CAPTURING MORE WORLD KNOWLEDGE IN THE REQUIREMENTS SPECIFICATION

Sol J. Greenspan and John Mylopoulos
Department of Computer Science
University of Toronto
Toronto, Ontario, CANADA

Alex Borgida
Department of Computer Science
Rutgers University
New Brunswick, New Jersey, USA

"The real problem is the mass of detailed requirements; and the only solution is the discovery or invention of general rules and abstractions which cover the many thousands of cases with as few exceptions as possible."
- C. A. R. Hoare

Abstract

The view is adopted that software requirements involve the representation (modeling) of considerable real-world knowledge, not just functional specifications. A framework (RMF) for requirements models is presented and its main features are illustrated. RMF allows information about three types of conceptual entities (objects, activities, and assertions) to be recorded uniformly using the notion of properties. By grouping all entities into classes or metaclasses, and by organizing classes into generalization (specialization) hierarchies, RMF supports three abstraction principles (classification, aggregation, and generalization) which appear to be of universal importance in the development and organization of complex descriptions. Finally, by providing a mathematical model underlying our terminology, we achieve both unambiguity and the potential to verify consistency of the model.

1. INTRODUCTION

Requirements definition is the task of gathering all of the relevant information to be used in understanding a problem situation prior to system development. The documentation of this information is called the requirements specification. The form and content of the requirements specification can have a tremendous impact on the task of software throughout its lifetime.

Experience over the last decade has led to some important observations that point to the need for improved requirements specification languages. First, it appears that more attention to requirements, including a better understanding of the problem situation, pays off in reduced total life-cycle effort and cost [3]. Secondly, it has been learned that it is difficult, indeed, to "get the requirements right"; some common problems are ill-defined terms, inconsistencies, ambiguities, and the tendency to mix requirements with design decisions [1].

Much of requirements definition involves such tasks as: defining terms in the domain of discourse, stating, clarifying and agreeing on assumptions and constraints, and discussing and negotiating the needs and objectives of an organization (business, government, industry). Whatever the application "world" (e.g. airline reservations, manufacturing, hospital administration, etc.) there is a body of "knowledge" used to interpret and understand that world.

For example, in considering the development of a variety of information systems for a large hospital in Toronto, we have found it necessary to become intimately familiar with a wide range of subject matters: medical knowledge, hospital procedures and policies, available therapies (drugs, surgery, etc.), legal responsibilities to government, and so on. We believe that this kind of real world knowledge needs to be captured in a formal requirements specification. The ability to efficiently design appropriate computer systems and enable them to evolve over their lifetime depends on the extent to which this knowledge can be captured.

Most current requirements languages concentrate primarily on functional specifications, which give a high-level target system description in terms of the functions to be performed by the ultimate system (with an emphasis on what the system is supposed to do but not how). In functional specification approaches, the world knowledge, whose importance we have been discussing, is often not an explicit part of the requirements specification. The knowledge is, at least initially, scattered throughout documents and the minds of people across the organization.

0270-5257/82/0000/0225$00.75 © 1982 IEEE

Our research addresses the development of languages and tools for requirements modeling, a specification approach directed toward high-level specifications which capture world knowledge directly and naturally in the specification. In this paper we present a framework for requirements modeling called RMF. An RMF model describes some "slice of reality". RMF guides the information gathering job of requirements definition. The resultant description should be useful in determining what the problem situation is and what solutions are possible. The design of RMF emphasizes the need for structuring the description to ease the task of finding answers to questions during system design and implementation and in the face of changing requirements.

Section 2 of the paper introduces our modeling approach and discusses the central theme, abstraction mechanisms. Section 3 is the core of the paper, presenting the main features of RMF along with illustrative examples. Section 4 discusses related work and fills in some background material. Finally, Section 5 discusses some interesting aspects of the framework and of our ongoing research.

2. MODELING AND ABSTRACTION

2.1 A modeling approach

The major goal of RMF is to form a synthesis of some modeling principles which we believe are essential to the requirements modeling task; but we stop short of providing a specific language incorporating them. We apply these principles to answer two key questions about requirements modeling: (1) What kind of information should be captured in a requirements model? (2) How should a requirements model be structured?

We have chosen a representation method in which a requirements model consists of a collection of conceptual entities (or simply entities) defined by their inter-relationships with other entities. Thus, the first question above is answered by (i) choosing appropriate specification units, or entity categories, and (ii) deciding what kinds of relationships are allowed within and between entities of each entity category; later on in the presentation we shall call the relationships properties and the types of relationships property categories.

RMF offers three kinds of specification units: object, activity, and assertion. As discussed in Section 4, these entity categories have been used successfully, in one form or another, across a wide range of modeling endeavors.

For a long time it has been asserted that abstraction is the best tool we have toward the intellectual manageability of complex descriptions [11,14]. The framework is based on the premise that effective structuring of large descriptions such as requirements models depends on the use of good abstractions. Below we introduce the abstraction mechanisms used in the framework.

2.2 Abstraction mechanisms

An abstraction mechanism is any descriptive facility that allows certain kinds of information to be included while precluding other, "lower-level" or "less important" details. In Software Engineering, abstraction is usually equated with the suppression of design decisions or implementation details. In this sense a requirements model should be "very abstract"; indeed, it is proposed as the "most abstract" specification for use in Software Engineering.

