intermediate documents or persistent artifacts [35]. All resource subclasses can have




```
LEGEND:
 Object Classes:                    Relation Classes:
  O1 -- Resource                     R0 -- is-a-subclass
  O2 -- Simple-Resource              R1 -- has-member-resources
  O3 -- Aggregate-Resource           R2 -- has-neighbors
```

Fig. 1. The object hierarchy of the generic resource model.

Resource	*Simple-resource*	*AggRS*	*has-members*	*has-neighbors*
Software-System	Function	Module	has-functionalities	
Document	Paragraph	Section	has-component-docs	realized-by-docs
Agent	People	Team	has-Teammembers	report-to-agents
Tool	Program	Toolkit	has-tool-calls	has-next-calls
Process	Action	Task	has-subtasks	has-successors

Fig. 2. The five specialized resource model.

neighbors as long as they are the objects of the same subclass. **Has-member-resources / member-of-AggRS** form an aggregation link between a **AggRS** and its member resources.

One additional pair of object relation classes (not shown in the Figure), **has-resource-specs / resource-spec-of**, describe links between resources and the **resource-spec** class. It is this specification, together with the **resource-spec** class that defines relationships across different resource classes, which is a central part of the definition of **AggRS**. **Resource-spec** is a specification of resources that serves as a pointer to the desired resources for a particular purpose. Its definition is shown in Fig. 4. Its attributes include **resource-type**, **resource-condition**, **maximum-quality**, **minimum-quality**, and **matched-resource-instances**. When fully specified, a **resource-spec** refers to a set of resources whose class is a subclass of **resource-type**, and whose attribute values match those in **resource-condition**. When a resource-spec is linked to a resource through has-resource-specs, it constitutes a conceptual bridge from a resource to another set of resources that match the description in the resource-spec, which is determined dynamically. Accordingly, we give examples of this web of relations in later sections.

Resource has an attribute called state. Values of state for a resource class describe stages of computational processing for the class. Among all subclasses of resources, state values of simple-resource and the definition of state transition are defined by users. State values of AggRS are a collection of its member resource classes.

Next, we discuss specializations of the generic resource model.

3.2. Specialized object models

Software development object models can be developed through specialization of the generic resource model. They represent different software objects in software development. On the other hand, these specialized object models inherit the characteristics of the generic resource model. On the other hand, other specifications are added to describe characteristics that are specific to the kind of software objects they describe. While we have defined a number of specialized object models, we now focus on five subclasses of **resource**, and a subclass of **AggRS**.

3.2.1. Five specialized resource models

We can identify five specialized **resource** models using the generic resource model. Each describes a single class of software objects. They are defined through multiple inheritance of attributes, subclasses, and relations from the generic resource model classes [4]. They include models for **software-system**, **document**, **agent**, **tool**, and development **process**. Fig. 2 lists elements of these specialized models as subclasses of objects and relations of the generic resource model. For example, the document model

[4] In this paper, we do not provide a complete enumeration of all of the attributes, relations, or default values for these classes, but instead focus on an illustrative subset that we and some of our research sponsors have employed in software development projects. Similarly, we do not describe the rules and computational methods that manipulate these entities, as examples can be found in our related work [31,28,33].

consists of **paragraph** as its **simple-resource** and **section** as its **AggRS**. As such, a document consists of sections and paragraphs to form a decomposition hierarchy with a sequential order of the first section, the second section, etc. As another example, a development process model consists of decomposible **tasks** and non-decomposible **actions** with **has-successors** to specify its execution order. Fig. 3 shows the generic resource model and its specialized individual object models. Together they represent the URM. In the figure, the classes of the individual models are represented by the symbols labeled with the circles. For example, **Document** is a subclass of **resource**, therefore, its symbol in the circle is **O1**, which is the label for **resource**.

3.2.2. A specialized AggRS model: software-development-model

Software-development-model is a specialized **AggRS** which, as before, consists of a collection of **resource** classes connected through web relations. These web relations enable the manipulation of different resource classes in different ways.

The **Software-development-model** describes the resource infrastructure where a software system is

