Improving Model Quality Using Diagram Coverage Criteria

Rick Salay and John Mylopoulos

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

Abstract. Every model has a purpose and the quality of a model ultimately measures its fitness relative to this purpose. In practice, models are created in a piecemeal fashion through the construction of many diagrams that structure a model into parts that together offer a coherent presentation of the content of the model. Each diagram also has a purpose – its role in the presentation of the model - and this determines what part of the model the diagram is intended to present. In this paper, we investigate what is involved in formally characterizing this intended content of diagrams as *coverage criteria* and show how doing this helps to improve model quality and support automation in the modeling process. We illustrate the approach and its benefits with a case study from the telecommunications industry.

Keywords: Modeling, Model quality, Diagrams.

1 Motivation

All models are created for a purpose [4]. For example, in a software development context, models may be used to communicate a software design, to help a designer work through alternative ideas, to support the work of various stakeholders, to enable a particular type of analysis, etc. The quality of a model corresponds to how well it can support its purpose by providing the information required by it – i.e. the purpose of a model determines its intended content. Thus, if the intended content can be characterized in terms of *content criteria* such as the required notation, coverage, accuracy, level of abstraction, etc. we can consider model quality to be measured by the degree to which the model meets these criteria.

In practice, a model is often manifested as a set of diagrams, possibly of different types, that decompose and structure the content of the model. The prototypical example of this is the UML which defines a single metamodel for UML models and identifies thirteen types of diagrams that can be used with it [13]. Like the models they present, each diagram has a purpose and plays a particular role in the presentation of model content, hence they too have content criteria. In particular, the content criteria relating to coverage, or *coverage criteria*, for a diagram identifies the part of the information carried by a model that is intended to be presented within the diagram. For example, in a UML model of a communication system, one class diagram may be intended to show the different types of communicating entities while another is intended to show the different types of messages that an entity of type *Terminal* can send.

P. van Eck, J. Gordijn, and R. Wieringa (Eds.): CAiSE 2009, LNCS 5565, pp. 186–200, 2009.

Coverage criteria are not typically modeled and if they are made explicit at all, it is only through some informal means such as comments or as part of the name of the diagram. However, the explicit and precise expression of coverage criteria is a fruitful activity because it helps improve the quality of models in several ways. Firstly, it improves model comprehension because it provides information that allows model consumers to properly interpret the content of diagrams and assess their overall quality. For example, without explicit coverage criteria it may not be clear whether the associations in the class diagram of communicating entity types represent all or just some of the associations between these entities. Secondly, it can be used to identify types of defects that are not detectable through other means. For example, coverage criteria can be used to detect when the class diagram of communicating entity types contains classes it shouldn't or doesn't contain classes that it should. Finally, the coverage criteria can be used with change propagation mechanisms to properly maintain the intended content of diagrams as the model evolves.

In [11] we describe how the types of relationships that exist between models and between diagrams play a role in describing the intentions about content. In this paper, we explore the relationship between a diagram and a model in greater depth. In particular, we make the following contributions:

- The notion of diagram coverage criteria is introduced as a new kind of information that can be included in a model.
- Four kinds of modeling defects are identified that can only be detected using coverage criteria.
- A systematic approach for defining formal coverage criteria is presented and the validity conditions that coverage criteria must satisfy are specified.
- A strategy for parameterizing coverage criteria is defined that allows reuse of coverage criteria to reduce specification effort and to allow diagrams with standard types of intentions to be auto-generated.
- Empirical results are presented of the application of the approach to a medium size UML model with 42 diagrams.

The rest of the paper is structured as follows. Section 2 introduces the concepts related to diagram coverage criteria and illustrates them using examples. Section 3 formalizes these concepts and provides a systematic way of defining coverage criteria. Section 4 describes the results of applying the approach to a UML case study in the telecommunications domain. Finally in Section 5 we discuss related work and in Section 6 make some concluding remarks.

2 Diagram Coverage Criteria

In order to illustrate the idea of diagram coverage criteria we utilize examples from a UML case study taken from a standards document for the European Telecommunications Standards Institute (ETSI) [7]. The case study consists of three UML models: a context model (4 diagrams), a requirements model (6 diagrams) and a specification model (32 diagrams) and details the development of the Private User Mobility dynamic Registration service (PUMR) – a standard for integrating telecommunications networks in order to support mobile communications. Thus, for example, PUMR allows

an employee using a mobile phone at her home company with a private exchange to roam to other private exchanges seamlessly. More specifically, it describes the interactions between Private Integrated Network eXchanges (PINX) within a Private Integrated Services Network (PISN). The following is a description from the document:

> "Private User Mobility Registration (PUMR) is a supplementary service that enables a Private User Mobility (PUM) user to register at, or de-register from, any wired or wireless terminal within the PISN. The ability to register enables the PUM user to maintain the provided services (including the ability to make and receive calls) at different access points." [7, pg. 43]

Consider diagram 65 from the specification model as shown in Figure 1. The intent of this class diagram is to show the detail for the types of response messages that can be exchanged between communication entities when they are trying to connect. The class *PUM Responses* is the abstract base class for these classes. This intention implies that the following constraints must hold between diagram 65 and the specification model:

1. Every class that is included in this diagram must either be *PUM Responses* or be a direct subclass of it.
2. Every direct subclass of *PUM Responses* in the specification model is included in this diagram.
3. For each class included in this diagram, every attribute in the specification model is included in this diagram.
4. No associations are included in the diagram.