In RMF we propose the use of a set of complementary abstraction mechanisms for descriptive purposes. A first abstraction mechanism, aggregation, allows an entity to be viewed as a collection of components, or parts. A second abstraction mechanism is classification, which allows a new entity, the class, to capture common characteristics shared by a group of entities. A third abstraction, generalization, captures the common characteristics of several classes.

Classification allows one to consider only the characteristics shared by the instances of a class and to ignore individual differences. Aggregation allows one to consider an entity while ignoring further detail about the components. Generalization allows one to consider only those properties that a collection of classes have in common without considering the classes' individual differences.

A principal design goal of our RMF is to apply the abstraction mechanisms uniformly over all the entity categories. That is to say, there are, in an RMF model, classes for object, activity, and assertion entities; entities of all categories can have component parts; and classes of each entity category are organized according to their generality/specificity.

We feel that a specification language exhibiting such uniformity will be much easier to formalize, understand and use.

3. THE FEATURES OF RMF

We first show in terms of objects how the abstraction mechanisms work, and then we extend the presentation to activities and assertions as well.

3.1 Tokens, classes, and metaclasses

Entities are stratified into classification levels according to whether they are considered "individuals", called tokens, collections of tokens called classes, classes of classes called metaclasses, and so on. The tokens of a class are called its instances; similarly, a class is said to be an instance of a metaclass.

Simple examples of object tokens are john-smith (representing a particular person) and 7 (representing the number 7). PERSON is an example of a class, whose instances are tokens such as john-smith, while INTEGER is a class whose instances would include 7.

An example of a metaclass is PERSON_CLASS, whose instances are classes of persons, such as PERSON, PATIENT, PHYSICIAN, NURSE.

In addition to its instances, a class bears additional information, but for this we need the notion of properties.

For the remainder of the paper we will use lower case letters and digits for token identifiers and upper case letters for class and metaclass identifiers. The suffix "_CLASS" will be use for metaclass identifiers.

3.2 Properties -- factual and definitional

Entities can be related to other entities by participating in properties. Properties consist of three items of information: a subject, an attribute (or property name), and a value. To express the property of the token john-smith that his age is 23, we could write

<john-smith, age, 23>

where the subject, attribute, and value are john-smith, age (an identifier), and 23, respectively. This property expresses "factual" information about the subject and thus is termed a factual property.

Factual information alone is clearly not adequate for requirements modeling. We have introduced classes (and metaclasses) for the purpose of defining and describing collections of entities that, presumably, are grouped together because some uniform conditions hold over all of them. What is needed is a facility for specifying generic information that pertains to each of the instances of a class (or metaclass). For example, we may want to specify for the class PERSON that each person has an age which is an AGE_VALUE. The RMF feature used for this is called a definitional property. The triple

(PERSON, age, AGE_VALUE)

is used for now to represent the above information. The three components are again called subject, attribute, and value. (Note that we have used angular brackets and parentheses to distinguish the two kinds of properties.)

It may be helpful to think of a definitional property as defining a function, e.g.,

age: PERSON --> AGE_VALUE

whose domain is the property subject and whose range in the property value. Evaluation of the function for an instance of the domain results in a corresponding factual property, e.g.

age(john-smith) = 23.

The most important point to notice here is that 23 is, and must be, an instance of AGE_VALUE. In general, for a class C with definitional property (C,a,V), for every instance x of C, it must be the case that a(x) is an instance of V. The close correspondence between definitional and factual properties is called the property induction principle. The principle requires as well that every factual property of an object be induced by a definitional property of some containing class.

It is useful to be able to associate factual properties to classes as well as to tokens. For example, the information

"the average age of persons is 21"

"the number of nurses is 200"

should be considered factual rather than definitional, that is

<PERSON, average-age, 21>

<NURSE, cardinality, 200>

since the information pertains directly to the subjects rather than to their instances.

The inclusion of metaclasses in the framework allows the property induction principle to be extended. In order to allow the specification of the above two factual properties, it would be necessary to have also specified definitional properties that induce them, such as

(PERSON_CLASS, average-age, AGE_VALUE)

(PERSON_CLASS, cardinality, NUMBER).

At this point in the paper, we have described how two abstraction principles, classification and aggregation, are incorporated into the framework. The classification abstraction is supported by the "instance of" relationship between tokens and classes and between classes and metaclasses; we consider the three levels to be adequate for most modeling purposes. The grouping of all the properties of an entity relates the entity to other entities which may in turn be the subjects of other properties; this supports the aggregation abstraction. The property induction principle relates these two dimensions in a coherent way.

3.3 Objects, activities, assertions

As implied by the examples above, objects represent the "things" in the world, such as persons, numbers, equipment, documents, etc.

```
PERSON_CLASS PATIENT
   association ward: HOSPITAL_WARD,
               primary-physician: PHYSICIAN,
               consulting-physician: PHYSICIAN,
   inserted-by register: ADMIT_PATIENT,
   initially phys-ward?:
      primary-physician.specialty ≠ ward
        => consulting-physician.specialty = ward,
   updated-by transfer:
                 TRANSFER(self,new-ward:WARD),
   removed-by release: RELEASE(self),
end {PATIENT}
```

Figure 1

Figure 1 is a description of the object class PATIENT, giving general information about patients for a particular hospital. The class is defined to be an instance of the metaclass PERSON_CLASS and has a number of definitional properties consisting of

<attribute> : <value>

pairs and grouped into property categories such as association, inserted-by, and initially.

