The Model Role Level – A Vision

Rick Salay[1] and John Mylopoulos[1],

[1] Department of Computer Science, University of Toronto
Toronto, ON M5S 3G4, Canada.
{rsalay, jm}@cs.toronto.edu

Abstract. Models are used widely within software engineering and have been studied from many perspectives. A perspective that has received little attention is the characterization of the role each model plays within a modeling project. We refer to this as *model intent*, and the collection of roles for all models as the *role level* within a project. Knowing the intent of a model supports model comprehension by providing the correct context for interpretation and enhances model quality by clearly defining what information it must contain. Furthermore, formal expression of this intent enables automated support for model analysis. Despite the value that knowledge of model intent can provide, there are no adequate means in the current state of modeling practice for expressing this information, apart from informal documentation. The focus of this paper is to provide a framework for understanding model intent, distinguish it from related modeling concepts and discuss its uses.

Keywords: Model intent, Model quality, Modeling, Roles.

1 Introduction

Modeling is (at last!) a core activity in software engineering, largely due to a series of standards established by the Object Management Group (OMG). Not surprisingly, the topic is being studied from many perspectives, including metamodeling [e.g., 13], formal model semantics [e.g., 4], horizontal and vertical consistency (refinement) [e.g., 8], model transformations [e.g., 3] and model management [1]. However, a perspective that has received little attention is the role that model *intent* plays in modeling and this is the focus of this paper.

When modelers create a collection of models for a particular software engineering project, they do so with an idea of what role each model plays in the collection – i.e., what information it should contain and how this information is related to the information in other models. The specification of these roles is what we refer to as the expression of model intent. Elsewhere we have described how to express model intent formally [16, 17] and how this can support model comprehension, quality, automation and evolution. In this paper, we explore the foundations behind model intent and offer a vision of how this new kind of information can be used in modeling practice. A key contribution of this paper is the identification of the "role level" as being an important, but typically unarticulated, aspect of modeling activity alongside the "model level" at which the content of models is created.

We begin in Section 2 by giving an illustration of what we mean by model intent. Then in Section 3 we analyze the concept of model intent to distinguish it from other related concepts and to reveal its unique characteristics. In Section 4, we discuss the implications of treating the role level as a first class notion within modeling practice. Finally, we discuss related work and draw some conclusions.

2 An Illustration

Throughout this paper we will draw on a hypothetical transportation system development project as source of illustrative examples. The transportation system concerns itself with different modes of transport (e.g., trains, cars, etc.) and has subsystems and software dealing with different aspects of their management (e.g., traffic control, tolls, etc.). Fig. 1 shows a class diagram *DTollPrice* for that project. This diagram is syntactically well-formed and according to the usual semantics of class diagrams we can interpret it as being equivalent to the following set of statements:

- Class *Vehicle* has subclasses *Car*, *SUV* and *Truck.*
- Class *TollTicket* has subclasses *SingleTripTicket* and *MonthlyTicket.*
- Class *Vehicle* has attributes *weight* and *numPassengers* of type *int.*
- Class *Truck* has an attribute *cargo* of type *Ctype.*
- Class *TollTicket* has an *authorizes* association to class *Vehicle* and an attribute *purchasePrice* of type *real*.
- Class *MonthlyTicket* has an attribute *discount* of type *real.*

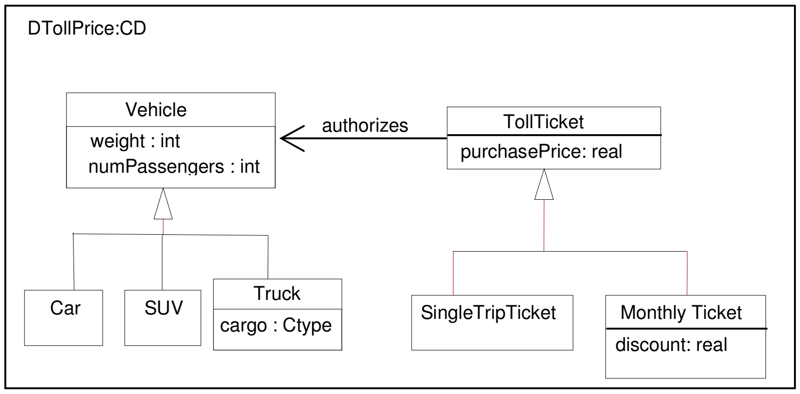

Fig.1.Transportation system diagram dealing with toll price.

