Chapter IV

Matching Models of Different Abstraction Levels: A Refinement Equivalence Approach

Pnina Soffer, Haifa University, Israel

Iris Reinhartz-Berger, Haifia University, Israel

Armon Sturm, Ben-Gurion University of Negev, Israel

Abstract

This chapter deals with the reuse of models, which assists in constructing new models on the basis of existing knowledge. Some of the activities that support model reuse, such as model construction, retrieval, and validation, may involve matching models on the basis of semantic and structural similarity. However, matching for the purposes of retrieval and validation relates to models of different abstraction levels, hence structural similarity is dif-

ficult to assess. This chapter proposes the concept of refinement equivalence, which means that a detailed model is a refinement of an abstract model. It emphasizes the use of refinement equivalence for the purpose of validating a detailed application model against an abstract domain model in the context of a domain analysis approach called application-based domain modeling (ADOM). We discuss the structural characteristics of refinement operations in object-process methodology (OPM) models, and present an algorithm that detects refinement equivalence.

Introduction

The benefits of applying reuse at various stages of system design and implementation have been widely recognized. The reuse of software components has been addressed for over 40 years, and the idea has been extended to other and more abstract design artifacts, such as design models and specifications (Eckstein, Ahlbrecht, & Neumann, 2001; Kim, 2001; Reinhartz-Berger, Dori, & Katz, 2002; Zhang & Lyytinen, 2001), requirements models (Lai, Lee, & Yang, 1999; Massonet & Lamsweerde, 1997; Sutcliffe & Maiden, 1998), conceptual models (Pernici, Mecella, & Batini, 2000), enterprise models (Chen-Burger, Robertson, & Stader, 2000), method engineering models (Ralyte & Rolland, 2001), and others. When the reusable artifact is a model, the purpose of reuse is to assist in constructing a new model, either within the same domain, or within another domain by analogical reasoning.

Reuse is a major underlying motivation for the emergence of the domain engineering discipline. Domain engineering supports the notion of a domain, defined as a set of applications that use common concepts for describing requirements, problems, and capabilities. The purpose of domain engineering is to identify, model, construct, catalog, and disseminate a set of software or business artifacts that can be applied to existing and future systems in a particular domain. A subfield of domain engineering is domain analysis, which captures and specifies the basic elements of the domain and the relationships among these elements, representing this understanding in a useful way. Domain analysis is, therefore, a discipline that deals with creating reusable models of a domain and reusing these models for creating specific applications.

Reuse environments of models in general, and domain analysis environments in particular, should provide support to at least part of the following activities: (a) construction of reusable models and their storage, possibly in a repository, (b) retrieval of models (or parts of them) that meet the requirements of a developed application, (c) adaptation of the reusable models to the current application needs, and (d) validation of the adapted models. These activities may employ in some cases a model matching operation, which is the focus of this chapter.

In the context of domain analysis, two types of reusable models can be used. One is a generic domain model at a high level of abstraction that has to be specialized in adaptation to the current needs. The second type is a complete and detailed model, whose level of abstraction is the same as that of the application. It may be reused as it is, or modified to the specific needs, but without a change in its abstraction level.

The abstraction level of the reusable model affects the nature of the above discussed activities. First, reusable models of a high abstraction level are constructed by abstracting a collection of domain applications and analyzing their commonalities and variation points. Model matching may be employed for detecting the common aspects of the collection of application models that are being generalized.

Second, the role of a repository is of much importance for low-level reusable models since a large number of these may be stored, and each may include slightly different details. In contrast, high-level domain models specify common aspects of domain applications; hence, a large number of such models is not required.

Third, in general, the retrieval of a model can be either index based or model based. Index-based retrieval uses indices that characterize the models, while model-based retrieval matches an input model (query) given by the user with the models stored in the repository (Mili, Mili, & Mili, 1995). While index-based retrieval is relatively simple and quick, model-based retrieval is more accurate, relying on a higher volume of information rather than on a classification represented by indices. Retrieval of a high-level model is relatively simple due to the low number of models and the clear distinction between them, hence, index-based retrieval is appropriate. Retrieval of a low-level detailed model is more complicated since there may be a number of different models for a given domain, and retrieval seeks the one that matches partial information available about the particular current needs. Model-based retrieval, relying on all the information captured in a model, enables the selection of the model that best fits the user's query. It may use a preliminary partial model or some facts about the modeled domain as an input query, and retrieve a detailed model (or detailed models) that matches the input model.

Fourth, the adaptation of a high-level model to the current needs is an instantiation operation, yielding an application model that should match the domain model. This matching should be verified by a validation activity. The adaptation of a detailed model can be done by modification (which can be controlled through defined variation points) or by integration with other models. Validation in this case should follow the variation points and check that their specified constraints are not violated.

In summary, model matching can be used for the activities of constructing a reusable model, retrieving it, and validating an application model against the reusable one. When model matching is used for retrieval, the expected output is a similarity measure, while when it is used for construction or validation, the focus is on identifying specific matches and mismatches between the models.

This chapter deals with the assessment of structural similarity between two models of a different abstraction level. Soffer (2005) addressed this issue emphasizing its relevance for the retrieval of a detailed model. Here we address the scenario of validating an application model against a domain model. Addressing this scenario, we decided to rely on an existing domain analysis approach in order to relate to concrete details rather than taking a generic view, which might overlook the complexity of the task. The domain analysis approach we use is application-based domain modeling (ADOM; Reinhartz-Berger & Sturm, 2004; Sturm & Reinhartz-Berger, 2004), which facilitates the instantiation of an application model from a domain model and its validation against the domain model.

According to ADOM, when a domain model is instantiated to an application model, the entities in the resulting application model are classified as instances of the entities in the domain model. Furthermore, the application models may include multiple instances of domain-model entities, as well as additional entities. Hence, an application model can be considered as a refinement of the domain model. The validation of an application model against the relevant domain model employs model matching for verification purposes.

Due to the difficulty of assessing structural similarity with respect to models of different abstraction levels, we seek for refinement equivalence rather than structural similarity.

Refinement equivalence is a situation where a detailed (application) model can be perceived as a refinement of a more abstract (domain) model. To this end, we first need to establish an understanding of the nature of the refinement of models. The chapter discusses several types of refinement operations and indicates their structural characteristics, demonstrated by using the object-process methodology (OPM) as a modeling language. Understanding the consequences of model refinement is the basis for an algorithm that identifies structural equivalence of two models.

The remainder of the chapter is organized as follows. The next section briefly introduces the OPM modeling language and provides an overview of the ADOM approach. The following section discusses different refinement operations and illustrates their outcome in an OPM model. Then we describe a rule-based algorithm for identifying structural equivalence of OPM models in the context of validating an application model against a domain model. Following that, a review of related work is presented, and finally a concluding discussion.

Overview of Adom and Opm

This section starts with a brief introduction to OPM, then provides an overview of the ADOM approach in general and the ADOM-OPM dialect in particular.

Object-Process Methodology

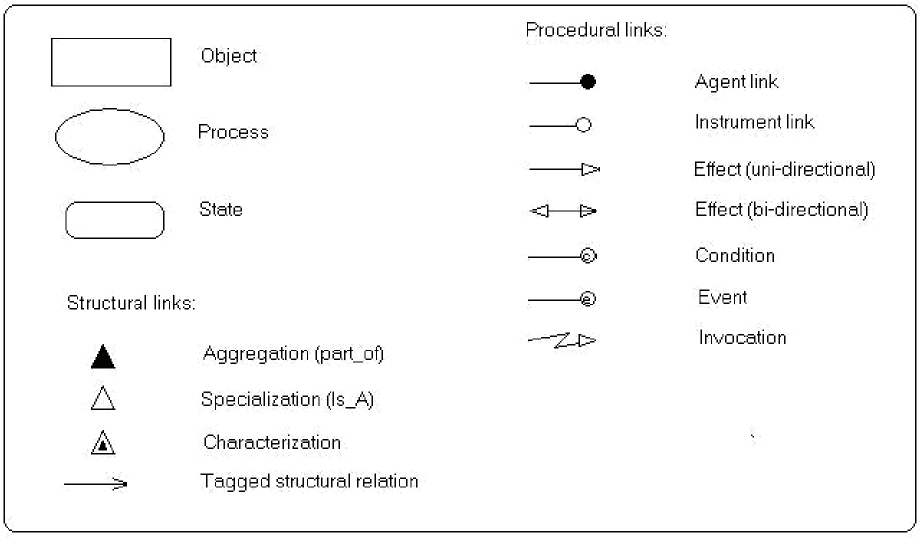
OPM, whose details are provided in Dori (2002), has been applied for various purposes at different development phases and tasks, such as conceptual requirements modeling (Soffer, Golany, Dori, & Wand, 2001), enterprise resource planning (ERP) system modeling (Soffer, Golany, & Dori, 2003), Web application design (Reinhartz-Berger et al., 2002), real-time systems specification (Peleg & Dori, 1999), algorithm specification (Wenyin & Dori, 1998), and others.