The properties of PATIENT relate each patient to a ward and two physicians, one primary and the other consulting. Patients are "created" through an ADMIT_PATIENT activity, updated through a TRANSFER activity and removed through a RELEASE activity. When a patient is first created, it must be the case that if the ward to which a patient is assigned is not the specialty of his/her primary (care) physician, then it must be the specialty of the consulting physician.

```
ACTIVITY_CLASS ADMIT_PATIENT
     input p: PERSON,
     control w: WARD,
             phys, consulting-phys: PHYSICIAN,
     output pt: PATIENT,
     triggered-by a1: ARRIVAL(p),
     precondition already-in?: NOT INST(p,PATIENT),
           room-left?:
                  PATIENT.cardinality <
                                 PATIENT_MAX,
     postcondition admitted?: IN_HOSPITAL(p),
     part check-id: CHECK_ID(p),
          put: CHOOSE_WARD(w,phys,consulting-phys),
          admit: INSERT(p,PATIENT),
          increment: INCREMENT(PATIENT.cardinality),
          urinalysis: PERFORM_URINALYSIS(p),
          blood-count: PERFORM_BLOOD_COUNT(p),
          blood-pressure: PERFORM_BLOOD_PRESSURE(p),
          temp: TAKE_TEMP(p),
end {ADMIT_PATIENT}
```

Figure 2

In Figure 2, we present a definition of the activity class ADMIT_PATIENT. It consists of one input property, a person, and one output property, a patient; also, it has three control properties, a ward and two physicians. The activity is triggered by instances of the assertion class, ARRIVAL, which is instantiated each time a person arrives at the hospital for admission.

ADMIT_PATIENT also includes two preconditions (already-in? and room-left?) and a postcondition (admitted?) that must be true before and after the activity, respectively. The preconditions assert that the person must not be already a patient of the hospital, and also that the number of patients (PATIENT.cardinality) is less than the hospital capacity (PATIENT_MAX). The postcondition asserts that the person has indeed been admitted once the ADMIT_PATIENT activity is over.

Finally, the "body" of ADMIT_PATIENT is defined by several part properties which specify the components of an ADMIT_PATIENT activity. The components involve checking the person's ID, choosing a ward and assigning a primary care and a consulting physician, inserting the person into the PATIENT class and incrementing the cardinality property of PATIENT; also, some tests are performed (urinalysis, blood-count, blood-pressure, and temperature).

We present an example of an assertion class in Figure 3 to underscore the uniform treatment of the entity categories. The assertion class IN_HOSPITAL has one argument, a patient, and asserts through its part property that the person is now physically

```
ASSERTION_CLASS IN_HOSPITAL
     argument p: PERSON,
     part patient?: INST(p,PATIENT),
          present?: PHYSICALLY_PRESENT(p),
end {IN_HOSPITAL}
```

Figure 3

present at the hospital.

We close this section by giving a definition of the metaclass PERSON_CLASS, in Figure 4. As stated in the previous section, PATIENT is entitled to a cardinality property only because it is an instance of this metaclass. Note that METACLASS is a built-in metametaclass that has all metaclasses as instances. Two other built-in metametaclasses have been found useful because they allow references to all entities and to all generic entities, respectively: (i) ENTITY, which has as instances all entities in a specification, including itself, and (ii) CLASS, which has as instances all classes, metaclasses, and the three built-in metametaclasses METACLASS, CLASS, and ENTITY.

```
METACLASS PERSON_CLASS
     association average-age: AGE_VALUE,
                 cardinality: NUMBER,
end {PERSON_CLASS}
```

Figure 4

3.4 Generalization

To support the generalization abstraction mechanism, a new relationship, subclass, is offered which can be declared between two classes or two metaclasses. For example, suppose PERSON has been defined as shown in Figure 5. (Note: The part properties of a data class instance do not change values, while the association properties do.) Then, changing the first line of the definition of PATIENT (Figure 1) to

PERSON_CLASS PATIENT subclass of PERSON

makes PATIENT a subclass or specialization of PERSON and PERSON a generalization of PATIENT.

```
PERSON_CLASS PERSON
     part name: PERSON_NAME,
          sin: SOCIAL_INSURANCE_#,
          ohip: ONTARIO_INS_#,
     association address: ADDRESS,
                 age: AGE_VALUE,
end {PERSON}
```

Figure 5

What does it mean to say that PATIENT is a specialization of PERSON? Well, for one thing we expect that every instance of PATIENT is, under all circumstances, also an instance of PERSON. Indeed, the semantics of becoming an instance of a class include becoming an instance of all of the generalizations of the class. Conversely, when an object ceases to be an instance of a class, it also ceases to be an instance of its specializations.