These constraints constitute the coverage criteria for diagram 65. Assume that we identify three roles for stakeholders dealing with diagrams: definer, producer and consumer. The diagram definer asserts that such a diagram must exist and what it is intended to contain. The producer creates the content and the consumer uses it for their purposes. Expressing the coverage criteria explicitly and precisely is useful for all three roles. The definer can articulate the intent of the diagram and effectively communicate this to the producer. The producer can use this to assess whether they are conforming to this intent. Constraints (1) and (4) ensure that nothing is included that does not belong in the diagram while constraints (2) and (3) ensure that everything that does belong is included. The consumer can use the constraints to properly interpret the content of the diagram. For example, without (3) it may not be clear to the consumer whether or not the attributes for these classes shown in the diagram are all the attributes for these classes or there are some more that have been omitted from the diagram. Thus, while diagrams are typically assumed to be incomplete relative to the model, the coverage criteria provide the consumer with information about the ways in which the diagram *is* complete. If formalized, the coverage criteria are useful for automated support of the management of the diagram content in order to ensure that the intent of the diagram is maintained as the specification model evolves. For example, if the producer adds a class to the model via diagram 65 and does not make it a subclass of *PUM Responses*, this violates constraint (1) and can be flagged as such. On the other hand, if a subclass of *PUM Responses* is added to the specification model by some other means, such as manually through another diagram, change propagation, round-trip engineering, etc., the violation of constraint (2) can trigger the "repair" action of adding it to diagram 65.

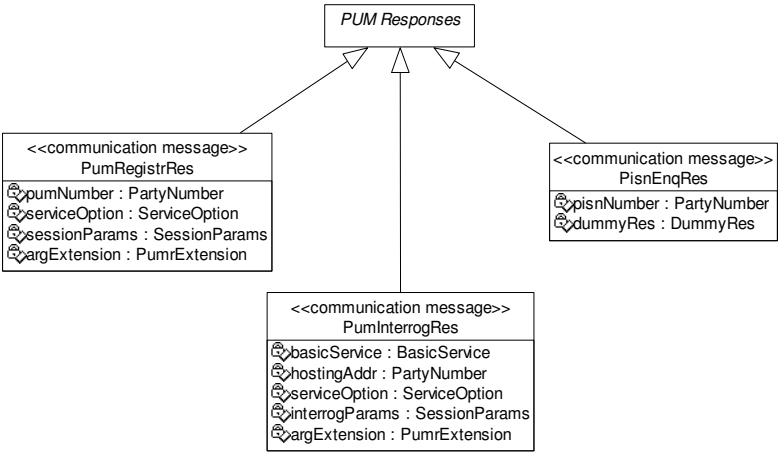

Fig. 1. Diagram 65 is a class diagram showing PUMR response message types carried in a connect signal

Note that the constraints in coverage criteria are different from constraints that are expressed within the metamodel. Metamodel constraints include invariants that must hold in the modeled domain (e.g. a class can't be a subclass of itself), completeness constraints from the modeling process (e.g. every use case requires an activity that describes it) and stylistic constraints (e.g. only single inheritance is permitted). Since diagrams do not exist within the model, these constraints do not address the content of diagrams. In contrast, coverage criteria are wholly concerned with the relationship between diagrams and the model. Furthermore, since the diagram intent is defined relative to the model, the coverage criteria is comprised of information that exists at the model level rather than at the metamodel level. For example, if model evolution causes some coverage criterion to be violated, a valid response may be to change the coverage criterion rather than the diagram. This represents a decision on the part of the model definer that the intent of a diagram has evolved.

The coverage criteria for diagram 65 are constraints that are mostly expressed in terms of the generic concepts in the metamodel (i.e. class, subclass, association, etc.) except for the mention of the specific class *PUM Responses*. The diagram has a special existential relationship to this element since it doesn't make sense to have a diagram that shows the subclasses of *PUM Responses* unless there exists a class in the model called *PUM Responses*. Thus, we consider it to be a *precondition* for the existence of diagram 65 that the model contain this class. When the precondition for a diagram is the required existence of a single model element, then the diagram is often a "detailing" of this element. For example, diagram 65 could be considered to be a detail view of the *PUM Responses* class. This special case has been leveraged by modeling tools [5, 6] to define a natural navigation path between diagrams containing the detailed element and the diagrams that are detail views of this element.