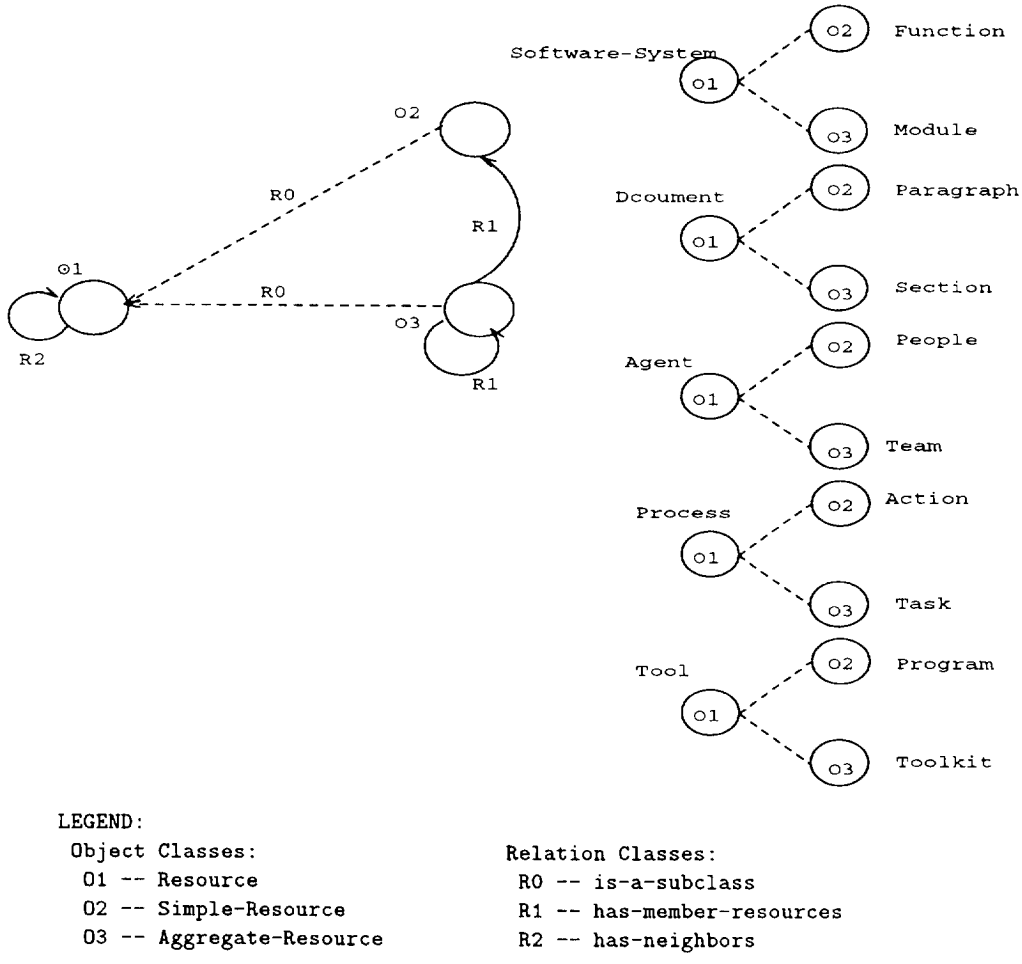


Fig. 3. The object hierarchy of the unified resource model.

produced. It consists of a development **process** model as its enactment plan, a **software** model as the basis for its product, an **agent** model for its performers, a set of optional other resource models as its input, and an optional **tool** model. Further, the **software** model is a specialized **AggRS** that has a software system model and a document model.

The web relations in **software-development-model** describe relationships among the individual models and are centered around the development process. **agent-role-spec** specifies the roles of agents assigned to perform the process tasks or actions. **tool-spec** specifies resources that are used as tools in the process. **required-resource-spec** specifies resources that are input to the process. **successful-provided-resource-spec** specifies resources that are output from the process when the process is successful; **failed-provided-resource-spec** specifies resources that are output from the process if its enactment breaks down or fails [29,36,37]. Accordingly, each of the five subclasses are defined according to the schema for **resource-spec** highlighted in Fig. 4.

4. An example of a software-development-model

To provide a realistic illustration of the use of the URM, we first present a subclass of the **software-development-model** called **DOD-STD-Model**. This model describes the development approach and products directed by MIL-STD-2167A [38], which is a public standard that guides the development of large software systems embedded in military applications

```
{{resource-spec
  is-a:                                    resource
  resource-type:                           [resource-classes]
  resource-condition:                      [(attribute value)]
  minimum-quantity:                        [integer]
  maximum-quantity:                        [integer]
  selection-condition:                     true
  resource-possession:                     [allocated-resource-instances] }}

{{agent-role-spec
  is-a:                                    resource-spec
  agent-role-spec-in-process:              [attached-process] }}

{{tool-spec
  is-a:                                    resource-spec
  tool-spec-in-process:                    [attached-process] }}

{{required-resource-spec
  is-a:                                    resource-spec
  required-resource-spec-in-process:       [attached-process] }}

{{successful-provided-resource-spec
  is-a:                                    resource-spec
  successful-provided-resource-spec-in-process:  [attached-process] }}

{{failed-provided-resource-spec
  is-a:                                    resource-spec
  failed-provided-resource-spec-in-process:  [attached-process] }}
```

Fig. 4. Definition of resource-spec.

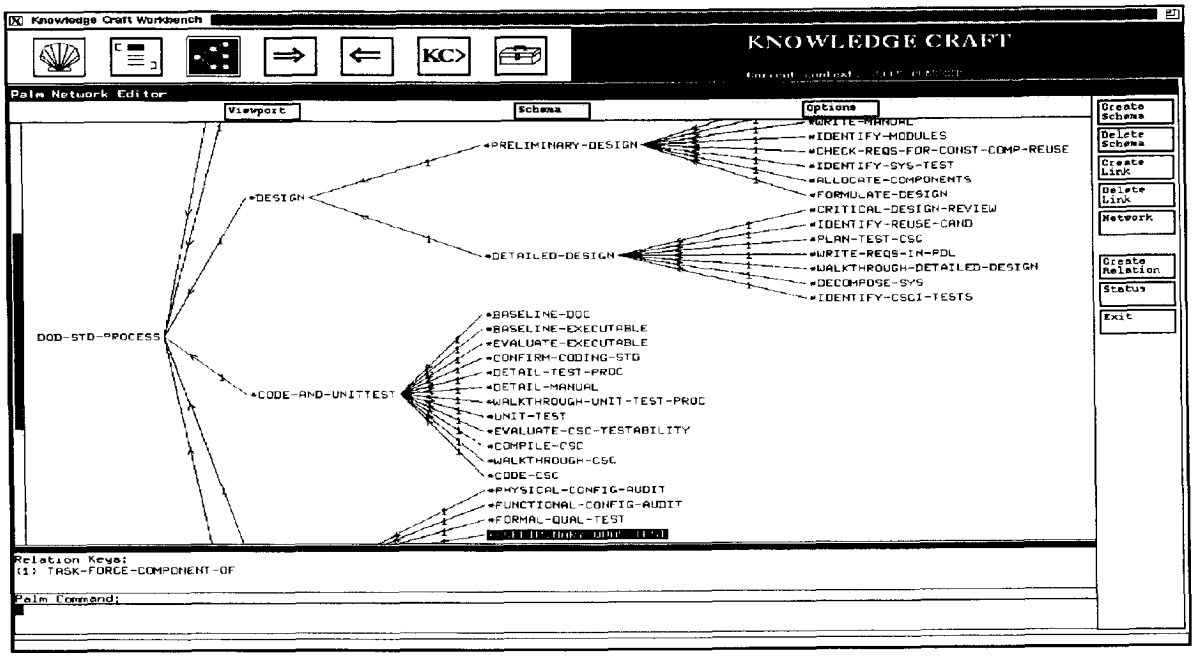


Fig. 5. The DOD-STD-process model: task decomposition.