OPM incorporates two equally important classes of entities: objects and processes. While object-oriented methods encapsulate processes in objects, and business-process modeling methods represent activities detached from the objects they affect, OPM unifies the system structure and behavior into a single representation. It uses a single graphic tool, the object-process diagram (OPD) set, as a single view of all the system aspects, both structural and dynamic. Structure is expressed by objects connected with structural relations, such as characterization (e.g., between an object and its attributes), aggregation (part of), specialization (is-a), and general tagged structural relations (specifying any other relation named by a tag). The behavior of a system is represented by a set of procedural links, which can be classified into three classes of links: enabling links, transformation links, and triggering links. Enabling links (e.g., instrument links) relate an object to a process when the presence of the object is required for the process to occur, but this occurrence does not affect the object state or value. Transformation links (e.g., effect links) relate an object to a process that changes the object state or value (including its creation and destruction). Triggering links (e.g., event links) relate a transformation of an object (reflected in its state or value) to a process it triggers.

Similar to other modeling languages (e.g., DFD), OPM allows the refinement of a model by zooming into processes and unfolding the structure of objects to enable a top-down analysis. The resulting model is a hierarchical OPD set, which specifies all the aspects of a system at a spectrum of detail levels.

A part of OPM notation is given in Figure 1.

Figure 1. OPM notation



Application-Based Domain Modeling

ADOM is a generic domain analysis approach, facilitating the creation of domain models, their instantiation for creating application models, and the validation of the resulting application models. Being influenced by the classical framework for metamodeling presented in OMG (2006), the ADOM approach is based on a three-layered architecture: application, domain, and language. The application layer, which corresponds to the model layer (M1), consists of models of particular systems, including their structure and behavior. The language layer, which corresponds to the metamodel layer (M2), includes metamodels of modeling languages, such as UML (unified modeling language), OPM, and so forth. The intermediate domain layer consists of domain models. The ADOM approach enforces constraints among the different layers; in particular, the domain layer enforces constraints on the application layer, while the language layer enforces constraints on both the application and domain layers.

Including language metamodels as an upper layer, the ADOM approach is language independent. However, in practice, language-specific ADOM dialects must be used. Such dialects include ADOM-UML (Reinhartz-Berger & Sturm, 2004; Sturm & Reinhartz-Berger, 2004) and ADOM-OPM (Sturm, Dori, & Shehory, 2006), which is the dialect used in this chapter, too.

ADOM-OPM extends OPM with two new features: (a) a multiplicity indicator, which is attached to entities at the domain layer and constrains the number of entities of that kind that can appear in a particular application model in that domain, and (b) a role, which is a stereotype-like element emphasizing additional semantics for an OPM entity. Roles are used within application models, classifying entities as instances of domain-model entities. These two features establish the relationships between domain and application models. When an application model is created, its entities are assigned roles that correspond to the entities of the domain model, and the links among them are bound to preserve the corresponding link structure of the domain model. Additional entities can appear in the application model (without assigned roles) as long as they do not violate the domain constraints.

Validating an application model against the domain model entails checking that (a) the multiplicity constraints, specified by the multiplicity indicators, are not violated, that is, the number of entities in the application model that are classified with a certain role complies with the multiplicity indicator of the domain-model entity, and (b) the link structure of the application model is equivalent to the link structure of the domain model, considering their corresponding entities.

Refinement Equivalence

This section discusses different refinement operations and provides observations that characterize their structural impact in an OPM model in order to establish an in-depth understanding of model refinement in general. It should be noted that for the purposes of model retrieval and validation, matching may address models at different abstraction levels. The retrieval of a complete and detailed model requires its matching against a preliminary or partial input model, which is at a higher abstraction level than the retrieved model. Similarly, the validation of an application model against a domain model requires the matching of a low-level detailed (application) model against a high-level (domain) model. However, model matching as addressed in the literature so far has mainly dealt with models whose abstraction levels are identical. Two common similarity aspects (or measures) that are usually checked are entity similarity and structural similarity. Entity similarity assessment (also called semantic similarity) aims at identifying entities that are semantically similar in the models that are being matched. It may employ mechanisms of various accuracy and complexity levels, ranging from the identification of identical entity name and type (Soffer, Golany, & Dori, 2005), through thesaurus-based affinity measurement (Castano, De Antonellis, Fogini, & Pernici, 1998; Ralyte & Rolland, 2001), to concept hierarchy-based distance measurement (Chen-Burger et al., 2000; Lai et al., 1999). Structural similarity assessment, on the other hand, typically follows the links among the entities in one model and searches for parallels in the other model (Chen-Burger et al.; Massonet & Lamsweerde, 1997; Ralyte & Rolland; Sutcliffe & Maiden, 1998). This is sometimes termed neighboring-entities search. According to these two similarity assessments, two models are considered matching if they include the same entities and the same links to some extent. However, in case the models that should be matched are not at the same level of abstraction, then one cannot expect both models to have the same structure and set of links. Rather, while a high-level model specifies a set of entities and relationships among them, the low-level model includes the same entities (or their instances) along with other entities. Therefore, the model structure might be different, including all the other entities that exist in the detailed model and the links among them.

Since the instantiation of a domain model to an application model is a specific case of refinement, specific implications with respect to the application model validation shall be indicated. The most notable characteristic of this specific case is that entities of the application model are classified as instances of entities in the domain model. Hence, semantic similarity assessment techniques (e.g., Palopoli, Sacca, Terracina, & Ursino, 2003; Ralyte & Rolland, 2001) are not needed for matching these models.

We view an OPD as a directed and labeled graph whose nodes are entities (objects and processes) and edges are both structural and behavioral links among the entities. A refinement operation inserts new nodes and edges into an existing graph. These additional parts may replace existing edges, thus they may form paths between nodes that were directly linked in the original graph.

We shall examine and characterize the results of two types of refinement operations: refinement of structure and refinement of behavior. Specifically, we aim at identifying conditions under which a path can be considered equivalent to a given link.

Definition 1: Let A and B be entities, and let P be a path between A and B. P is equivalent to a link of type *l* if and only if a link *l* between A and B can be replaced by P through a refinement operation.

Notation: $P \cong l$.

Refinement of Structure

The paths established when structure is refined can replace both structural and procedural links that originally existed with the entity whose structure is being refined. We shall examine these two categories of links and characterize the path that replaces them in a refined model.

Structural links: When more structural details are revealed, a direct structural link in the abstract model can be replaced by a path including structural links and entities. This is demonstrated in the example shown in Figure 2, in which a characterization link between Warehouse and Number of Locations in the abstract model (a) appears as a path including both specialization and characterization links in the refined model (b). The refinement indicates that only a warehouse in which inventory locations are managed is characterized by the attribute Number of Locations.

In general, a path including a number of structural links can always be abstracted to a specific link type independently of the order in which these links appear.

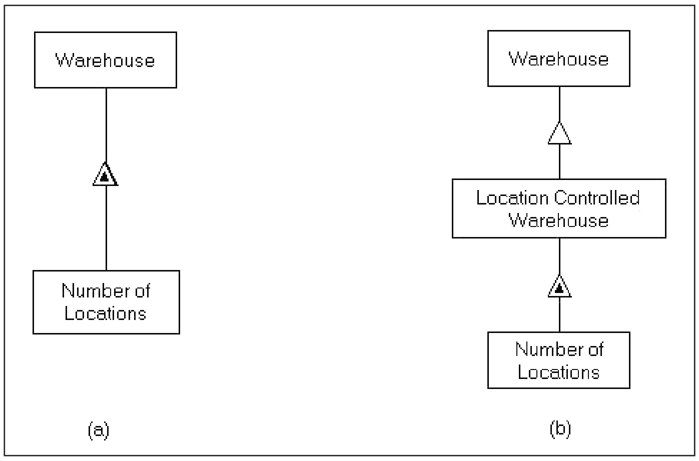
Definition 2: Let L be a set of link types. $l \in$ L is dominant with respect to L if and only if P $\cong$ l is true for every path P that includes l together with any $r \in$ L.

Notation: $D_L = l$.