The coverage criteria for diagram 65 are strong enough that for any UML model that satisfies the precondition, the content of the diagram is uniquely determined – i.e. the coverage criteria constitutes a query on those specification models that satisfy the

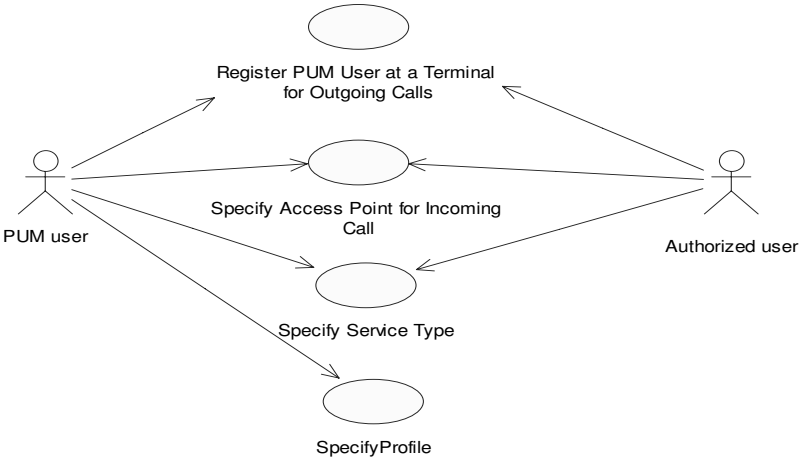

Fig. 2. Diagram 44 is a use case diagram showing the registration use cases

precondition. The intuition here is that when the producer fills in a diagram they must always have some principle in mind by which they decide what should be included and what should not be included and following this principle results in a unique diagram[1]. This principle is exactly the coverage criteria. Thus, there is a general pattern for coverage criteria: there is a (possibly empty) precondition and the coverage criteria uniquely determine the content of the diagram on any model satisfying the precondition.

Now consider diagram 44 from the requirements model shown in Figure 2. The intent of this diagram is to show all use cases related to the registration of PUM users within a network. The coverage criteria can be expressed as follows:

1. Every use case that is included this diagram is a registration use case.
2. Every registration use case (in the requirements model) is included in this diagram.
3. Every actor (in the requirements model) associated with a use case included in the diagram is also included in the diagram.
4. Every association (in the requirements model) between any elements in the diagram is also included in the diagram.

Like the coverage criteria for diagram 65 this lists a set of diagram inclusion constraints that pick out a unique diagram for each model; however, unlike diagram 65, the truth of these conditions cannot be fully determined from the content of the requirements model alone. In particular, there is no information in the requirements model that can be used to determine whether or not a particular use case is a registration use case. This highlights another benefit of articulating the coverage criteria – it exposes contextual information that is assumed when interpreting the diagram and that may be missing from the model. One response to this is to extend the model

[1] Note that we are not suggesting that the information in a diagram can only be presented in one way but rather that what information is included in the diagram is determined completely by the principle the producer is following.

to include this information. In this case, this could be done in several ways ranging from formal to informal including: adding a use case called "Registration" which these use cases specialize, adding a profile at the metamodel level with an attribute to indicate the type of use case, using a naming convention on use cases to indicate the type of use case, annotating the use cases with comments, etc.

Another response to this situation is to treat the inclusion of a use case in the diagram as *meaning* that the use case is a registration use case. In this case, diagrams are not only used as views on the model but also to extend the model itself. Since diagrams are typically considered to only be relevant to the presentation of a model and not its content, this approach has the drawback that the information may not be preserved in further refinements of the model (e.g. into the code) and hence would be lost. This suggests that the first response may be preferred if this information is needed in downstream processes – i.e missing context information should be viewed as a case of model incompleteness.

2.1 Parameterized Coverage Criteria

In many cases, it is possible to generalize the diagram intention and coverage criteria by replacing certain constants by parameters. For example, let *DirectSubclass*[*c*:class] represent the coverage criteria for a type of class diagram having the generalized intention that the diagram shows a class *c* and the detail of all of its direct subclasses:

1. Every class that is included in this diagram must either be *c* or be a direct subclass of it.
2. Every direct subclass of *c* in the specification model is included in this diagram.
3. For each class included in this diagram, every attribute in the specification model is included in this diagram.
4. No associations are included in the diagram.