purchased by the US government [5] **DOD-STD-Model** includes (i) a software product model, DOD-STD-SDF; (ii) a development team model, DOD-STD-Agent; and (iii) a software development process model, DOD-STD-Process. The development staff or team model is an addition we provided to MIL-STD-2167A for the completion of **software-development-model**. Since MIL-STD-2167A does not specify which types of software development tools to use, we do not include a tool model here, although such a tool model can be created. We now turn to examine the individual resource models and then the web of relations that connect them [6]. Fig. 6 gives a definition of **Formulate-Design**, which is one of the DOD-STD-Process tasks that is a component subtask of Preliminary-Design, as shown in Fig. 5.

In DOD-STD-Process, we use **component-of** to define component (sub)tasks or actions. We also use **has-successors** to specify the sequential or concurrent execution order among the tasks [7]. When exe-

[5] The use of other public standards, such as those grouped under the ISO 9000 banner, serve similar purposes in guiding the realization of high-quality software development products. There are also other military standards for software development such as MIL-STD-498, which is intended to supercede MIL-STD-2167A. While new standards seek to rectify technical shortcomings in MIL-STD-2167A, there nonetheless remains the problem of identifying, formulating, and codifying a software development model which can demonstrate conformity to the standards, what ever they may be.

[6] In this paper, we only identify a partial set of classes and instance values, as our model of software development for MIL-STD-2167A includes hundreds of object and relation classes, dozens of (sub)processes, documents, and organizational roles. < FN > . < H2 > The DOD-STD-Process model < /H2 > < P > The DOD-STD-Process specifies component tasks such as System Requirement Analysis, Design, Coding, Testing, and Integration. Fig. 5 shows a view of the multi-level task decomposition of DOD-STD-Process < FN > We use a commercially available CASE tool, KnowledgeCraft from The Carnegie Group, as a knowledge-base/model management system and graph browser within the Articulator environment. We use the graph browser to visualize the decomposition structure of selected relationships within the web of relationships that characterize a software development model.

[7] Process tasks that incorporate iteration or conditional choices utilize atomic actions to denote the begin and end decision points.

{{Formulate-Design

is-a:	task
component-of:	Preliminary-Design
has-predecessors:	none
has-successors:	Allocate-Components
	Identify-System-Test

Fig. 6. The DOD-STD-process model: a task definition.

cuted, a task can start only after its immediate predecessors are complete. For example in Fig. 6, **Formulate-Design** has two successors: **Allocate-Components** and **Identify-System-Test**. Thus, depending how we specify the successors of a task and other task constructors, we can achieve different task execution orders, including sequential, concurrent, iterative, or conditional [27].

4.1. The DOD-STD-SDF model

The DOD-STD-SDF specifies the set of software documents and software system executables required by MIL-STD-2167A, as well as their relationships [39,40]. SDF represents the Software Development Files that are the products of software development. Fig. 7 shows the system product decomposition of DOD-STD-SDF. Fig. 8 provides a compositional view of some of the DOD-STD-SDF documents. This set of documents is designed around an object called a Computer Software Configuration Item (CSCI), and includes a description of its requirements, design, coding, testing, and integration. As such, Fig. 7 shows the kinds of the document for each CSCI. When the model is instantiated, the set of the documents should be instantiated for each of the CSCI identified in the Requirements Specification. Another property of DOD-STD-SDF is that it defines the relation **realized-by-doc** among the documents. MIL-STD-2167A requires developers to keep track of the realization of initial development requirements among implemented CSCIs and their unit modules. To satisfy this need, we use **realized-by-doc** to link a single software requirement to its design, coding, and testing in order to maintain its consistency throughout software development. Then this relation can be retrieved to identify the actual allocation of all software system requirements. In Fig. 8 **Software-Top-Level-Design-Doc** is realized by **Software-Detailed-Design-Doc**, which in turn realized by **CSCI** and **Software-Test-Plan.**

