JOURNAL OF INFORMATION SYSTEMS
Vol. 20, No. 2
Fall 2006
pp. 37–63

Policy-Level Specifications in REA Enterprise Information Systems

Guido L. Geerts
University of Delaware

William E. McCarthy
Michigan State University

ABSTRACT: The Resource-Event-Agent (REA) enterprise model is a widely accepted framework for the design of the accountability infrastructure of enterprise information systems. Policy-level specifications define constraints and guidelines under which an enterprise operates, and they are an extension to the REA enterprise model, adding the "what should, could, or must be" to the "what is." This paper aims both at comprehensive understanding of policy-level definitions as part of REA enterprise systems and at understanding of the semantic constructs that enable such definitions. We first explore two distinctive semantic abstractions essential to policy-level specifications: typification and grouping. The typification abstraction links instances of an object class to concepts for which they are concrete realizations, while the grouping abstraction aggregates objects into collections. We next present a number of patterns for the semantic modeling of policies. Following, we look at policy-level applications for REA enterprise information systems. We explore type and grouping definitions for the REA primitives (resource, event, agent) and discuss enterprise applications for three different kinds of policy definitions: knowledge-intensive descriptions, validation rules, and target descriptions. Our discussion of specific enterprise applications includes internal control applications (e.g., limit checks), variance analysis based on standard specifications (e.g., bills of materials), and budgeting applications.

Keywords: policy-level specifications; grouping; REA enterprise information systems; typification.

I. INTRODUCTION

An integral part of enterprise information systems development is the specification of an enterprise or conceptual model. A conceptual model is an abstract representation of reality defined in terms of semantic abstractions. Abstraction is a mental process where some characteristics of a set of objects are selected for analysis and where other characteristics that are not relevant are excluded. Examples of semantic abstractions include classification, aggregation, and generalization/specialization (McCarthy 1987; Batini et al. 1992; Odell 1998; Larman 2002). An object class or entity set classifies a set

We acknowledge the helpful comments of Del DeVries, Severin Grabski, Pavel Hruby, Jesper Kiehn, the editor, two anonymous reviewers, numerous MS and MBA students at the University of Delaware and Michigan State University, workshop participants at the University of Delaware, participants of the AAA Intensive Workshop on Teaching AIS, and participants at presentations given at the 2005 AAA Annual Meeting and the 2006 Midyear Meeting of the AAA Information Systems Section. Financial support for this work was provided by the University of Delaware, Lerner College of Business and Economics.

of objects (instances) with similar characteristics or behavior, such as the people engaged (instances) by a specific organization as employees (object class). Aggregation is a semantic abstraction that describes a composite object (whole) in terms of the objects of which it consists (parts). Aggregation is used to associate different properties of a class together (such as the names, addresses, and telephone-numbers of employees), to associate instances of different classes together by physical containment (such as a bike consisting of a frame, a saddle, and wheel parts), and to depict group-membership associations (such as a company consisting of a collection of departments or a union consisting of a collection of employees). Generalization/specialization is a semantic abstraction used to model supertypes of a class (such as inventory as a generalization of raw-material and finished-goods) or subtypes of a class (such as salesperson, cashier, and buyer as specializations of employee). There are some differences in how these abstractions are used in different computing paradigms. For example, conceptual models defined for object-oriented systems consider behavioral specifications, while those defined for database systems pay far less heed to integrated expression of procedures. For simplicity, we ignore such differences in this paper and focus on the core abstractions as they apply across paradigms.

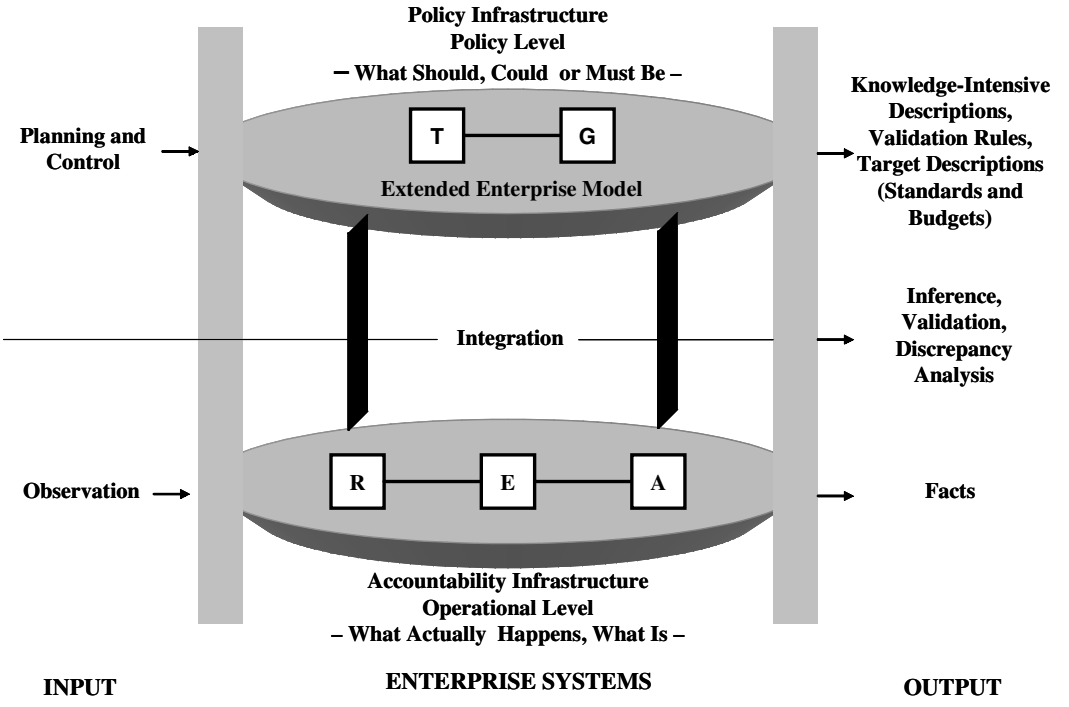
The Resource-Event-Agent (REA) enterprise model (McCarthy 1982) is a widely accepted framework for the conceptual design of the accountability infrastructure of an enterprise information system; that is, a representation of the resource flows within and between firms in terms of what is currently occurring or what has occurred in the past. However, many applications require an extension to the REA enterprise model with a policy infrastructure that describes what should, could, or must be occurring sometime in the future. An important aspect of policy-level specifications is linking them with current operations. For example, information about how to make an engine is linked with the actual manufacturing of the engine. Discrepancies between what should have occurred and what actually occurred can then be analyzed further: Why did manufacturing the engine take longer than the time prescribed by the engineering standards? Two distinct abstractions that are instrumental in the semantic modeling of the policy infrastructure are typification and grouping. The typification abstraction captures concept descriptions that apply to a set of objects. The grouping abstraction corresponds to the group-membership special form of aggregation, and it groups objects together in collections.

This paper focuses primarily on the use of the typification and grouping semantic abstractions to specify policy-level extensions to REA enterprise systems. The remainder of the paper is organized as follows. Section II discusses policy-level extensions to REA enterprise systems: the different kinds of policy definitions, the integration of the policy and accountability infrastructures, and the heuristic nature of policy specifications. Section III presents a comprehensive study of the typification and grouping abstractions as well as a number of patterns for the definition of policies. Section IV discusses policy applications for REA enterprise information systems. First, type and grouping definitions for the REA primitives are explored. Next, enterprise applications for three different kinds of policy definitions are discussed in detail: knowledge-intensive descriptions, validation rules, and target descriptions. Finally, section V presents the conclusion.

II. A POLICY-LEVEL EXTENSION TO REA ENTERPRISE SYSTEMS

Figure 1 presents a policy-level extension to Resource-Event-Agent (REA) enterprise systems. The lower part of Figure 1 represents the economic activities that actually happen in a company; input here results from observations (or automated measurements) of these

FIGURE 1
Policy-Level Specifications in REA Enterprise Systems



activities. The upper part of Figure 1 represents the economic activities that should, could, or must happen in a company, and input there results from planning and control activities. ''T'' and ''G'' represent the two key semantic abstractions we are going to use in this paper to model policies: Typification and Grouping. The middle layer shows that integrating the policy and accountability infrastructures enables inference, validation, and discrepancy analysis.

The accountability infrastructure or operational level of REA enterprise systems is well-documented in the literature (McCarthy 1982, Dunn et al. 2005). In essence, the REA model is a pattern for the semantic definition of business processes. Phenomena captured by the REA model include the economic activities that take place in a company, the resources that are acquired and consumed, and the agents who are accountable for economic activities.

The Business Rules Group (2000) defines a policy as ''a general statement of direction for an enterprise.'' In this paper, the term ''policy'' refers to a description of economic phenomena that could, should, or must occur. We distinguish among the following three types of policy definitions: knowledge-intensive descriptions, validation rules, and target descriptions. A knowledge-intensive description defines characteristics of a concept that apply to a group of objects. Such characteristics can take the form of a policy definition: e.g., the price of any bottle of Chanel No. 5 is $75. Actual instances, e.g., an actual bottle of Chanel No. 5, can then derive the policy-based characteristic through inference: ''If a

bottle is of type Chanel No. 5 then its price must be $75.'' A validation rule represents permissible values, and a common application of validation rules in enterprise systems is preventive controls. For example, the salary for an employee should be validated against the salary range defined for his or her employee type, such as ''A staff member should earn between $25,000 and $40,000.'' Target descriptions provide benchmarks regarding economic phenomena, and they can take at least two different forms: standards and budgets. Generally, standards are specifications to be followed; however, they may be tweaked (like changing a cookie recipe). Standards often refer to engineering information, for example: ''How much raw material does it take to manufacture a bike?'' or ''What are the best practices for assembling cars?'' Budgets provide quantified performance measures most often related to a specific time period such as ''How many cars do we expect to sell in the second quarter of 2006?''

The upper right side of Figure 1 illustrates knowledge-intensive descriptions, validation rules, and target descriptions as the main output of the planning and control activities and the mechanisms to define the policy infrastructure. The middle right side of Figure 1 shows enterprise applications that rely on the integration of the accountability and policy infrastructures. For example, through inference, policies defined for concepts can be applied to their actual instantiations, validation can insure that descriptions of actual enterprise phenomena do not violate the rules defined as part of the policy infrastructure, and discrepancy analysis can compare actual phenomena with their target descriptions (a common accounting application of this last case is variance analysis).

The accountability and policy infrastructures of REA enterprise systems strongly differ in nature. The accountability infrastructure defines ''what is'' and is structured following a set of normative, domain-specific modeling rules. The REA transaction pattern helps to structure the description of business processes by defining the domain-specific object classes (resource, event, and agent) and associations (stock-flow, duality, and participation) that should be part of such a description. The template further embeds the following domain-specific structuring rules: (1) an economic event must be part of a duality association; that is, economic events resulting in an inflow of resources must be linked with economic events resulting in an outflow of resources, and (2) all economic resources must participate in an inflow and an outflow association. The first rule affects economic consideration between resource flows, while the second insures the integration of business process descriptions into an enterprise value chain. The policy infrastructure, on the other hand, represents mechanisms designed to plan, control, and evaluate economic activities, and it is essentially nonnormative in nature; i.e., there are no domain-specific rules mandated to structure policy definitions. A business entity may often decide to eschew policy-level extensions for one or more elements of the accountability infrastructure. For example, certain companies could decide not to record budgetary information for a specific economic event (e.g., they do not define or maintain purchase budgets) or single-product companies would have no need for product type definitions.