Now consider how this interpretation is affected as we disclose different aspects of the intent regarding this model with the following series of statements.

- (I1) *DTollPrice* is a (proper) submodel of the design model *TransSysDesign* -- We now know that this is not a complete model by itself but is part of a larger one and so we expect there to be other submodels showing other parts of the design. Furthermore, we know that there may be other classes not shown here that may be related to these classes. We also know that this model is described at the "design level" of detail (vs. implementation level, for example).
- (I1.1) The classes in *DTollPrice* are aggregated within the class *TransportationSystem* not shown in this diagram -- This elaborates (I1) and we can now infer the statement "classes *Vehicle* and *TollTicket* have aggregation relationships to class *TransportationSystem*"
- (I2) Diagram *DTollPrice* shows the parts of a transportation system dealing with toll payment -- This gives us some sense of the purpose of the model and basis for assessing the relevance of the current content to this purpose.
- (I2.1) All and only the attributes of vehicles that affect toll price are shown -- This elaborates (I2) and we can now infer the statement that "*weight*, *numPassengers* and *cargo* are the only vehicle attributes that affect toll price"
- (I2.2) All types of toll tickets are shown -- This elaborates (I2) and we can now infer the statement that "*SingleTripTicket* and *MonthlyTicket* are the only types of *TollTicket*"

From a model user perspective, I1 helps us to understand the broader context of the model and its relationship to other model artifacts, I2 helps us to understand the purpose of the model and hence the rationale for its content, and I1.1, I2.1 and I2.2 allow us to infer additional statements that augment the standard semantics of the model. From a modeler perspective, asserting these statements helps to clarify what information to include or exclude from *DTollPrice*. For example, I2.1 forces the modeler to think about whether there are any other attributes of vehicles that may affect toll price. As the transportation system design evolves and other modelers modify *DTollPrice*, these statements provide guidance to help them remain conformant to the original intentions – or perhaps, to challenge and change the intention if that is appropriate. Finally, if some of these statements can be formalized, then conformance checks and repairs can be performed in an automated way by modeling tools. We now turn to an in-depth analysis of the concept of model intent.

3 Analyzing Model Intent

Model intent is a kind of information about models. As such it exists in the *development world* [13] that consists of artifacts such as models, activities such as modeling and actors such as modelers. We assume that a modeler works in the context of a particular *modeling project* that consists of an evolving set of interrelated models. Note that the modeling project can be part of a larger effort such as a software development project, documentation project, etc. Furthermore, if a development methodology is being followed, we think of the modeling project as being part of an instance of this methodology. We begin by identifying some key stakeholder roles relative to a model in the context of a modeling project:

- *Modeler*
    - *Definer*: The model definer is a modeler who decides that the model must exist and defines what information it should contain.
    - *Producer*: The producer is a modeler who creates the content of the model in accordance with the requirements of the definer.
- *Consumer*: The consumer uses the model to satisfy their goals.

The modeler role is subdivided into the definer and producer roles to reflect the fact that a modeling project may involve many modelers and that the modeler who decides that a given model must exist may not be the same one that creates the model. For example, a senior designer on a software project may play the definer role for a certain model that a junior designer must create as producer. The junior designer may then play the definer role with respect to how they wish to subdivide the model into various submodels and then play the producer role in creating them. We assume that the model intent emerges from the activities of the definer role and that any expression of model intent is created by a definer. Both the producer and consumer may use these expressions of model intent to support their activities.

Now consider the framework in Fig. 2 showing the different kinds of model intent that can arise within a modeling context. The entry point into the framework is through the need arising for a model due to the information requirements of some stakeholders such as software developers, testers, users, business decision makers, etc. For example, assume that the modeler is responding to the need of a group of software developers for the UML model *TransSysDesign* representing the detailed design of a transportation control system that the developers must build. This generates an initial *existential intent* on the part of the modeler that identifies that the model *TransSysDesign* must be created within the project. At this point, as the modeler considers the purpose of the model, they recognize that *TransSysDesign* is related to other models in well defined ways (arrow *R*). For example, *TransSysDesign* must *satisfy* the requirements model *TransSysReq*, it must *refine* the architecture model *TransSysArch*, etc. All of these are intended relationships that the model must conform to.