Considering the example of Figure 2, it is clear that $D_{\{\text{Specialization, Characterization}\}}$ = Characterization as inheritance maintains characteristics along the hierarchy. Another example of this dominance is the attribute Number of Wheels, which characterizes a vehicle as well as a car, which is a specialization of a vehicle.

Observation 1: Let A and B be entities and P be a path from A to B. Let L be the set of link types included in P. If $D_L = l$, then P $\cong$ l.

Figure 2. Example of refinement involving a structural link

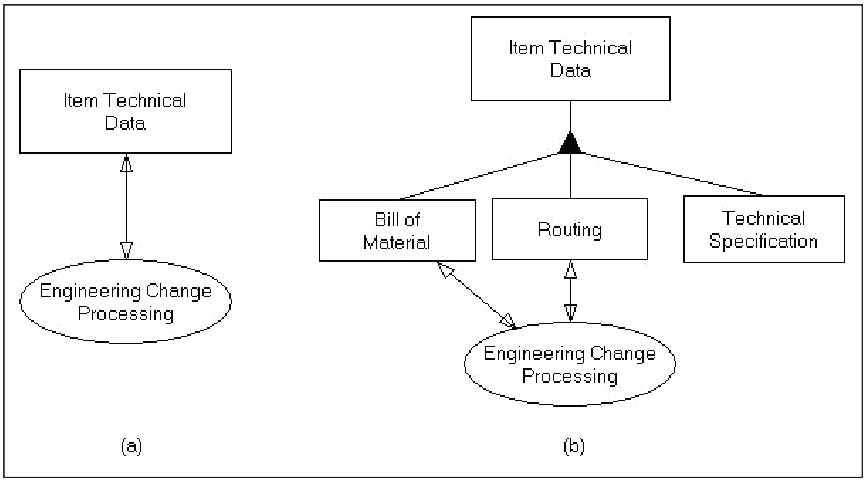


Observation 1 is a direct result of the definition of dominance with respect to a set of link types. It is useful for identifying equivalence regarding paths that include structural links since dominance can easily be established considering these link types, as in the above example. As another example of establishing dominance, consider the attribute Power that characterizes Engine. It characterizes the engine as well as the car of which the engine is part. Hence, characterization is dominant with respect to aggregation as well.

Procedural links: When a procedural link exists between an entity whose structure is being refined and another entity, the resulting path in the refined model consists of both structural and procedural links. As an example, Figure 3a shows an abstract model including an effect link between Engineering Change Processing and Item Technical Data. A refined model (Figure 33b) shows that Item Technical Data is composed of Bill of Material and Routing, which are affected by Engineering Change Processing. A third part of Item Technical Data, Technical Specification, remains intact as it is not even connected to the process.

In general, a refined model may specify the interaction of a process with attributes, parts, or specializations of an entity, whereas an abstract model simply specifies an interaction with the entity.

Figure 3. Example of a procedural link in structure refinement



Observation 2: Let A, B, and C be entities. Let P be a path from A to B so that A is linked to C and C is linked to B by a procedural link of type *l*. If the link from A to C is (Characterization) $\vee$ (Aggregation) $\vee$ (Specialization), then P $\cong$ *l*.

The proof of Observation 2 is by a simple demonstration that such refinement is possible (e.g., Figure 3). Note that Observation 2 does not imply the dominance of procedural links with respect to structural links since there may be paths that cannot be abstracted to a procedural link. For example, in Figure 3b, the path between Engineering Change Processing and Technical Specifications is not equivalent to the effect link included in it.

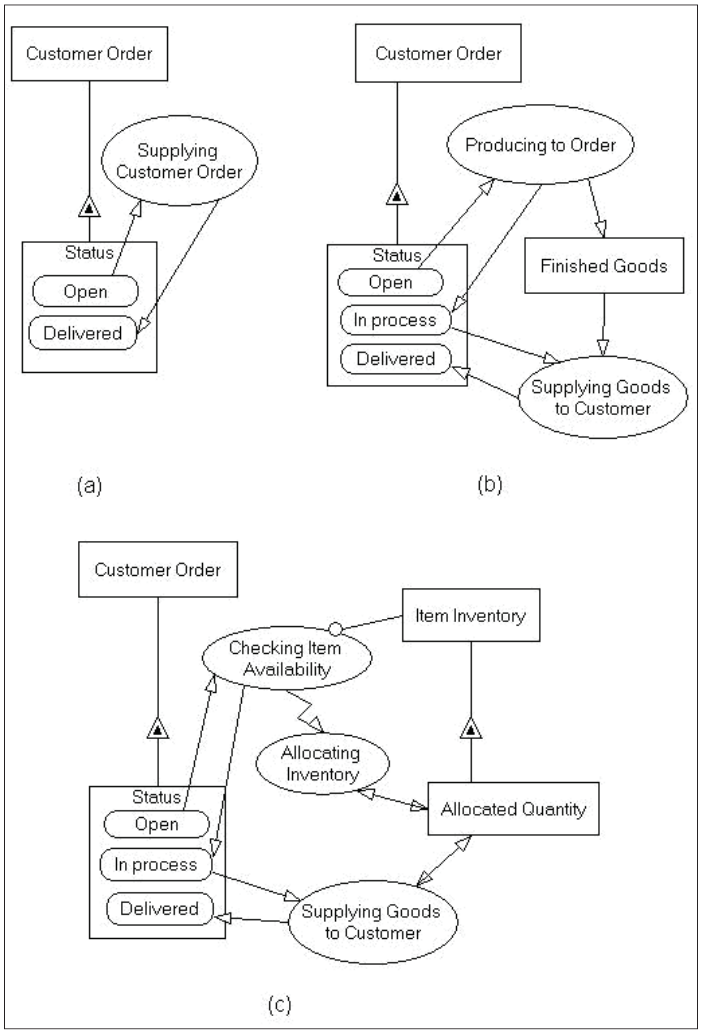
Refinement of Behavior

In general, the refinement of behavior is more difficult to identify than the refinement of structure for reasons that are explained below. Nevertheless, this difficulty is partly overcome when dealing with ADOM's classified entities. We shall first address the general case of refinement when no entity classification is used, and then explain how it becomes easier when ADOM-related models are addressed.

The behavior of a system or a domain is captured by processes. A process can be refined into a sequence of activities (subprocesses) that comprise it. Such a sequence is modeled as a path leading from an initial state (or input objects) to a final state (or output objects). The subprocesses in a refined process may interact with other objects besides the ones the higher level process interacts with, but these objects can be considered internal, meaning that in the abstract view of the process, the interaction is not observed. For example, consider two people who perform a task together. The interaction and allocation of work between them is internal in the sense that it is not of interest to others as long as the job is done.

The difficulty in identifying a refined process lies in the fact that unlike the refinement of structure, in which a link is replaced by a path, when a process is refined, an entity is replaced by a path (or several paths). Therefore, the initial and final states are the only reference points available. However, this information is not always sufficient for a conclusive identification of refinement equivalence. Consider a process of a high level of abstraction (e.g., building a house), having an initial state (existing plans, resources) and a final state (a house built). This process can be refined into many different processes, all having the same initial and final states and subset of interactions (stakeholders, authorities, building materials) as the abstract one. Yet, while being all equivalent to the abstract model, these refined processes are not equivalent to one another. As a detailed example, consider the abstract process of Supplying Customer Order in Figure 4a, which can be refined into the two different processes in Figure 4b and c. These two refined processes have identical initial and final states, Open Customer Order and Delivered Customer Order, respectively, as does the abstract process. However, while

Figure 4. An abstract model and two possible refinements



both processes can be considered equivalent to the abstract model, they are not equivalent to one another (in their internal division into subprocesses, additional inputs and outputs, etc.). It is therefore easier to formulate a necessary condition rather than a necessary and sufficient condition for refinement equivalence of processes.

Observation 3: Let m1 be a model portion in which process A transforms an initial state s_1 into a final state s_2. Let E1 be the set of entities directly linked to A in m1. Let m2 be a model portion that refines m1. Then m2 consists of a path P and a set E2 of entities that are directly linked to the entities of P so that P is from an initial state s_1 to a final state s_2 and E1 $\subseteq$ E2.

Note that the initial and final states are not necessarily explicitly represented in an abstract model, in which case the inputs and outputs of the process should be considered in a similar manner to the states.