III. POLICY DEFINITIONS WITH THE TYPIFICATION AND GROUPING SEMANTIC ABSTRACTIONS

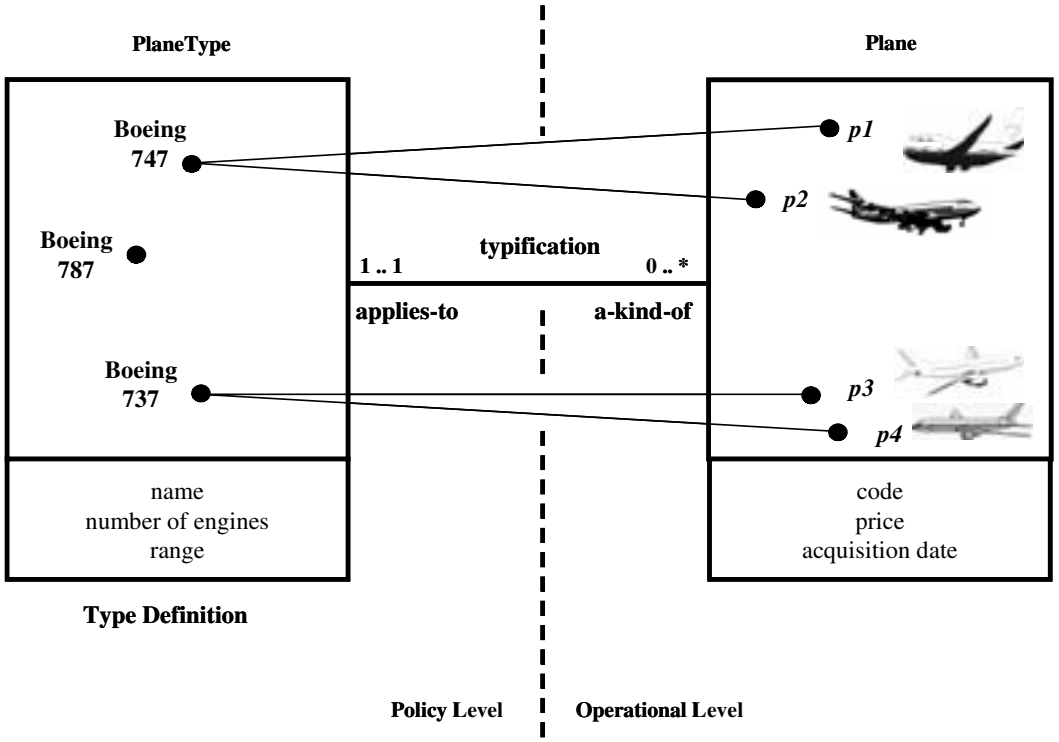
The Typification and Grouping Semantic Abstractions

The concept of typification goes back to Plato and other Greek philosophers (Tarnas 1991) where type images are described as ''archetypal forms.'' For example, a horse is a real thing, while ''horseness'' is an abstract concept (Sowa 1999; McCarthy 2002). In addition to philosophers, typification has been studied extensively by computer science

researchers including Smith and Smith (1977a, 1977b), Brodie (1981), Sakai (1981), Hammer and McLeod (1981), Fikes and Kehler (1985), and more recently, Goldstein and Storey (1994), Hay (1996), Fowler (1997), Odell (1998), Scheer (1998), Eriksson and Penker (2000), and Silverston (2001). While differences in notation, application domain, and computing paradigm exist among these researchers, they all present the idea of typification as a semantic abstraction useful for capturing concept descriptions that apply to a set of objects.

An example of typification is illustrated in Figure 2.[1] Instances of PlaneType represent concepts or type definitions, and the Plane-PlaneType association represents a typification abstraction. The use of the a-kind-of and applies-to roles in Figure 2 further elucidates the semantics of typification. The definition of a Boeing 747 applies to all its realizations {p1,p2}[2], while in the opposite direction, p1 is a-kind-of Boeing 747. The instances of a type definition, such as PlaneType in Figure 2, are considered to possess archetypal essences

FIGURE 2
Typification



[1] We use most of the conventions of a UML (Unified Modeling Language) class diagram (Booch et al. 1999) in this paper. For simplicity reasons, we do not use UML's notation for aggregation: a diamond at the aggregate end. For illustration purposes, we use black circles to represent instances of an object class; e.g., in Figure 2, a Boeing 747 is an instance of the PlaneType object class. When we use a UML class diagram, we adopt the common convention of using ''camel case'' to describe our examples.

[2] We use {} to represent instances.

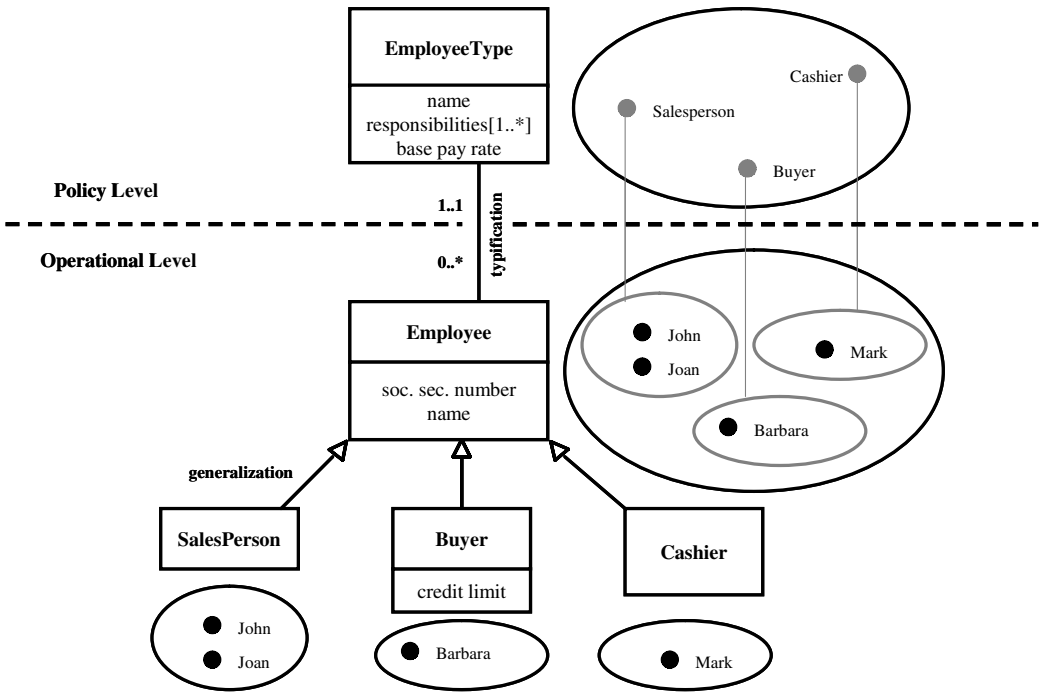
that portray ideal or typical components of objects, such as a Boeing 747 having four engines and a range of 7,500 miles. The descriptions of the type class are definitional in nature, such as giving the distinguishing characteristics of a Boeing 747. The example in Figure 2 further illustrates that it is possible to have a type definition for which there are no current realizations (Boeing 787).

Typification and generalization abstractions are intertwined as seen in Figure 3 which illustrates the following:

(1) typification: the explicit linking between actual employees (instances of Employee: {John, Joan, Mark, Barbara}) and different employee categories (instances of EmployeeType: {Salesperson, Buyer, Cashier}).
(2) generalization: the object class Employee is decomposed in the generalization plane, and employees are assigned to their appropriate subtypes—John and Joan become instances of Salesperson, for example.

While both abstractions represent the same categories (Salesperson, Buyer, Cashier), the information captured differs substantially. Instances of the entity Employee represent actual employees with a social security number, a name, etc. A set of employees becomes a subtype if it has differential properties, differential participation in relationships, or (in an object-oriented environment) differential behavior. For example, a credit limit is a characteristic to be recorded for instances of Buyer only. On the other hand, instances of

FIGURE 3
Typification and Generalization



EmployeeType represent concepts, and they further describe each of the subtype object classes. An example of information gathered for employee type instances is the particular job responsibilities of a type. So, instead of recording additional information for all instances of a subtype (e.g., all buyers), the type definitions define the characteristics of the subtypes themselves (e.g., What is a buyer? What are the characteristics of a buyer?). The semantic abstraction in Figure 3, where the type instances are also the names of object classes, is labeled a ''generic entity'' in Smith and Smith (1977b) and a ''power type'' in Odell (1998). Odell (1998, 28) describes a power type as follows: ''an object type whose instances are subtypes of another object type.''

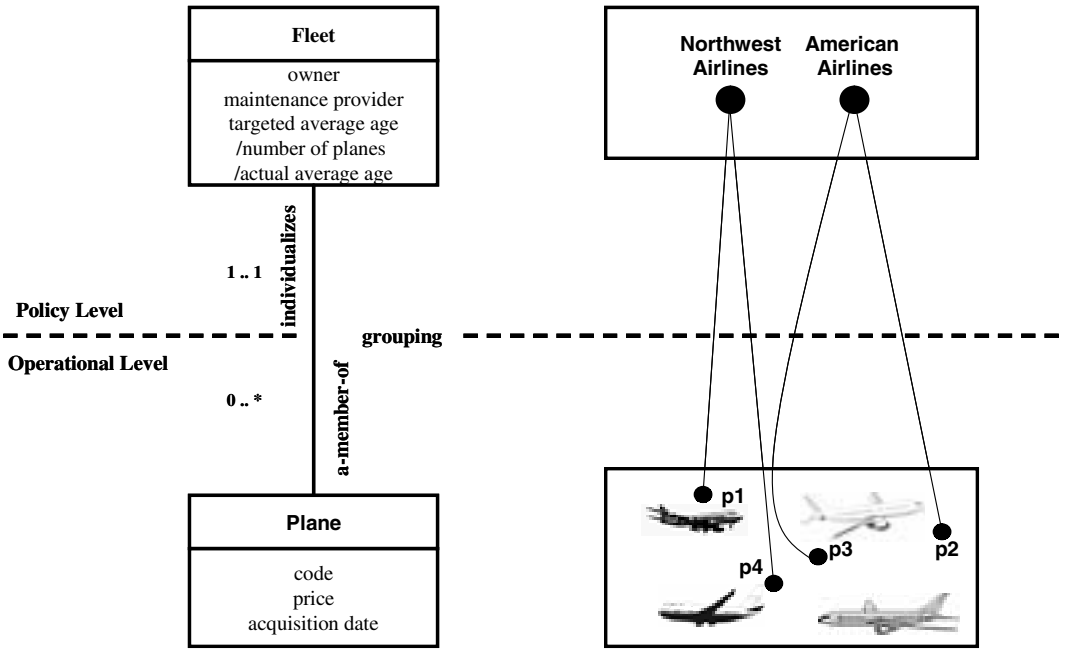
While type definitions and subtypes capture two different kinds of information needs for the same object class, each of them may exist without the other. When there are no differentiating individual characteristics or behaviors of interest among groups of employees, the subtypes can be left out. However, EmployeeType can still be used to record the job responsibilities for specific categories of employees. Similarly, it is possible to record specific characteristics for a subtype without being interested in generic or static information for any of the subtypes. Most generally, the use of the typification abstraction in a model warrants additional consideration of expansion in the generalization sense. However, in this paper, we choose to focus on policy-level modeling issues, so our examples concentrate largely on typification being used by itself.