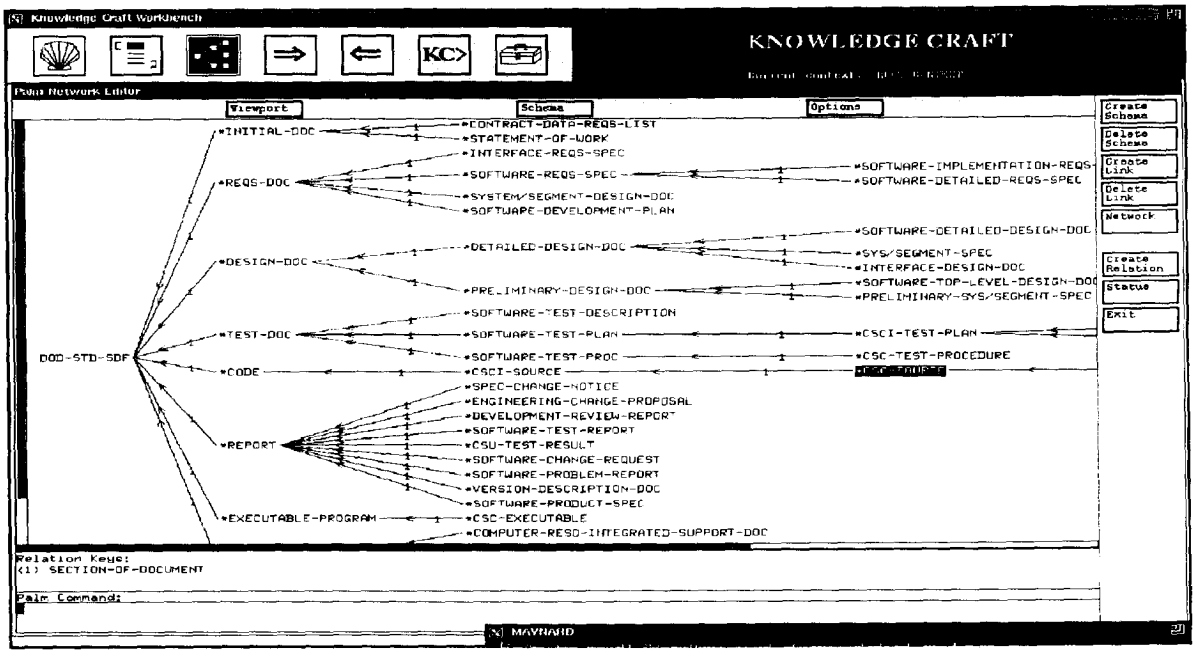


Fig. 7. The DOD-STD-SDF model: product decomposition.

{{Software-Top-Level-Design-Doc

is-a:	document
section-of-document:	design-doc
functional-description:	STLDD for each CSCI
realized-by-doc:	Software-Detailed-Design-Doc}}

Fig. 8. The DOD-STD-SDF model: a document definition.

4.2. The DOD-STD-Agent model

A development team that builds software in a manner conforming to MIL-STD-2167A typifies many situations where software systems are currently produced. The development team consists of different agent roles that are involved in the DOD-STD-Process model including Manager, Designer, Quality Assurance (QA) people, Engineer and other Roles. Fig. 9 shows the roles in DOD-STD-Agent and the definition for a DOD-STD-Agent role **Software-Designer**.

4.3. Web relations in the DOD-STD-model

In DOD-STD-Model, DOD-STD-Process, DOD-STD-SDF, and DOD-STD-Agent are linked through the following web relations. Fig. 10 gives a definition of these web relations for the subtask **Formulate-Design** in DOD-STD-Process.

Has-successful-provided-resource-spec specifies resources in DOD-STD-SDF that are output from the subtasks in DOD-STD-Process. Ideally, every single resource in DOD-STD-SDF must be produced by some subtask in DOD-STD-Process. Otherwise, the resource will not be created in a real software development since it is not accounted for in DOD-STD-Process. If problem situations arise, **has-failed-provided-resource-spec** is also defined to show, for example, that a problem report is created.

Has-required-resource-spec specifies resources that are input to each of the subtasks in DOD-STD-Process. Since MIL-STD-2167A does not contain information about other types of input resources, this

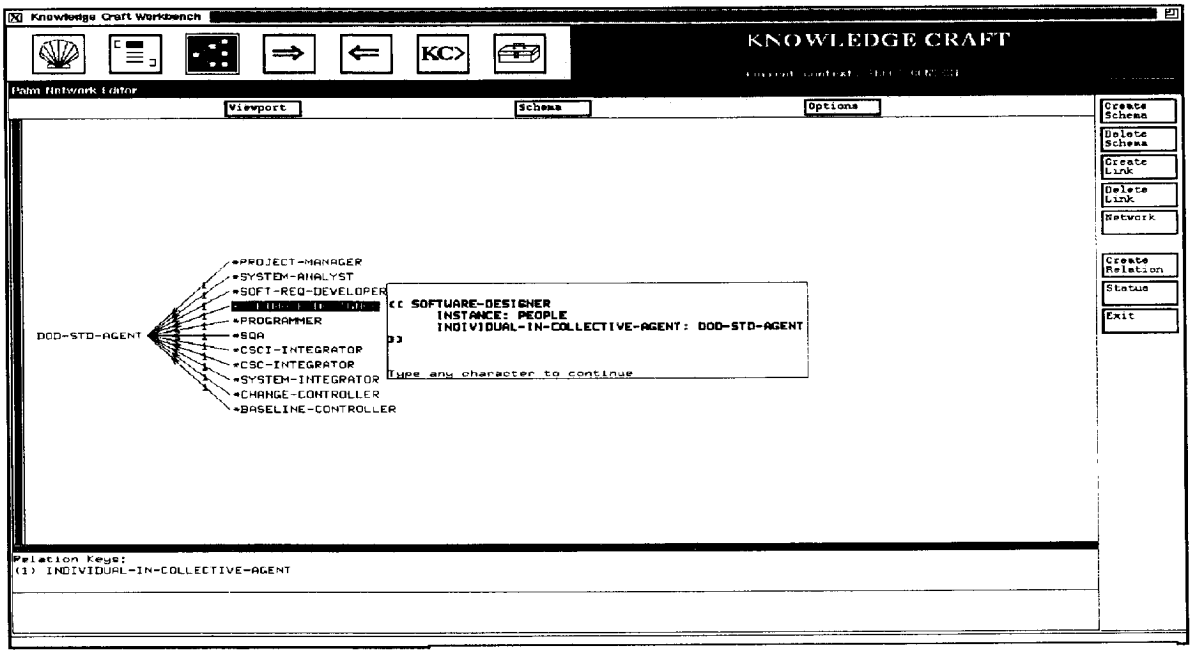


Fig. 9. The DOD-STD-Agent model: role decomposition.

web relation only has resources in DOD-STD-SDF as its inputs. This means a resource in DOD-STD-SDF, created earlier by a subtask in DOD-STD-Process, can then used by later subtasks.

Has-agent-role-spec specifies the roles in DOD-STD-Agent that are involved in each subtask in DOD-STD-Process. Each subtask in DOD-STD-Process has at least one role as its performer. Extra qualification attributes such as skill requirements and prior experiences can also be added.