Another aspect of specialization concerns the definitional properties of the two classes involved. All definitional properties of PERSON are inherited by PATIENT; so, by virtue of being declared a specialization of PERSON, PATIENT has, in addition to the properties specified in Figure 1, also a name, a social insurance number, an address, etc.

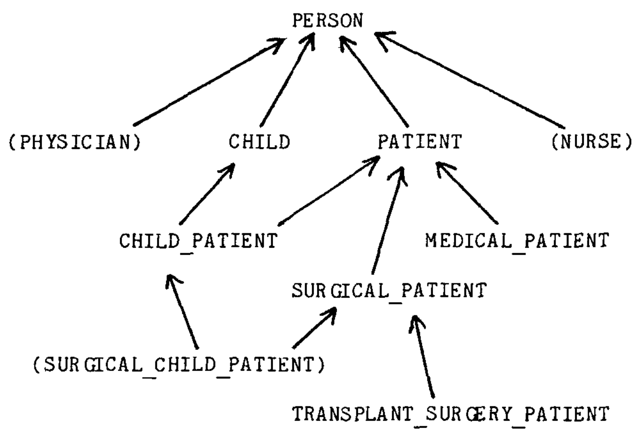
Property inheritance allows for economy of expression in a specification because a definitional property need only be mentioned once for the most general class to which it is applicable. Inheritance also serves as a memory aid, since knowing that a class is a subclass of another allows one to concentrate on the additional information needed to describe the subclass.

```
PERSON_CLASS CHILD subclass of PERSON
     association age: CHILD_AGE_VALUE,
                 guardian: PERSON,
     invariant guardian.age > 30,
end {CHILD}

PERSON_CLASS SURGICAL_PATIENT subclass of PATIENT
     association blood-type: BLOOD_TYPE,
                 surgery: SURGERY_TYPE,
end {SURGICAL_PATIENT}

PERSON_CLASS TRANSPLANT_SURGERY_PATIENT
                         subclass of SURGICAL_PATIENT
     association donor: PERSON,
end {TRANSPLANT_SURGERY_PATIENT}

PERSON_CLASS CHILD_PATIENT
                         subclass of CHILD, PATIENT
     association nurse: NURSE,
end {CHILD_PATIENT}
```

Figure 6

To illustrate the importance of generalization, suppose we have already defired the class PERSON and its specialization PATIENT. A number of other object classes are also relevant for our hospital example. CHILD specializes PERSON by restricting its age property to allow only values in CHILD_AGE_VALUE. SURGICAL_PATIENT as well as TRANSPLANT_SURGERY_PATIENT are specializations of PATIENT. CHILD_PATIENT gives an example of a class that has more than one immediate generalization. Figure 6 includes the definitions of all these object classes while Figure 7 summarizes the subclass relation for the object classes defined so far.

Specialization opens the door to a form of stepwise refinement that is based on the introduction of detail for special cases. Moreover, this form of refinement is not applicable only to object classes. Consider, for example, some specializations of ADMIT_PATIENT, as shown in Figure 8. The first, ADMIT_CHILD_PATIENT, simply



(The classes in parentheses have not been defined.)

Figure 7

assigns a nurse to the child patient in addition to all the things done for other patients. The second specializes ADMIT_PATIENT for surgical patients where a blood test is done for possible transfusion. ADMIT_SURGICAL_CHILD_PATIENT, the third, is a specialization of the previous two activities and therefore inherits all their definitional properties; in addition, it has a definitional property of its own which obtains permission for surgery from the child's guardian.

```
ACTIVITY_CLASS ADMIT_CHILD_PATIENT
                         subclass of ADMIT_PATIENT
     input p: CHILD,
     control n: NURSE,
     output pt: CHILD_PATIENT,
     part find-nurse: FIND_NURSE(n,w),
          admit: INSERT(P,CHILD_PATIENT),
end {ADMIT_CHILD_PATIENT}

ACTIVITY_CLASS ADMIT_SURGICAL_PATIENT
                           subclass of ADMIT_PATIENT
     output pt: SURGICAL_PATIENT,
     triggered-by a1: ARRIVAL(p)
                          AND SURGERY_NEEDED(p),
     part blood-typing: PERFORM_BLOOD_TYPING,
end {ADMIT_SURGICAL_PATIENT}

ACTIVITY_CLASS ADMIT_SURGICAL_CHILD_PATIENT
        subclass of ADMIT_CHILD_PATIENT,
                           ADMIT_SURGICAL_PATIENT
     part obtain-permission:
                   OBTAIN_PERMISSION(p,p.guardian),
end {ADMIT_SURGICAL_CHILD_PATIENT}
```

Figure 8

Note that redefinitions of definitional properties must be consistent with the properties they replace. For example, the value of the age property of child, CHILD_AGE_VALUE, must be a specialization of AGE_VALUE (Figures 5 and 6). Similarly, the redefinitions of properties such as p and pt in ADMIT_CHILD_PATIENT are all consistent with the properties of ADMIT_PATIENT they replace (Figures 2 and 8). An interesting application of this consistency rule involves properties whose value is an assertion class such as the a1 property of ADMIT_PATIENT (Figure 2). For ADMIT_PATIENT the value of a1 is the assertion ARRIVAL(p), while for ADMIT_SURGICAL_PATIENT (Figure 8) the value of a1 is the stronger assertion ARRIVAL(p) AND SURGERY_NEEDED(p).