Grouping has been extensively studied by researchers including Brodie (1981) and Motschnig-Pitrik and Storey (1995). Grouping is considered as a special form of aggregation, and it groups objects into collections based on one or more common characteristics. The grouping abstraction emphasizes the properties of the set object class, and it is considered as a modeling option whenever set-level characteristics are of interest.

Figure 4 illustrates an example of grouping. Planes are grouped into collections named fleets, based on ownership. Plane is the member object class, Fleet is the grouping object class, instances of Fleet represent grouping definitions, and the Plane-Fleet association represents a grouping abstraction. The a-member-of and individualizes (Taivalsaari 1996) roles in Figure 4 further elucidate the semantics of grouping.

Next, we discuss four key characteristics of the grouping abstraction. First, a grouping association represents membership in a collection and not an is-a-kind-of association which is the case for typification. For example, a plane is a member of a fleet but not a kind of fleet. There is typically no close relationship between grouping and generalization and, as a result, it is unlikely that a grouping object class is a power type (i.e., it is unlikely that its instances are subtypes of its member class). For example, instances of Fleet (e.g., Northwest Airlines) are not subtypes of the Plane object class. Second, a grouping object class describes set-level characteristics; i.e., characteristics that apply to the collection as a whole and that are shared by all members of the collection. Examples of set-level characteristics in Figure 4 are owner, maintenance-provider, and targeted-average-age. Third, a characteristic of grouping abstractions is the near omnipresence of derived attributes. The Fleet object class in Figure 4 has two derived attributes: (1) /number-of-planes—the number of planes in the fleet, and (2) /actual-average-age—the average age of a plane in the fleet. The UML notation for a derived attribute is a slash (/) in front of the attribute. The inclusion of a derived attribute as part of the actual information system will be based on parameters such as its access statistics and the cost of computing the attribute. A generally applied heuristic is to include derived attributes that are infrequently updated but are frequently accessed (Batini et al. 1992). Fourth, as described by Motschnig-Pitrik and Storey (1995), membership associations are nontransitive, as is illustrated by the following example. The facts that

FIGURE 4
Grouping



the plane with id 87B5 is a member of the United fleet and that United is a member of the Star Alliance does not imply that the plane with id 87B5 is a member of the Star Alliance; airlines are members of alliances, not planes. Table 1 summarizes some important differences between the typification and grouping semantic abstractions.

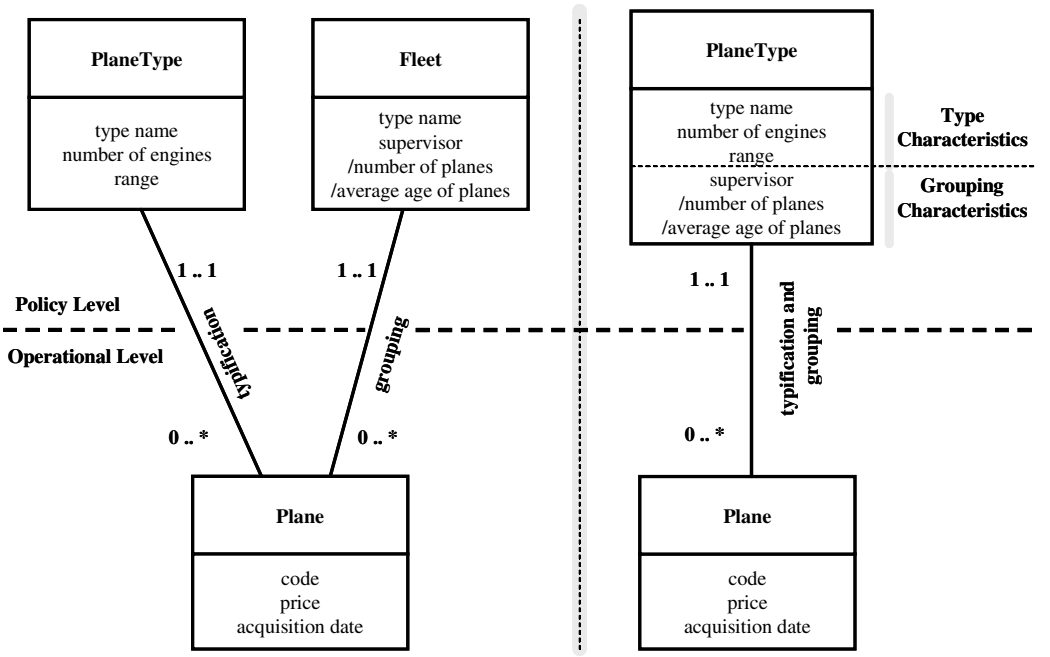
The semantics of the typification and grouping abstractions can further be extended by the definition of multiplicities. Type definitions always apply to a number of instances, and their upper multiplicity is many (*). For example, the definition of a Boeing 747 applies to all current and future planes of that category. Similarly, grouping definitions represent collections; therefore, they are associated with multiple members and their upper multiplicity is many (*). For example, a fleet has many airplanes. Exclusive categorization and membership can be expressed by an upper multiplicity of one: a plane has only one type and a plane is member of only one fleet. However, shared classification and membership are also possible: an employee can have many skills (shared classification) and an employee can be assigned to more than one department (shared membership).

Finally, there are some issues related to typification and grouping that need further clarification. From a practical point of view, the distinction between typification and grouping is not always definitive. This is illustrated by the example in Figure 5. Fleet now groups planes by plane type (instead of ownership) and records set-level characteristics shared by the collection of Boeing 747s, Boeing 737s, etc.—supervisor, number-of-planes, and average-age-of-planes. When type definitions are used for grouping, the type and grouping object classes are typically merged into one hybrid object class as is shown on the right-hand side of Figure 5. The PlaneType object class now represents both a type definition

TABLE 1
Typification versus Grouping

	Typification	Grouping
Nature of Abstraction	**is a-kind-of** (applies to). The plane with id 87B5 is **a-kind-of** Boeing 747. Type definitions present archetypal essence and are timeless in nature; i.e., they apply to all current and future realizations.	**is a-member-of** (individualizes). The plane with id 87B5 is **a-member-of** the Northwest fleet. A grouping defines a collection with common characteristics shared by all its members.
Power Types	Integral to the notion of typification.	It is unlikely to have a grouping class whose instances are subtypes of its member class.
Derived Attributes	Not usually relevant. Instances of types do not represent collections.	Highly relevant due to the collection nature of groupings.
Transitivity	Transitive. If a plane is a-kind-of Boeing 747 and a Boeing 747 is a-kind-of jet, then the plane is a-kind-of jet.	Nontransitive. If the plane with id 87B5 is a-member-of the United fleet and United is a-member-of the Star Alliance, that does not imply that the plane with id 87B5 is a-member-of the Star Alliance.

FIGURE 5
Hybrid Represenation of Type and Grouping Definitions



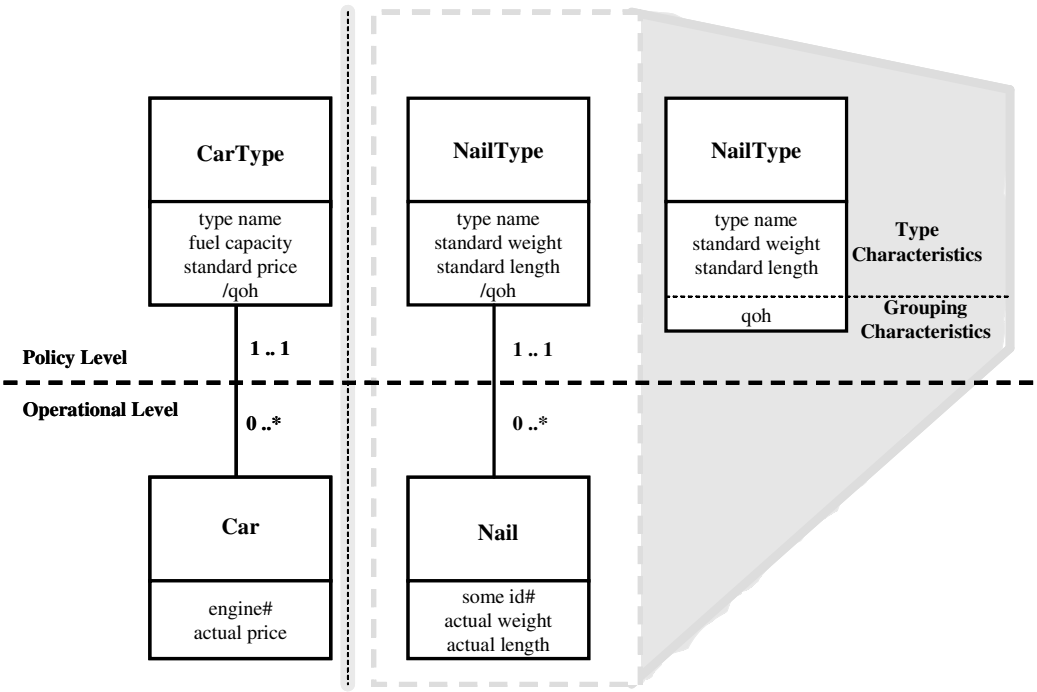
and a grouping definition. This is a common situation for recording inventory information in enterprise systems: both the characteristics (typification) and the number available (grouping) of each of the item types are recorded.

Figure 6 extends this issue with a common trade-off in building enterprise information systems. We use ''Car'' as the prototypical example of resources that are individually traced. Engine# and actual-price are characteristics recorded for instances of Car (i.e., for specific cars). Instances of CarType represent both type definitions (type-name, fuel-capacity, standard-price) and grouping definitions (quantity-on-hand [qoh]). We use ''Nail'' as the prototypical example of resources for which it is not economical to uniquely identify instances. The model for Nail is further compromised as is illustrated on the right side of Figure 6. Information about individual nails is no longer recorded, but the qoh attribute summarizes information about the actual number of nails available. Qoh is no longer a derived attribute. Such a compromised definition is common for mass-produced, inexpensive products.