Before the modeler actually creates *TransSysDesign*, they must decide what information should be in it based on what information they think would satisfy the needs of the developers (arrow *C*). This gives rise to an intention about what the content of *TransSysDesign* should be and we refer to a characterization of this as *content criteria*. The modeler may then recognize that this information should not be created as a single monolithic model but must instead be decomposed into multiple related "partial" models (arrow *D*). Doing this involves both the identification of new models (arrow *I*) and the expression of the decompositional structure of the set of partial models (arrow *E*) that we call *decomposition criteria.*

We now examine each of these kinds of model intent more closely.

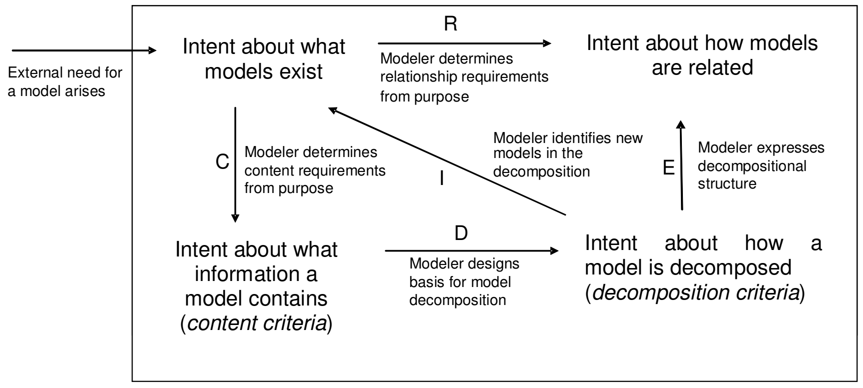
External need for a model arises
Intent about what models exist
R
Modeler determines relationship requirements from purpose
Intent about how models are related
C
Modeler determines content requirements from purpose
I
Modeler identifies new models in the decomposition
E
Modeler expresses decompositional structure
Intent about what information a model contains (*content criteria*)
D
Modeler designs basis for model decomposition
Intent about how a model is decomposed (*decomposition criteria*)

Fig. 2. A framework for model intent.

3.1 Expressing Existential Intent – Model Roles

An important distinction that is not often made clear in research on modeling and is central to the concerns of this paper is that of model roles vs. the models that play those roles. We define a *model role* within a modeling project as a reification of the need for a model that satisfies a particular purpose while a model is an artifact that can play (or realize[1]) one of these roles. Thus, we assume that every model must have a purpose. Furthermore, a model role is not merely an attribute of a model – it has its own existence independently of the models that play it. The acknowledged existence of a model role within a project represents an *existential intent* on the part of the modeler (as definer): an expectation that a model playing this role should exist within the project.

In practice, when actors in a modeling project talk about models, they are usually talking about model roles rather than the actual artifacts that play those roles. For example, consider the following typical sentences:

1. "Where is the error handling model?"
2. "Who will create the structural model of the traffic light controller?"
3. "Here is the latest version of the toll ticket purchase flowchart."

In these sentences, the phrases "error handling model", "structural model of the traffic light controller" and "the toll ticket purchase flowchart" all refer to model roles rather than models themselves. Only in sentence (3) is an actual model referred to as the referent of the word "Here". The lifetime of a model role is longer than that of the models playing the role and can exist even if no model plays the role. Furthermore, different models can play a given role at different times although only one can play

[1] We will use the terms *play* and *realize* interchangeably. Thus a model is a realization of a role.

the role at a time. For example, in sentence (2) it is clear that no structural model exists yet so the model role precedes the existence of a model playing the role. In sentence (3) the referent of “Here” is the model that is playing the role at the time that the sentence was uttered.

The Model Level vs. the Role Level. In the current state of practice, most explicit modeling activity takes place at the *model level* rather than at the *role level*. That is, modelers spend most of their time expressing the content of particular models rather than their intentions about the models. The role level is a level of abstraction on the model level since model roles identify and say something about models without giving their content.

Abstraction is a powerful mechanism for managing complexity by supporting top-down understanding but in order for a level of abstraction to be useful it must provide some form of summarization of the details that are omitted [12]. We will assume that a model summary is any property that abstracts from the content of the model. Thus, the model intent at the role level is one such summary and we argue that this summary in particular is a key one for supporting a stakeholder’s comprehension of a collection of interrelated models.