Specialization can also be used to structure assertion class definitions. The IN_HOSPITAL assertion class, for instance, can be specialized by specializing its arguments, by adding conjuncts (parts), or even by redefining some of its parts. Thus, for child patients, IN_HOSPITAL might be specialized (see Figure 9) to check that the patient is in a ward accompanied by a nurse.

```
ASSERTION_CLASS CHILD_IN_HOSPITAL
                     subclass of IN_HOSPITAL
   argument p: CHILD
   part in-ward?: IN_WARD(p),
        with-nurse?: WITH_NURSE(p),
end {CHILD_IN_HOSPITAL}
```

Figure 9

We close this section by pointing out that metaclasses (and metametaclasses) are also organized into specialization hierarchies, as suggested in Figure 10.

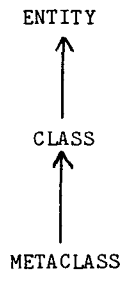
4. RELATED WORK

In this section we argue the advantages and utility of the requirements modeling framework. We justify our choices of specification concepts (namely, the entity categories, property categories, and abstraction mechanisms) by demonstrating that they are based on a consensus over a wide variety of specification and modeling experience. Moreover, our framework subsumes the features of important requirements languages.

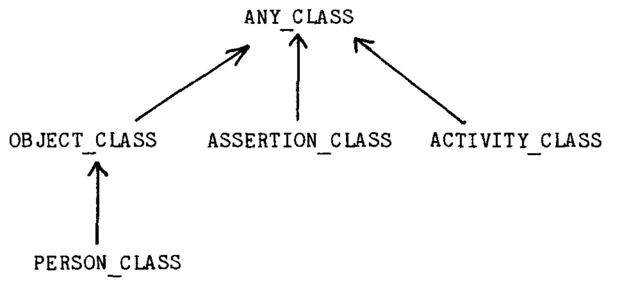
4.1 Related requirements languages

The achievement of a highly uniform framework is a goal partly inspired by Softech's SADT [22]. SADT offers data and activity concepts and uses the same graphical box and arrow notation for describing both. A data concept is defined (decomposed) in a diagram showing the data subparts as boxes interconnected by arrows representing activities. Activity objects are defined by a "dual" kind of diagram in which activity boxes are interconnected by data arrows.

RMF's "object" and "activity" correspond to SADT's data and activity, and RMF adds a third, complementary specification unit, assertions, to facilitate the explicit specification of information which in an SADT model would usually be specified in accompanying natural language text. In addition, RMF makes explicit the use of abstraction principles, which we believe the modeler tends to



Generalization hierarchy for metametaclasses



Generalization hierarchy for metaclasses

Figure 10

use implicitly during requirements modeling to interpret an SADT model.

By way of analogy with SADT's three arrow types (Input/Control/Output) between data and activity, we have considered various kinds of relationships between RMF's three entity categories as candidate property categories. RMF property categories offer some explicit interpretations for the relationships represented by the SADT arrows.

A second language which supports requirements specifications is PSL (Problem Statement Language) [26] which was the first automated facility for storing and managing "problem statements". A problem statement is a functional specification in the form of a data-oriented target-system description. Such a functional specification differs from a requirements model by making design decisions that determine system boundaries. "System structures" are distinguished from internal process and data structures that are used to capture characteristics of the target system.

PSL does offer a number of useful relationship types which support our choice of property categories. These relationship types fall into several groups, some of which are roughly as follows: (i) System Flow -- a process may receive input data and generate output data; (ii) Data Derivation -- a process may use data to derive or update data; (iii) System Dynamics -- events occur when a condition becomes true/false or upon inception/termination of a process, and an event may trigger a process. (Most of the words used in (i)-(iii) above stem from the PSL vocabulary of keywords.) Many of the RMF property categories coincide with these useful PSL relationships, and all PSL statements can be expressed in RMF. Moreover, by applying the notions of symmetry inspired by SADT, we discovered other useful relationships. The resultant symmetry among RMF relationships is what permits a rather concise formalization of RMF.

RSL (the Requirement Statement Language) is part of perhaps the most comprehensive project to date to examine and improve the state of the art [2]. The language itself is a functional specification language oriented to real-time systems such as command and control systems. It offers a set of relationships similar to but somewhat more concise than PSL, with the major differences being due to its real-time-system orientation. RSL allows members of an "entity-class" to be members of one subordinate "entity-type". However, there is no enforced relationship between data associated with an entity-class and data associated with its entity-types. This is a simple form of a kind of subclass relationship and an example of where RMF attempts to provide a more general modeling facility.

RSL also offers a general graphical control flow specification feature called an Rnet. The Rnets describe partial ordering among processes, have a "subnet" notation for suppressing details, and provide for control flow events to "trigger" other events. Thus, Rnets provide a formal structure for specifying information about events and conditions. RMF uses its assertion objects in several roles (property categories) to express these kinds of information.

4.2 Semantic database modeling and knowledge representation

As illustrated by a recent workshop [9], researchers in several areas of Computer Science, notably in Artificial Intelligence (AI) and Data Base Management, have independently concluded that real-world modeling is of paramount importance for building computer systems, albeit each of these areas has goals and perspectives that differ somewhat from those of Software Engineering.