Policy-Level Associations

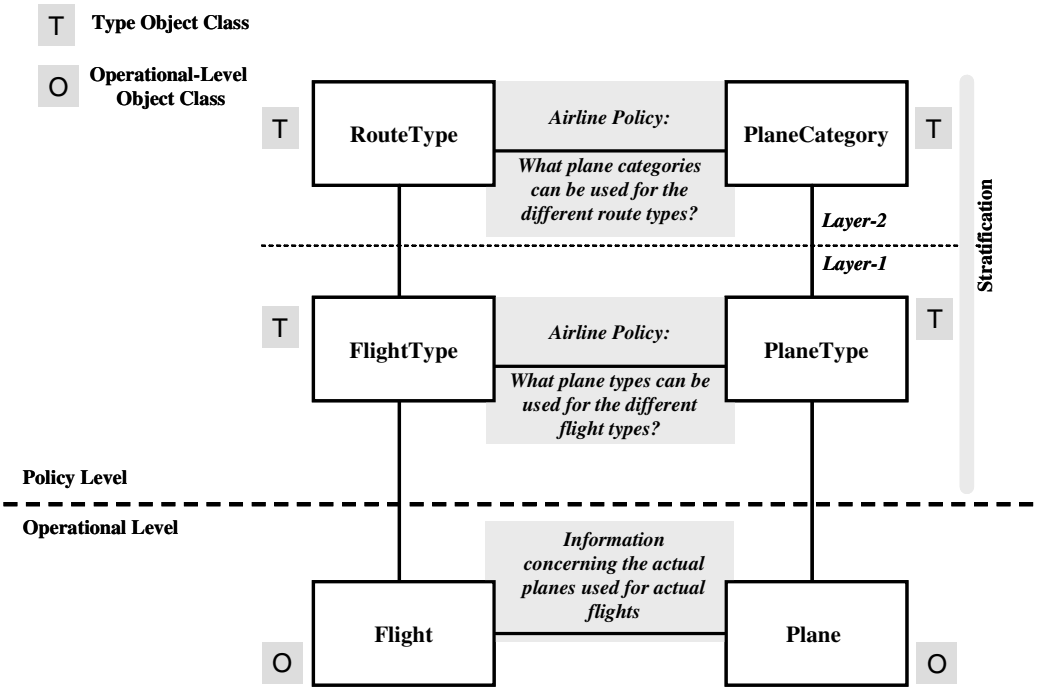
An association is a relationship between object classes ''that indicates some meaningful and interesting connection'' (Larman 2002, 153). The examples in Figure 2 and Figure 4 illustrate typification and grouping as associations that link operational-level object classes

FIGURE 6
Compromised Definition for Mass-Produced Inexpensive Products



with policy-level object classes. However, associations can also be defined between policy-level object classes[3] as is illustrated in Figure 7. Stratified definitions represent a first kind of policy-level association. On the right side of Figure 7, individual planes are classified in terms of their type (Boeing 747), and types are then further classified as categories (jet, propeller plane). Similarly, the left side of Figure 7 illustrates a stratified definition for flights. Instances of FlightType describe the common characteristics of scheduled flights. For example, the scheduled departure time for the daily flight from Philadelphia to Tampa is 8:00 a.m. An instance of Flight describes an actual flight taking place. For example, the flight actually left Philadelphia at 8:15 a.m. Instances of RouteType (e.g., international flights) represent common characteristics that apply to flight types (and thus flights). For example, a passport is required for all international flights. By contrast with the flight and plane typifications, the following example illustrates a stratified grouping definition: basketball players (Magic Johnson) are grouped into teams (the Michigan State University basketball team), and teams are then further grouped into athletic conferences (Big Ten). Another example of a stratified grouping definition was discussed above: Plane → Fleet → Alliance.

FIGURE 7
Policy-Level Associations



[3] Our use of associations that directly link one abstract class with another abstract class for purposes of specifying enterprise policies is very strongly related to the artificial intelligence notion of heuristic match, advanced originally by Clancey (1985). See McCarthy and Outslay (1989, 24–26) for an accounting example of a heuristic match decision.

The diagram in Figure 7 also illustrates two nonstratification, policy-level associations: (1) FlightType-PlaneType: the types of plane that should be used for the different flight types (for example: the daily flight from Philadelphia to Tampa should use a Boeing 757), and (2) RouteType-PlaneCategory: the plane categories that should be used for the different routes (for example: jet planes should be used for international flights). Both associations represent rules that restrict the permissible occurrences of the Flight-Plane association at the operational level. Similar policy-level associations can also be defined between groupings (for example: which team [grouping] of stockbrokers manages a portfolio [grouping] of stocks), or between a type and a grouping (for example: what plane types [type] can be part of a fleet [grouping]).

Policy Definitions

The policy infrastructure consists of policy-level object classes and associations. The following are three mechanisms that can be used for the actual definition of policies: attributes, associations, and association classes. Each of these is described in a paragraph below.

First with respect to attributes, they can define characteristics of concepts that are then applied to operational objects through inference. For example, all Boeing 747s have four engines, and if a plane is classified as a Boeing 747, then it must have four engines. Similarly, sale price can be defined for a product type and then be applied to all its instantiations. Second, attributes can define validation rules. For example, the fact that the take-off weight of a Boeing 747 cannot exceed 413 tons can be recorded as an attribute of PlaneType called maximum-take-off-weight. The actual take-off weight of a flight with that type of airplane is then validated against this rule. Third, attributes can also define targets. For example, the expected departure time of each flight type will be recorded, and discrepancies with actual departure times are then used to analyze delays.

An association can define both valid and invalid interactions between policy-level object classes. The example in Figure 7 defines valid interactions between plane categories and route types. An example of a rule defined by this association is that jets should be used for international flights. Associations can also express invalid interactions that should not occur at the operational level. The policies defined in Figure 7 by the association between RouteType and PlaneCategory could be rephrased in the following way: the plane categories that cannot be used for each of the route types. An example of such a policy would be that propeller planes cannot be used for international flights. Associations can also be used to define targets. For example, the FlightType-PlaneType association in Figure 7 could be used to describe the following best practice information: the recommended plane types for each flight type.

An association class represents an association that has attributes of its own. The attributes can then be used to define policies in the same ways as described above. The existence of an association class at the policy level does not necessarily imply the existence of a corresponding association class at the operational level. Consider the example where a targeted-fuel-consumption attribute is defined for the association between FlightType and PlaneType. However, the actual fuel consumption will be recorded as an attribute of the Flight object class as there is only one plane per flight. To determine variances, the targeted-fuel-consumption attribute of the association class (what should be) will be compared with the actual-fuel-consumption attribute of the Flight object class (what actually occurred).

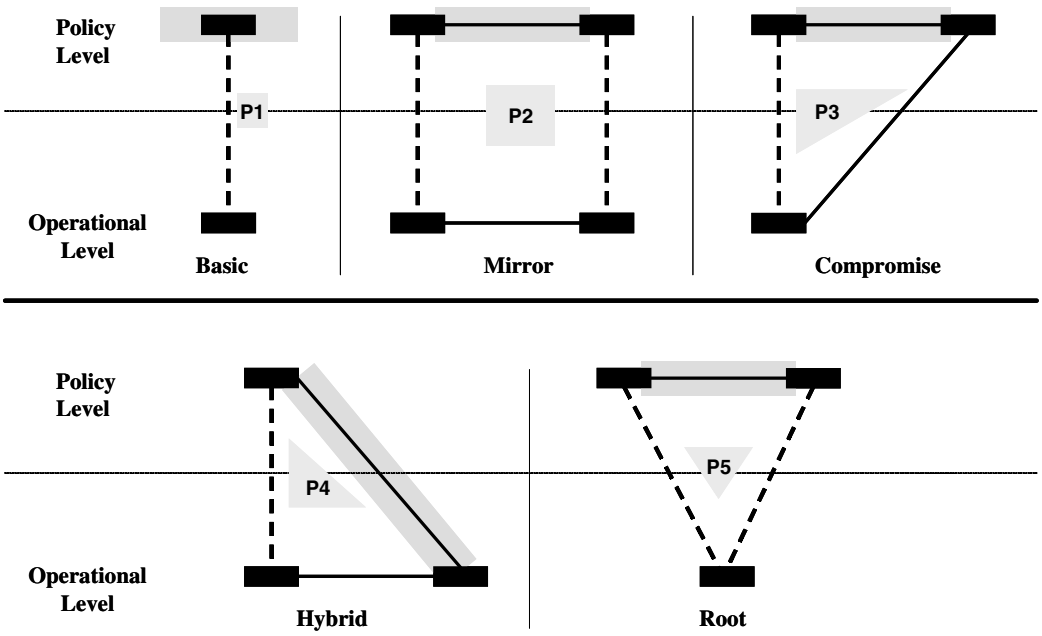
Patterns for Policy-Level Specifications

The definition of a policy-level architecture typically requires the application of one or more of the patterns[4] shown in Figure 8. The following notation is used in Figure 8 to draw attention to the composition of each of the patterns: black rectangles represent object classes, dotted lines represent typification or grouping abstractions, solid lines represent all associations that are not typification or grouping abstractions, and gray shaded areas represent policy definitions. Figure 9 presents an integrated example that illustrates each of the five patterns.

The underlying structure of the ''Basic'' pattern (P1) is a single typification or grouping association and policies are defined with attributes. In Figure 9, the Flight-FlightType association represents an instantiation of the basic pattern with the scheduled-departure-time attribute defining the policy: the time a flight should take off. The actual-departure-time attribute of Flight allows comparing the targeted departure time (what should be) with the actual departure time (what is). The other four patterns (P2, P3, P4, P5) in Figure 8 have the following two characteristics in common: (1) they all integrate at least one typification or grouping abstraction, and (2) they use an association (or association class) for the definition of policies. Patterns P2, P3, and P4, define policies regarding the content of an operational-level association. On the other hand, pattern P5 defines restrictions or guidelines related to multiple classifications (a-kind-of or a-member-of) of the same object class.

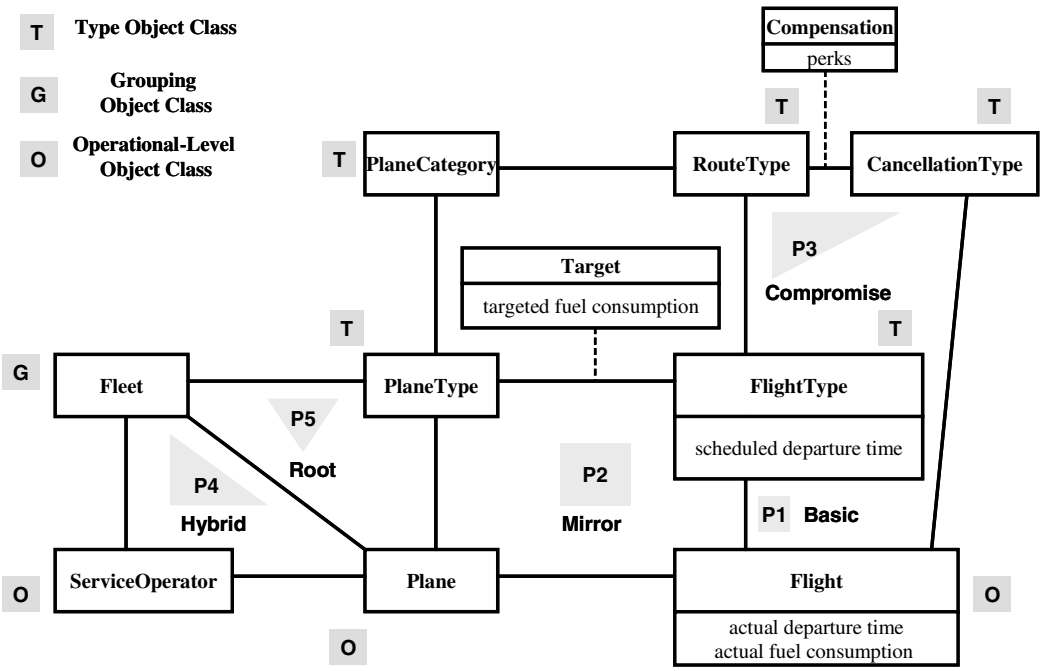
A ''Mirror'' pattern (P2) has the following configuration: (1) two associations, one at the operational level and one at the policy level, and (2) connections between the two

FIGURE 8
Patterns for Policy-Level Specifications



[4] We use ''pattern'' here in the same sense as Fowler (1997).

FIGURE 9
Stereotypical Patterns Applied



associations through typification and/or grouping abstractions. Figure 9 illustrates the following instantiation of the mirror pattern:

- operational-level association → Plane-Flight;
- policy-level association → PlaneType-FlightType; and
- two typification connection associations →Plane-PlaneType and Flight-FlightType.