Now the coverage criteria for diagram 65 in Figure 1 could be expressed as *DirectSubclass*[*PUM Response*]. An obvious benefit of parameterized coverage criteria is reuse as it reduces the incremental effort to define the coverage criteria for different diagrams when the coverage criteria have the same form. However, there are other benefits as well. Formalized parameterized coverage criteria can be used to define a library of common types of diagrams that can then be used to automatically *generate* diagrams of these types and hence reduce modeling effort. For example, a model that is produced by a reverse engineering tool can be quickly structured by generating diagrams using different parameterized coverage criteria and various parameters. Furthermore, the generalized intent can be used to generate a meaningful diagram name (and other metadata) that reflects the intent of the diagram. For example, *DirectSubclass*[*c*:class] can be applied to various classes to produce the diagram of its subclasses and generate the name "The direct subclasses of *c*" for each diagram.

3 Formalization

The objective of this section is to encode coverage criteria formally in order to be precise about its structure, allow the definition of validity conditions that must hold

and for characterizing the types of defects that can be detected. In order to express metamodels in a formal way, we have chosen to use order-sorted first order logic with transitive closure (henceforth referred to as FO+) as the metamodeling formalism rather than using either MOF or Ecore with OCL. There are a number of reasons for this. Firstly, first order logic is widely known and has comparable expressive power to the other metamodeling approaches. Secondly, it has a textual representation that is more convenient when discussing formal issues. Finally, its semantics are formal and notions such as logical consequence and consistency are well defined.

Using FO+ we can define the metamodel of a model type as a pair $\langle \Sigma, \Phi \rangle$ where Σ is called the *signature* and defines the types of model elements and how they can be related while Φ is a set of axioms that defines the well-formedness constraints for models. Thus, a metamodel $\langle \Sigma, \Phi \rangle$ is an FO+ theory and each finite model (in the model theoretic sense) of this theory will be considered to be a model (in the modeling sense) that is an instance of this metamodel.

For example, we define the metamodel (abstract syntax) of a simplified UML class diagram as follows.

CD = (1)

sorts class, assoc, attr, string

pred startClass: assoc × class
endClass: assoc × class
attrClass: attr × class
className:class × string
subClass: class × class

constraints

// startClass, endClass, attrClass and className are functions[2]

$\forall a$:assoc $\exists! c$:class · startClass(a, c)

$\forall a$:assoc $\exists! c$:class · endClass(a, c)

$\forall a$:attr $\exists! c$:class · attrClass(a, c)

$\forall c$:class $\exists! s$:string · className(c, s)

// a class cannot be a subclass of itself

$\forall c$:class · ¬TC(subClass(c, c))

The signature Σ_{CD} consists of the pair $\langle sorts_{CD}, pred_{CD} \rangle$ where $sorts_{CD}$ is the set of element types that can occur in a model while $pred_{CD}$ is a sets of predicates used to connect the elements. We will say $\Sigma_{T1} \subseteq \Sigma_T$ to mean that $sorts_{T1} \subseteq sorts_T$ and $pred_{T1} \subseteq pred_T$. The **constraints** section describes Φ_{CD}. Note that the quantifier $\exists!$ means "there exists one and only one" and the operator TC takes a predicate and produces its transitive closure.

[2] In FO+ we express functions as appropriately constrained predicates rather than including functions directly into the logic in order to treat this in a uniform way with other well-formedness constraints.

Here we are treating a type of diagram (i.e. class diagrams) in the same way as a type of model by giving it a metamodel defining its abstract syntax. We do this since in this paper we are not interested in the notational aspects of a diagram, only in what subsets of a model they can be used to present. Thus, we will take a diagram to be equivalent to the *submodel* it picks out from a model.

Assume that $T = \langle \Sigma_T, \Phi_T \rangle$ is the metamodel of some model type and $T1 = \langle \Sigma_{T1}, \Phi_{T1} \rangle$ is the metamodel for a type of submodel of T. For example, we could have UML $= \langle \Sigma_{UML}, \Phi_{UML} \rangle$ as the model type and CD $= \langle \Sigma_{CD}, \Phi_{CD} \rangle$ as the type of submodel we are interested in. Since T1 is a type of submodel of T we will assume that $\Sigma_{T1} \subseteq \Sigma_T$. Note that in general, the constraints of a type of submodel may be either stronger or weaker than the constraints of the model. For example, in UML, communication diagrams can only represent interactions in which message overtaking does not take place [13] and so the constraints on communication diagrams are stronger than on interactions within a UML model.