One of the central themes of AI is the Representation of Knowledge [8], which has been found indispensable for simulating human behavior (e.g., natural language understanding) and for building "expert" systems. Semantic Networks (see [7] for a review) have been used in AI for over a decade as ways of representing and especially organizing world knowledge through the notions of "nodes" (for entities) and "links" of various types (indicating types of relationships). The abstraction principles used in RMF are directly supported by many versions of semantic networks. Generalization, under the heading of "IS-A hierarchies", has received considerable attention in AI. The basic organization of RMF, modulo the assertion classes, has been directly influenced by this AI research, especially PSN [15]. The popularity of semantic networks in AI provides independent motivation to our work and further confidence in its appropriateness.

In the field of Data Bases, semantic or conceptual models (see [17] for an overview) have gained increasing popularity as ways of describing database schemata which enhance comprehensibility, and hence facilitate database design and maintenance. In fact, the terms "aggregation" and "generalization" were introduced in [24] in the context of database design. Semantic data models, however, concentrate by and large on the specification of objects (i.e. "data"). Increasing attention is being given to specification of constraints on the validity of data, and to a lesser extent to specification of activities. As discussed in [18], extending current modeling capabilities with respect to logical information and activities is essential to improving on current semantic models.

The Taxis model [19] is one of the few semantic data models that extends the use of abstraction facilities beyond data. In fact, Taxis uses aggregration, generalization, and classification for organizing relations, transactions, exceptions, and "scripts" for user interfaces [5]. Our current work on requirements modeling has been carried out within the framework of Taxis, with the purpose of providing a higher-level specification language that would be appropriate for expressing, as a special case of requirements specification, Corporate Requirements, the need for which is described in [16]. We expect, as well, that the use of the same abstraction principles in the RMF as in Taxis will enhance the utility of the RMF for information system design using Taxis.

Our approach is consistent with views in [29] that strongly advocate the use of semantic modeling in Software Engineering. Among other semantic modeling work relevant to requirements modeling, we note the work reported in [10],[23], and [25], which similarly emphasize the importance of modeling real-world phenomena as a system analysis approach. [23] presents a conceptual modeling approach based on semantic networks and uses an IS-A hierarchy for organizing concepts. Another language similar in spirit to RMF is presented in [27]; it is based on variations of the same abstraction principles provided by RMF.

We point out that although RMF has its roots in previous work in AI and Data Bases, it provides novel capabilities. These include assertions as another category of entities, property categories for defining (abbreviating) pertinent information types, and the uniform application of the abstraction principles to all object categories.

Of course, any descriptive framework based on classes must acknowledge Simula as a precursor. We also acknowledge Smalltalk (see Byte magazine, August 1981) as having influenced our basic framework; both RMF and Smalltalk are "object-oriented" (this is a different sense of the word "object" than used elsewhere in this paper) in that each specification unit encapsulates the description of some conceptual entity. Just how the ideas of these (as well as of certain other) programming languages compare to our framework is quite interesting, but such a discussion is beyond the scope of this paper.

5. AN ASSESSMENT OF RMF

5.1 Concerning methodology

At the heart of many software development methodologies lies one or more abstraction mechanisms, which allow us to ignore details at some level, plus a refinement principle which provides for the guided and gradual reintroduction of details across the abstraction dimension. The aggregation abstraction forms the core of software design methodologies such as "stepwise refinement" (e.g., [28]). Similarly, the "implementation" dimension is the basis for the abstract machine and abstract data type related methodologies (e.g., [20]). The generalization abstraction has not been exploited in Software Engineering as have the other dimensions. Yet, it is our contention that it is an invaluable organizational tool for system description in general, and for requirements modeling in particular.

The main idea of specification guided by generalization is that a model can be constructed by modeling first the most general classes, and then proceeding to more specialized classes. For example, in modeling a hospital world, one might consider first the concepts of patient, doctor, admission, treatment, etc. Later, the modeler can differentiate between child patients, heart patients, internists and surgeons, surgical and medical treatments, etc. At each step, only the information (properties) appropriate to that level are specified. (We do not rule out the need to iterate, i.e. to go back to revise previous level.)

Generalization is the appropriate principle to exploit when the difficulty of modeling is due to a large number of details rather than due to the complexity of the system/world; a hierarchy of classes organized along this dimension provides a convenient structure for distributing information (expressed uniformly as properties in RMF) and associating it where it most naturally belongs. Such stepwise refinement by specialization [6] is orthogonal and complementary to the more usual "stepwise refinement by aggregation", whose main effect is to decompose complex situations into a number of less complex ones. Both kinds of refinement are orthogonal and complementary to a third dimension, the progression from "world-oriented" specifications to specifications of a more and more completely implemented system.

5.2 An underlying model for RMF

Since descriptive languages are notoriously ambiguous, we are working on a detailed formal definition of a language based on RMF. We limit our discussion to an outline of the underlying formalism and the advantages of such a definition. (For a detailed presentation, see [13].)