The configuration of the mirror pattern can be used for the definition of a variety of policies as is illustrated by the following two examples. First, the PlaneType-FlightType association in Figure 9 determines the permissible content of the Plane-Flight association; that is, when a plane is assigned to a flight, then the plane's type (typification) must be one of the plane types approved for the flight's flight type (typification). Stated differently, valid instances of the Plane-Flight association—and thus its possible content—are determined by the instances of the PlaneType-FlightType policy-level association. Second, the Target association class defines the targeted fuel consumption per flight type per plane type. The actual fuel consumption is recorded as an attribute of the Flight object class. While the configuration (mirror pattern) is the same, the nature of the policy and the mechanisms used for its definition differ when compared to the first example. The policy defines a target (the expected fuel consumption) instead of a validation rule. Attributes are used as a mechanism for the description of the targeted values and the actual values.

Policy definitions, other than the ones illustrated in Figure 9, can be accommodated by the mirror pattern. For example, the instances of a policy-level association can define targets which are the expected combinations. Finally, with respect to the mirror pattern, Figure 9

further illustrates how it can be stratified. The instances of the PlaneCategory-RouteType association further restrict the possible instances of the PlaneType-FlightType association and thus the Plane-Flight association.

While presented by itself in Figure 8, the ''Compromise''[5] (P3) pattern is really a variation of the mirror pattern. It occurs when, similar to the ''Nail'' prototype discussed earlier in the paper, one of the typification or grouping abstractions in the mirror pattern is compromised. Consider the example in Figure 9. The policy is defined between RouteType and CancellationType. The missing object class in Figure 9 is CancellationEvent which would describe the actual events resulting in flight cancellations such as a winter storm, a mechanical problem, or a strike. Similar to the ''Nail'' prototype, the airline has decided not to record the actual cancellation events, but only to record the type of cancellation such as weather, mechanical, or labor. The Compensation association class describes the perks to be given when a flight gets cancelled. For example, a hotel room and a meal need to be offered (perks) when an international flight (route type) gets canceled due to weather (cancellation type). It needs to be emphasized that the association between Flight (operational level) and CancellationType (policy level) is not a typification or grouping abstraction; a cancellation type is a characteristic of a flight but a flight is not of type weather, mechanical or labor. Similar to the other patterns, the ''Compromise'' pattern configuration can be used to define different kinds of policies. The example in Figure 9 determines (inference) the compensation for a cancelled flight based on its route type and cancellation type. The RouteType-CancellationType association could also express valid types of cancellations per route type, and this information could then be used as a validation rule that determines the permissible cancellation types per flight. Finally, the compromise example in Figure 9 shows an interesting construction. Flight is the subject of a stratified typification definition, and cancellation type is defined per route type and not flight type.

The ''Hybrid'' pattern (P4) is a variation of the mirror pattern where one of the operational-level object classes does not participate in a typification or grouping abstraction. A policy is defined as a hybrid association between a policy-level object class and an operational-level object class. This pattern most likely occurs when one of the operational-level object classes has a limited extension (i.e., a small number of instances), and policies are defined for actual instances. In Figure 9, the Fleet-ServiceOperator association defines a policy (validation rule) as part of a hybrid pattern; that is, the list of individual operators that are authorized to service a fleet. Instances of the Plane-ServiceOperator association are then validated against this policy. For example, a plane can only be serviced by a service operator who is authorized to serve the fleet of which the plane is a member, so the Fleet-ServiceOperator association determines the permissible content of the Plane-ServiceOperator association.

The distinctive characteristic of the ''Root'' pattern (P5) is that the same operational-level object class (root) participates in more than one typification/grouping association. In Figure 9, the root object class Plane participates in both a typification (Plane-PlaneType) and a grouping (Plane-Fleet) association. The Fleet-PlaneType association defines the policy that determines the composition of the fleet, and when a plane is assigned to a fleet, the policy will validate whether the plane's type has been approved. In general, the root pattern governs dependencies between different classifications (a-kind-of, a-member-of) of the same object class. The dependency for the example in Figure 9 is the following: an object may become a member of a collection only when the object's type is approved to be part of

[5] Our use of the term compromise here is related to the notion of implementation compromise described in Rockwell and McCarthy (1999).

that collection. Similar to the other patterns, the configuration of the ''Root'' pattern can be used to define different kinds of policies. For example, we could define the targeted average age per plane type per fleet using the same instantiation of the root pattern in Figure 9. This policy defines a target, and the mechanism used for its definition is an association class attribute.

We end this section with some additional comments on the enumerated patterns. First, as illustrated by the examples above, they define basic configurations that serve as starting points for the definition of different kinds of policies (knowledge-intensive descriptions, validation rules, and target descriptions), enabled by different data modeling mechanisms (attributes, associations, and association classes). Second, the patterns are heuristic in nature, and each of them is subject to some modeling consistency issues that must be evaluated (like all heuristics) on a case-by-case basis. An example of such discrepancy for the hybrid pattern is illustrated in Figure 10. The PlaneType-Pilot association defines a policy (validation rule) that determines which plane types a pilot is approved for. However, the association between Plane and Pilot—to be governed by the policy—is not explicitly modeled but derived instead from the Pilot-Flight and Flight-Plane associations. And third, the patterns presented in this paper are nonexhaustive. For example, none of the five patterns deal with policy definitions for heterogeneous groupings (Motschnig-Pitrik and Storey 1995).

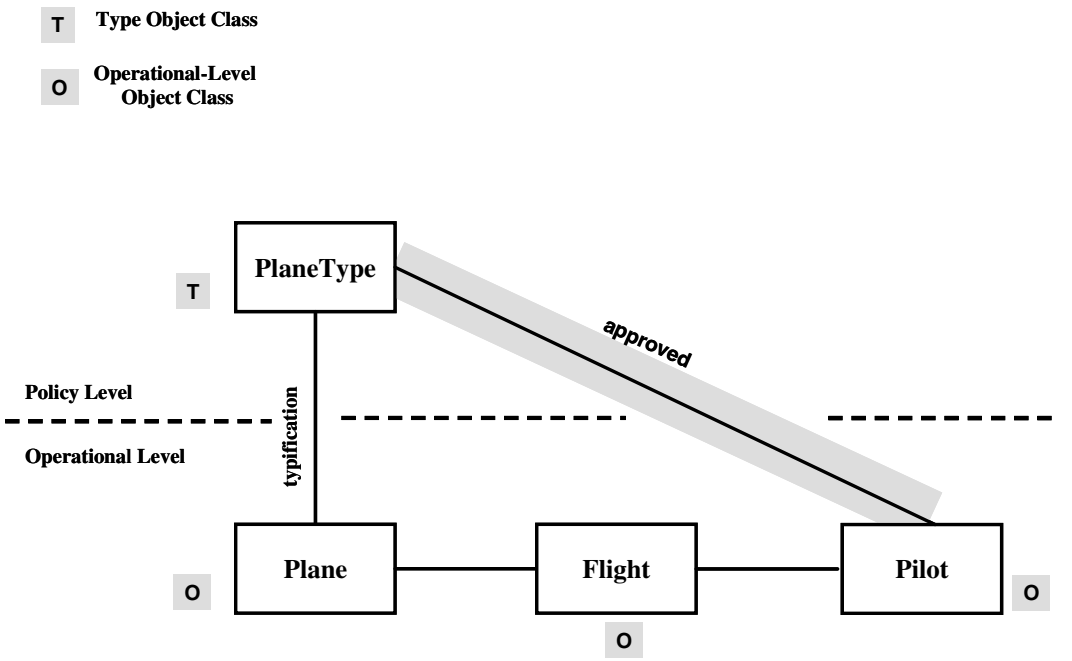
IV. POLICY APPLICATIONS FOR REA ENTERPRISE INFORMATION SYSTEMS

The accountability infrastructure of an REA enterprise system describes the actual economic activities that take place—What is or what has actually occurred. The policy infrastructure defines constraints and guidelines under which an enterprise operates—What could, should, or must be. Two key semantic abstractions of the policy infrastructure are type and grouping definitions, and their integrated use results in improved economy of presentation and informativeness. Policies defined for concepts and groupings need to be presented only once, and they are then shared by all their current and future instantiations and members. Further, once an object is defined as a realization of a concept or as a member of a grouping, additional information can be inferred. Type definitions are somewhat natural, but grouping definitions are largely idiosyncratic. The policy-level information needed to plan, control, and evaluate enterprise activities will depend on factors such as organizational context and management style. The following examples for resources, events, and agents should help readers understand the range and nature of type and grouping definitions in REA enterprise systems, as well as the kind of policies specified for them.

Type Definitions for Resources, Events, and Agents

Resources are often categorized based on technical specifications, such as octane rating for gas {regular, midgrade, premium}. Best practices (which cars need premium gas) or price policies (what unit price to charge) can then be established for the resource types. Events are often categorized based on their method of execution, such as the mode of sales {distributor sales, direct customer sales, internet sales}. Another example is the method of payment {cash, check, credit card} where a standard charge can be established for each of the different modes. Internal agents are often categorized based on skills or roles (salesperson, buyer, truck-driver, etc.), and authorization and compensation policies can then be established by skill. External agents are often categorized based on qualifications (e.g., creditworthiness of customers and quality rankings for vendors).

FIGURE 10
Heuristic Nature of Patterns



Grouping Definitions for Resources, Events, and Agents

Enterprises often manage resource collections such as a fleet of airplanes (airline), a fleet of cars (rental company), a chain of stores, or a portfolio of stocks. Examples of policy definitions for resource collections include: ways to compose the resource collection (the types of cars that should be part of the rental fleet) and the minimum and/or maximum number of members in the resource collection (the minimum number of stocks that can be in a portfolio). A somewhat different example of a resource collection often used in manufacturing applications is ''lot'' defined by Silverston (2001, 85) as: ''a grouping of items of the same type generally used to track inventory items back to their source; it is often the result of a production run.'' An example of a characteristic for lots is expiration-date which can be used in a policy such as the following: do not sell items that have expired. Event collections are often managed as a unit, and examples include the following: a package of ten car washes, a season pass to a theme park, and a season ticket package for a certain sports team. Examples of policy definitions for event collections include: events must occur during a specific time period (valid for admission during summer) and restrictions on the number of members (ten car washes). Agent groupings are often used to represent groups of people that share specific tasks, goals, commitments, etc., and examples of such groupings are committees, departments, and teams. Examples of policy definitions for agent groupings include: the way they should be composed (best practices regarding the composition of a team in terms of skills) and the minimum and/or maximum number of members (the minimum number of employees on a team).

The policy infrastructure of an enterprise system defines constraints and guidelines under which an enterprise operates. In section two of this paper, we distinguished among three different kinds of policy definitions: knowledge-intensive descriptions, validation rules, and target descriptions. Next, we discuss enterprise applications for each of these policy definitions in more detail.

Knowledge-Intensive Descriptions

Knowledge-intensive descriptions can take different forms, and a first example is illustrated in Figure 11. The model is an instantiation of the basic pattern (typification) applied to an economic resource (product). The price attribute of ProductType represents the policy which is the amount the customer should pay for the product. The price is inferred for individual products based on their participation in the typification association. We, therefore, show price as a derived attribute of Product. From a business perspective, it is assumed that the price is standardized (i.e., no discounts or other discrepancies are allowed for this particular company). The example in Figure 11 is for a resource, but it can be applied to other REA primitives as well. For example, an hourly rate is determined per employee type (agent) or a tax rate is determined per sales type (event).