To see this, consider the following different possible summaries of diagram *DTollPrice* in Fig. 1:

1. *DTollPrice* contains seven classes
2. *DTollPrice* contains some details for classes *Vehicle* and *TollTicket*
3. *DTollPrice* contains all information related to toll ticket pricing
4. *DTollPrice* contains classes *Vehicle* and its three subclasses and class *TollTicket* and its two subclasses

These are all valid summaries of *DTollPrice* and give different amounts of information about it. However, of these, (3) is distinguished because it defines the role intended to be played by the model and expresses the model intent. In general, the choice of summary used should correspond to the task for which the abstraction will be used. For example, if the goal is to efficiently decide which model has more than nine elements, then a summary like (1) is most appropriate. What we suggest is that the summary that is most appropriate for supporting the task of *model comprehension* is the one that is drawn from the purpose of the model because this explains why it contains the information it contains. Explicitly modeling the role level as part of normal modeling activity benefits all stakeholders by providing this useful level of abstraction on the collection of models in a project.

Within a project, the model level says things about the application domain while the role level says things about the model level. For example, *DTollPrice* says things that must hold in the transportation system while the intent that *DTollPrice* show the information related to toll ticket pricing says something about this model, not the transportation system. In general, we expect the intent about a model to define the *kinds* of information that the model should express and this should not bias the information to be true or false about the application. Although this seems like a straightforward stratification of information, the clean separation of levels is not always possible. For example, simply asserting that there exists a model *DTollPrice* with the intent described above already assumes that the transportation system *has* a type of entity called a “toll ticket”. If it didn’t then the existence of *DTollPrice* would not make sense. Thus, the model intent can also be dependent on particular facts about the application and this causes a “tangling” between the model level and role level. We give other examples of this tangling in subsequent sections of this paper.

Model Roles vs. Model Types. The type of a model is defined by its modeling language and this is typically characterized by a metamodel, has a notation and has associated tools such as editors. A model role is a *use* of a modeling language in a particular context. Thus, model roles and model types are related but distinct concepts.

A given model type can be used for many roles and a given role could be modeled using potentially many types; however, the combinations are not arbitrary. For example, a Statechart may be used to play the roles “behaviour of class *Car*”, “process for purchasing a toll ticket” or “behaviour of the traffic network” but it could not be used to play the role “organizational structure of toll operators” because Statecharts do not provide the appropriate concepts required for this modeling task. On the other hand, the role “process for purchasing a toll ticket” could be modeled using a Statechart, Flowchart, UML Activity Diagram or UML Sequence Diagram but not using a Class Diagram.

3.2 Expressing Intent about Content – Role Constraints

We assume that no model is created without a purpose and this purpose is the key driver of model intent. Within a software development context, Lange [6] has classified some of the possible purposes a model may have such as to support testing, analysis, communication, etc. If the context is broadened to include entire organizations that produce software, then the possible purposes of models can include such things as support for persuasion (sales & marketing), training (human resources) and decision making (marketing & management). [2]

Although the possible purposes of a model seem quite diverse, the only thing a model can actually provide is a *set of information*. Thus, we may reduce the question of the model’s purpose to that of what information it should provide and by what means – i.e., what the requirements for the model are. With software artifacts we can have both functional and quality (or, non-functional) requirements. Similarly, with models we can state its *content* requirements (what information it should provide) and its quality requirements (how it provides the information).

The content requirements of a model define what information belongs in the model and what does not. The quality requirements of a model specify more generic properties that the information must satisfy and these can be captured by model quality metrics such as complexity, balance, conciseness, precision, flexibility, etc. Model quality metrics have received significant research attention [e.g., 6, 11]. While both types of requirements can be expressed at the model role level, our focus in this

[2] Of course, one may come up with “unorthodox” purposes for a model, e.g. to impress my boss, to decorate my wall, etc. however we do not consider these types of purposes in this paper.

paper is on content requirements. The modeler (as definer) interprets the content requirements as a *set of constraints* that the information in the model must satisfy – i.e., as a specification for the model content. We call this specification the content criteria of the model.

Kinds of Model Constraints. At this point we need to consider more carefully the term "constraint" used in the above exposition. The concept of constraint is a very general one and different kinds of constraints have different functions within modeling. To help characterize these kinds, consider the taxonomy of constraints shown in Fig. 3. *Type constraints* are due to the rules for correct usage of the modeling language used by a model independently of the particular purpose for which the model is used. These include semantic constraints that ensure that the content is semantically interpretable and consistent and syntactic constraints that ensure that the model is renderable using a particular notation.