Observation 3 provides a necessary condition that might not be sufficient for the identification of equivalence. When the lower level model is a result of an instantiation operation of a domain model, its entities are assigned roles that correspond to domain-model entities. In other cases, we need a way to relate the subprocesses in a refined model to a process in the abstract model. For that purpose, we note that it is likely that at least one of the subprocesses in a refined model bears a name that can be identified as similar to the general process' name as appears in the abstract model. Such resemblance can be detected by existing affinity detection techniques, which are not the focus of this chapter. This can be explained by a tendency to name the process in the abstract model after the main activity that constitutes the essence of the process. In fact, such tendency is not unique to process models. Suggesting a semiautomatic procedure for abstracting a database schema, Castano et al. (1998) refer to a "representative" element of the detailed schema, whose name should be given to the generalizing element in the abstracted schema. When refining an abstract process to lower abstraction levels, details of other activities are revealed. In the example of Figure 4, Supplying Goods to Customer can be identified as similar to Supplying Customer Order.

In such cases, we expect the refined model to include a path from the initial state to the similarly named process (or, in ADOM-based models, to the process whose role corresponds to the process in the domain model) and to the final state. A path is also expected to relate the process to other entities that interact with it in the higher-abstraction-level model. If such paths exist in a detailed model, and if they are equivalent to the links of the abstract model, than the detailed model can be considered as a refinement of the abstract one. Observation 4 indicates a condition under which a path that may include a number of processes and objects or states is considered as equivalent to a specific type of procedural link.

Observation 4: Let A be an object or a state of an object, B be a process, and P be a path between A and B. Let *l* be the procedural link by which A is related to P, then $P \cong l$.

Note that the direction of the path can be from the object to the process or backward, depending on the specific links involved.

Observation 4 can be justified when abstracting the entire path (processes and objects) to a process (named after its representative activity, B). The link that determines the nature of the interaction between this abstracted process and the object is the link relating the object to the path. In the example of Figure 4b and c, the path from the state Open of Customer Order Status to Supplying Goods to Customer is equivalent to the direct link from Open to Supplying Customer Order in 4a.

Observation 4 provides a sufficient condition for identifying refinement equivalence. However, this condition, though sufficient, is not a necessary one. It is based on the assumption, discussed above, that the abstract process is named after its main activity. This assumption is not necessarily always true. For example, a production process can be refined into processes of cutting, drilling, milling, and so forth. In such cases, the path between the initial and final states in the abstract model has to be matched against the path in the detailed model. That path can be decomposed into individual links for this purpose. As explained above, when application-model processes bear roles that classify them as corresponding to domain-model processes, the naming difficulty does not exist. Thus, Observation 4 can conclusively identify refinement equivalence.

Tracking Refinement Equivalence

The previous section identified conditions that enable the detection of refinement equivalence. When an application model is validated against a domain model, the following steps can be taken: (a) The names of the entities that have a role assigned to them in the application model are replaced by their roles, (b) satisfaction of the multiplicity constraints specified in the domain model is determined, and (c) the links among the entities in the domain model are matched by corresponding links in the application model. In case such corresponding link is not found, an equivalent path is searched for between the source entity and the destination entity of the link.

This section describes a rule-based algorithm that identifies refinement-equivalent paths with respect to a given link type. The algorithm is basically a path-searching algorithm applying rules, which follow the discussion and observations of the previous section, to assure that the path found is indeed equivalent to the link being matched.

Searching for an Equivalent Path

Consider a pair of OPDs <A, D>, where A is the application model and D is the domain model being matched. Assume A is searched for a path between two entities that are directly related in D. The steps of the search shall first be informally described, and then specified formally. Each step of the search partitions A into two sets of entities: One is the set of entities to which a path from the source entity is already established, and the other is the set of entities that are not yet explored. Starting from the source entity, each step follows a link and moves one entity from the unexplored set to the set of entities that are connected to the source. The choice of link to be followed is based on the search rules, whose details are given below. The steps repeat until a direct link is found from the connected set of entities to the destination entity, or until all the links have been exhausted and it is clear that the searched-for path does not exist. The algorithm seeks to establish the existence of a path that is not necessarily the shortest path, hence no backtracking is performed and the number of steps is at most the number of entities in A minus one.

The formal specification of the search applies the following notation.

s: the source entity of the link in D whose equivalent path is being searched for in A:

d: the destination entity of the link in D whose equivalent path is being searched for in A:

- $L_M(e_1, e_2)$: Let e_1 and e_2 be entities; then $L_M(e_1, e_2)$ is a Boolean variable whose TRUE value indicates the existence of a direct link from e_1 to e_2 in model M (M is either the application model A or the domain model D).
- $\text{Link}_M(S_1, S_2)$: Let S_1 and S_2 be nonoverlapping sets of entities in model M; then $\text{Link}_M(S_1, S_2)$ is an indicator expressing the existence of a direct link from an entity in S_1 to an entity in S_2.

$$\text{Link}_M(S_1, S_2) = \begin{cases} 1 & \text{if } \exists\, e_1 \in S_1,\, e_2 \in S_2, \text{ such that } L_M(e_1, e_2) = \text{TRUE} \\ 0 & \text{otherwise} \end{cases}$$

- S_M: the set of entities in model M
- $C_i(M, s)$: the set of entities in model M to which a path from *s* has been found until the *i*th step of the search
- $U_i(M, s)$: The set of entities in model M whose relationship with *s* has not yet been investigated by the *i*th step of the search

In the context of the application model, $C_i(A, s)$ and $U_i(A, s)$ partition S_A so that at each step *i* of the search, $S_A = C_i(A, s) + U_i(A, s) + \{d\}$. In other words, each entity in A belongs either to the set of entities that have already been established as linked to *s* (including *s* itself) or to the set of entities whose relationship with *s* is unknown yet, or to the set that holds *d* only.

Lemma: Let an application model A be searched for a path from *s* to *d* at the *i*th step of the search. A path from *s* to *d* exists only if $\text{Max}\,[\text{Link}_A(C_i(A, s), \{d\}), \text{Link}_A(C_i(A, s), U_i(A, s)) * \text{Link}_A(U_i(A, s), \{d\})] = 1$.

Proof: Assume a path exists. It can lead from $C_i(A, s)$ directly to *d*, then $\text{Link}_A(C_i(A, s), \{d\}) = 1$. Otherwise, it leads from $C_i(A, s)$ to some entity $e \in U_i(A, s)$ and from *e* to *d*. Then $\text{Link}_A(C_i(A, s), U_i(A, s)) = 1$ and $\text{Link}_A(U_i(A, s), \{d\}) = 1$.

Assume a path does not exist. Then $\text{Link}_A(C_i(A,s),\{d\}) = 0$ and the following are true:

1. If $\text{Link}_A(C_i(A, s), U_i(A, s)) = 1$, then $\text{Link}_A(U_i(A, s),\{d\}) = 0$.
2. If $\text{Link}_A(U_i(A, s),\{d\}) = 1$, then $\text{Link}_A(C_i(A, s), U_i(A, s)) = 0$.

Note that the above lemma is one sided; that is, it does not imply that if $\text{Max}\,[\text{Link}_A\,(C_i(A, s),\ \{d\}),\ \text{Link}_A\,(C_i(A, s),\ U_i(A, s))*\text{Link}_A\,(U_i(A, s),\{d\})] = 1$, then a path exists. Rather, this is a necessary condition for the existence of such a path.

The initial state of the search is $C_0(A, s) = s$, $U_0(A, s) = S_A - \{s, d\}$. At each step, if the condition specified in the lemma is satisfied, one entity is moved from $U_i(A, s)$ to $C_i(A, s)$ by following a link, implying that a relation of this entity to *s* is established. The steps repeat until either a path is found, that is, $\text{Link}_A(C_i(A, s),\{d\}) = 1$, or the condition of the lemma is not satisfied; that is, the searched-for path does not exist. The search rules ensure that the found path is equivalent to the link being searched for.

Figure 5 specifies the equivalence path search algorithm. This algorithm employs the following operations.

Fold_Structure (entity): A folding operation of structural relations in OPM is an abstraction operation in which a detailed OPD portion, including structural relations such as characterization, aggregation, and specialization, is

Figure 5. Equivalent path search algorithm

```
Current = s
Fold_Structure (d)
Exclude_Links
Do while (Link_A(C_i(A, s),U_i(A, s))*Link_A(U_i(A, s),{d}) = 1)
 AND (Link_A(C_i(A, s),{d}) <> 1)
      If Link_Type is procedural then Fold_Structure(Current)
      Exclude_Links
      Verify_Equivalence
      If Link_Type is structural then Compute_Cardinality
      Select_Entity
End Do
If    (Link_A(C_i(A, s),{d}) = 1) AND (Path_Cardinality =
      Link_Cardinality) AND (Condition) then Path_Found =
      TRUE
Else  Path_Found = FALSE
```

replaced by an OPD portion of a higher abstraction level. The entities that provide the structure details of the entity being folded (which is the parameter of this operation) are not shown in the abstracted OPD. Other entities, which are originally related to the structure details, are related directly to the folded entity.