A second example of a knowledge-intensive description is illustrated in Figure 12 where price is defined as an attribute of the association class PricePolicy. This is an instantiation

FIGURE 11
Knowledge-Intensive Descriptions, Validation Rules, and Target Descriptions

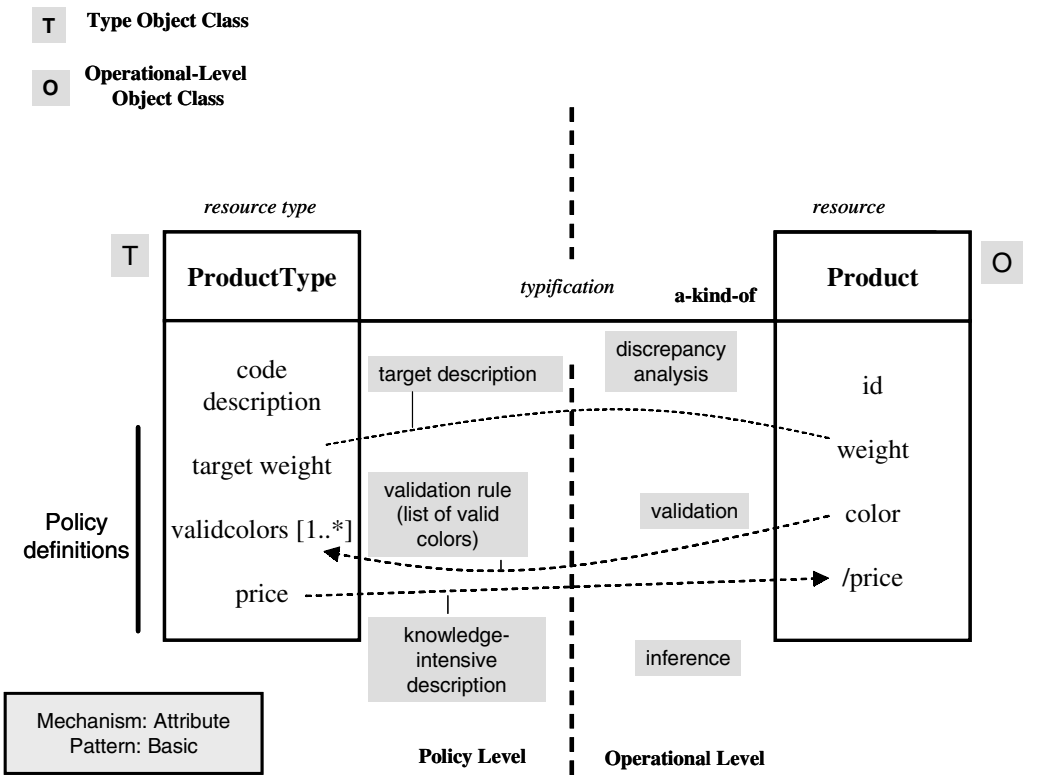
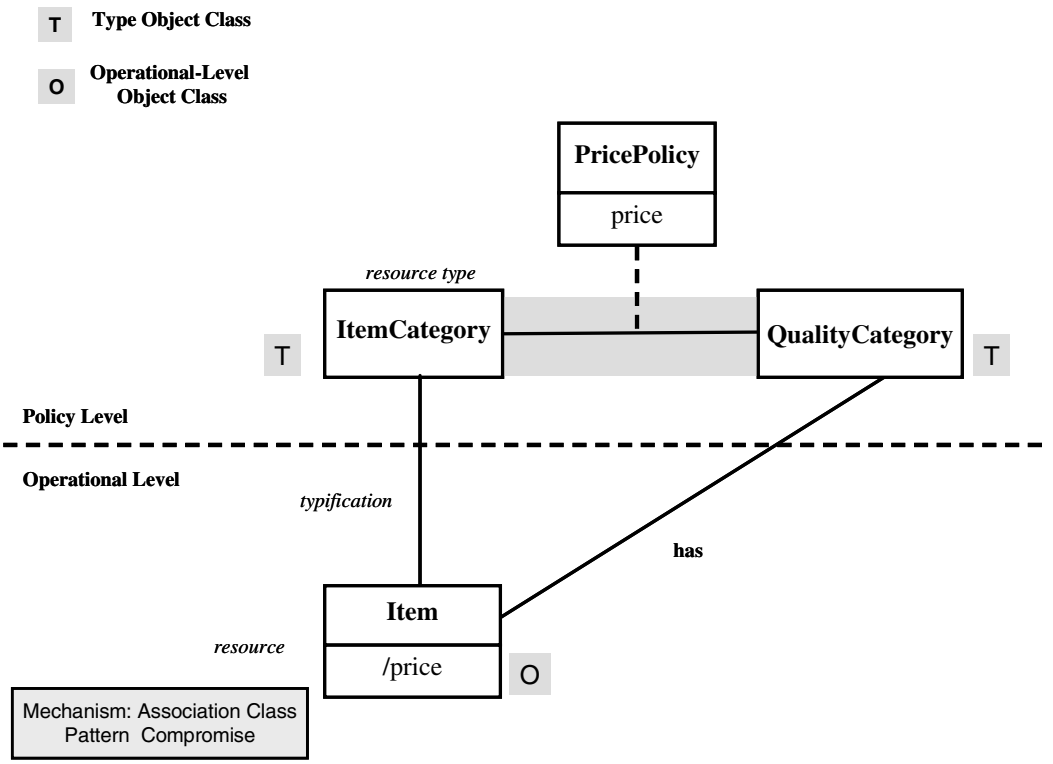


FIGURE 12
Knowledge-Intensive Descriptions



of the compromise pattern. Instances of QualityCategory denote generic categories often represented by labels such as excellent, average, and poor. The compromise pattern indicates that it is not economical to describe the individual quality conditions for each of the item instances. A price policy is defined for all possible combinations of ItemCategory and QualityCategory, and the actual price of an individual item is then determined by the intersection of its item category and its quality category.

Validation Rules

Validation rules define constraints that must be true, and they differ from knowledge-intensive descriptions in that they do not define derivable information but rules to which actual phenomena (operational level) must adhere. Figure 11 illustrates the use of attributes for the specification of a validation rule: the color of a product must be one of the valid colors defined for its product type. The dotted line with its arrowhead pointed to ProductType indicates that the actual color is validated against the list of valid colors. Common applications of this pattern in enterprise systems are limit checks and range checks. Again, such validation rules can be specified for REA primitives other than resource. The following is an example of a validation rule for an internal agent (employee): the maximum salary for a specific job type (for example: the highest possible salary for an employee with job type engineer is $190,000). Validation rules can also be defined by

TABLE 2
Shipment Policy Decision Table

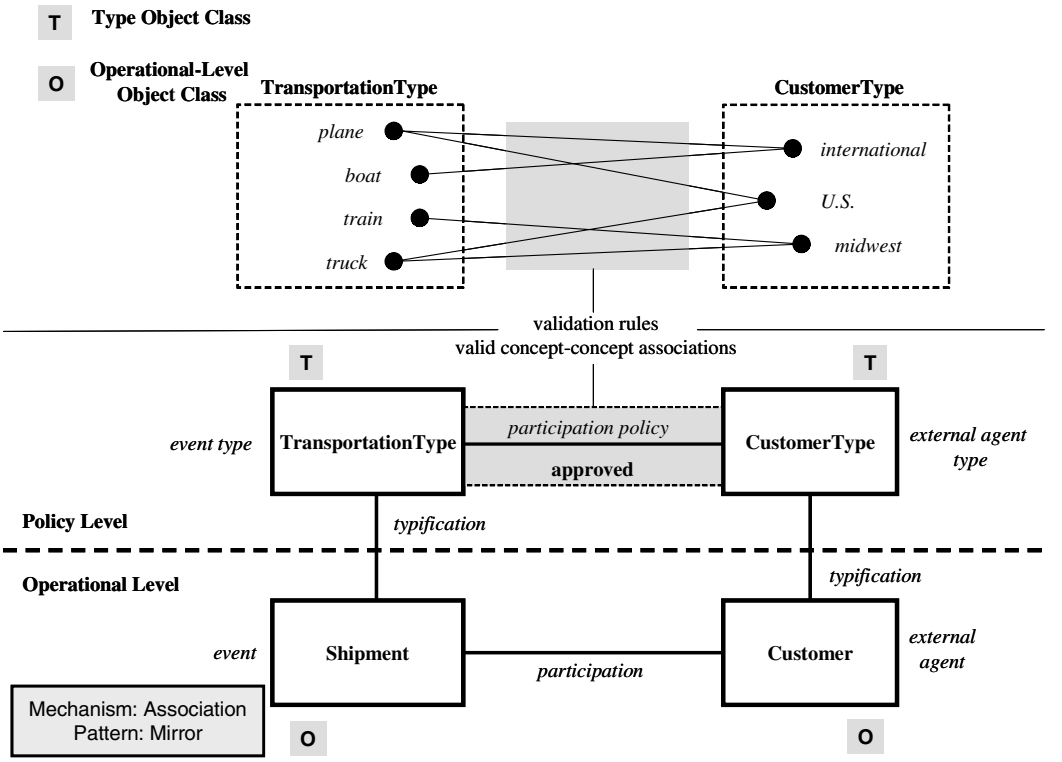
Transportation Type \ Customer Type	Midwest	U.S.	International
Plane		X	X
Boat			X
Train	X		
Truck	X	X	

means of association class attributes. For example, the price policy in Figure 12 could define a minimum price.

Associations are another mechanism for the definition of validation rules as we will illustrate with three examples next, each using a different pattern: mirror, hybrid, and root.

Table 2 shows a decision table that represents a number of policies regarding shipping, such as ''international customers can be serviced only by plane or boat.'' Figure 13 illustrates how the policies in Table 2 can be integrated into an REA enterprise model

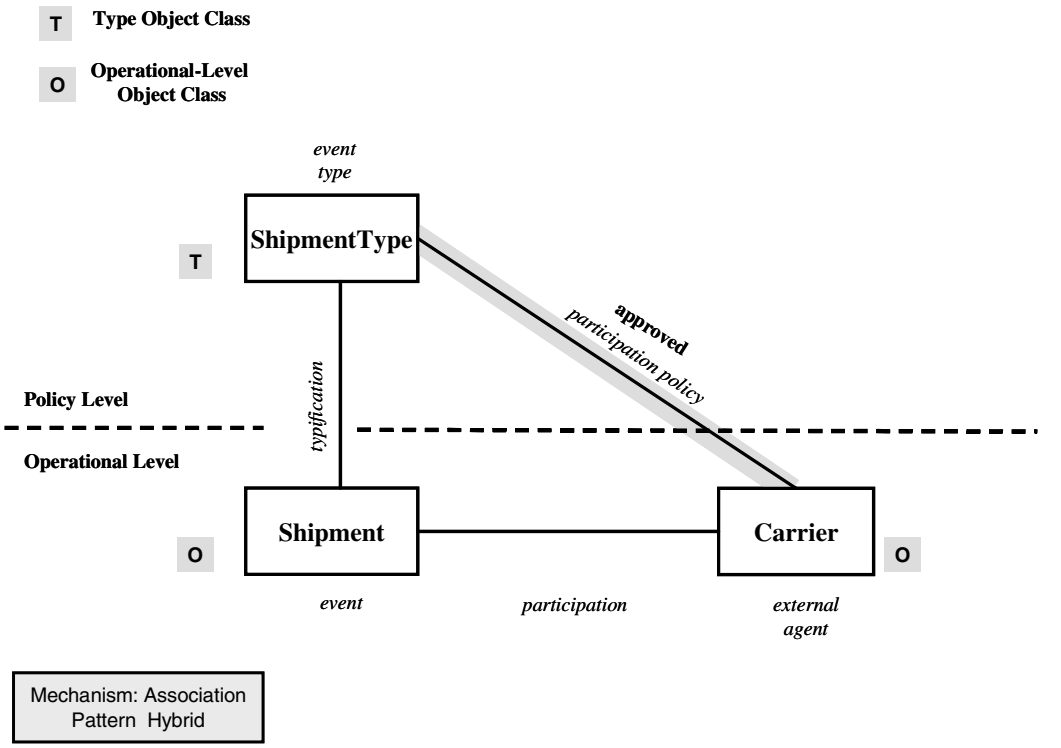
FIGURE 13
Validation Rule Definitions



as validation rules. The upper part of Figure 13 shows how instances of the TransportationType-CustomerType association define ''what valid associations do exist between transportation types and customer types.'' The REA enterprise model in the lower part of Figure 13 is an instantiation of the mirror pattern, with the TransportationType-CustomerType association being the policy definition. Typification associations link the descriptions of the actual shipments and their customers with the policies, and they enable validation judgments such as ''a shipment to a customer is valid when a valid association exists between the shipment's transportation type and the customer's type.''