Role constraints are due to the intended purpose of the model and thus are usage context dependent. From a linguistic point of view, these carry information about pragmatics and determine how the same model could be interpreted differently in different usage contexts. Role constraints can be further subdivided as follows. *Method-level constraints* are role constraints due to the development methodology being followed to create the content. These include constraints that require the existence of models playing particular roles, constraints on the sequence of modeling activities, constraints on content due to modeling conventions, etc. *Project-level constraints* are role constraints due to the modeler's design decisions and interpretations of stakeholder needs within a particular project (i.e., an instance of a development method). These include constraints that require the existence of models playing particular roles, constraints that define what content is required by the purpose and constraints that govern the decomposition of a model. For example, consider the different kinds of constraints on the content of *DTollPrice* shown in Table 1. We give a suggestive formalization of each constraint using logic[3]. Constraint (C3) is an example where the model level is tangled with the role level since the existence of state machine models in the project is dependent on the

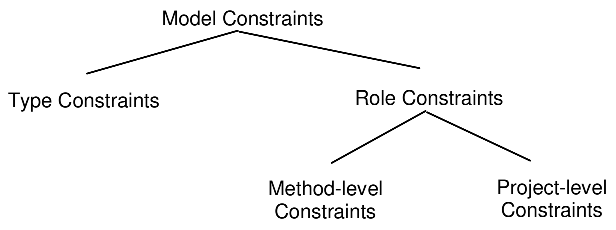

Fig. 3. A taxonomy of constraints that apply to models.

[3] TC means transitive closure. Readers interested in the details of our formalization approach, please see [16, 17].

occurrence of classes within a particular model. Most of the research on model constraints within software engineering has focused on type constraints and associated notions like consistency [e.g., 8].

A key difference between project-level role constraints and the other two kinds is the level of generality of the constraint. Type constraints are common to all models of the same type (i.e., same language) and are typically defined as part of the metamodel for the model type. Method-level role constraints are common to all models playing the same type of role. If these are expressed, they are found as part of the method definition. In contrast to both of these, project-level role constraints are specific to a particular model role within a particular project. This means that we view these role constraints to be *part* of the modeling project rather than outside or above it. Thus, a violation of such a constraint could be addressed either by changing the model content or by changing the constraint itself to reflect a change in the intent about the content. For example, if constraint (C4) is violated by *DTollPrice* because it contains the toll ticket attribute *ticketFormat* that is unrelated to toll ticket pricing, then a valid response on the part of the definer is to recognize that the intent of *DTollPrice* has evolved (e.g., to expressing all toll ticket details) and it now should allow this attribute.

A software method typically identifies the abstract input and output roles for development activities (e.g., requirements gathering, design, etc.) as well as the constraints between these roles. Since these are defined at the method level scope that spans multiple projects we could consider them to be expressing the intent of the *development organization* rather than the intent of modelers. As such, the method level can only be used to express a limited amount detail regarding these roles. For example, the Rational Unified Process (RUP) [5] identifies the existence of an "elaboration phase architecture model" but in a particular project this would typically consist of multiple partial models (or diagrams) to address different concerns, different stakeholder views, different modeling languages, different decompositions, etc. Each of these models in the elaboration phase has a distinct role, and thus, a distinct model intent associated with it and this cannot be expressed at the method level.

Table 1. Examples of different kinds of model constraints.

Kind of constraint	**Constraint involving *DTollPrice***
Type	(C1) *DTollPrice* (as a class diagram) can't contain a class that is a subclass of itself. $\forall m{:}\text{CD}\ \forall c{:}m.\text{Class} \cdot \neg\text{TC}(\text{subclassOf}(c, c))$
Role (method-level)	(C2) *DTollPrice* (as a Design model) cannot use multiple inheritance. $\forall m{:}\text{Design}\ \forall c{:}m.\text{Class} \cdot \neg\exists c_1, c_2{:}\text{DTollPrice.Class} \cdot \text{subclassOf}(c, c_1) \wedge \text{subclassOf}(c, c_2)$
Role (project-level)	(C3) Every class of *DTollPrice* requires a corresponding Statemachine Diagram to show its behaviour. $\forall c{:}\text{DTollPrice.Class}\ \exists m{:}\text{TransSysProject.SMD} \cdot \text{behaviourOf}(m, c)$ (C4) *DTollPrice* contains all and only attributes related to toll ticket pricing. $\forall a{:}\text{DTollPrice.Attr} \cdot \text{partOfComputation}(a, \text{tollTicketPrice})$

3.3 Expressing Intent about Relationships

The purpose of a model may require that its content constrain the content of other models or be constrained by the content of other models. We refer to a role constraint that is intended to hold between particular roles as an *intended model relationship*. Specifically, when such a constraint is intended to hold between certain roles then it means that the possible combinations of models that can play these roles are restricted because only certain combinations satisfy the constraint.