Now assume that M:T is a model and Msub:T1 is intended to be a submodel of it. We will interpret this to mean that the constraint Msub $\subseteq$ M must hold. That is, each element and relation instance in Msub must also be found in M. If in addition, we state that Msub has coverage criteria CC(Msub, M), then intuitively this will mean that CC contains the preconditions and the constraints that further limit which submodels Msub can be. Formally, we can express CC as a set of constraints on the combined signatures of T1 and T using a special type of metamodel that includes the metamodels of T1 and T and additional constraints showing how these models are related [11]. As an example, we will express the coverage criteria for diagram 65 as follows:

CC_{M65}(M65:CD, M:UML) = M65.CD + M.UML + (2)

subsort M65.class $\leq$ M.class, M65.assoc $\leq$ M.assoc, M65.attr $\leq$ M.attr (3)

constraints

// precondition

$\exists$ *mc*:M.class · M.className(*c*) = "PUM Responses" $\wedge$ (4)

// inclusion constraints

$\forall c$:M.class · ($\exists c1$:M65.class · $c1 = c$) $\Leftrightarrow$ ($c = mc \vee$ M.subClass(c, mc)) $\wedge$ (5)

$\forall a$:M.assoc · ($\exists a1$:M65.assoc · $a1 = a$) $\Leftrightarrow$ *FALSE* $\wedge$ (6)

$\forall a$:M.attr · ($\exists a1$:M65.attr · $a1 = a$) $\Leftrightarrow$ ($\exists$ c:M65.class · M.attrClass(a) = c) $\wedge$ (7)

$\forall c1, c2$:M65.class · M65.subClass($c1$, $c2$) $\Leftrightarrow$ (M.subClass($c1$, $c2$) $\wedge$ $c2 = mc$) $\wedge$ (8)

$\forall a$:M65.attr, c:M65.class · M65.attrClass(a, c) $\Leftrightarrow$ M.attrClass(a, c) $\wedge$ (9)

$\forall c$:M65.class, s:M65.string · M65.className(c, s) $\Leftrightarrow$ M.className(c, s) (10)

Line (2) indicates that CC_{M65} imports the signature and constraints for CD and UML and to avoid name clashes these are "namespaced" with "M65" and "M", respectively. Thus, for example, the sort M65.class is distinct from the sort M.class. Line (3) asserts that the elements in M65 are subsets of the elements in M. The precondition (4) asserts the constraints that must hold in M for the diagram to exist – in this case that M must contain a class named "PUM Responses". The use of a

precondition distinguishes the case of a diagram not existing from the case that the diagram exists but is empty.

The inclusion constraints define the constraints that must hold between the content of M65 and M and are defined in the scope of the precondition so that the variable *mc* is bound. These encode the constraints for diagram 65 expressed in words in section 2. Thus, constraints 1 and 2 are expressed by (5), constraint 3 is expressed by (7) and (9) and constraint 4 is expressed by (6). Note that parameterized coverage criteria can be defined by allowing free variables in the definition. For example, if we allow *mc* to remain a free variable in the above then we define the parameterized coverage criteria *DirectSubclass*(M65:CD, M:UML)[*mc*: class].

The coverage criteria above are written in a standardized form. If we assume that we are expressing coverage criteria CC(Msub:T1, M:T) then the form is:

$$\begin{aligned}
&\text{CC(Msub:T1, M:T2)} = \text{Msub.T1} + \text{M.T2} + \\
&\quad \textbf{subsort } \textit{for each } S \in \textit{sorts}_{T1}, \text{Msub.}S \leq \text{M.}S \\
&\quad \textbf{constraints} \\
&\quad \text{// precondition} \\
&\quad\quad \textit{precondition} \wedge \\
&\quad \text{// inclusion constraints} \\
&\quad\quad \textit{for each } S \in \textit{sorts}_{T1}, \\
&\quad\quad\quad \forall x\text{:M.}S \cdot (\exists x1\text{:Msub.}S \cdot x1 = x) \Leftrightarrow Q_S(x) \wedge \\
&\quad\quad \textit{for each predicate } P\text{:}S_1 \times \ldots \times S_n \in \textit{pred}_{T1} \\
&\quad\quad\quad \forall\, x_1\text{:Msub.}S_1, \ldots, x_n\text{:Msub.}S_n \cdot \\
&\quad\quad\quad\quad \text{Msub.}P(x_1, \ldots, x_n) \Leftrightarrow \text{M.}P(x_1, \ldots, x_n) \wedge\ Q_P(x_1, \ldots, x_n) \wedge
\end{aligned} \tag{11}$$