4.4. Composition of software development models

In order to more rapidly build large and complex software development models (SDMs), we would

{{Formulate-Design

is-a:	task
has-agent-role-spec:	role-for-formulate-design
has-required-resource-spec:	required-for-formulate-design
has-successful-provided-resource-spec:	succ-provided-for-formulate-design}}
has-failed-provided-resource-spec:	failed-provided-for-formulate-design}}

{{role-for-formulate-design

is-a:	agent-role-spec
agent-role-spec-in-process:	Formulate-Design
resource-type:	Software-Designer
resource-condition:	(state ready)
maximum-quantity:	1
resource-possession:	none }}

{{required-for-formulate-design

is-a:	required-resource-spec
required-resource-spec-in-process:	Formulate-Design
resource-type:	System/Segment-Design-Doc, Software-Reqs-Spec
resource-condition:	(state ready), (state reviewed)
maximum-quantity:	1, 1
resource-possession:	none }}

{{succ-provided-for-formulate-design

is-a:	provided-resource-spec
provided-resource-spec-in-process:	Formulate-Design
resource-type:	Software-Top-Level-Design-Doc
resource-condition:	(state none)
maximum-quantity:	1
resource-possession:	none }}

{{failed-provided-for-formulate-design

is-a:	provided-resource-spec
provided-resource-spec-in-process:	Formulate-Design
resource-type:	Problem-Report
resource-condition:	(state none)
maximum-quantity:	1
resource-possession:	none }}

Fig. 10. The web relation for formulate-design.

like to draw upon SDMs that might exist elsewhere, or be stored within shared wide-area SDM repositories [12,41]. Assuming that translators exist or can be built for mapping SDM notations into the meta-model notation described above, we must next face the problem of how to compose independently developed SDMs into larger models. Thus, our interest here is in moving toward viewing the composition of models [8] as a process, rather than simply as a "cut and paste" or "copy and edit" task. However, without loss of generality, we will limit our discussion here to the composition of software development process models in order to highlight the problems we address.

In our view, there are three central problems in composing models of software development processes within SDMs. The first is that of finding or *matching* a candidate software process model for composition (e.g., a model for an object-oriented design (OOD) technique [42]) with its target software development infrastructure. The second is *planning* for how to construct a match, when the matching activity fails to produce an acceptably compatible match [29,37]. The third entails inserting or *integrating* the candidate SDM into a compatible process model representation. All three of these activities entail complex representations and reasoning processes. Thus, our effort here is aimed at outlining the nature of these problems, as well as how the URM can be utilized to formulate solutions.

In our view, new software development processes and the software engineering tools that support them, require that a web of compatible resources be in-place or available to facilitate successful process model composition. This resource web can be thought of as a signature [43] or configuration of technological and organizational resources, e.g., OOD support tools, staff trained in OOD, and OOD artifacts or documents. Thus, the matching problem requires that the resource web for each process model composition candidate must be explicitly represented. It also requires that the resource web for the software production infrastructure in each organization seeking to adopt the new development process also be explicitly represented and described. Subsequently, if a match between the resource web and the organizational structures can be found, then it is possible for the candidate software development process model to be successfully composed with the target. If a match cannot be found, it is very unlikely that the candidate process model can be easily adopted and integrated.

Clearly, it is likely that an exact match will not always be found (cf. [43]). Therefore, it is necessary to establish some criteria or measure of acceptable distance from an exact match, i.e., acceptable partial match metrics. At this point, we have developed a set of 35 heuristics for classifying the disparity between currently available resource webs and those needed to satisfy planned and scheduled software development processes [29,36,37]. In short, these heuristics examine which classes of the necessary resources are unavailable, available but already in use elsewhere, available but broken, similar type available, or available, as well as with multi-resource combinations (e.g., process A subsumes process B which requires a certain class of tool and certain class of staff). This classification mechanism thus serves to provide the basis for reasoning about how to achieve the best partial match with respect to the resources available.

The planning problem is of course related to the matching problem. While the matching problem may conform to a static pattern-matching and problem-solving scenario, the planning problem represents the dynamic scenario. That is, when matching results in only a partial match, then what can be done to revise or reconfigure the candidate or target SDM resource webs to achieve a better or best match?

The task of evolving a candidate SDM resource web into a configuration that better matches the one bound to the target resource infrastructure is the essence of the composition planning problem. This is a generative process planning problem [5]. This means that the plan must be generated to incrementally transform the candidate SDM into the desired target resource configuration, given available resources. The transformations may include adding or removing resources, restructuring resource configuration (web) relations, revising resource attributes or attribute values. In turn, the plan represents a partial ordering of which transformations to apply. Thus, there remains the problem of how to implement the

[8] Alternatively, the "tailoring" and specialization of generic SDMs citeusaf-stsc.