The model relationships that occur between models are often instances of typical *relationship types* encountered in Software Engineering such as *submodelOf*, *refinementOf*, *aspectOf*, *refactoringOf*, *projectionOf*, *transformationOf*, *realizationOf*, etc. The use of relationship types allow sets of constraints that are commonly expressed between roles to be packaged and reused. They also provide a meaningful level of abstraction on the constraints expressed at the role level by summarizing the intent. For example, if an instance of the relationship type *UMLrefines* (specializing *refinementOf*) is intended to hold between two UML model roles, then this carries a different meaning for the stakeholder than another relationship type that says that one model is a *submodelOf* of the other. Thus, relationship types are both a reuse mechanism and an abstraction mechanism that simplify the expression of model intent.

3.4 Expressing Intent about Model Decomposition

Up to this point in our analysis, we have only considered model intent expressed as role constraints on a model or between particular models. Another common scenario at the role level is that the modeler intends that a required model (i.e., for which there is a model role) be decomposed into a collection of interrelated models rather than being created as a single model. There are many possible reasons to do this and the reason defines the purpose of the decomposition. For example, in order to manage complexity, the model may be decomposed into smaller parts and into different levels of abstraction. A model may be decomposed because it is not renderable as-is and it must be split into diagrams that have well defined notations. This is the case with the UML – it has no single notation for the entire modeling language and so a UML model must always be decomposed into diagrams in order to be rendered. Another reason to decompose a model is to support some task – e.g., in order to assign the parts to different development teams.

Note that the purpose of the decomposition may underdetermine it since there may be many possible decompositions that can satisfy this purpose. However, we argue that when a modeler decides to decompose a model, they do not break it up in an arbitrary way but rather they have an intent about how this should be done – i.e., they have some decomposition criterion. Furthermore, following this criterion does not simply yield a set of partial models of the whole - it must also define the intent for each of these models, since they, like all models, must each have a well defined purpose.

Thus, the intent regarding a decomposition is driven partly by the purpose of the decomposition and partly by the modeler's design decisions on how to achieve this purpose. For example, assume that we have the model role *VehicleTypes* in the transportation system design and the purpose of this model is to show all the types of vehicles that are used in the transportation system. Now, assume that the modeler (as definer) decides that this model is too complex and must be decomposed. Consider the following two possible decompositions:

- D1 = {*LightVehicleTypes, MidrangeVehicleTypes, HeavyVehicleTypes*}
- D2 = {*PassengerVehicleTypes*, *CommercialVehicleTypes*, *ServiceVehicleTypes*}

D1 represents a decomposition of *VehicleTypes* on the basis of vehicle weight while *D2* is a decomposition on the basis of vehicle function. Both satisfy the purpose of managing complexity and both define the intent for each constituent model but each has a different basis for the decomposition. Thus, the basis for the decomposition is also part of the decomposition criteria. Note that the bases for both these decompositions come from the application domain and so they are examples of tangling between the model and role levels.

Since decomposition can be applied recursively it is natural to have hierarchical decompositions. Note however that the decompositional structure is normally only a role level phenomenon and is not evident at the model level. There are two reasons for this. First, there is no approach in common usage for identifying collections of models that decompose another model[4]. Second, when a decomposition occurs, often only the constituent models actually exist as artifacts and the decomposed model is "implicit." For example, if the decomposition *D1* is used for *VehicleTypes*, and we have a set of three models that realize the roles in *D1*, then the model that realizes *VehicleTypes* can be derived from these (by composing them). However, if the decomposition is sufficient to satisfy the stakeholders information requirements then this model does need to actually be materialized.