In each inclusion constraint, Q_i represents a formula called the *inclusion condition* that may involve bound variables in the precondition. Intuitively, the inclusion conditions pick out the parts of M that belong in Msub and provide a systematic way of defining coverage criteria. Based on this form, the coverage criteria can be seen to consist more simply of the precondition and a set of inclusion conditions. For example, the coverage criteria for diagram 65 could be expressed more compactly as the set of definitions:

$$\begin{aligned}
&\textit{precondition} := \exists\, mc\text{:M.class} \cdot \text{M.className}(c) = \text{“PUM Responses”} \\
&Q_{class}(c) := (c = mc \vee \text{M.subClass}(c, mc)) \\
&Q_{assoc}(a) := \textit{FALSE} \\
&Q_{attr}(a) := \exists\, c\text{:M65.class} \cdot \text{M.attrClass}(a, c) \\
&Q_{subClass}(c1, c2) := (c2 = mc) \\
&Q_{attrClass}(a, c) := \textit{TRUE} \\
&Q_{className}(c, s) := \textit{FALSE} \\
&Q_{startClass}(a, c)^3 := \textit{FALSE} \\
&Q_{endClass}(a, c) := \textit{FALSE}
\end{aligned} \tag{12}$$

[3] Inclusion constraints for startClass and endClass were omitted in (11) since these must always be empty relations because there are no associations.

When the coverage criteria is expressed in terms of inclusion conditions it is clear that for every M that satisfies the precondition, the inclusion constraints specify a unique submodel Msub of M. This is because there is a constraint for each sort and predicate of Msub that determines exactly what subset of these from M are included in Msub. To ensure that the resulting submodel Msub is also always well formed – i.e. that it satisfies the constraints Φ_{T1} – we must add the following validity conditions:

$$M.\Phi_T \cup Msub.\Phi_{T1} \cup \Phi_{CC} \not\models FALSE \tag{13}$$

$$M.\Phi_T \cup \Phi_{CC} \models Msub.\Phi_{T1} \tag{14}$$

Here, $M.\Phi_T$ and $Msub.\Phi_{T1}$ are the imported versions of the constraints of T and T1 found in CC(Msub:T1, M:T2) and Φ_{CC} are the set of subsort, precondition and inclusion constraints. Condition (13) says that the constraints in CC must be consistent and condition (14) guarantees that the submodel defined by the coverage criteria for each T-model is a well formed T1-submodel. As a simple example of a case where candidate coverage criteria does not satisfy condition (14), consider that in (12) we can change the definition of the inclusion condition $Q_{attr}(a)$ to be $Q_{attr}(a) := TRUE$. This now says that M65 contains all attribute elements of M but still only a subset of the class elements. This clearly can result in M65 being an ill-formed class diagram since it can now contain attributes with no corresponding class – i.e. attrClass is no longer necessarily a function. In particular, this is due to the fact that for the modified constraints $\Phi_{CC_{M65}}$ we have that:

$$\Phi_{UML} \cup \Phi_{CC_{M65}} \not\models \forall a{:}M65.attr\ \exists! c{:}M65.class \cdot attrClass(a, c) \tag{15}$$

We can directly relate the structure of coverage criteria to the types of defects that coverage criteria can be used to detect as shown in Table 1. The first two types of defects can occur when an instance of the submodel violates the coverage criteria – either by excluding intended information or by including unintended information. The third type of potential defect indicates that the coverage criteria cannot be fully characterized using information in M. This was the case with diagram 44 discussed in Section 2 since the inclusion condition identifying a registration use case could not be expressed. As discussed there, the corrective action required depends on whether the information represented by the inclusion conditions is considered to be needed by downstream processes using the model. The fourth type of potential defect is the case where the inclusion condition *can* be specified using information in the model but this information is only "weakly modeled" using some informal scheme such as naming conventions. For example, if a convention is used to prefix all registration use cases with the string "Reg_", this would allow the inclusion condition to be defined by checking for this prefix. The potential problem with this is that the semantics of these conventions may be lost in downstream uses of the model unless they are recorded with the model in some way. Thus it may be preferable to promote this information to "first class" status by modeling it directly – e.g. assuming registration use cases all specialize a use case called "Registration".

Table 1. Defect types

Defect Type	Description	Occurrence criteria
Missing information in Msub	Msub does not contain some information from M that it should.	An instance of an inclusion constraint in which right hand side is satisfied but the left hand side is not.
Too much information in Msub	Msub contains some information in M that it should not.	An instance of an inclusion constraint in which the left hand side is satisfied but the right hand side is not.
Missing information in M	The coverage criteria of Msub cannot be formally expressed in terms of the content of M.	One or more formulas Q_i cannot be formally expressed using information in M.
Weakly modeled information in M	The coverage criteria of Msub is expressed in terms of content in M that is not modeled in the metamodel of M.	One or more formulas Q_i are expressed using informal criteria such as naming conventions.

4 Application to the PUMR Example

In this section we discuss the results of our analysis to express the content criteria for the 42 diagrams over the three UML models in the PUMR example. Since we did not have access to the original definers of these diagrams, we relied on the documentation [7] of the diagrams to infer their intentions. Fortunately, the documentation is substantial and detailed so that we have a high level of confidence that our results are reasonable. To give a sense for the diversity of coverage criteria of the PUMR diagrams, we summarize a few in Table 2.