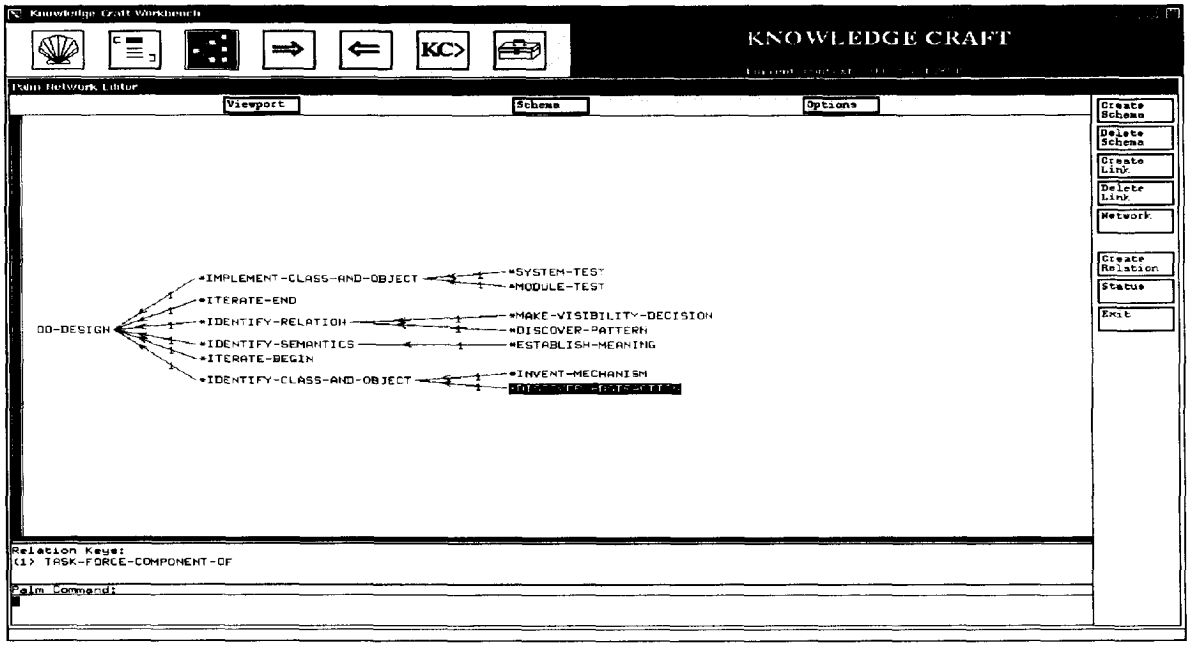


Fig. 11. The Booch's ODD Process model: task decomposition.

{{Formulate-OOD-Design

is-a:	task
has-agent-role-spec:	role-for-Formulate-OOD-Design
has-required-resource-spec:	required-for-formulate-OOD-design
has-successful-provided-resource-spec:	succ-provided-for-formulate-OOD-design}}
has-failed-provided-resource-spec:	failed-provided-for-Formulate-OOD-Design}}

{{required-for-formulate-OOD-design

is-a:	required-resource-spec
required-resource-spec-in-process:	Formulate-OOD-Design
resource-type:	System/Segment-Design-Doc, Software-Reqs-Spec
resource-condition:	(state ready), (state reviewed)
maximum-quantity:	1, 1
resource-possession:	none }}

{{succ-provided-for-formulate-OOD-design

is-a:	provided-resource-spec
provided-resource-spec-in-process:	Formulate-OOD-Design
resource-type:	Software-Top-Level-Design-Doc, Software-Object-Hierarchy
resource-condition:	(state none), (state none)
maximum-quantity:	1, 1
resource-possession:	none }}

Fig. 12. Some updated web relations for Formulate-ODD-Design.

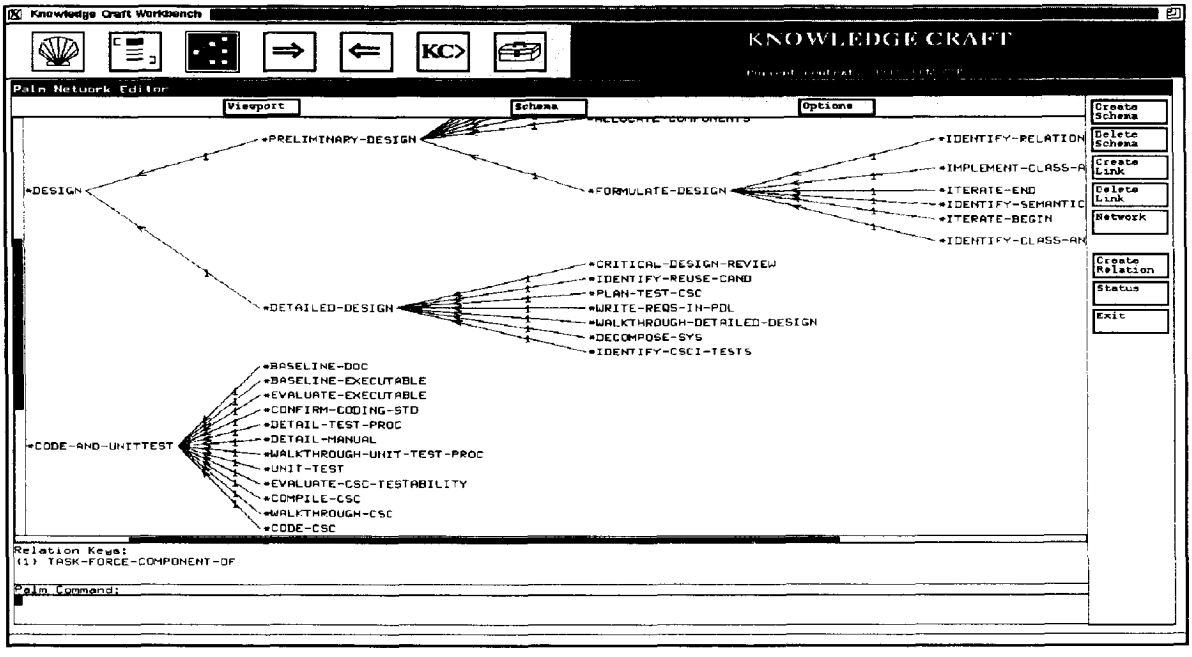


Fig. 13. The updated DOD-STD-Process: task decomposition.

plan that might be derived through a generative planning process. At this point, our software support environment for building and composing SDMs (described later) does not support generative planning, thus this task must still be performed manually.

Finally, there is the composition integration problem. Here we can describe our solution strategy with an example. Our starting point assumes that we have a candidate SDM whose web of resource requirements is satisfied with an upward-compatible development process infrastructure [9]. Let us consider the problem of how to model the (static) integration of an OOD process into the DOD-STD-Process. Fig. 11 displays a model of the OOD process derived from Booch [42]. In order to integrate this process model into the DOD-STD-Process, the matching problem boils down to determining if and where to incorporate the OOD process with the current design task hierarchy.