Figure 14 shows an REA enterprise model where validation rules are defined with the same mechanism (association), but a different pattern (hybrid). Shipment is an economic event, Carrier is an external agent, and Shipment-Carrier is a participation association. The Shipment-ShipmentType association categorizes shipments as being of type next day, second day, international, etc. It is likely that a company works (has agreements) with a limited number of carriers, and the policy definition expresses what types of shipments are approved for each individual carrier. The policy is defined per individual carrier instead of per carrier type. The REA enterprise model in Figure 14 enables the following validation: a carrier is valid for a shipment when the combination of the shipment's carrier and the shipment's type occur as a valid instance of the policy association (ShipmentType-Carrier).

FIGURE 14
Validation Rule Definitions



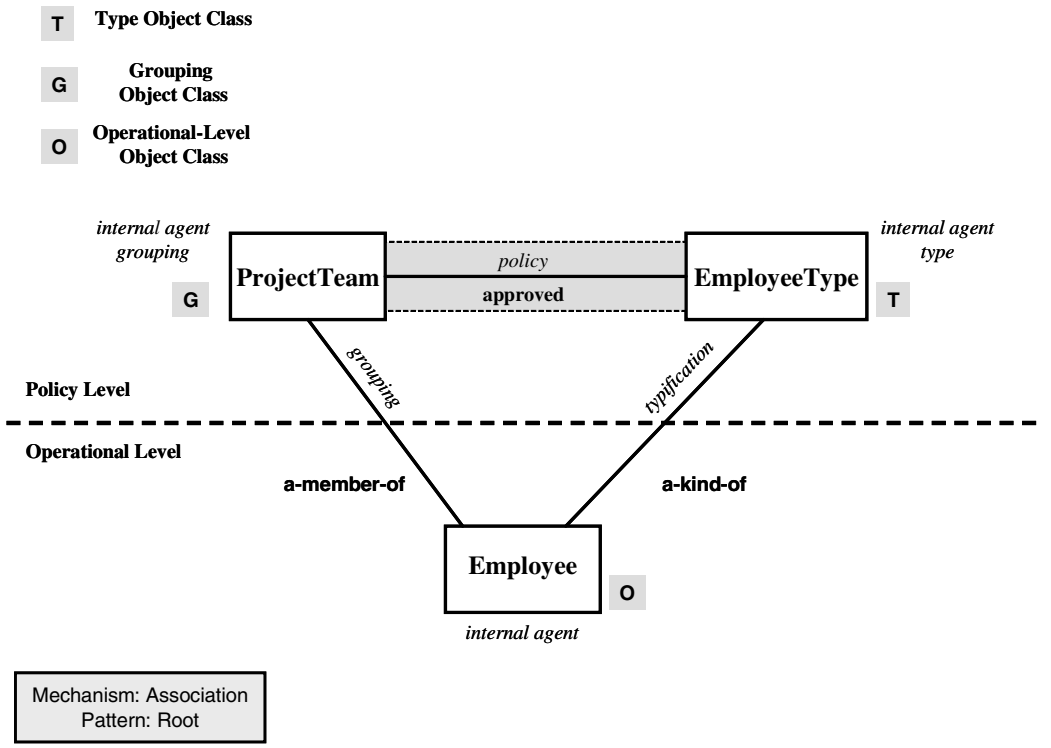
In a root pattern, the same object class participates in two different typification/grouping abstractions. For the example in Figure 15, the root class employee, an internal agent, participates in both a grouping (project team) and a typification (employee type) association. The ProjectTeam-EmployeeType policy association defines how an employee's membership in a project team depends on the employee's type. The REA enterprise model in Figure 15 enables the following validation: when an employee is assigned to a project team (Employee-ProjectTeam association), clearance must be obtained that the employee's type is valid (approved) for that project team.

Target Descriptions

Target descriptions define ''what should be,'' and two common applications in enterprise systems are standards and budget specifications. Figure 11 illustrates the use of attributes for the definition of a standard specification. The target-weight attribute of the ProductType object class defines how much we expect instances of that product type to weigh (i.e., its standard weight). The weight attribute of the Product object class defines how much the item actually weighs. The typification association links actual products with their product types and thus with their target descriptions. Discrepancies (variances) between the actual weight and the standard weight can then be analyzed.

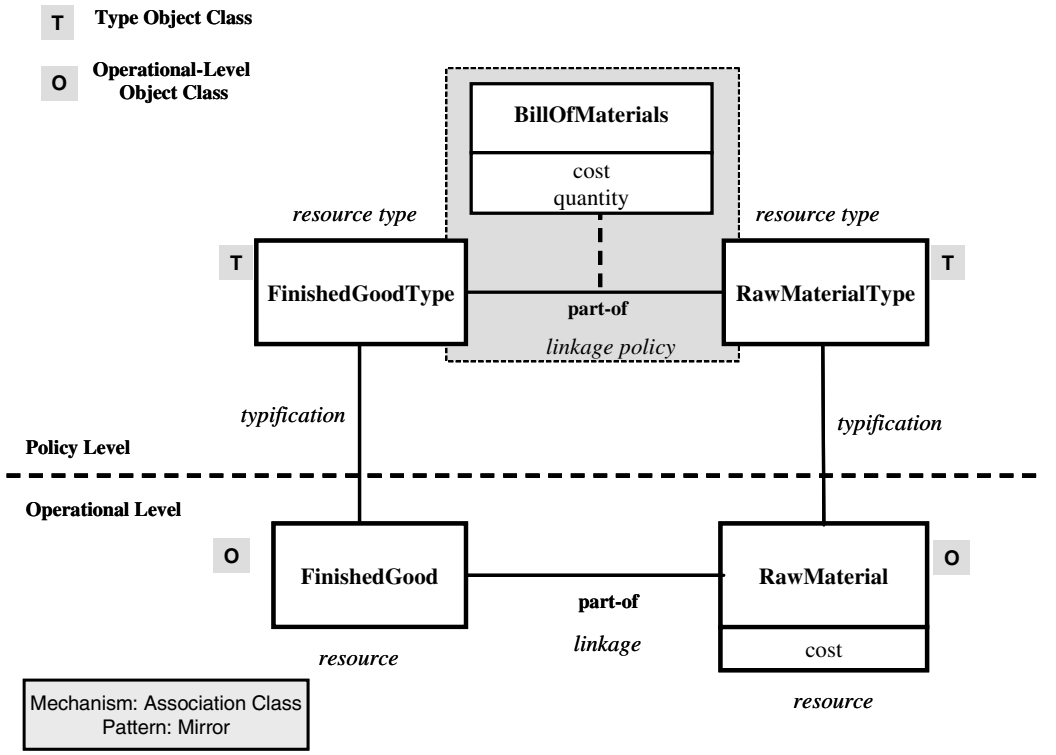
Standards often capture engineering information like ''the proper way to do something.'' An example of a standard specification that captures engineering information is a Bill of

FIGURE 15
Validation Rule Definitions



Materials (BoM): ''a listing of all the assemblies, subassemblies, parts, and raw materials that are needed to produce one unit of a finished product'' (Siegel and Shim 1995, 44). A BoM defines the way to manufacture a finished good, and Figure 16 illustrates the integration of BoM specifications into an REA enterprise model as an instantiation of the mirror pattern. The part-of association at the operational level represents a whole-part (aggregation) association between a finished good and its components. It describes the actual resources consumed by an identifiable instance of finished good (e.g., the components [resources] needed to manufacture the Boeing 747 with ID ''87B5''). Geerts and McCarthy (2000) use ''linkage'' to name the association between two economic resources. The part-of association is mirrored at the policy level defining the raw material types needed to manufacture an instance of a finished good type (e.g., the raw materials types needed for the manufacturing of a Boeing 747). BoM specifications typically include targets such as the expected quantity (standard quantity) and the expected cost (standard cost) of the raw material types needed. For the example in Figure 16, this information is recorded by the cost and quantity attributes of the BillOfMaterials association class. This information can then be used for variance analysis. Often, BoM specifications are more difficult than the one illustrated in Figure 16, and more complex variations of BoM enterprise models are discussed at length in Hay (1996) and Scheer (1998). All these variations make extensive use of policy specifications.

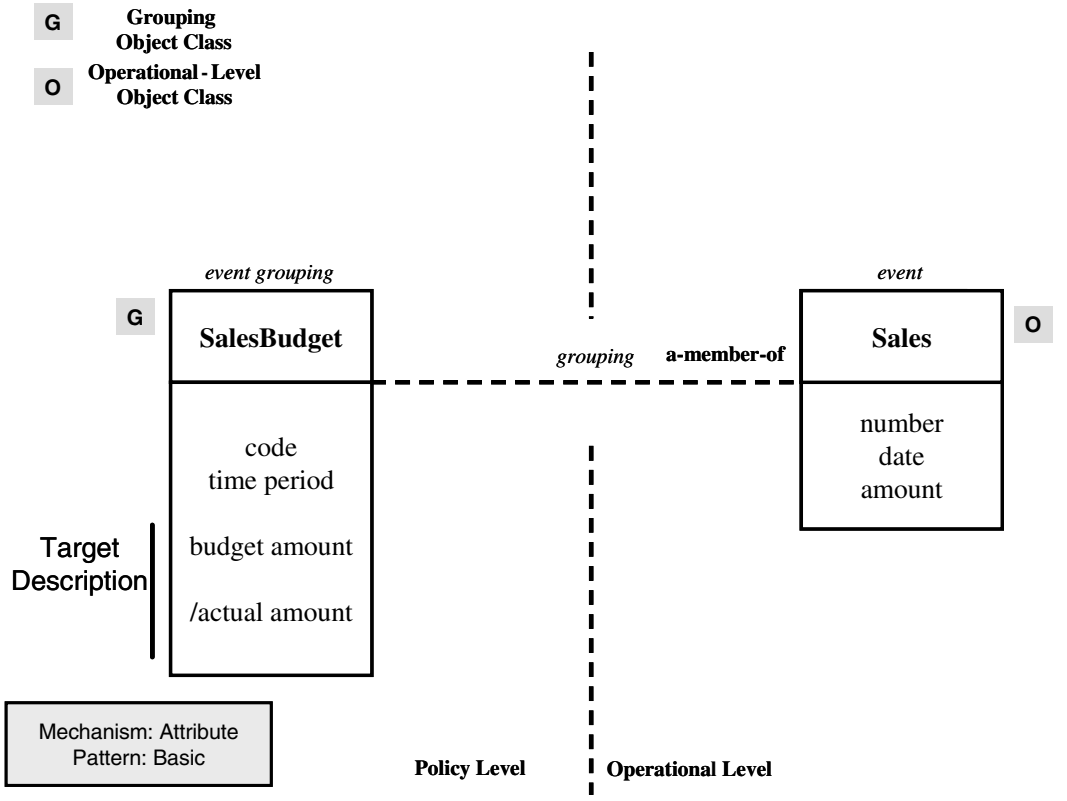
FIGURE 16
Target Descriptions: Standard Definition



Standards can also be defined for other REA primitives. For example, the policy definition in Figure 15 can also be used as a standard definition instead of as a validation rule. The ProjectTeam-EmployeeType association would then describe the expected composition of projects teams in terms of employee types (skills), and discrepancy analysis would compare actual with expected compositions.