This operation is employed only when the link, whose equivalent path is searched for, is a procedural link. Its role is to replace paths created through refinement of structure by their equivalent procedural links on the basis of Observation 2.

Exclude_Links: This operation excludes links that cannot be included in the path. Links can be excluded from the search for three reasons. The first reason is that they cannot be part of the path according to the search rules, in which case they are excluded at the beginning of the search. The second reason is that their direction is opposite of the search direction. At every step of the search, the unidirectional links from the entities of $U_i(A, s)$ to the entities of $C_i(A, s)$ are excluded from the search. The last reason applies to inheritance (is-a) links, which may be included in a path in both directions, from the special to the general as well as the other way. When going up the relation, the links to other specializations of the general entity cannot be included in the path.

Select_Entity: At every step of the search, all the links from the entities of $C_i(A, s)$ to the entities of $U_i(A, s)$ are arranged according to priorities defined by the search rules. The first link according to this order is selected and the entity it relates to is moved to $C_i(A, s)$ and becomes the Current entity.

Verify_Equivalence: The search rules specify for a given link the link type that must be included in the path and its required position (at the source, at the destination, or anywhere in the path). If the required position is at the source or destination of the path, then all the links from *s* or to *d* (respectively), which are not of the mandatory type (i.e., are not of the type that must be in that position in the path in order to preserve the nature of the interaction), are excluded from the search at the first step by the Exclude_Links operation. As a result, a Boolean variable Condition is assigned a TRUE value. If the required position is anywhere in the path, the Condition is verified by a set of indicators EC_e, defined next.

Let *e* be an entity in $C_i(A, s)$; then $EC_e = 1$ if and only if a link of the mandatory type is in the path from *s* to *e*.

Starting at $EC_s = 0$, and letting *e* be moved from $U_i(A, s)$ to $C_i(A, s)$ through a link of type *t* from an entity $a \in C_i(A, s)$, then:

$$EC_e = \begin{cases} 1 & \text{if } (EC_a = 1) \text{ or } (t \text{ is of mandatory type}) \\ 0 & \text{otherwise} \end{cases}$$

When a path is found, $EC_d = 1$ implies that it includes at least one link of the mandatory type (according to the conditions specified by the search rules), in which case Condition = TRUE.

Compute_Cardinality: This operation is performed only when structural relations are searched for. The cardinality of a link is defined as <SL, SU, DL, DU>, where SL is the source lower participation constraint, SU is the source upper participation constraint, DL is the destination lower participation constraint, and DU is the destination upper participation constraint.

Let *e* be an entity in $C_i(A, s)$; then the aggregated cardinality of the path from *s* to *e* is denoted by $<SL_e, SU_e, DL_e, DU_e>$, where *s* holds <1, 1, 1, 1>.

Let *a* be moved to $C_i(A, s)$ through a link whose cardinality is <SL, SU, DL, DU> from entity $e \in C_i(A, s)$, then $SL_a = SL_e * SL$, $SU_a = SU_e * SU$, $DL_a = DL_e * DL$, $DU_a = DU_e * DU$.

For example, assume an item is supplied by zero to three suppliers, a supplier has one to two contact persons, and a supplier can supply one or more (1...*m*) items. The aggregated cardinality of the path between an item and a purchasing contact person is <1, *m*, 0, 6>.

Search Rules

The search for an equivalent path employs rules of two types: link selection rules and equivalence conditions. Both rule types are defined for each type of link in OPM. A link selection rule defines the types of links that can be included in an equivalent path and provides searching priorities for the search algorithm. It is applied by the Exclude_Links operation, which excludes all the irrelevant links from the search, and by the Select_Entity operation, which uses the priorities given for selecting the entity to be moved from $U_i(A, s)$ to $C_i(A, s)$. An equivalence condition defines conditions for a path to be equivalent to a link of a certain type. It is employed by the Verify_Equivalence and Exclude_Links operations. Conditions may specify link types that must be included in a path and their required positions that can be at the source of the path, at its destination, or at any point in the path.

A link selection rule is of the following form:

Link Selection (Link Type): {Set of Types}

Link Type is the type of link to which the path is to be equivalent, while Set of Types is an ordered set of link types. All the link types in the set can be included in a path, which is equivalent to Link Type. Their order in the set determines the priority in which the search algorithm considers links in the examined OPD when searching for a path.

On the basis of Observation 1, the Set of Types specified for structural link types satisfies $D_S = l$, where l is the Link Type and S is the Set of Types.

For example, the link selection rule for aggregation, which is a structural link that denotes a whole-part relation and is dominant with respect to specialization (is-a) relations only, is:

Link Selection (Aggregation): {Aggregation, Specialization}

For procedural link types, the Set of Types is defined on the basis of Observation 4. According to this observation, the link that determines the equivalence is the one related to the source or destination object without restrictions on the types of links in the path. Hence, the Set of Types for procedural link types includes all the types of links in OPM.

The order of the types in the Set of Types always sets the relevant Link Type as the first priority for the search algorithm. For procedural link types, it lets the algorithm prefer procedural links over structural ones.

An equivalence condition is of the following form:

Equivalence Condition (Link Type): Mandatory Type must be located at Required Position in the path

Mandatory Type is a link type that is necessarily included in the path in order to preserve the nature of the interaction, where Required Position is the exact position where it should appear (the possible values are at Source Position, at Destination Position, and Anywhere).

Mandatory Type is, with one exception, the Link Type itself. The exception is an invocation link, which represents the triggering of a process by the completion of another process. This can also be modeled as an event created by the first process and triggering the second one. In this case, an event link replaces the invocation link.

For structural link types, the Required Position is Anywhere, since the link selection rules ensure the dominance of the specific link type with respect to the links in the path. Hence, their position in the path is of no importance as long as they are present. For procedural link types, the Required Position, according to Observation 4, depends on the link type. Links whose direction is from the object to the process (e.g., instrument links) require the Mandatory Type at the source of the path, while links that lead from the process to the object (e.g., result links, which are unidirectional effect links) require the Mandatory Type at the destination of the path.

For example, below are the equivalence conditions for aggregation links (i.e., structural links that denote whole-part relations) and instrument links (i.e., procedural links that denote input objects that are not changed by the process; these links are directed from the object to the process).

Equivalence Condition (Aggregation): Aggregation must be located Anywhere in the path

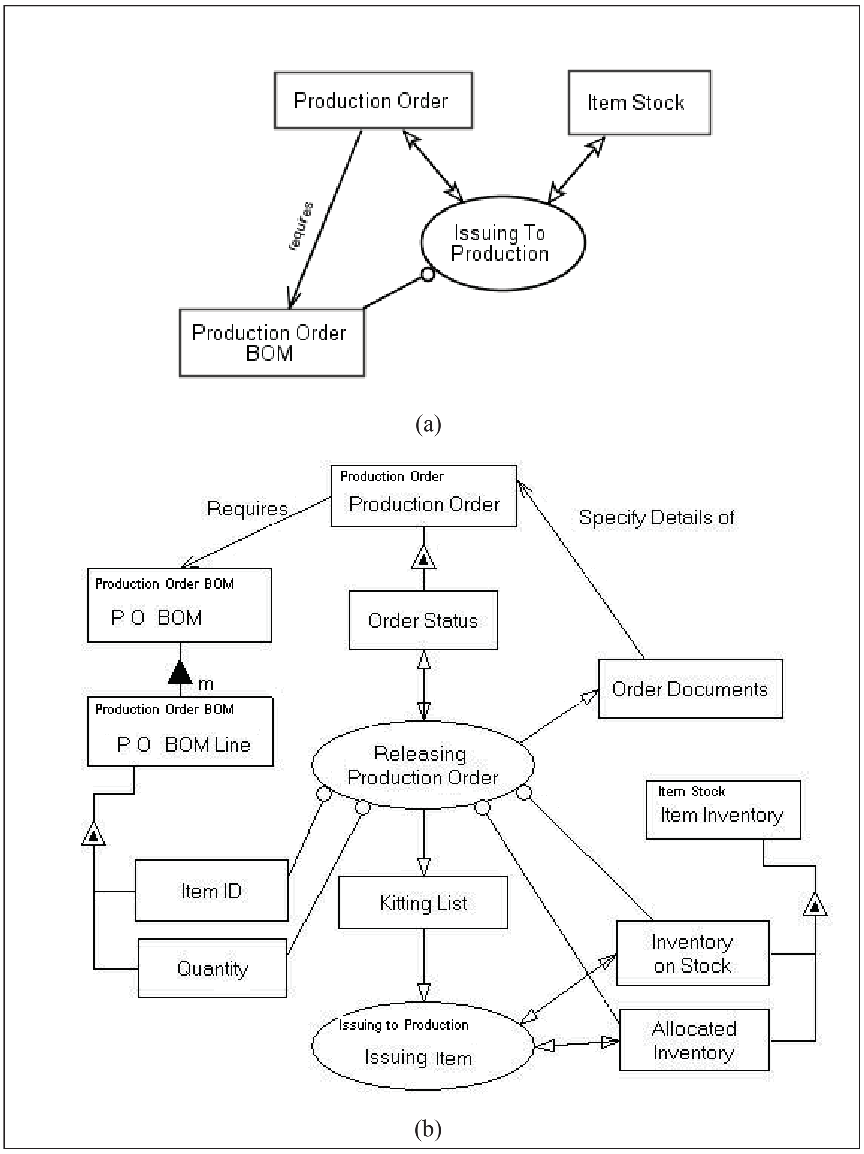
Equivalence Condition (Instrument Link): Instrument Link must be located at Source Position in the path