Since neither the MIL-STD-2167A nor the DOD-STD-Process preclude an OOD process, then we make our choice for where to include it so that the resources required during the OOD process match those provided by its predecessor tasks, and whether the resources required by the successor tasks of OOD match in an upward-compatible manner.

In this case, we incorporate OOD as the process for **Formulate-Design** as part of the activity for **Preliminary-Design** [10]. Fig. 12 shows how the object schemas for these design tasks are updated so that they match the resources provided by the predecessor task (e.g., the **Software-Reqs-Spec** outlined in Fig. 10), and those required by the succeeding preliminary design tasks. With this done, we show the updated DOD-STD-Process model task hierarchy in Fig. 13 which indicates that the model of Booch's OOD process has been successfully integrated as **Formulate-OOD-Design** and hence composed into the DOD-STD-Process. What now remains is updat-

[9] Recall, the resource infrastructure refers to the software systems, documents, tools, and organization of agents that are required to perform a software development process.

[10] MIL-STD-2167A does not specify the process of how to **Formulate-Design**. Thus, any development effort that seeks to conform to this standard must create or acquire such a design process model, then integrate it into the overall development process.

ing the DOD-STD-SDF model so that the outputs from Booch's OOD process match or encompass those required by **Formulate-Design**. Thus, we have shown through this simple example that we can use the URM to help us begin to better understand and support the composition of software process models.

5. Conclusion

The URM appears to be a powerful modeling formalism for investigating interactions and impacts among different classes of commonly seen software objects and process models. Our strategy focused on the representation and relationships among software systems, processes, products, tools, and agents that conform to the generic and specialized resource models that constitute the URM we have described. Our experience is using the URM in the Articulator meta-model demonstrates that we and others can successfully integrate and interoperate independently developed PSDSEs. We believe the formulation of SDMs via composition and the process life cycle may lead to the rapid integration and interoperation of SDMs in different notations through shared wide-area SDM repositories. However, this is an area for further research attention.

Our experience also indicates that a dynamic team-based endeavor is required in order to achieve mature SDMs that can be used in industrial settings. Further, our team effort most likely succeeds as a result of rapid SDM prototyping, incremental development, iterative refinement, and the reengineering of ad hoc SDMs or instances.

Looking into the future, we anticipate the Articulator meta-model and the URM can be used in other application fields, based on their ability to represent a wide range of resources and an intricate web of relationships among them. In this regard, our research is now extending into other areas, such as modeling of other aspects of development organizations as well as other organizational processes (e.g., workflow automation, order fulfillment, new product development, financial book closings) and their resource infrastructures [1]. Subsequently, our goal is to demonstrate the definition, integration, enactment, and composition of process-directed environments and applications for these process models.

Acknowledgements

Preparation of this report was supported in part by contracts and grants from AT&T Bell Laboratories, Andersen Consulting, Hewlett-Packard, IBM Canada Ltd., McKesson Water Products Co., Northrop-Grumman B-2 Division, Office of Naval Research (contract number N00014-94-1-0889), and Holosofx Inc., as well as the USC Center for Operations Management, Education and Research (COMER). However, no endorsements are implied. We also acknowledge contributions from Garry Brannum, Prasanta Bose, Mike Debellis, Ming-June Lee, Frank Luo, John Noll and Larry Votta.

References

[1] W. Scacchi and P. Mi. Modeling, Integrating, and Enacting Complex Organizational Processes, International Journal Intelligent Systems for Finance, Accounting, and Management (1993). Previous version presented at the 5th International Conference on Intelligent Systems for Finance, Accounting, and Management, Stanford University (December 1993).

[2] P. Devanbu, R.J. Brachman et al., LaSSIE: A Knowledge-based Software Information System, in: Proceedings of the 12th International Conference on Software Engineering, Nice, France March 1990 (1990) 249–269.

[3] W. Emmerich, G. Junkerman et al., Merlin: Knowledge-based Process Modeling, in: Proceedings of 1st European Workshop on Software Process Modelling (1991) 181–186.

[4] P.K. Garg, T. Pham et al., Matisse: A Knowledge-Based Team Programming Environment, Technical Report HPL-92-104, Hewlett-Packard Laboratories, International Journal Software Engineering and Know. Engineering (1992).

[5] K.E. Huff and V.R. Lesser, A Plan-Based Intelligent Assistant That Supports the Process of Programming, ACM SIGSOFT Software Engineering Notes 13 (1988) 97–106.

[6] G.E. Kaiser, Rule-Based Modeling of the Software Development Process, in: The 4th International Software Process Workshop, New York, NY (1988) 84–86.

[7] P. Mi and W. Scacchi, A Knowledge-based Environment for Modeling and Simulating Software Engineering Processes, IEEE Trans. on Knowledge and Data Engineering 2, No. 3 (1990) 283–294.

[8] M.Oivo and V.Basili, Representing Software Engineering Models — The TAME Goal Oriented Approach, IEEE Trans. Software Engineering 18, No. 10 (1992) 886–898.

[9] B. Ramesh and V. Dhar. Representation and Maintenance of Process Knowledge for Large Scale Systems Development, in: Proceedings of 6th Knowledge-based Software Engineering Conference (1991) 223–231.