Figures 3 summarizes the defects found due to our analysis. The bars correspond to the types of defects described in Table 1. To some extent, the low number of missing/additional info defects found for Msub could be attributed to the fact that in

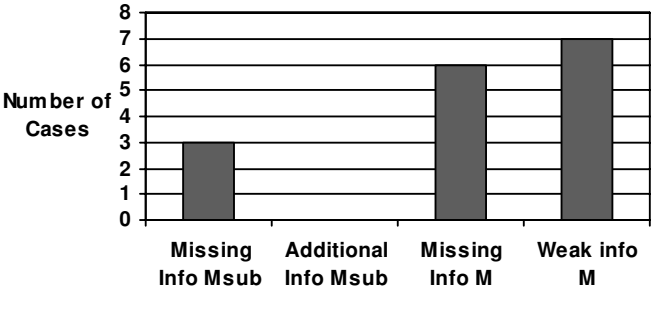

Fig. 3. Defects found in PUMR example

Table 2. Examples of coverage criteria from PUMR analysis

Diagram	Diagram Type	Summary of coverage criteria
62	Class Diagram	The communication classes in the QSIG package that are used by classes in the PUMR package.
73	Class Diagram	All error code classes in the PUMR package and their attributes.
52	Object Diagram	All the objects and links used in registration and de-registration interactions.
58	Statemachine Diagram	The content of statemachine "Registration Processing" except the content of composite state "Registration Request" (the content of "Registration Request" is shown in diagram 59).

Table 3. Parameterized coverage criteria used in the PUMR example

Parameterized coverage criterion	Diagram Type	Summary of coverage criteria	Inst.
DirectSubSuper[class]	Class Diagram	Immediate subclasses and superclasses of the class	2
DirectSubclass[class]	Class Diagram	Immediate subclasses of the class	4
DirectAgg[class]	Class Diagram	Classes directly aggregated by the class	3
NameContains[string]	Class Diagram	Classes whose name contains the string	6
FullActivity[activity]	Activity Diagram	The full contents of the activity	4
FullSequence[interaction]	Sequence Diagram	The full contents of the interaction	5
FullState[state]	State machine Diagram	The full contents of a composite state	1

expressing the coverage criteria we were trying to find the criteria that would best fit the existing diagrams. It is interesting that despite this, there were some clear errors that we were able to find. Cases that exhibited the third type of defect included diagram 44 shown in Figure 2 where there was no way to express the inclusion condition on registration use cases using information in the model. All of the examples of the fourth type of defect relied on naming conventions to identify a type of element. For example, diagram 70 shows the different enquiry message classes. These are all identifiable by having the stereotype "communication message" and by having a name with the prefix "PumEnq".

Of the 42 diagrams, 25 could be seen to clearly be instances of more general parameterized coverage criteria types. This is elaborated in Table 3. Certain "large objects" (e.g. activities, interactions and statemachines) in a UML model have dedicated

diagram types. The most common coverage criteria is to show the full content of these objects in a diagram (e.g. *FullActivity*[activity], *FullSequence*[interaction]). The variety of coverage criteria that can be associated with these depend on their ability to show partial information. For example, statemachine diagrams can also be used to show the content of a single composite state and so we have *FullState*[state]. This capability can be used to decompose the presentation of a large statemachine across several diagrams. This is the case with diagrams 58 and 59 – together they depict the statemachine showing registration processing. A similar possibility exists with large interactions and activities but no instances of these occur in the PUMR example.

5 Related Work

Most of the work relating to diagrams deals with their role in providing a notation for a model. For example, in [1] Baar formalizes the relationship between the metamodels for the concrete syntax and the abstract syntax, in [2], Gurr defines conditions under which a notation is effective, etc. In contrast, our interest is in how diagrams delineate submodels and impose structure on a model and this bears a closer connection to the work on heterogeneous collections of related models.

The problem of inter-view consistency has been much studied, especially with UML (e.g. see [3]) and along with this, investigations into generic constraint management mechanisms such as change propagation and conformance detection have been pursued [10, 9]. Our focus is on the identification and elaboration of a new class of constraints that can characterize an aspect of modeling intention and can be of use in modeling. Thus, our concerns are somewhat orthogonal to but complementary with this work.

In another direction, generic configurable modeling environments have emerged such as MetaEdit+[6] and the Generic Modeling Environment (GME) [5]. The use of the diagram structure here is for the navigation from model elements in one diagram to other diagrams showing more detail about the element (e.g. its internal structure). Such a navigation approach is limited to expressing the intent of diagrams that detail model elements and cannot express more complex intentions such as the "dependency" class diagram containing all classes in package P1 used by classes in package P2. Furthermore, the types of detailing diagrams that can be expressed are restricted to those that can be defined by revealing/hiding particular element types defined in the metamodel. More complex coverage criteria such as only showing "direct" subclasses in diagram 65 are not possible.