As explained above, the two types of rules are based on Observation 1, which addresses structural links when structure is refined, and on Observation 4, which addresses procedural links when behavior is refined. Observation 2, which addresses procedural links when structure is refined, is not applied as part of the rule base, but is taken into account by the Fold_Structure operation performed by the search algorithm.

Exemplifying the Equivalent-Path Search Algorithm

The algorithm steps are illustrated by an example given in Figure 6: Figure 6a is part of a domain model, while Figure 6b is an application model that should be matched against the domain model. The domain model specifies the main concepts as well as their multiplicity constraints. For example, Pro-

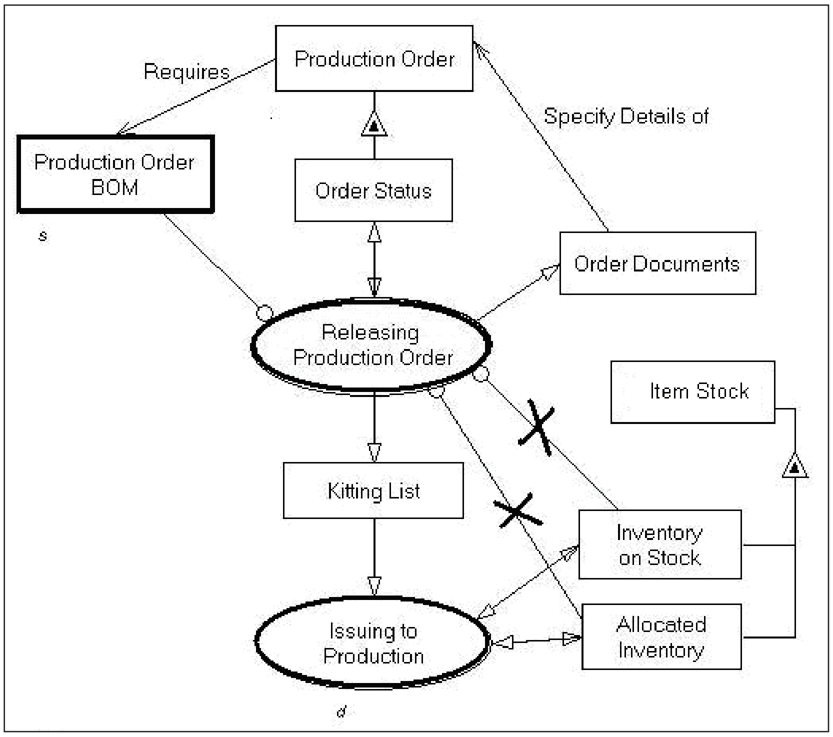
Figure 6. Refinement Equivalence Example



duction Order and Issuing to Production are indicated as mandatory single entities (the 1..1 at the right lower corner of the entities), meaning they must be instantiated exactly once in any application model of the domain, while Production Order BOM and Item Stock are indicated as mandatory multiple entities (the 1..m at the right lower corner of the entities), meaning they must appear at least once in any application model in the domain. Correspondingly, some of the application-model entities have roles (at their left upper corner) that relate them to the domain-model entities, while others are additional application-specific entities. Note that the number of role-classified entities in the application model is consistent with the multiplicity indicators specified in the domain model for each role.

None of the procedural links specified in Figure 6a appears as a direct link in Figure 6b. Nevertheless, they are all matched by equivalent paths in the application model. The domain model specifies that a process of Issuing to Production affects the Production Order and the Item Stock, and uses the Production Order BOM (which specifies the required materials). In the application model, a process of Releasing Production Order precedes Issuing

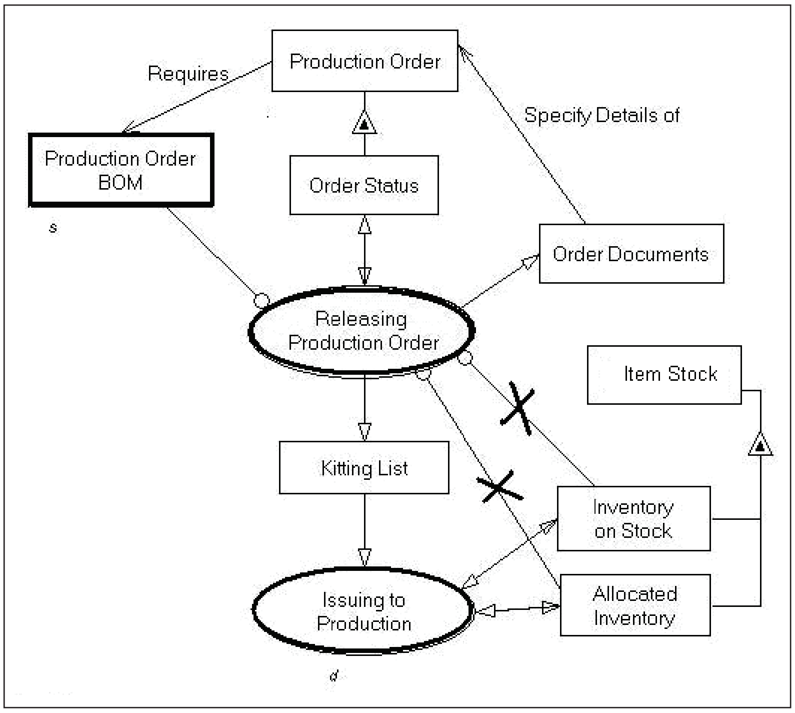
Figure 7. Search Algorithm 1st Step



Item (whose role is Issuing to Production), using Item Inventory (whose role is Item Stock) information as well as the item ID and quantities specified by PO BOM Lines, which are parts of the PO BOM (both have a role of Production Order BOM). The process of Releasing Production Order creates Order Documents (a set of documents, specifying details of the production order, to be used in the production process) and a Kitting List, which is a list of items to be prepared in kits before they can be issued to production. The Issuing Process uses the Kitting List and affects the Item Inventory.

We shall follow the steps of the search algorithm for tracking an equivalent path that matches the instrument link from Production Order BOM to Issuing to Production in the domain model of Figure 6(a) in the application model of Figure 6(b). Two entities in that model are classified with the role of Production Order BOM. However, since one is part of the other, we will use the whole as the source of the searched path, as illustrated in Figures 7 to 10. The search in Figures 7 to 10 is performed after the names of the entities have been replaced by their roles (whenever they have one), according to the first validation step.