Another common enterprise application of target descriptions is budget specifications. Horngren and Foster (1991, 172) define a budget as ''a quantitative expression of a plan of action and an aid to coordination and implementation.'' Examples of budgets include the following: determining the estimated cost for the manufacturing of a custom-built home, determining how many units should be sold in the next quarter, and determining the amount of advertising allowed in a quarter. Figure 17 defines a sales budget and shows that budget information primarily is of two types: (1) static estimates done in advance (the budget amount), and (2) derived attributes representing arithmetic roll-ups (the actual amount). The budget-amount defines the estimated total sales amount for the specified time period; thus, it applies to all sales transactions that will occur during that period. Actual-amount is a derived attribute calculated from the Sales amount attribute, and it is used for discrepancy analysis. The actual amount represents a running total that will increase over time. The dotted association line in Figure 17 indicates that membership in time periods can be

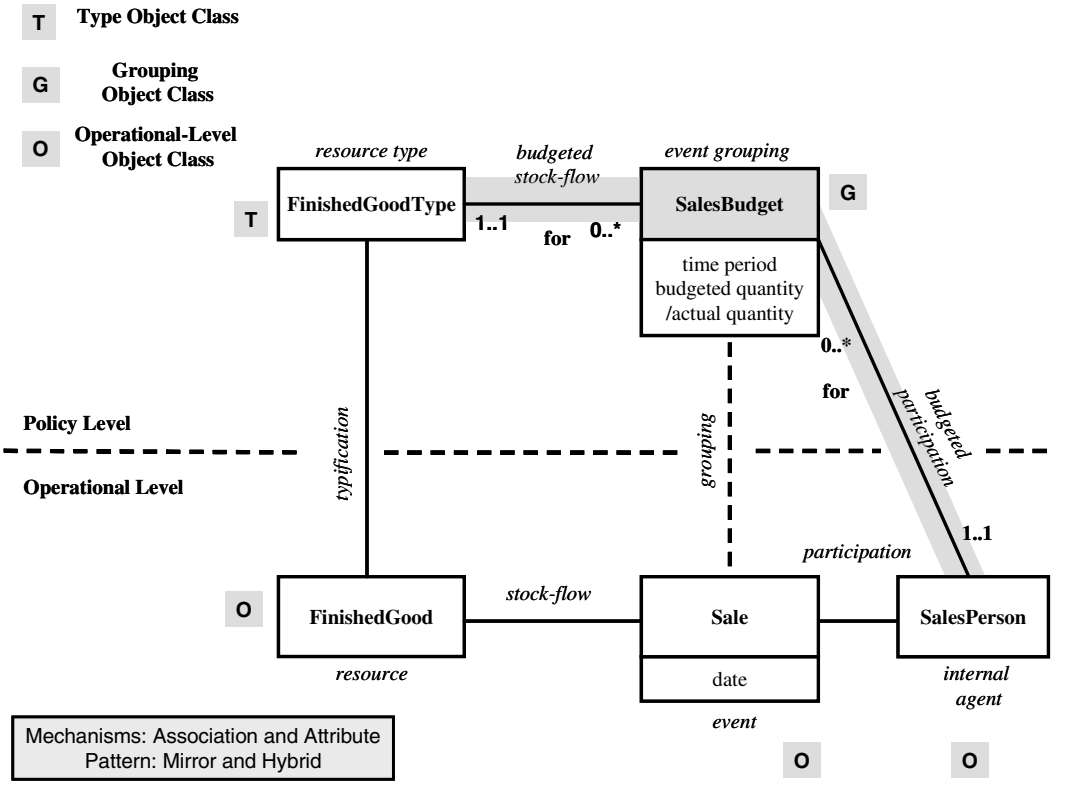
FIGURE 17
Target Descriptions: Budget Definition



determined procedurally by applying time functions to the date attribute; i.e., the grouping association does not have to be recorded explicitly.[6]

Budgets are often broken down into components that define who and what. For REA models, the who and what can be expressed in terms of agents and resources, such as the following: the number of items per finished good type (resource) each salesperson (agent) should sell (event) in Spring 2006. The REA enterprise model for this budget specification is illustrated in Figure 18. Instances of the SalesBudget object class define the budgeted quantity per FinishedGoodType per SalesPerson per time period. Stated differently, both ''for'' associations in Figure 18 are part of the budget definition. The budgeted stock-flow association is part of a mirror pattern instantiation. The budgeted participation association is part of a hybrid pattern instantiation, since budgets are specified for individual salespeople. The model in Figure 18 does not explicitly define the Sale-SalesBudget grouping association. A sale becomes a member of a sales budget (grouping) when (1) its date occurs in the sales budget time period, (2) its finished good is of the type specified for the sales budget (the mirror pattern), and (3) its salesperson is the salesperson for whom the budget

FIGURE 18
Target Descriptions: Broken-Down Budget Definition



[6] An alternative presentation for time period often used in practice is the definition of a start date and an end date.

is defined (the hybrid pattern). Stated differently, the policy associations are used to derive budget membership procedurally. However, for the case where the Sale-SalesBudget grouping association is explicitly modeled (i.e., a sale is explicitly assigned to a sales budget), the policy associations define constraints. For example, a sale cannot be a member of a sales budget when the sale's finished good is not of the type defined for the sales budget (the mirror pattern).

V. CONCLUSION

In this paper, we have extended the REA accountability infrastructure with policy-level definitions that specify ''what should, could, or must be.'' In the initial part of the paper, we laid the groundwork by studying the two key semantic abstractions that enable the definition of policies: typification and grouping. Following this exposition, we presented a number of patterns for the semantic modeling of policies. In the second half of the paper, we discussed policy applications for REA enterprise systems. We first presented typification and grouping extensions for REA primitives. Next, we differentiated between three different kinds of policy definitions in enterprise systems—knowledge-intensive descriptions, validation rules, and target descriptions—and we discussed applications for each of them. We noted that (1) knowledge-intensive descriptions enable the definition of policies that can be inferred (such as ''what the price must be''), (2) that validation rules present constraints that must be met (often in the form of a set of permissible values), and (3) that target descriptions represent guidelines as to ''what should be.''

REFERENCES

Batini, C., S. Ceri, and S. B. Navathe. 1992. *Conceptual Database Design. An Entity-Relationship Approach*. Redwood City, CA: Benjamin/Cummings.

Booch, G., J. Rumbaugh, and I. Jacobson. 1999. *The Unified Modeling Language Reference Manual*. Reading, MA: Addison-Wesley.

Brodie, M. L. 1981. Association: A database abstraction for semantic modeling. In *Entity-Relationship Approach to Information Modeling and Analysis*, edited by P. P. Chen, 583–608. Amsterdam, the Netherlands: North Holland.

Business Rules Group. 2000. Defining business rules. What are they really? Available at: http://www.businessrulesgroup.org.

Clancey, W. J. 1985. Heuristic classification. *Artificial Intelligence* 27: 289–350.

Dunn, C. L., J. O. Cherrington, and A. S. Hollander. 2005. *Enterprise Information Systems: A Pattern-Based Approach*. Boston, MA: McGraw-Hill.

Eriksson, H-E., and M. Penker. 2000. *Business Modeling with UML.* New York, NY: John Wiley & Sons.

Fikes, R., and T. Kehler. 1985. The role of frame-based representation in reasoning. *Communications of the ACM* (September): 904–920.

Fowler, M. 1997. *Analysis Patterns: Reusable Object Models*. Reading, MA: Addison-Wesley.

Geerts, G. L., and W. E. McCarthy. 2000. The ontological foundation of REA enterprise information systems. Working paper. Available at: http://www.msu.edu/user/mccarth4/.

Goldstein, R. C., and V. C. Storey. 1994. Materialization. *IEEE Transactions on Knowledge and Data* 6 (5): 835–842.

Hammer, M., and D. McLeod. 1981. Database description with SDM: A semantic database model. *ACM Transactions on Database Systems* (September): 351–386.

Hay, D. C. 1996. *Data Model Patterns*. New York, NY: Dorset House Publishing.

Horngren, C. T., and G. Foster. 1991. *Cost Accounting.* Englewood Cliffs, NJ: Prentice Hall.

Larman, C. 2002. *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design and the Unified Process*. Upper Saddle River, NJ: Prentice Hall.

McCarthy, W. E. 1982. The REA accounting model: A generalized framework for accounting systems in a shared data environment. *The Accounting Review* (July): 554–578.

———. 1987. On the future of knowledge-based accounting systems. In *The D.R. Scott Memorial Lecture Series*, edited by K. Harmon, T. Howard, and J. Parker, 19–42. Columbia, MO: The University of Missouri.

———, and E. Outslay. 1989. An analysis of the applicability of artificial intelligence techniques to problem-solving in taxation domains. *Accounting Horizons* (June): 14–27.

———. 2002. Book Review of *Knowledge Representation: Logical, Philosophical, and Computational Foundations* by John F. Sowa. *The Accounting Review* (July): 695–697.

Motschnig-Pitrik, R., and V. C. Storey 1995. Modeling of set membership: The notion and issues. *Data & Knowledge Engineering* 16: 147–185.

Odell, J. J. 1998. *Advanced Object-Oriented Analysis & Design Using UML.* New York, NY: Cambridge University Press.

Rockwell, S. R., and W. E. McCarthy. 1999. REACH: Automated database design integrating first-order theories, reconstructive expertise, and implementation heuristics for accounting information systems. *International Journal of Intelligent Systems in Accounting, Management, and Finance* (September): 181–197.

Sakai, H. 1981. A method for defining information structures and transactions in conceptual schema design. *Proceedings of the Seventh International Conference on Very Large Databases.* Cannes, France: 225–234.

Scheer, A-W. 1998. *Business Process Engineering: Reference Models for Industrial Enterprises*. Heidelberg, Germany: Springer-Verlag.

Siegel, J. G., and J. K. Shim. 1995. *Dictionary of Accounting Terms.* 2nd edition. New York, NY: Barron's.

Silverston, L. 2001. *The Data Model Resource Book. Volume 1. A Library of Universal Data Models for All Enterprises.* New York, NY: John Wiley & Sons.

Smith, J. M., and D. C. P. Smith. 1977a. Database abstractions: Aggregation. *Communications of the ACM* (June): 405–413.

———, and ———. 1977b. Database abstractions: Aggregation and generalization. *ACM Transactions on Database Systems* (June): 105–133.

Sowa, J. 1999. *Knowledge Representation: Logical, Philosophical, and Computational Foundations.* Pacific Grove, CA: Brooks/Cole Publishing.

Taivalsaari, A. 1996. On the notion of inheritance. *ACM Computing Surveys* 28 (3): 438–479.

Tarnas, R. 1991. *The Passion of the Western Mind*. New York, NY: Ballantine Books.