Aspect oriented modeling (AOM) bears some similarity to our work and a wide variety of approaches for AOM have been proposed [12]. Here the idea is to provide a means for separately maintaining and developing *aspects* - subsets of the information in a model relating to particular concerns such as security or customization - and then allowing these to be woven together to produce the complete model when necessary. Since there is no consensus on what exactly an aspect is, it is difficult to clearly differentiate our work from this – is every diagram with an intent a valid aspect? The practice of AOM suggests otherwise. Aspects are typically associated with software concerns that crosscut the model whereas we have no such bias. The motivation of AOM is to provide techniques for separating concerns in a manageable way whereas

ours is to articulate the intent of diagrams in order to improve the quality of models. AOM emphasizes the independence of aspects whereas we focus on how submodels are related and are interdependent.

6 Conclusions and Future Work

All models and diagrams of a model have a purpose that circumscribes their contents through content criteria. Moreover, their quality can be assessed by how well their content meets these content criteria. In this paper, we focus on a particular type of content criteria for diagrams called coverage criteria. Coverage criteria for a diagram specify the part of the model that a given diagram is intended to contain. Although coverage criteria are not typically expressed explicitly, we propose that doing so can improve the quality of models by improving model comprehension by stakeholders, allowing the detection of defects that previously could not be detected and supporting automated change propagation and generation of diagrams. We have specified a systematic way of defining coverage criteria using preconditions and inclusion conditions and we have formally defined the conditions under which these conditions are valid. Finally, we have shown the results of applying these concepts to an actual UML case study consisting of 42 diagrams and the concrete benefits we obtained.

As part of future work, we are investigating how to extend this research to expressing coverage criteria about collections of diagrams. The motivating observation here is that the collection of diagrams presenting a model are typically structured further into related subgroups. For example, diagram 44 in Figure 2 can be grouped with another similar use case diagram (diagram 48) that shows the deregistration use cases and together, these two diagrams decompose the full set of use cases in the requirements model. If we take this subgroup of diagrams to be a kind of model (a *multi-model* [11]) then it can have its own coverage criteria that says that it consists of a set of use case diagrams that decompose the use cases in the requirements model. In this way, we hope to extend the ideas in this paper to characterize the intentions about the way collections of diagrams are structured.

Acknowledgments. We are grateful to Alex Borgida for his insightful comments on earlier drafts of this paper.

References

1. Baar, T.: Correctly defined concrete syntax for visual models. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 111–125. Springer, Heidelberg (2006)
2. Gurr, C.: On the isomorphism, or lack of it, of representations. In: Marriott, K., Meyer, B. (eds.) Visual Language Theory, pp. 293–306. Springer, Heidelberg (1998)
3. Huzar, Z., Kuzniarz, L., Reggio, G., Sourrouille, J.L.: Consistency problems in uml-based software development. In: Jardim Nunes, N., Selic, B., Rodrigues da Silva, A., Toval Alvarez, A. (eds.) UML Satellite Activities 2004. LNCS, vol. 3297. pp. 1—12. Springer, Heidelberg (2005)

4. Ladkin, P.: Abstraction and modeling, research report RVS-Occ-97-04, University of Bielefeld (1997)
5. Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason IV, C., Nordstrom, G., Sprinkle, J., Volgyesi, P.: The Generic Modeling Environment. In: Workshop on Intelligent Signal Processing, May 17 (2001)
6. MetaEdit+ website, http://www.metacase.com
7. Methods for Testing and Specification (MTS); Methodological approach to the use of object-orientation in the standards making process. ETSI EG 201 872 V1.2.1 (2001-2008), http://portal.etsi.org/mbs/Referenced%20Documents/eg_201_872.pdf
8. MOF™ Query / Views / Transformations (QVT) – Final Spec., http://www.omg.org/cgi-bin/doc?ptc/2005-11-01
9. Nentwich, C., Capra, L., Emmerich, W., Finkelstein, A.: xlinkit: a consistency checking and smart link generation service. ACM TOIT 2(2), 151–185 (2002)
10. Nuseibeh, B., Kramer, J., Finkelstein, A.: A Framework for Expressing the Relationships Between Multiple Views in Requirements Specifications. IEEE TSE 20(10), 760–773 (1994)
11. Salay, R., Mylopoulos, J., Easterbrook, S.: Managing Models through Macromodeling. In: Proc. ASE 2008, pp. 447–450 (2008)
12. Schauerhuber, A., Schwinger, W., Retschitzegger, W., Wimmer, M.: A Survey on Aspect-Oriented Modeling Approaches (2006), http://wit.tuwien.ac.at/people/schauerhuber
13. UML 2.0 Metamodel, http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-05.zip