The underlying model is based on a logic involving time, in which we can make assertions about the properties that any entity has with respect to special time entities called "situations". At any moment in time (i.e., in any situation), the "world" being described is characterized essentially by knowledge of what entities are instances of what classes (metaclasses,etc). Object classes have as instances those entities that are deemed to exist (i.e. to be relevant) at that time; an activity class has as instances activities that are occurring, or active, at that time; an assertion class is considered to have as instances assertions that are true at that moment.

Each period when an entity belongs to a class is characterized by an initial (insertion) time and a final (removal) time. During this period, the object is expected to have the factual properties induced by the definitional properties of the class. Thus, a description can, in fact, be expressed in the form of axioms defining the meaning of the "instance-of" and "subclass-of" relations.

Property categories can now be explained as designating axiom schemata, which provide templates for the axioms that represent properties in the respective property categories. For example, an initial condition for an object class expresses a condition that is true for each object that enters the class at its time of entry. This is captured in our logic by an axiom defining the property category initially as a predicate over properties of objects, involving the object, an assertion, and the insertion time (situation).

The axiom schemata give precise meaning to property categories, so that property categories can now be seen as abbreviations for commonly encountered restrictions on properties. The way is open for users of RMF to extend the list of property categories as dictated by the exigencies of special domains of discourse.

Such an underlying model relates an RMF description to formal semantics which will be useful for developing theoretical and pragmatic tools supporting the consistency of descriptions.

The big advantage of descriptions based on logic with time is that the descriptions are quite declarative. One has a view of the entire time-line (more precisely, over all relevant situations). Information that is typically represented by control flow specifications in other models is subsumed here by logical formulae involving situations plus information about the relationships between situations, which impose a partial (time) ordering on situations.

5.3 Uniformity

There are several senses in which the framework exhibits high uniformity. Given the "instance-of" relation and initial/final situations, it is straight-forward to define primitive insert/remove actions which add and delete entities from classes.

These actions are intuitive for objects. Applying the same ideas to assertions and activities, we model activation/termination of activities, and becoming true/false for assertions as the insertion/removal of entities of the respective categories.

The imposition of generalization hierarchies on each object category results in an interesting perspective as well. For objects, one can view an entity as starting out in a particular class and moving around on the hierarchy throughout its lifetime. For example, a person could become a child patient, later (by virtue of growing older) an adult patient, and so on. An activity's behavior can be viewed at several levels of generalization depending on what aspects of its participants (inputs, outputs, controls) its effects (preconditions, postconditions, and conditions it maintains), and components (parts) are associated at each level. For assertions, the imposition of a generalization hierarchy is particularly novel and interesting. Assertions viewed as entities are propositions whose (semantic) interpretation (i.e. specification of under what circumstances they are true) depends on the classes in which they reside. Clearly, in RMF, one assertion class is a specialization of another only if the former logically implies the latter. Property inheritance ensures consistency between assertion classes. Thus, the generalization abstraction organizes assertions according to both their arguments and their assertional import.

Concerning property categories, it turns out that most of those we have found useful can be defined in terms of a small number of items of information. There are many forms of uniformity (symmetry, duality) present. Many of the axiom schemata for the property categories are virtually identical except for, e.g., the entity category of the property subject or value, whether the insert vs. remove time is mentioned, the order of binding through quantifiers over classes and time, etc. As a simple example, initially, precondition, and inserted-by property categories all assert something about the insert time of the property subject, while postcondition and removed-by property categories all assert something about the remove time of the property subject. Parts and constraints are examples of property categories of objects that pertain to the entire instance interval and, in fact, their schemata are identical except for the fact that a part associates an object while a constraint associates an assertion.

5.4 Conclusion

We do not claim to have invented the abstraction mechanisms combined in the framework; rather, we have argued that they are independently motivated by several modeling endeavors. What we HAVE done is to combine them in a simple, constructive way, and we have explained some principles of interaction and their appropriateness for requirements specification.

When we say RMF "captures more world knowledge" than other specification techniques, we are referring to the semantic information that is conveyed by the three concept types, the kinds of relationships provided, and the use of assertions in roles where English is used in other techniques. More specifically, we mean that RMF captures world knowledge more formally (in the same sense that Predicate Calculus is more formal than English), and without resorting to more implementation oriented concepts.

We wish to stress that successful modeling depends not just on how one represents knowledge but on how one structures or organizes it. For example, Predicate Calculus would be adequate, from the point of view of expressibility, for representing knowledge; however, it does not provide good structuring facilities. We have argued in this paper that structuring/organizing a model should be based on useful abstraction mechanisms such as those offered by RMF.

In this paper we have bypassed discussing the important task of how the relevant terms of the domain of discourse are initially identified and recorded. We propose that this task should be done separately and thoroughly prior to RMF modeling. We intend to use an SADT-like technique to set up an initial ("structured") lexicon of the terms whose semantic relationships are of importance to the model. Our current research [13] investigates the connection between such a lexicon and the RMF model: how to proceed from the former to the latter and how to maintain consistency between them.

A common problem with using abstractions is that humans often over-abstract in an effort to establish regularity in their environment; thus, although at first sight all patients admitted to the hospital must have blood-pressure taken, some subclasses such as AMPUTEE may not, and even the most heartless hospital will not reject a patient because he doesn't know his health insurance number. One aspect of our current research concerns appropriate responses to such exceptional situations and how exception specifications serve as yet another abstraction principle in organizing large, detailed descriptions [4].