4 Uses of Model Intent

In this section, we briefly[5] consider the ways in which the role level and model intent can be used to support modeling. Since the role level constrains (via role constraints) the possible project configurations at the model level to those that conform to model intent, this can be utilized to detect model defects by determining whether models playing roles satisfy their role constraints. This includes the detection of existential intent violations when a role exists that has no corresponding model in the project. For models in the project that are found to be non-conformant, the role constraints can also be utilized to help guide the repair process by restricting the allowable modifications. When role constraints are formalized, these supports can be built into modeling tools and automated.

The typical approach to modeling based on the model level is bottom-up: models are created incrementally by adding content to them and new models are introduced as needed. During bottom-up modeling, the act of expressing the intent of a model is

[4] Of course, artifact grouping mechanisms such as folders, packages, etc. could be used informally to indicate this.

[5] For more detailed descriptions of how model intent can be used, please see [16, 17].

useful because it forces a modeler to clarify what the purpose of the model is and provides a way to ensure that the content is consistent with this purpose.

The fact that a model role can precede the existence of a model playing the role means that a definer can use role constraints as a way to direct modeling activities. Thus, in addition to bottom-up modeling, the existence of the role level creates an opportunity for *top-down modeling* by allowing the required models and their intent to be specified before content is created. While bottom-up modeling is more *organic*, top-down modeling is more *designed* since the "information architecture" of how content is distributed across multiple models can be prescribed. Top-down modeling also enables the management of a multi-person modeling process by allowing different model roles to be assigned to different modelers.

5 Related Work

The concept of "role" has been used in many contexts within computer science. In their work on social roles, Masolo et. al. [9] review these uses and identify some common ontological features of roles. A role is a kind of property and so it can have multiple players. This is clearly also the case with a model role, although it can only have a single player at a time. A role is always linked to a context. In the case of a model role this is a modeling project. A role has a relational nature and its definition may depend on the existence of other properties. We see this with model roles through the intended model relationships and the tangling with the model level. Finally, roles are *anti-rigid* and thus they are necessarily non-essential properties of the entities that play them. In this respect model roles differ from social roles since models are specifically constructed to play a particular role within a project. Although the model can still exist when it is not playing this role, its value as an "information bearer" is tied to its role; hence we consider the model role to be a "quasi" essential property of a model.

There is work relating to various types of "intention" within software engineering, although the specific issue of expressing model intent has not been a focus of academic research (as far as the authors can determine). For example, Yu defines the i* language for describing stakeholder intentions in early requirements engineering [19], while Ralyté and Rolland use the intention of a process step to support merging method fragments in method engineering [15]. The work of Mens et. al. regarding *intentional source-code views* to help with software maintenance [10] is more closely related to our interests because here a view of software is defined in a declarative way so that the "intent" of the view can be clearly understood. Despite this it is not concerned with models.

If we consider specifically model-related research that could potentially be used for expressing model intent, the early and influential ViewPoints framework [14] stands out as the "best fit." Here, a ViewPoint is similar to a model role in that it is an "envelope" that carries metadata about a model including constraints with other models. However, the ViewPoints framework uses this for managing decentralized development rather than for characterizing model purpose. Furthermore, role constraints are not clearly distinguished from type constraints and thus there is no systematic approach to managing the differences between violations to these kinds of constraints. Finally, there is no support for relationship types or for hierarchical decompositions of models.

Other heterogeneous modeling frameworks that have emerged more recently, such as the Generic Modeling Environment (GME) [7] and the Atlas Model Management Architecture (AMMA)[2], focus entirely on models and do not make the distinction with model roles. AMMA does have a special model type called a MegaModel for expressing metadata about models and bears some similarity to our macromodel language for the role level [17]. Unfortunately, there is no special support for expressing role constraints at this level.

Finally, the Software Process Engineering Metamodel (SPEM) [18] used for defining development methods effectively captures the notion of a model role with its concept of a *work product* used by tasks. Despite this, it is limited to use at the method level and not the project level. Furthermore, although relationships between work products can be expressed, these are limited to process sequencing constraints rather than constraints on the content of models realizing the work products.

6 Conclusions

Much of modeling research and practice has focused on the model level where the information about the application domain is created. In this paper, we argue that there is also another level of information that is central to modeling but has not been subject to significant examination. The model role level is where model intent exists and the focus of this paper is to analyze this notion and present a vision of how it can positively impact modeling.