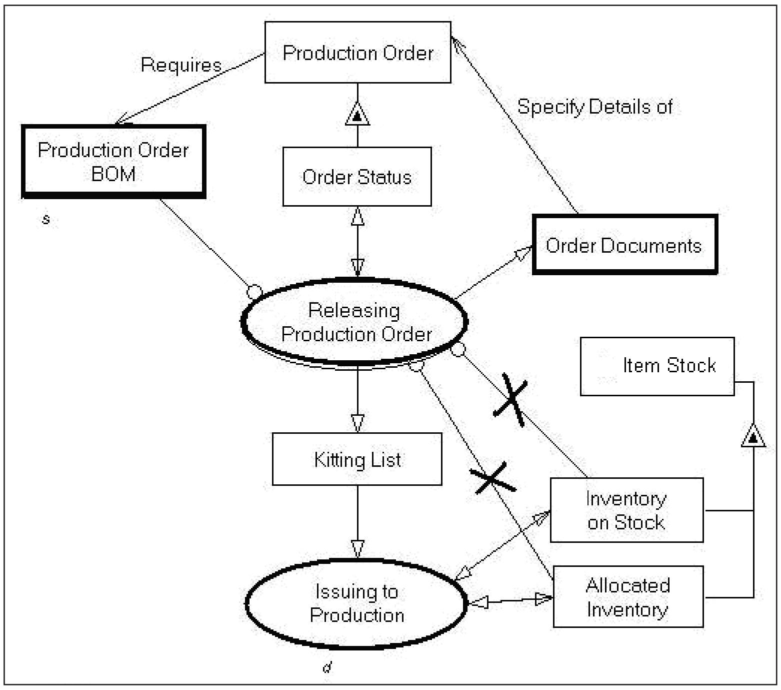
Figure 8. Search Algorithm 2nd Step



Step 1 (see Figure 7)**:** C_0(A, s) includes the source entity, Production Order BOM (highlighted). The source entity is the Current entity, and a Fold_Structure(Current) operation is performed. As a result, its structural details are not seen, and the instrument links originally related to these details are now related directly to Production Order BOM itself. U_0(A, s) includes all the other entities in the model, except the source entity, Production Order BOM, and the destination entity, Issuing to Production (highlighted). C_0(A, s) is linked to U_0(A, s), which is linked to the destination entity, thus the condition of the lemma is satisfied.

Step 2 (see Figure 8)**:** Following the instrument link, C_1(A, s) includes Releasing Production Order in addition to Production Order BOM. Note that the equivalence condition of an instrument link requires that the first move should be through an instrument link, and it is satisfied. Two instrument links that lead to Releasing Production Order are excluded from the search by the Exclude_Links operation since their direction is opposite of the path direction. C_1(A, s) is still linked to U_1(A, s), which is linked to the destination entity.

Figure 9. Search Algorithm 3rd Step



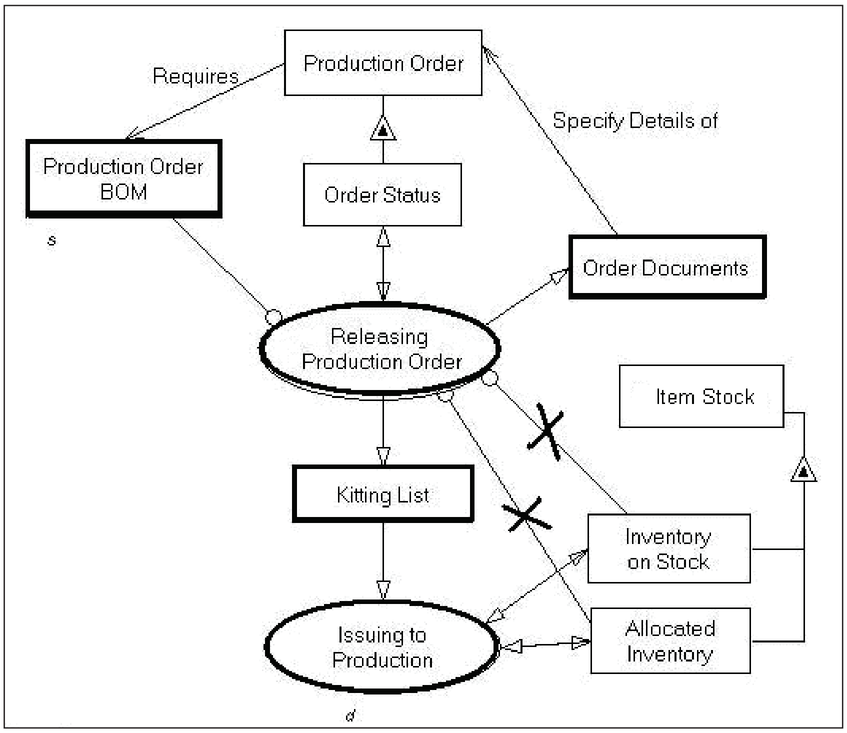
Step 3 (see Figure 9)**:** Following the effect link, Order Documents is included in $C_2(A, s)$. Note that this is a random choice from the three effect links that lead from Releasing Production Order. $C_2(A, s)$ is still linked to $U_2(A, s)$, which is linked to the destination entity.

Step 4 (see Figure 10)**:** Following the next effect link from $C_2(A, s)$, Kitting List is now added to $C_3(A, s)$. $C_3(A, s)$ is now linked to the destination entity, thus establishing a path that meets the equivalence conditions, and is therefore equivalent to the direct link of the domain model.

Note that Step 3 is actually redundant and could be avoided by a different choice of link. Nevertheless, by addressing all the links of the $C_i(A, s)$ set, the algorithm is able to simply look one step ahead at a time and avoid a recursive backtracking.

The complexity of the search algorithm is $O(|S_A|)$, where $|S_A|$ is the number of entities in A. The search is performed for each link in D when the models

Figure 10. Search Algorithm 4th Step



are matched. Hence, the complexity of the matching is $O(|S_D|^2 * |S_A|)$. Note that $|S_D|$ is expected to be significantly smaller than $|S_A|$.

Related Work

Model similarity has been addressed by several disciplines. The ones that are relevant to this work are the disciplines of reuse and schema analysis and matching. The difference in abstraction level between matched models has not, to the best of our knowledge, been explicitly addressed in the reuse literature. Kim (2001) presents an object-oriented model reuse application in which an initial model, including classes and nonspecific links, serves as a basis for retrieving an existing complete model. The retrieved model is then modified and adapted to the current needs using modification rules, whose details are not presented. No details are available about how a complete model is retrieved and evaluated, how this retrieval considers the nonspecific links of the input model, and how structurally different from each other the models retrieved are.

Structural similarity plays an important role in the works that deal with analogical reasoning (Massonet & Lamsweerde, 1997; Sutcliffe & Maiden, 1998), where models designed for a certain domain are applied to other domains by analogy. The retrieval is based on structural properties of the model and on semantics, which is based on generalizations. In Sutcliffe and Maiden, the models to be retrieved include a number of layers, each dealing with different information types, going from an abstract layer to a detailed one. The matching with the input information interactively follows these layers of specific information types, and the user is required by the system to provide enough information to discriminate between existing models. Hence, the structural similarity deals with models of the same abstraction level. In Massonet and Lamsweerde, while the entities of an input model are generalized to a higher level in an is-a hierarchy, their link structure is expected to remain the same and serves as a basis for structural similarity assessment.

Other works that apply reuse for method engineering (Ralyte & Rolland, 2001) and for enterprise modeling (Chen-Burger et al., 2000) use simple structural similarity assessment along with semantic similarity based on affinity (Ralyte & Rolland) or on a generalization hierarchy (Chen-Burger et al.). The model used by Ralyte and Rolland includes multiple abstraction levels. Hence, there might be a match between the abstraction level of a query model and one of the levels of the reusable models, but it is not explicitly addressed and verified. None of the above reviewed works relates to model matching for validation purposes as proposed in this chapter.

Schema-matching literature focuses on the semantic mapping of one schema to the other. While semantic similarity in the reuse literature is mostly affinity based, or in some cases relies on is-a hierarchies, semantic matching in the schema-matching literature sometimes combines the affinity of terms with structural considerations. Schema matching maps elements of one schema to elements of another schema rather than compute similarity measures between the two schemas. Hence, each pair of elements is thoroughly examined and structural aspects, such as attributes and is-a relations, are taken into account (Madhavan, Bernstein, & Rahm, 2001; Rahm & Bernstein, 2001; Rodriguez & Egenhofer, 1999). In some cases, paths are sought where direct links do not exist (Palopoli et al., 2003). Nevertheless, dealing with schemas means dealing with a low level of abstraction. Some schemas may be more detailed than others, and the techniques suggested are aimed at overcoming such differences rather than at dealing with models that are basically at different abstraction levels. Typical to this situation is the use of the term "structural equivalence" of schemas (Algaic & Bernstein, 2001), which relates to a consistent mapping of schema elements from one schema to another and backward in the lowest abstraction level. It is defined as structural as opposed to semantic equivalence, which relates to integrity constraints as well.

The similarity assessment of entities, presented by Rodriguez and Egenhofer (1999), relates to parts, functions, and attributes of two entity classes. Their similarity measure uses a function that provides asymmetric values for entity classes that belong to different levels of abstraction. While addressing single entity classes, they take contextual information into account for the similarity measurement. However, context information of an entity cannot be considered equivalent to a view of the entity as a part of a model, including relationships with other entities.