[10] W. Schafer, B. Pueschel and S. Wolf, A Knowledge-based Software Development Environment Supporting Cooperative

Work, Internatinoal Journal Software Engineering and Know. Engineering 2, No. 1 (1992) 79–106.
[11] M. Wood and I. Sommerville, A Knowledge-based Software Components Catalogue, in: P. Brereton, Ed., Software Engineering Environments (Ellis Horwood Limited, 1988) 116–133.
[12] P. Mi, M. Lee and W. Scacchi, A Knowledge-based Software Process Library for Process-driven Software Development, in: Proceedings of the 7th Knowledge-Based Software Engineering Conference, McLean, VA (1992).
[13] A.M. Geoffrion, An Introduction to Structured Modeling, Management Science 33, No. 5 (1987) 547–589.
[14] A. Geoffrion, The formal aspects of structured modeling, Operations Management 37, No. 1 (1989) 30–52.
[15] A. Geoffion, The SML language for structured modeling: levels 1 and 2, and 3 and 4, Operations Research 40, No. 1 (1992) 38–76.
[16] B. Czejdo and M. Taylor, Integration of Information Systems using an Object-Oriented Approach, Computer Journal 35, No. 5 (1992) 501–513.
[17] M. Lenard, An Object-Oriented Approach to Model Management, Decision Support Systems 9, Nos. 3–4 (1993) 67–74.
[18] M. Jarke, M. Jeusfeld and T. Rose, A Software Process Data Model for Knowledge Engineering in Information Systems, Information Systems 15, No. 1 (1990) 86–115.
[19] V. Dhar and M. Jarke, On Modeling Processes, Decision Support Systems 9, No. 1 (1993) 39–49.
[20] P.K. Garg, P. Mi, T. Pham, W. Scacchi and G. Thunquest, The SMART Approach to Software Process Engineering, in: Proceedings of the 16th International Conference Software Engineering, IEEE Computer Society (1994) 341–350.
[21] F. Leymann and W. Altenhuber, Managing Buiness Processes as an Information Resource, IBM Systems Journal 33, No. 2 (1994) 326–348.
[22] M.Freeman and P.Layzell, A Meta-Model of Information Systems to Support Reverse Engineering, Information and Software Technology 36, No. 5 (1994) 283–294.
[23] B. Blum, Characterizing the Software Process, Information and Decision Technologies 19, No. 4 (1994) 215–232.
[24] H. Bhargava and R. Krishnan, Computer Aided Model Construction, Decision Support Systems 9, Nos. 3–4 (1993) 91–112.
[25] A. Sen, A. Vinze and T. Feng-Liou, Construction of a Model Formulation Consultant: The AEROBA Experience, IEEE Transactions on Systems, Man, and Cybernetics 22, No. 5 (1992) 1220–1232.
[26] D. Dolk and J. Kottemann, Model Integration and a Theory of Models, Decision Support Systems 9, Nos. 3–4 (1993) 51–66.
[27] P. Mi and W. Scacchi, Process Integration in CASE Environments, IEEE Software 9, No. 2 (1992) 45–53.
[28] W. Scacchi, The Software Infrastructure for A Distributed System Factory, Software Engineering Journal 6, No. 5 (1991) 355–369.
[29] P. Mi and W. Scacchi, Modeling Articulation Work in Software Engineering Processes, in: Proceedings of the 1st International Conference on the Software Process (1991) 188–201.
[30] W. Scacchi and P. Mi, Experiences with Process Modeling, Analysis, and Simulation of Formalized Process Models, in: Position Paper at the 8th International Software Process Work, Dagstuhl, Germany, IEEE Computer Society (1993).
[31] L. Votta, Comparing One Formal to One Informal Process Description, in: Position Paper at the 8th Internatinoal Software Process Work, Dagstuhl, Germany, IEEE Computer Society (1993).
[32] W. Scacchi and P. Mi, Modeling, Integrating, and Enacting Software Engineering Processes, in: Proceedings of the 3rd Irvine Software Symposium, Irvine Research Unit in Software, University of California at Irvine (1993).
[33] R. Balzer and K. Narayanaswamy, Mechanisms for Generic Process Support, in: Proceedings of the 1st ACM SIGSOFT Symposium on Foundations Software Engineering, ACM, Software Engineering Notes 18, No. 5 (1993) 9–20.
[34] R. Conradi, C. Ferstrom, A. Fuggetta and B. Snowden, Towards a Reference Framework for Process Concepts, in: Software Process Technology, Second European Workshop on Software Process Technology (EWSPT '92) (Springer-Verlag, Lecture Notes in Computer Science 635, September 1992) 3–17.
[35] S.C. Choi and W. Scacchi, SOFTMAN: An Environment for Forward and Reverse CASE, Information and Software Technology 33, No. 9 (1991).
[36] P. Mi, Modeling and Analyzing the Software Process and Process Breakdowns, PhD Thesis (Computer Science Dept., University of Southern California, 1992).
[37] P. Mi and W. Scacchi, Articulation: An Integrated Approach to Diagnosis, Re-planning, and Re-scheduling, in: Proceedings of the 8th Knowledge-Based Software Engineering Conference, Chicago, IL (1993) 77–85.
[38] US Department of Defense, Defense System Software Development, Document DOD-STD-2167A, Technical Report, Department of Defense (1988).
[39] S.C. Choi and W. Scacchi, Assuring the Correctness of Configured Software Descriptions, ACM Software Engineering Notes 17, No. 7 (1989) 67–76.
[40] P.K. Garg and W. Scacchi, A Hypertext System to Manage Software Life Cycle Documents, IEEE Software 7, No. 3 (1990) 90–99.
[41] J. Noll and W. Scacchi, Integrating Diverse Information Repositories: A Distributed Hypertext Approach, Computer 24, No. 12 (1991) 38–45.
[42] G. Booch, Object Oriented Design with Application (The Benjamin/Cummings Publishing Company, Inc., 1991).
[43] A.M. Zaremski and J.M. Wing, Signature Matching: A Key to Reusem, in: Proceedings of the 1st ACM SIGSOFT Symposium on Foundations Software Engineering, ACM, Software Engineering Notes 18, No. 5 (1993) 182–190.