We approached the analysis by first recognizing that when modelers create a collection of models in order to satisfy the requirements of stakeholders, they do so with an intent about what role each model plays in the collection. We then presented a framework that defines four aspects of model intent at the role level: the existential intent for a model that arises in response to the need for information by stakeholders, the content criteria that expresses what information the model is intended to contain, model relationships that express how models are intended to constrain one another and the decomposition criteria that expresses the intent behind how a model is decomposed into a collection of models. Explicitly articulating model intent has benefits for all stakeholders by supporting model comprehension, improving model quality and enabling automated support for model analysis.

Although we have laid foundations here and in our other work [16,17], making the use of the role level a practical reality requires further research. In particular, because expressions of model intent can be used both for supporting model automation and comprehension, they have the conflicting requirement of both being formal and being understandable by non-technical stakeholders. In addition, the added burden of expressing model intent may discourage its use and we need techniques such as reuse and automated inference of intent to help minimize this effort. Finally, while our work focuses mainly on the project level, more research must be done on how the role level can enrich method specification languages such as SPEM [18] by using model intent to constrain the content of artifacts used by a method.

References

[1] Bernstein, P.: Applying Model Management to Classical Meta Data Problems. In Proc. Conf. on Innovative Database Research, pp. 209--220 (2003)

[2] Bezivin, J., Jouault, F., Rosenthal, P., and Valduriez, P.: Modeling in the Large and Modeling in the Small. Lecture Notes in Computer Science, Num. 3599, Springer-Verlag GmbH, pp. 33--46 (2005)

[3] Czarnecki, K. and Helsen, S. : Feature-based survey of model transformation approaches. IBM Syst. J., Vol. 45, No.. 3, pp. 621--645 (2006)

[4] Harel, D. and Rumpe, B.: Meaningful modeling: what's the semantics of" semantics"? IEEE Computer, vol. 37, no. 10, pp. 64--72 (2004)

[5] Kruchten, P.: The rational unified process: an introduction. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA (2000)

[6] Lange, C.F.J. and Chaudron, M.R.V.: Managing Model Quality in UML-Based Software Development. 13th IEEE International Workshop on Software Technology and Engineering Practice, pp.7--16 (2005)

[7] Ledeczi A., Maroti M., Bakay A., Karsai G., Garrett J., Thomason IV C., Nordstrom G., Sprinkle J., Volgyesi P.: The Generic Modeling Environment, Workshop on Intelligent Signal Processing, May 17 (2001)

[8] Lucas, F. J., Molina, F., Toval, A.: A systematic review of UML model consistency management, Information and Software Technology (2009)

[9] Masolo, C., Vieu, L., Bottazzi, E., Catenacci, C., Ferrario, R., Gangemi, A. and Guarino, N.: Social roles and their descriptions. In Proc. of KR'04, pp. 267--277 (2004)

[10] Mens, K., Mens, T. and Wermelinger, M.: Maintaining software through intentional source-code views. In proc. of the 14th international conference on Software engineering and knowledge engineering (2002)

[11] Moody, D. L.: Metrics for evaluating the quality of entity relationship models. Lecture notes in computer science. Springer, pp. 211--225 (1998)

[12] Moody, D. L.: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Transactions on Software Engineering, pp. 756-779, November/December (2009)

[13] Mylopoulos, J., A. Borgida, M. Jarke, and M. Koubarakis, Telos: Representing Knowledge About Information Systems. ACM Transactions on Information Systems, vol. 8, no. 4, pp. 325--362 (1990)

[14] Nuseibeh, B., Kramer J. and Finkelstein, A. : A Framework for Expressing the Relationships Between Multiple Views in Requirements Specifications. IEEE TSE, vol. 20, no. 10, pp. 760--773 (1994)

[15] Ralyté, J. and Rolland, C.: An assembly process model for method engineering. In Proc. CAiSE 2001, pp. 267--283 (2001)

[16] Salay, R., Mylopoulos, J.: Improving Model Quality Using Diagram Coverage Criteria. In Proc. CAiSE 2009, pp. 186--200 (2009)

[17] Salay, R., Mylopoulos, J., Easterbrook, S.M.: Using Macromodels to Manage Collections of Related Models. In Proc. CAiSE 2009, pp.141--155 (2009)

[18] Software Process Engineering Metamodel V2.0 specification. Object Management Group., http://www.omg.org/technology/documents/formal/spem.htm

[19] Yu, E.S.K. and Mylopoulos, J.: Modelling organizational issues for enterprise integration. Proc. Int. Conf. On Enterprise Integration and Modelling Technology. Turin, Italy (1997)