A more holistic view of schema analysis, including a variety of techniques for schema abstraction, matching, and reuse, is presented in Castano et al. (1998). Schema abstraction is an operation in the opposite direction compared to our discussion of refinement operations. The ERD schemas addressed limit the discussion to structural links only, without addressing the representation of behavior. Yet, their abstraction operation is consistent with our opposite-direction refinement, and applying the algorithm presented here to their examples of detailed and abstract schema yields a match. A number of schema similarity measures are presented there, dealing mainly with semantics and, to a limited extent, model structure, particularly with attributes. Interestingly enough, their fuzzy similarity measure is asymmetric and may indicate that schema *a* matches schema *b* to a higher extent than in the other direction. This is explained as being a result of differences in the abstraction level between the two schemas.

Our approach can be classified according to the extensive classification of schema-matching approaches presented by Shvaiko and Euzenat (2005). It is a structure-level approach (computes mapping elements by analyzing how entities appear together in a structure), syntactic (interprets the input in function of its sole structure following some clearly stated algorithm), and graph based (addressing children, leaves, and relations of entities). However, this classification does not relate to differences in the abstraction level of the matched schemas, and this issue is not addressed by any of the works surveyed there.

In summary, the main contribution of this chapter as compared to related earlier model-matching works is in explicitly addressing models of different abstraction levels, representing both the structure and behavior of a domain of applications.

Conclusion

The reuse of models requires activities that in many cases employ model matching. In this chapter, we stressed that differences in the abstraction level are likely to exist between models, specifically in the retrieval and validation activities, and therefore refinement equivalence is a better measure than structural similarity. Refinement equivalence is identified when a detailed model can be considered a refinement of a model of a higher abstraction level. In this chapter we discussed the notion of refinement equivalence as an enabler of validating a detailed application model against an abstract domain model in the context of the ADOM approach for domain analysis.

The discussion of refinement operations and the observations that characterize their impact on model structure, as well as the search algorithm, address OPM models. However, ADOM is language independent and can be used with other modeling languages as well. Other modeling languages are different mainly in the separation of structural and behavioral aspects of the modeled domain (and applications). Yet, the notion of refinement equivalence is of relevance to models independently of the modeling language. Some of the observations made in this chapter can easily be generalized and become applicable to other modeling languages. For example, Observation 1, which deals with the dominance of structural relations in a path, is not specific to OPM only. Hence, when dealing with models that capture structural information only (e.g., ERD, UML class diagrams), the algorithm can be applied using the search rules that relate to structural links only, omitting the Fold_Structure operation. Regarding the behavioral aspects, generalization is less straightforward. In multiview modeling languages, such as UML, consistency among views might also need consideration.

An equivalent-path search algorithm is, naturally, language specific, and apparently needs to be developed for each modeling language. However, the algorithm presented here is mainly a path-searching algorithm, while specific features of the OPM links are captured by the equivalence rules. Hence, the main body of the algorithm might be applicable to other modeling languages while the unique features of the language might affect mainly the equivalence rules.

The search algorithm that enables refinement-equivalence identification has been implemented in a reuse application that supports business-process alignment and gap analysis in the implementation of ERP systems (Soffer et al., 2005). The application matches abstract enterprise requirement models with a detailed model of the ERP system, and retrieves the parts that match the requirements.

Future research should extend the refinement-equivalence concept and apply it to other modeling languages that serve in reuse applications, such as UML.

References

Algaic, S., & Bernstein, P. A. (2001). A model theory for generic schema management. In *Proceedings of DBPL* (LNCS 2397, pp. 228-246). Berlin, Germany: Springer-Verlag.

Castano, S., De Antonellis, V., Fogini, M. G., & Pernici, B. (1998). Conceptual schema analysis: Techniques and applications. *ACM Transactions on Database System, 23*(3), 286-333.

Chen-Burger, Y. H., Robertson, D., & Stader, J. (2000). A case-based reasoning framework for enterprise model building, sharing and reusing. In *Proceedings of the ECAI Knowledge Management and Organization Memories Workshop*, Berlin, Germany.

Dori, D. (2002). *Object process methodology: A holistic systems paradigm.* Heidelberg, Germany: Springer Verlag.

Eckstein, S., Ahlbrecht, P., & Neumann, K. (2001). Increasing reusability in information systems development by applying generic methods. In *Advanced information systems engineering* (LNCS 2068, pp. 251-266). Berlin, Germany: Springer-Verlag.

Kim, Y. J. (2001). An implementation and design of COMOR system for OOM reuse. In *Active Media Technology, 6th International Computer Science Conference* (LNCS 2252, pp. 314-320). Berlin, Germany: Springer-Verlag.

Lai, L. F., Lee, J., & Yang, S. J. (1999). Fuzzy logic as a basis for reusing task-based specifications. *International Journal of Intelligent Systems, 14*(4), 331-357.

Madhavan, J., Bernstein, P. A., & Rahm, E. (2001). Generic schema marching with Cupid. In *Proceedings of the VLDB Conference*, Rome.

Massonet, P., & Lamsweerde, A. V. (1997). Analogical reuse of requirements frameworks. In *Proceedings of the Third IEEE Symposium on Requirements Engineering (RE'97)* (pp. 26-37).

Mili, H., Mili, F., & Mili, A. (1995). Reusing software: Issues and research directions. *IEEE Transactions on Software Engineering, 21*(6), 528-561.

OMG. (2006). *Meta-object facility (MOF™), version 2.0.*

Palopoli, L., Sacca, D., Terracina, G., & Ursino, D. (2003). Uniform techniques for deriving similarities of objects and subschemes in heterogeneous databases. *IEEE Transactions on Knowledge and Data Engineering, 15*(2), 271-294.

Peleg, M., & Dori, D. (1999). Extending the object-process methodology to handle real time systems. *Journal of Object Oriented Programming, 11*(8), 53-58.

Pernici, B., Mecella, M., & Batini, C. (2000). Conceptual modeling and software components reuse: Towards the unification. In *Information systems engineering: State of the art and research themes* (pp. 209-220). London: Springer-Verlag.

Rahm, E., & Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal, 10*(4), 334-350.

Ralyte, J., & Rolland, C. (2001). An assembly process model for method engineering. In *Advanced information systems engineering* (LNCS 2068, pp. 267-283). Berlin, Germany: Springer-Verlag.

Reinhartz-Berger, I., Dori, D., & Katz, S. (2002). Open reuse of component designs in OPM/Web. In *Proceedings of the 26th Annual International Computer Software and Applications* (pp. 19-24).

Reinhartz-Berger, I., & Sturm, A. (2004). Behavioral domain analysis: The application-based domain modeling approach. In *Proceedings of the 7th International Conference on the Unified Modeling Language (UML2004)* (LNCS 3273, pp. 410-424). Berlin, Germany: Springer-Verlag.

Rodriguez, M. A., & Egenhofer, M. J. (1999). Putting similarity assessments into context: Matching functions with the user's intended operations. In *Proceedings of CONTEXT'99* (LNAI 1688, pp. 310-323). Berlin, Germany: Springer-Verlag.

Shvaiko, P., & Euzenat, J. (2005). A survey of schema-based matching approaches. *Journal on Data Semantics, 4*, 146-171.

Soffer, P. (2005). Refinement equivalence in model-based reuse: Overcoming differences in abstraction level. *Journal of Database Management, 16*(3), 21-39.

Soffer, P., Golany, B., & Dori, D. (2003). ERP modeling: A comprehensive approach. *Information Systems, 28*(6), 673-690.

Soffer, P., Golany, B., & Dori, D. (2005). Aligning an ERP system with enterprise requirements: An object-process based approach. *Computers in Industry, 56*(6), 639-662.

Soffer, P., Golany, B., Dori, D., & Wand, Y. (2001). Modelling off-the-shelf information systems requirements: An ontological approach. *Requirements Engineering, 6*(3), 183-198.

Sturm, A., Dori, D., & Shehory, O. (2006), Domain modeling with object-process methodology. In *Proceedings of the Eighth International Conference on Enterprise Information Systems*.

Sturm, A., & Reinhartz-Berger, I. (2004). Applying the application-based domain modeling approach to UML structural views. In *Proceedings of the 23rd International Conference on Conceptual Modeling (ER2004)* (LNCS 3288, pp. 766-779). Berlin, Germany: Springer-Verlag.

Sutcliffe, A., & Maiden, N. A. (1998). The domain theory for requirements engineering. *IEEE Transactions on Software Engineering, 24*(3), 174-196.

Wenyin, L., & Dori, D. (1998). Object-process diagrams as an explicit algorithm specification tool. *Journal of Object-Oriented Programming, 12*(2), 52-59.

Zhang, Z., & Lyytinen, K. (2001). A framework for component reuse in a meta-modelling-based software development. *Requirements Engineering, 6*(2), 116-131.