We also believe (calendar, clock) time to be essential to requirements modeling, since many requirements involve expressing things about time. The time model proposed in [10] would be appropriate and fits directly into our framework as object modeling; we would extend it to activity and assertion entities within RMF.

Finally, within the Taxis Project at the University of Toronto, we are applying the same general principles to different phases of Software Engineering, and we believe they will be a key factor in developing a unified approach to Software Engineering.

Acknowledgments: The authors wish to thank Brian Nixon for technical assistance during preparation of the manuscript, Theresa Miao for typing help, and, finally, the members of the Taxis Group for their constant support.

6. REFERENCES

[1] Bell, T. E., and T. A. Thayer, "Software Requirements: Are They Really a Problem?", Proceedings of the Second International Conference on Software Engineering, San Francisco, October 1976, pp. 61-68.

[2] Bell, T. E., D. C. Bixler, and M. E. Dyer, "An Extendible Approach to Computer-Aided Software Requirements Engineering," in [21], pp. 49-60.

[3] Boehm, B., "Software Engineering: R&D Trends and Defense Needs," in Wegner, P. (editor), Research Directions in Software Technology, MIT Press, 1979.

[4] Borgida, A., "Flexible handling of exceptions: a prospectus for research," Dept. of Computer Science, Rutgers Univ., Feb. 1982.

[5] Borgida, A., J. Mylopoulos, J., and H. K. T. Wong, "Methodological and Computer Aids for Interactive Information System Design," in Automated Tools for Information System Design,, H.-J. Schneider and A. Wasserman (editors), IFIP, North-Holland, 1982.

[6] Borgida, A., J. Mylopoulos, and H. K. T. Wong, "Taxonomic Software Specifications," in M. Brodie, J. Mylopoulos, and J. Schmidt (Eds.), Perspectives on Conceptual Modelling, Springer-Verlag, 1982.

[7] Brachman, R. J., "On the Epistemological Status of Semantic Networks," in [12].

[8] Brachman, R. and B. Smith (editors), Special Issue on Knowledge Representation, SIGART No. 50, February 1980.

[9] Brodie, M. L., and S. N. Zilles (eds.), Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park, CO, 23-26 June 1980, SIGPLAN Notices, Volume 16, No. 1, Jan.1981.

[10] Bubenko, J. A., "Information Modeling in the Context of System Development," IFIP 80.

[11] Dijkstra, E. W., "Notes on Structured Programming," Structured Programming, Academic Press, 1972.

[12] Findler, N. (Editor), Associative Networks, Academic Press, 1979.

[13] Greenspan, S. J., Ph. D. Thesis on Requirements Modeling, Dept. of Computer Science, University of Toronto (forthcoming).

[14] Hoare, C. A. R., "Notes on Data Structuring," in Structured Programming, Academic Press, 1972.

[15] Levesque, H. J., and J. Mylopoulos, "A Procedural Approach to Semantic Networks," in [12], pp. 93-120.

[16] Lum, V., et al., 1978 New Orleans Data Base Design Workshop Report, IBM Research Report RJ2554, San Jose, July 1979.

[17] McLeod, D. and R. King, "Semantic Database Models," in Principles of Database Design, S. B. Yao (editor), Prentice Hall, 1981.

[18] McLeod, D., and J. Smith, "Abstraction in Databases," in [9].

[19] Mylopoulos, M., P. A. Bernstein, and H. K. T. Wong, "A Language Facility for Designing Interactive Database-Intensive Application," ACM Transactions on Database Systems, Volume 5, Number 2, June 1980, pp. 185-207.

[20] Parnas, D., "On the Criteria to be Used in Decomposing Systems Into Modules," Comm. ACM, Vol 15, No. 12, December 1972, pp. 1053-1058.

[21] Ross, D. T. (guest editor) Special Issue on Requirements Analysis," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977.

[22] Ross, D. T., "Structured Analysis(SA): A Language for Communicating Ideas," in [21], pp. 16-34.

[23] Roussopoulos, N., "CSDL: A Conceptual Schema Definition Language for the Design of Data Base Applications," IEEE Transactions on Software Engineering, Volume SE-5, Number 5, September 1979, pp. 481-496.

[24] Smith, J., and D. Smith, "Database Abstractions: Aggregation and Generalization," TODS, Vol. 2, No. 2, 1977, pp.105-133.

[25] Solvberg, A., "A Contribution to the Definition of Concepts for Expressing Users' Information Systems Requirements," Proc. International Conf. on Entity-Relationship Approach to Systems Analysis and Design, December 10-12,1979, pp. 359-380.

[26] Teichroew, D. and E. Hershey, III, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," in [21], pp.41-48.

[27] Wilson, M., "A Semantics-Based Approach to Requirements Analysis and System Design," Proc. COMPSAC 79, Nov. 1979, pp.107-112.

[28] Wirth, N., "Program Development by Stepwise Refinement," Comm ACM, April 1971.

[29] Yeh, R. et al., "Software Requirement Engineering: A Perspective," Dept. of Computer Science, Univ. of Texas, Austin, 1979.